NTNU

Norwegian University of
Science and Technology

A Virtual Security Net and Soft Motion Controller for a 7-DOF Redundant Cooperative Robotic Manipulator

Håkon Grøtte

Master of Science in Cybernetics and Robotics
Submission date: June 2018
Supervisor: Jan Tommy Gravdahl, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Preface

This thesis is made as a completion of the Master of Science education in Cybernetics and Robotics at NTNU. The research has been made in collaboration with SINTEF Ocean and the Department of Engineering Cybernetics at NTNU.

Several persons and institutions have contributed to this master thesis, both academically and practically. I would like to thank my supervisor at NTNU, Jan Tommy, for contributing with constructive feedback. Furthermore, I want to thank Franka Emika for providing rapid aid to questions regarding their product and SINTEF Ocean for providing a high-standard workplace, the Panda robotic manipulator and some high-value nutrition for the student. Finally, John Reidar Mathiassen from SINTEF deserves a significant acknowledgment for being available and assistive throughout the entire research period.

The following paragraph describes the implementation and code the author has found existing libraries for. However, significant understanding and modification was required to tailor the APIs to fit this thesis's problems.

The path planning and obstacle avoidance algorithm, described in section 4.1, was implemented by Grøtte in [1]. The code required to solve the Quadratic Programming problem is mainly created by Mattingley and Boyd [2].

The remaining code is created solely by the author of this thesis. Moreover, an informative summary of this thesis' contributions is outlined in Chapter 3.

Abstract

This thesis presents the study and implementation of a virtual security net for a cooperative robotic manipulator. The input to the security net includes a goal-position and orientation for the end-effector paired with the desired approach direction. The security net ensures that the corresponding output is a safe trajectory from start pose to goal pose that the end-effector successfully tracks. Moreover, the security net consists of a global path planning and obstacle avoidance method, a trajectory generation technique based on waypoints, and a reliable soft motion controller that possesses considerable tuning capabilities. Comprehensive literature studies have been made in several relevant fields, and this has formed a basis for the development process of the security net. In brief details, the path planning algorithm is based on a distance transform (DT) on a three-dimensional workspace to avoid obstacles and control the speed of the robot. Moreover, the graph search algorithm A^* is used in conjunction with the DT to calculate an optimal path for the robot in cluttered environments. The trajectory generation technique creates cubic a B-spline which satisfies C^2 continuity, which a velocity guidance logic is built on. Finally, the soft motion controller is defined as a Quadratic Programming problem in order to conveniently minimize tracking errors and induce penalties on undesirable behaviors, as well as providing a reliable and simple way to handle robot constraints.

Keywords: *robotic manipulator*, *obstacle avoidance*, *path planning*, *distance transform*, *A* algorithm*, *motion control*, *trajectory generation*, *guidance system.*

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation and objective

The two paragraphs antecedent to the block diagram in this section are inspired by [1].

Robotic systems have been used in the industry for decades. Hence, robots have become an integral part of modern society. Until recently, they were primarily designed to complete programmed, repetitive tasks. The reasons for expanding the workspace of which robots can operate are many, including economic and human safety; it is desirable to have intelligent robots that can cooperate with humans and operate without imposing a health risk to humans within the robot's physical reach. For these reasons, numerous strategies have been developed for robots to help them process information and adapt to dynamic environments. Aspects of these strategies include perception and modeling, decision making, collision avoidance and path planning, and motion control.
The term robot can convey many different meanings in the mind of the reader. In this report, it means an industrial robot, conventionally denoted a robotic manipulator, see Figure 1.1. In the case of robotic manipulators, the structure of the robot adds to the complexity of the proposed strategies in the literature. Consequently, the scope of this thesis commences with the main focus on the manipulator's end-effector, and how it can be safely moved to its destination. Moreover, the manipulator's joints are currently not considered in the obstacle avoidance problem.

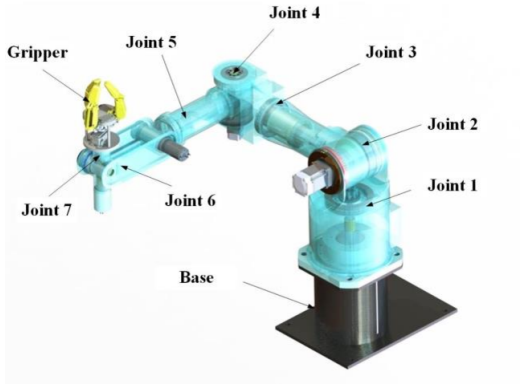


Figure 1.1: Example of a 7 joint robotic manipulator [3]

This thesis is part of SINTEF's Neodroid project, where the long-term goal is to create a reality-ready robot brain in virtual reality. Moreover, the Neodroid project wants to train a neural network AI in virtual reality first, and then continue training on an actual robot. After this transition, it is desirable to have a robust virtual security net to safely develop the AI on. The remaining parts of this paragraph attempt to introduce some desired behavioral attributes of such a security net. A learning robot that can perceive its surroundings in 3D needs to be able to explore and learn safely in its environment, even in the presence of perception errors and action errors. Perception errors include inaccurate 3D mapping and mapping obstacles where none exist. Action errors include deciding upon grasping actions at erroneous positions, due to errors in interpretation by the robot brain. To safely handle all of these types of errors, the robot will operate with a virtual security net consisting of a 3D perception and action components that move the robot efficiently and safely to the target pose, with minimal risk of collision. The robot shall move slowly near potential obstacles, so that a sensitive torque limit may be used to determine whether there actually is an obstacle worth stopping for.

The block diagram in Figure 1.2 presents a descriptive overview of how the different parts of the security net are implemented and how they form the total system when combined.

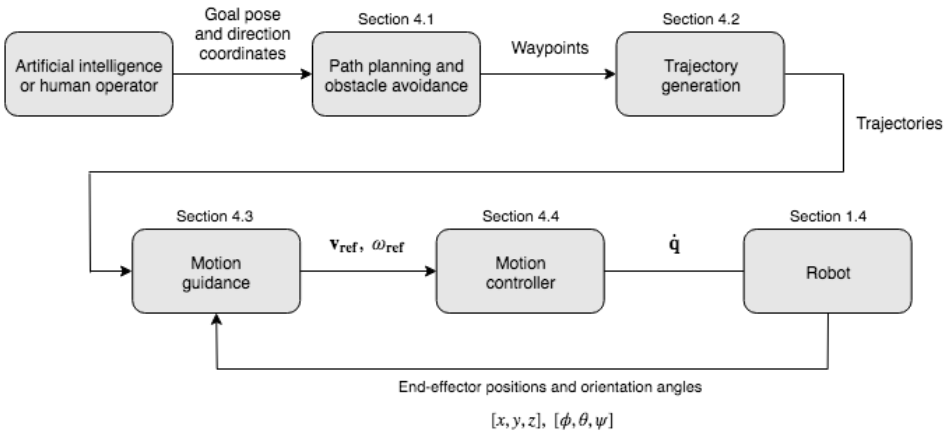


Figure 1.2: Block diagram of system

1.2 Outline

The thesis is organized as follows. The reader interested in the background material, consisting of an extensive literature study followed by a description of the robotic manipulator used, as well as relevant theory can read Chapters 1 and 2, respectively. Otherwise, the reader is encouraged to continue from Chapter 3. Chapter 3 is a terse summary of this thesis's contributions to the scientific field. Chapter 4 thoroughly describes the implementation procedure for all methods found in the block diagram in Figure 1.2. These methods include path planning and obstacle avoidance in Section 4.1, trajectory generation in Section 4.2, motion guidance in Section 4.3, and the motion controller in Section 4.4. Chapter 5 introduces experiments and results for the security net, in addition to particular methods that deserved individual testing. All experiments and results are discussed in Chapter 6. Chapter 7 and 8 describes future work that can be looked into and concluding remarks, respectively. Finally, some appendices with both pertinent and obsolete contents can be found in the end part of this thesis.

1.3 Related work

1.3.1 Path planning and obstacle avoidance

Most work related to path-planning and obstacle avoidance for the robot's end-effector was made by the same author that conducted this report. A detailed description of that work can be found in [1]. The idea is to implement the procedure derived there and test the robot's ability to safely move the end-effector from a starting pose to a goal pose. Consequently, no literature study is made for the path

planning and obstacle avoidance fields in this report. A summarized description of the method is outlined in chapter 4.

1.3.2 Position vs Velocity Control for end-effector

The following study is outlined to form the basis towards an opinion regarding whether position or velocity control is best suited for this thesis.

Foka and Trahanias introduce a methodology for avoiding obstacles by controlling the robot's velocity in [8]. They argue that without velocity control, the robot has to make detours or follow a suboptimal path in order to avoid collision avoidance with obstacles. On the other hand, especially considering dynamic environments, they state that the robot can in many cases avoid making the aforementioned suboptimal actions if it can either increase its speed to bypass the obstacle or decrease its speed to let the obstacle move away from the robot. They use a Partially Observable Markov Design Process to control the movement of the robot and provide several results of their proposed methodology.

In [9], Wardani et al. try to optimize trajectory length and energy consumption when avoiding obstacles. Additionally, they consider nonconstant velocity to make the problem more substantial and connected to real-life scenarios. They argue that in order to avoid obstacles smoothly, the robot needs to control its velocity. The report's main results showcase that their application of penalty parameters play an important role in obtaining optimal results; there exist penalty parameter values that are the most optimal for different systems.

In [10], Belkhouche states that navigation in dynamic environments requires the knowledge of obstacles' velocities. Furthermore, the paper says that for dynamic environments it a necessity to control the speed of the robot. He tries to solve the path planning problem in dynamic environments by reducing arbitrary moving objects to stationary objects. His solution combines linear navigation laws with the notion of the virtual plane[1] and velocity, as well as orientation windows.

Owen and Montano address the motion planning challenge for dynamic environments by mapping it into a velocity space in [11]. The method is applied to robots subject to both kinematic- and dynamic constraints. An immediate benefit of using velocity space is that it yields the opportunity to directly calculate velocity commands for the robot. They provide several experiments and their results show promise.

In [12], Zelanak et al. argue why velocity control is the clear choice for reactive robot motion: Velocity control provides smoother motion, is more suitable for low control frequencies, more robust to control signal variation, and it reduces collision forces more effectively. The primary advantage of position control is inherent safety with regards to signal delays. To support these points, the paper presents

[1] The virtual plane is a transformation of an observer that allows the mapping of moving obstacles to stationary obstacles.

several experiments based on a cooperative manipulation task performed by a stiff robotic manipulator. They conclude their paper by stating that velocity control is more natural than position control when humans interact with compliant robots; it reduces contact forces, and it is shown to be highly robust to slow or randomly-delayed control signals. In addition, they mention one drawback with velocity control: If a software glitch or a severe network delay occurs, it's possible for the hardware-level servo controllers to continue moving the robot, posing a danger both to the robot and its environment. Under position control, the robot will stop at the last commanded position, which is inherently safer.

The following paragraph is a terse discussion based on the preceding study. Furthermore, a decision regarding control method is made.

Considering the fact that the long-term goal of the Neodroid project is to develop a cooperative robot, in addition to the preceding study, it seems that velocity control is best suited for this thesis. Furthermore, when controlling velocity it is straightforward to control the kinetic energy the robot has, utilizing the formula

$$E_k = \frac{1}{2}mv^2. \tag{1.1}$$

This becomes especially important when humans consistently are within the robot's physical reach.

1.3.3 Trajectory generation

The following literature study presents popular methods of trajectory generation to achieve continuous profiles for the robot motion. Commencing is a terminology lecture describing the difference between a path and a trajectory in this context, inspired by Hlavác's presentation [13].

A path consists of ordered waypoints in the space[2], which the robot's end-effector should follow. Moreover, a path is usually planned globally taking into account obstacle avoidance similar to what was done in [1], outlined in section 4.1. On the other hand, a trajectory "approximates" the desired path waypoints, usually by a class of polynomial functions. Trajectories do not need global information. Furthermore, it is specified and designed locally, often with parts of the path covered by individual trajectories. In this context, it is desired that pieces of trajectories join smoothly, which induces that a single trajectory design takes into account only neighboring trajectories from the path. It is desirable to have a continuous profile in position, velocity, acceleration, and jerk.

In [14], Horsch and Jüttler describe an algorithm for interpolation of Cartesian positions by a rational spline motion. They propose an interpolation scheme based

[2]here: operational space = Cartesian space

on cubic spline functions and possesses the following important features: Each segment of the spline motion results from a local construction of neighboring positions (waypoints) and all computations are constructed to support real-time calculations. Moreover, the construction yields a rational C^1-spline motion, which produces continuous velocity profiles. Their research focuses on industrial robots, in which it is often desirable to set intermediate end-effector poses between start- and goal pose. The spline segments are constructed by using cubic Bernstein polynomials, which are represented in B-spline form.

In [15], Sencer and Shamoto propose a trajectory for Cartesian motion systems that utilizes a corner-smoothing algorithm in order to satisfy C^2 continuity transitions and minimum curvature geometry at junction points of consecutive segments. Moreover, their proposed method is designed to allow smooth movements between linear so-called $G01-$segments, without having to temporarily stop the motion at sharp corners. The algorithm fits minimum curvature quintic (5^{th} order) B-splines to blend adjacent straight lines together. Additionally, the cornering error is controlled analytically allowing the user to set the desired cornering tolerance. To generate smooth feed motion along the entire path, the feed rate is reduced on the fly at high curvature sections. The jerk profile is utilized in this work.

Svejda et al. [16] deals with the problem of interpolation of generated end-effector paths for application in robotics. Their paper discusses two interpolation methods: Line interpolation with polynomial blending and cubic spline interpolation with recalculated feed rate. The latter method's ability to recalculate feed rate reduces undesirable peaks in acceleration and ensures demanded position, velocity and acceleration profiles along the trajectories. The authors argue that the former method is not very suitable for coincident points that are too close to each other when high precision is of greater importance (e.g. arc welding); it is more convenient for pick and place applications.

Gasparetto and Zanotto [17] present an analysis of an algorithm for optimal trajectory planning of robot manipulators where two possible primitives for building the trajectory are considered: cubic splines and fifth order (quintic) B-splines. Moreover, the proposed technique that was tested allows to set constraints on the robot motion, expressed as upper bounds on the absolute values of velocity, acceleration, and jerk. Their results suggest that it might be inferred that the cubic spline trajectory would be slightly preferable if very strict requirements are set on mean and maximum values of velocities, acceleration, and jerk of the robot joints. On the other hand, they remark that the behavior of the higher-order derivatives (acceleration and jerk) of the joints' trajectories turns out to be smoother when fifth-order B-splines are employed to generate the trajectory. Furthermore, the trajectory based on cubic splines features a discontinuous jerk of the robot joints. Hence, if very strict requirements are set on the trajectory smoothness, fifth-order B-splines are preferable with respect to cubic splines.

The following paragraph is a terse discussion based on the preceding study.

Based on the preceding study it seems that creating splines that ensure C^3 conti-

nuity (i.e. continuous jerk profile) is quite ambitious; the number of waypoints N used for the splines varies every time the path planning algorithm is run. Popular ways of obtaining C^3 continuity include creating up to C^2 continuity splines and reducing feed rate levels wherever necessary (e.g. in high curvature sections). It is also important to note that creating a C^{N-1} continuous curve has several instabilities for a high amount of waypoints N. Moreover, requirements placed on one stretch of high degree curves can have a very strong effect some distance away [7]. Hence, methods generating very high-degree curves are neglected. Consequently, this thesis develops a method that combines several cubic Bézier curves based on a controlled design (yielding a piecewise cubic curve called a B-spline) to ensure C^2 continuity for the entire spline. This stitching process is outlined in section 4.2.

1.3.4 Motion control

The following literature study presents prevalent methods of motion control for robotic manipulators, which will serve as a foundation for this thesis's implementation. Furthermore, the decision to use velocity control, derived in section 1.3.2, narrows the study. Commencing is a prerequisite theoretical description of the inverse-kinematics relationship for robotic manipulators, inspired by [18].

A conventional way of controlling robotic manipulators is to exploit the inverse-kinematics relationship. Consider the equation

$$\dot{x} = \begin{bmatrix} v \\ \omega \end{bmatrix} = J\dot{q}, \tag{1.2}$$

where $v \in \mathcal{R}^3$ is the translational velocity of the end-effector, $\omega \in \mathcal{R}^3$ is the rotational velocity of the end-effector, $\dot{q} \in \mathcal{R}^n$ contains the velocity of each joint. For future references, the dimension of x will be denoted m. The matrix J is called the manipulator Jacobian or Jacobian in short. At each robot configuration, it maps the joint velocity vector into the corresponding velocity of the end-effector. Multiplying both sides of (1.2) with the inverse of the Jacobian yields the inverse-kinematics relationship

$$\dot{q} = J^{-1} \begin{bmatrix} v \\ \omega \end{bmatrix}, \tag{1.3}$$

which is valid only when the Jacobian is a square matrix. For manipulators with $n \neq 6$ joints, the inverse Jacobian would have to be approximated. This is the case for redundant manipulators, which is when a manipulator has more than the minimal number of degrees of freedom required to complete a set of tasks. Consequently, the inverse kinematics problem for a redundant manipulator is ill-posed: there may exist infinitely many configurations of the robot which give the desired end-effector configuration. Since there may be an infinite number of joint trajectories which give the requisite end-effector path, additional criteria are used

to choose among them. One common solution is to choose the minimum joint velocity which gives the desired workspace velocity. This is achieved by choosing

$$\dot{q} = J^{+} \begin{bmatrix} v \\ \omega \end{bmatrix}, \tag{1.4}$$

where $J^{+} = J^{T}(JJ^{T})^{-1}$ is the Moore-Penrose generalized inverse of J. [18]

In [19], Siciliano presents a comprehensive tutorial of the literature on kinematic control of redundant manipulators. Most of the proposed approaches are based on the instantaneous or local resolution of redundancy at the velocity level through the use of the manipulator's Jacobian matrix. Global optimization techniques have also been proposed in the literature. However, Siciliano argues that they involve increased computational complexity which rules them out in practical real-time implementation for which the end-effector is continuously modified based on sensory feedback information. Another important point in purposely adopting redundancy is the avoidance of kinematic singularities, which occur when the Jacobian, at some configuration q, has a rank less than m. In this case, the manipulator loses its ability to move along or rotate about some direction of the task space, meaning that its manipulability is reduced.

The Moore-Penrose generalized inverse of the Jacobian defined in (1.4) might seem attractive in that it has a least squares property that generates the minimum norm join velocities. However, kinematic singularities are not avoided in any practical sense, since joint velocities are minimized only instantaneously and can then become arbitrarily large near singular configurations. To overcome this drawback, one might use the damped least-square inverse of the Jacobian in the form $J^{*} = J^{T}(JJ^{T} + \lambda^{2}\mathcal{I})^{-1}$, which is nonsingular in the whole workspace. Under this control, the problem becomes finding suitable values of λ which sets the weight of the minimum norm solution, $\|\dot{q}\|$, with respect to the minimum task tracking error $\|\dot{x} - J\dot{q}\|$.
In addition to the foregoing, a closed-loop algorithmic version of the open-loop solution (1.3) can be obtained if the task space vector $\dot{x}$ is replaced by $\dot{x} = \dot{x}_d + \Lambda e$, where $e = \dot{x}_d - \dot{x}$ denotes the error between the desired task trajectory $\dot{x}_d$ and the actual trajectory $\dot{x}$, and Λ is a positive definite (diagonal) matrix that suitably shapes the error convergence.
Moreover, a computationally cheaper solution is devised on the transpose of the Jacobian matrix, i.e.

$$\dot{q} = J^{T}\Lambda e. \tag{1.5}$$

A simple Lyapunov argument can show that (1.5) guarantees limited tracking errors and null steady-state ($\dot{x}_d = 0$) errors. An intrinsic advantage of this solution is that it may avoid numerical instabilities near singularities because the pseudoinverse is not required. [19]

In [20], Bi et al. try to control a redundant manipulator in consideration of multi-

ple performance criteria[3]. Moreover, the method is based on the classical gradient projection method, where different criteria are weighed by combining a task priority strategy with fuzzy inference. The authors explain the gradient projection method as follows: "on the basis of pseudo-inverse of the Jacobian matrix, a term proportional to the gradient of the criterion (a cost function H) is projected onto the null space of the Jacobian matrix so that the end-effector task is not affected." Furthermore, they argue that the gradient projection method has been proven to be the most popular and influential control strategy for kinematically redundant manipulators. Their fuzzy inference yields varying weighing criteria depending on changing urgency and importance. Hence, the task priority method becomes real-time and the resulting joint angle space is qualified to handle dynamic environments. The method is applied to a 10 DOF manipulator in a numerical study, where three performance criteria are taken into account: singularity avoidance, obstacle avoidance, and joint limitation avoidance.

Schuetz et al. [21] present a motion planning technique for redundant manipulators in uncertain environments based on tactile feedback. They argue that motion planning solely based on visual information performs poorly in cluttered environments because contacts with obstacles might be inevitable and thus a distinction between hard and soft objects has to be made. They try to use the tactile information (resulting force and torque) for minimizing contact forces while simultaneously pursuing the end-effector tasks as long as reasonable. Moreover, they project the external forces to their point of application to derive a one-dimensional system equation (per resulting force), from which they develop a suitable controller using feedback linearization with second-order dynamics. They test their proposed method on a 9 DOF manipulator with their control strategies in null space and can show a fast reaction to external forces and their significant reduction by evasive nullspace motion of the manipulator.

Chiaverini [22] presents a paper that commences by studying the application of existing singularity-robust methods to the case of kinematically redundant manipulators. Then, a new task-priority redundancy resolution (TPRR) technique is developed that, in addition to handling kinematic singularities, overcomes the effects of algorithmic singularities[4] that commonly plague kinematic singularity-robust methods. For the kinematic singularities, it is recognized that the damped least-squares solution with numerical filtering represents a good compromise between accurate tracking performance of the pseudoinverse solution and the capability of providing feasible joint velocities of the damped least-squares solution. Nevertheless, special care must be taken to handle the case of multiple kinematic singularities. As for algorithmic singularities, the TPRR technique avoids the inversion of the Jacobian matrix which is ill-conditioned in the neighborhood of the singularity, which yields continuous joint velocity solutions and accurate tracking of the end-effector task. Furthermore, it is computationally advantageous with re-

[3] e.g. maximization of joint availability, obstacle avoidance or minimization of joint torques.

[4] singular configurations at which the end-effector task and the constraint task conflict despite the redundant degrees of freedom. [22]

spect to classical TPRR techniques. Nevertheless, the paper recognizes that the use of damped least-squares inverse is problematic when both kinematic- and algorithmic singularities are expected. The method is tested on a 7 DOF manipulator to demonstrate its effectiveness.

1.4 Franka Emika's Panda robot

The robotic manipulator used in this thesis is the Panda robot created by Franka Emika, showcased in Figure 1.3. The robot arm is lightweight with impressive measurement opportunities and precision, and its design displays thought towards real-time human-robot interaction. In Franka Emika's own words: "The Arm is inspired by the agility, dexterity and sensitivity of the human arm. It is able to recognize and process even the slightest contacts to react within milliseconds. The Hand can grasp firmly and quickly for high performance manipulation. The Fingers can automatically be exchanged to optimally grasp a wide variety of objects."

Figure 1.3: Franka Emika's Panda 7 DOF robot with control box [4]

1.4.1 Control parameters specifications

The following section and all its subsections describe the physical requirements the Panda robot needs to obey. Most parts are extracted from the Franka Control Interface[5]. If a controller attempts to perform control without satisfying the underlying requirements, a fail-safe system will exit the control-loop without warning.

1.4.1.1 Joint trajectory requirements

The necessary conditions are

$$\begin{aligned} & q_{min} < q < q_{max}, \\ & -\dot{q}_{max} < \dot{q} < \dot{q}_{max}, \\ & -\ddot{q}_{max} < \ddot{q} < \ddot{q}_{max}, \\ & -\dddot{q}_{max} < \dddot{q} < \dddot{q}_{max}. \end{aligned} \tag{1.6}$$

The recommended conditions are

$$\begin{aligned} & q_{min,soft} < q < q_{max,soft}, \\ & -\dot{q}_{max,soft} < \dot{q} < \dot{q}_{max,soft}, \\ & -\tau_{jmax} < \tau_{jd} < \tau_{jmax}, \\ & -\dot{\tau}_{jmax} < \dot{\tau}_{jd} < \dot{\tau}_{jmax}. \end{aligned} \tag{1.7}$$

At the beginning of the trajectory, the following conditions should be fulfilled:

$$\begin{aligned} q &= q_d, \\ \dot{q} &= 0, \\ \ddot{q} &= 0. \end{aligned} \tag{1.8}$$

At the end of the trajectory, the following conditions should be fulfilled:

$$\begin{aligned} \dot{q} &= 0, \\ \ddot{q} &= 0. \end{aligned} \tag{1.9}$$

1.4.1.2 Cartesian trajectory requirements

The necessary conditions are

$$\begin{aligned} &-\dot{p}_{max} < \dot{p} < \dot{p}_{max}, \\ &-\ddot{p}_{max} < \ddot{p} < \ddot{p}_{max}, \\ &-\dddot{p}_{max} < \dddot{p} < \dddot{p}_{max}, \\ &q_{min} < q < q_{max}, \\ &-\dot{q}_{max} < \dot{q} < \dot{q}_{max}, \\ &-\ddot{q}_{max} < \ddot{q} < \ddot{q}_{max}, \end{aligned} \tag{1.10}$$

where the latter three conditions are derived from inverse kinematics.

The recommended conditions for Cartesian trajectories are equivalent to (1.7).

The start- and end trajectory requirements for Cartesian control, in addition to Cartesian space limit values, are omitted here because joint space trajectory is the more relevant one in this thesis.

1.4.1.3 Joint space limits

Name	**Joint 1**	**Joint 2**	**Joint 3**	**Joint 4**	**Joint 5**	**Joint 6**	**Joint 7**	**Unit**
q_{max}	2.9671	1.8326	2.9671	0.0873	2.9671	3.8223	2.9671	rad
q_{min}	-2.9671	-1.8326	-2.9671	-3.1416	-2.9671	-0.0873	-2.9671	rad
$q_{max,soft}$	2.8973	1.7628	2.8973	0.0175	2.8973	3.7525	2.8973	rad
$q_{min,soft}$	-2.8973	-1.7628	-2.8973	-3.0718	-2.8973	-0.0175	-2.8973	rad
$\dot{q}_{max}$	2.3925	2.3925	2.3925	2.3925	2.871	2.871	2.871	$\frac{rad}{s}$
$\dot{q}_{max,soft}$	2.1750	2.1750	2.1750	2.1750	2.6100	2.6100	2.6100	$\frac{rad}{s}$
$\ddot{q}_{max}$	16.5	8.25	13.75	13.75	16.5	22	22	$\frac{rad}{s^2}$
$\dddot{q}_{max}$	50000	50000	50000	50000	50000	50000	50000	$\frac{rad}{s^3}$

Table 1.1: Joint space limits

1.4.1.4 Denavit-Hartenberg parameters

Franka Emika uses the Denavit-Hartenberg convention to select frames of reference. They are as follows:

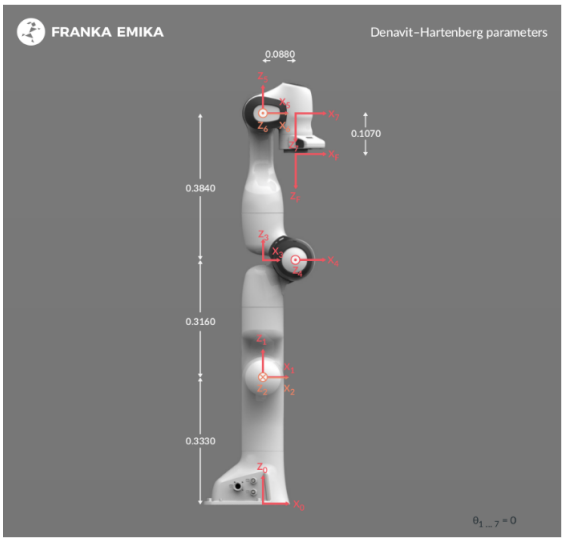


Figure 1.4: Panda's kinematic chain [5]

Joint	$a(m)$	$d(m)$	$\alpha(rad)$	$\theta(rad)$
Joint 1	0	0.333	0	θ_1
Joint 2	0	0	$-\frac{\pi}{2}$	θ_2
Joint 3	0	0.316	$\frac{\pi}{2}$	θ_3
Joint 4	0.0825	0	$\frac{\pi}{2}$	θ_4
Joint 5	-0.0825	0.384	$-\frac{\pi}{2}$	θ_5
Joint 6	0	0	$\frac{\pi}{2}$	θ_6
Joint 7	0.088	0	$\frac{\pi}{2}$	θ_7
Flange	0	0.107	0	0

Table 1.2: Denavit-Hartenberg (DH) parameters

1.5 Assumptions

Similar to the assumption made in [1], this thesis assumes that the occupancy grid mapping obstacles in three-dimensional space is manually filled with obstacles. The long-term goal of the Neodroid project[5] is to use Computer Vision to detect obstacles automatically, also in real-time operations.

[5] A SINTEF project this thesis contributes to.

Chapter 2

Theory

Sections 2.1-2.4 are taken from [1].

2.1 Occupancy-grid maps

Occupancy-grid maps represent environments as an array of cells. Each cell corresponds to an area in the physical environment and holds an occupancy value which indicates whether that area is occupied or free. The value representation in each cell can either be deterministic or stochastic. The former specifically states that a cell is occupied or available, regardless of how certain it is. This yields a boolean representation, e.g. one of the integers from the set $I = \{0, 1\}$. Conversely, the stochastic representation uses a probabilistic representation which could include sensor noise, environment knowledge etc. to weigh the probability value. Thus, its occupancy value could be all decimals in the range $p_i \in [0, 1]$.

For the 2D case, an occupancy grid has representation similar to that of an image. Hence, each cell can be considered as a pixel. In 3D, each pixel becomes a voxel.

2.2 Distance transformation

The distance transform (DT) of an occupancy-grid map provides a metric of the distance from each cell to the nearest denoted obstacle. Moreover, the DT is primarily performed on binary images or similar grid-based structures.

Multiple algorithms exist for calculating the DT of an image. This section summarizes the algorithm presented by Saito and Toriwaki [23] with the optimization proposed by Meijster et al. [24], where the Euclidean metric for computing distances

is used. Moreover, the two-dimensional Euclidean DT (EDT) problem will be described for a boolean occupancy-grid map M. Consequently, the problem of the EDT is to assign to every grid point (x, y) the distance to the nearest (here) obstacle in M. Thus, the two-dimensional output array becomes $dt[x, y] = \sqrt{EDT(x, y)}$, where

$$EDT(x, y) = MIN(i, j : 0 \leq i < m \wedge 0 \leq j < n \wedge b[i, j] : (x-i)^2+(y-j)^2). \quad (2.1)$$

The notation "$MIN(k : P(k) : f(k))$ for the minimal value of $f(k)$ when k ranges over all values that satisfy $P(k)$" is used. The minimum of the empty set is defined to be ∞, and the rule $z + \infty = \infty \ \forall \ z$ is used. Then, some calculation yields

$$EDT(x, y) = MIN(i : 0 \leq i < m : (x - i)^2 + G(i, y)^2), \quad (2.2)$$

where $G(i, y) = MIN(j : 0 \leq j < n \ \wedge \ b[i, j] : |y - j|)$.
Using those definitions, the algorithm can be summarized as follows: In a first phase each column C_x (defined by points (x, y) with x fixed) is separately scanned. For each point (x, y) on C_x, the distance $G(x, y)$ of (x, y) to the nearest obstacle point on $C_x \cap M$ is determined. In a second phase each row R_y (defined by points (x, y) with y fixed) is separately scanned, and for each point (x, y) on R_y the minimum of $(x - x')^2 + G(x', y)$ is determined, where (x', y) ranges over row R_y. For a more detailed description, see Meijster et al. in [24]. Furthermore, the authors argue that the time complexity of their proposed algorithm is $\mathcal{O}(n)$, if n is the total number of pixels.

The problem can be expanded to higher dimensions. Specifically, for a $d-$dimensional EDT, the problem must be separated into d phases, each solving a one-dimensional problem.

2.3 Graph theory

A mathematical graph is a set of nodes and edges. The nodes (also called vertices) are connected together by edges (also called links). Hence, for any graph, it is necessary to describe

- A set of all nodes.
- A set of all edges from each node.

For the graph in Figure 2.1 this becomes

- A set of all nodes $\{A, B, C, D, E\}$.
- A set of all edges from each node:
  - $A : \{A \rightarrow B, \ A \rightarrow C\}$

- $B : \{B \to A,\ B \to C,\ B \to D\}$
- $C : \{C \to A,\ C \to B,\ C \to E\}$
- $D : \{D \to B,\ D \to E\}$
- $E : \{E \to C,\ E \to D\}$

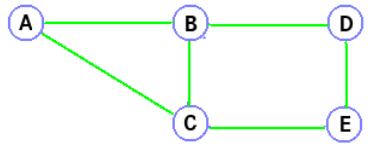


Figure 2.1: Unweighted graph with edges

A graph can also be weighted, where each edge has a respective weight (also called cost) assigned to it.

2.4 A* shortest path algorithm

A* is one of the most popular choices for pathfinding because it is fairly flexible and can be used in a wide range of contexts. It is similar to Dijkstra's algorithm in that it is guaranteed to find the shortest path. Furthermore, it utilizes a heuristic to select vertices closer to the goal, similarly to what Greedy Best-First-Search does.

A* starts off with a weighted graph G, the starting node *start* and the goal node *goal* as inputs. With *start* as the root node, A* creates a set $frontier$ that holds all nodes that are candidates for examining. Additionally, A* creates a set *closed* containing the nodes that have already been examined. A* keeps track of the generated path(s) by assigning parent-successor pairs for each node examined.

There is a loop that repeatedly pulls out the best node n from $frontier$ and examines it. If n is the goal node, A* is finished. If not, n is added to the *closed* set and its neighbors are examined. If a neighbor node a is already in *closed*, the cost calculated for the current path is compared with the previous cost assigned to a. If the new cost is better, a is added to or updated in $frontier$ with the improved cost. The cost function $f(n)$ in conventional A* is defined as

$$f(n) = g(n) + h(n), \tag{2.3}$$

where $g(n)$ represents the exact cost of the path from *start* to any node n (Dijkstra's cost function) and $h(n)$ represents the heuristic estimated cost (usually Euclidean distance) from n to *goal* (cost function in Greedy Best-First-Search).

As a preface to the following in-depth pseuedocode the following definitions are made:

- **frontier** A priority queue that stores pairs consisting of a node with a corresponding priority. It is ordered such that it returns the node with the lowest[1] priority value.
- **closed** A dictionary containing nodes that have been visited as keys and the current best (lowest) cost of the nodes as values.
- **camefrom** A dictionary storing parent-successor pairs for all nodes. Used to recalculate the shortest path found.
- **start** The starting node.
- **goal** The goal node.

A* shortest path

```
Initialize frontier, closed and camefrom with start;
while frontier not empty do
    current = lowest priority item from frontier;
    if current is goal then
        break;
    for neighbors of current do
        cost = g(current) + movementcost(current, neighbor);
        if neighbor not in closed OR cost less than closed(neighbor) then
            closed(neighbor) = cost;
            priority = cost + h(neighbor, goal);
            frontier(neighbor) = priority;
            camefrom(neighbor) = current;
    end
end
```

Algorithm 1: In-depth psuedocode for A* shortest path (general implementation)

After the algorithm has converged, the suggested path is reconstructed using **camefrom**.

2.5 Bézier curves

The following theory is inspired by [25] and [6].

In short, Bézier curves are the result of linear interpolation[2]. In mathematical terms, Bézier curves are a form of parametric functions. Furthermore, they are

[1] i.e. the best candidate.

[2] In mathematics, linear interpolation is a method of curve fitting using linear polynomials to construct new data points within the range of a discrete set of known data points.

polynomials of s, with the value of s fixed being between 0 and 1.[3] A Bézier curve of order n ($n = 1$ for linear, $n = 2$ for quadratic etc.) is composed of Bernstein basis polynomials of degree n in addition to a set of $n+1$ control points P_0 through P_n (see Figure 2.2). The first and last control points are always the endpoints of the curve; however, the intermediate points (if $n > 1$) generally do not lie on the curve. The explicit forms of a Bézier curve of order 1, 2 and 3 are as follows

$$B(s) = (1-s)P_0 + sP_1, \tag{2.4}$$
$$B(s) = (1-s)^2P_0 + 2(1-s)tP_1 + s^2P_2, \tag{2.5}$$
$$B(s) = (1-s)^3P_0 + 3(1-s)^2tP_1 + 3(1-s)s^2P_2 + s^3P_3, \tag{2.6}$$

where $s \in [0,\ 1]$. The general definition can be expressed explicitly as follows:

$$B(s) = \sum_{i=0}^{n} \binom{n}{i} (1-s)^{n-i} s^i P_i, \tag{2.7}$$

where $\binom{n}{i}$ are binomial coefficients.

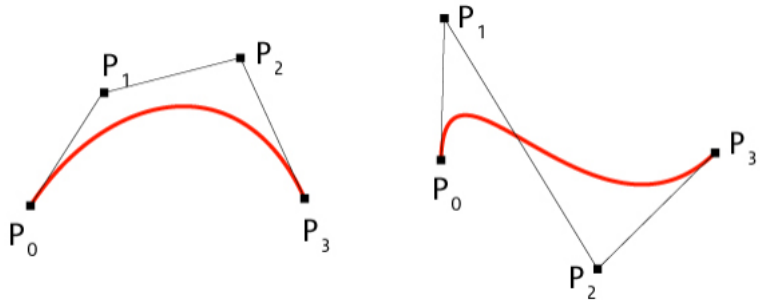


Figure 2.2: Example of two Bézier curves of degree 3.[6]

2.6 B-splines

The following theory is inspired by [25] and [7].

B-splines (or basis-splines) are piecewise continuous polynomial interpolation curves where the "single curve" is built by performing polynomial interpolation over a set of points. For instance, a cubic B-spline defined by twelve points will have its curve

[3] Usually t is used as a parameter. Here s is used to avoid confusion between function parameter and time parameter t.

built by evaluating the polynomial interpolation of four points, and the curve can be treated as a lot of different sections, each controlled by four points at a time, such that the full curve consists of smoothly connected sections defined by points $\{1, 2, 3, 4\}$, $\{2, 3, 4, 5\}$, ..., $\{8, 9, 10, 11\}$, and finally $\{9, 10, 11, 12\}$, for eight sections. In order to properly distinguish Bézier curves and B-splines, consider the difference to be this:

- for Bézier curves, the curve is defined as an interpolation of points
- for B-Splines, the curve is defined as an interpolation of curves.

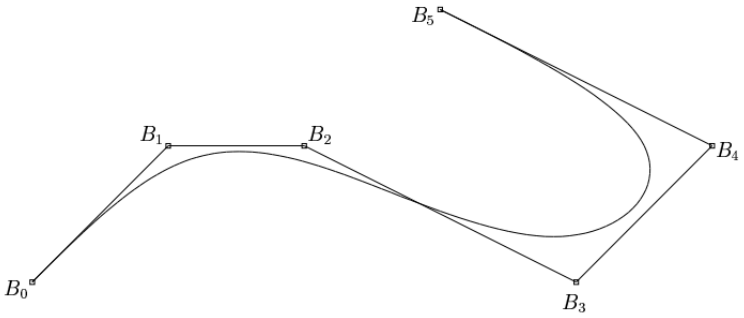


Figure 2.3: A relaxed[4]uniform cubic B-spline curve based on points B_0 through B_5. Moreover, this particular spline is interpolated by using two cubic Bézier curves [7].

[4]A cubic spline is relaxed if its second derivative is zero at each endpoint.

Chapter 3

Contributions

In this section, a summarized version of this thesis's contributions to the field is presented. There is a request for a reliable and fail-safe implementation regarding path planning, obstacle avoidance and motion control for robotic manipulators. The purpose of this thesis's implementation is to create a robust virtual security net that can make the robot cooperate in complex environments without posing a significant threat to anyone or anything within its range. The challenges were significant; for redundant manipulators, in particular, the research concerning velocity control is more limited than that of position control. Several reasons exist for this. Of huge importance is the fact that redundant manipulators have infinite joint configurations for a single end-effector pose. In summary, it is desired to have a solution that can handle practical tasks like "pick and place" operations, while simultaneously cooperating with a complex environment.

To obtain a cooperative robot that can also handle specific pick and place operations, this thesis suggests a solution that combines several methods to obtain a security net that successfully and safely moves the robot end-effector from a starting pose to a goal pose. With the block diagram depicted in Figure 1.2 in mind, the contributions of this thesis are summarized as follows:

- Path planning and obstacle avoidance
  - Utilizes a Euclidean distance transform (EDT), which calculates the distance to obstacles for all defined points of a volume surrounding the robot.
  - A special version of the A^* shortest path algorithm uses the EDT as input and finds respective waypoints the end-effector should follow.
- Trajectory generation
  - Based on the waypoints from the path planning, a relaxed cubic B-spline that satisfy C^2 continuity on the entire spline is generated.

- The spline contains a curve parameter s that informs how long the entire trajectory is in addition to how far the end-effector has traveled on it.

- Motion guidance

  - The direction of the translational velocity- and the rotational velocity references are inspired by line-of-sight (LOS) guidance and are dependant on the trajectory by using the curve parameter s to calculate most references.

  - The magnitude of the velocity is dependant on the value of the EDT at the end-effector's positions.

- Motion controller

  All the preceding bullet points are combined to form an input to the most significant contribution of this thesis, namely the motion controller. It ensures that a proper motion is exerted on the respective joints when following the trajectory. Furthermore, the controller solves a Quadratic Programming problem for each iteration, defined as

$$
\begin{aligned}
\text{minimize} \quad & J_c = \gamma \left\| v - v_{ref} \right\|_2^2 + \kappa \left\| \omega - \omega_{ref} \right\|_2^2 + \mu \dot{q}^T \dot{q} + \chi g(\dot{q}, q) \\
\text{subject to} \quad & \begin{bmatrix} v \\ \omega \end{bmatrix} = J\dot{q}, \\
& -\dot{q}_{max} \leq \dot{q} \leq \dot{q}_{max}, \\
& -\ddot{q}_{max} \leq \ddot{q} \leq \ddot{q}_{max}, \\
& -\dddot{q}_{max} \leq \dddot{q} \leq \dddot{q}_{max}.
\end{aligned}
\tag{3.1}
$$

  There are several terms in the cost function that deserve an explicit explanation. The solver calculates the most optimal change in the robot's joint velocities that allows the end-effector to follow desired velocity references in Cartesian space. This concludes the purpose of the $\gamma \left\| v - v_{ref} \right\|_2^2 + \kappa \left\| \omega - \omega_{ref} \right\|_2^2$ terms in the cost function, where γ and κ are weighting constants. μ and χ are also weighting constants. The $\mu \dot{q}^T \dot{q}$ term is added to impose restrictions on the joint motions. The final term $\chi g(\dot{q}, q)$ is included to guarantee that the physical joint limits of the respective joints are not violated. This term was necessary because the joint position constraint $q_{min} \leq q \leq q_{max}$ does not work in practice when optimizing with respect to joint velocities. [1] By defining the optimization variable as $z := \begin{bmatrix} v^T & \omega^T & \dot{q}^T \end{bmatrix}^T$, the QP problem can be put on standard form, which yields

[1] The reason for this is clarified in section 4.4.3.

$$\begin{aligned} \underset{z}{\text{minimize}} \quad & z^T Q z + q_0 z \\ \text{subject to} \quad & A_{eq} z = 0, \\ & A_{ineq} z \leq b. \end{aligned}$$

The observant reader quickly realizes that $q, \ddot{q}$ and $\dddot{q}$ have to be approximated using $\dot{q}$. This is part of the reason the joint angle constraint fails.

A detailed implementation description of all methods, as well as necessary experiments and discussions, follow in the upcoming chapters.

Chapter 4

Implementation

4.1 Path planning

The following section frequently use terms that have been thoroughly described in sections 2.1-2.4.

In order to minimize the end-effector's risk of colliding with its environment, a natural way to commence is to create a reference path that is safe yet effective. This thesis set out to test the path planning algorithm developed by Grotte in [1]. The method starts with defining a deterministic occupancy-grid around the robot and maps corresponding obstacles into it. Another term for such a construct is a voxel grid. The method proceeds by calculating the squared Euclidean Distance Transform (SEDT) for the entire voxel grid. Figure 4.1 displays a graphical representation of the output from the SEDT algorithm in 2D. The SEDT output is used to control the velocity of the end-effector; it is based on the distance to the closest obstacle at every given time.

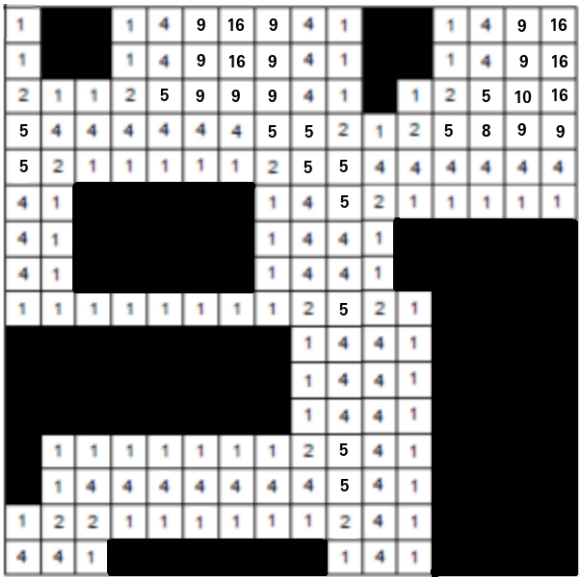


Figure 4.1: 2D grid example of calculated SEDT for all pixels. [1] Obstacles are marked black.

In order to create a path between a start coordinate and a goal coordinate, the A^* shortest path algorithm is used. Algorithms like A^* are convenient to run on graph-based structures. Hence, the voxel grid is transformed to a graph structure with a maximum of 26 neighbors per voxel, which yields much more optimal paths than using only 8 neighbors [1]. The A^* version used takes the time of traversing into account in its cost function. As previously mentioned, the SEDT is used to control the velocity of the end-effector. Consequently, if the cost function for A^* is defined in such a way as to maximize the time efficiency of traversing, the trajectory will prioritize paths that are further away from obstacles, because there the robot is allowed to move at greater speeds. Formulated in mathematical terms, in addition to the Euclidean distance heuristic term $h(n)$ defined in section 2.4, where n is a node in the graph, the cost function $f(n)$ will include the term

$$t(n) = K \times time = K\frac{distance}{speed} = K\frac{1}{v} = \frac{K}{min(v_{max}, v_{min} + d_{obj}(n))}, \tag{4.1}$$

which represents the time of traversing at each node in the graph. v_{max} and v_{min} are the maximum and minimum allowed speeds, respectively, and $d_{obj}(n)$ is the distance to the closest obstacle from node n. The final cost function thus becomes

$$f(n) = t(n) + c \times h(n) = \frac{K}{min(v_{max}, v_{min} + d_{obj}(n))} + c \times h(n), \tag{4.2}$$

where K and c are weighting terms.

4.1.1 This thesis's contributions to the path planning procedure

The implementation of the path planning algorithm in [1] only supports finding a path from a starting node to a single goal node. For this thesis, it is of interest to connect multiple temporary goals before a final main goal is reached. Hence, this expansion had to be developed in this thesis. The incentive for adding this extension is that it might help the robotic manipulator to avoid joint limits and in general cumbersome, ineffective joint configurations. Consider the base joint of the robot being rotated 2.35 radians $\approx 135°$, and that the end-effector is in the lower-right plane, see Figure 4.2. Now imagine specifying the end-effector goal coordinate at the blue circle. Without clever path planning, this might set the desired end-effector path as the shortest path to the goal coordinate, giving a high probability that the base joint will move towards its max joint limit defined in Table 1.1. With the new feature of allowing multiple goals, the path could utilize the arbitrarily positioned intermediate waypoints 1 and 2 in Figure 4.2 to avoid unnecessary joint pressure. In terms of Figure 4.2, the desired end-effector path would be similar to the green path in opposition to the undesired, red path.

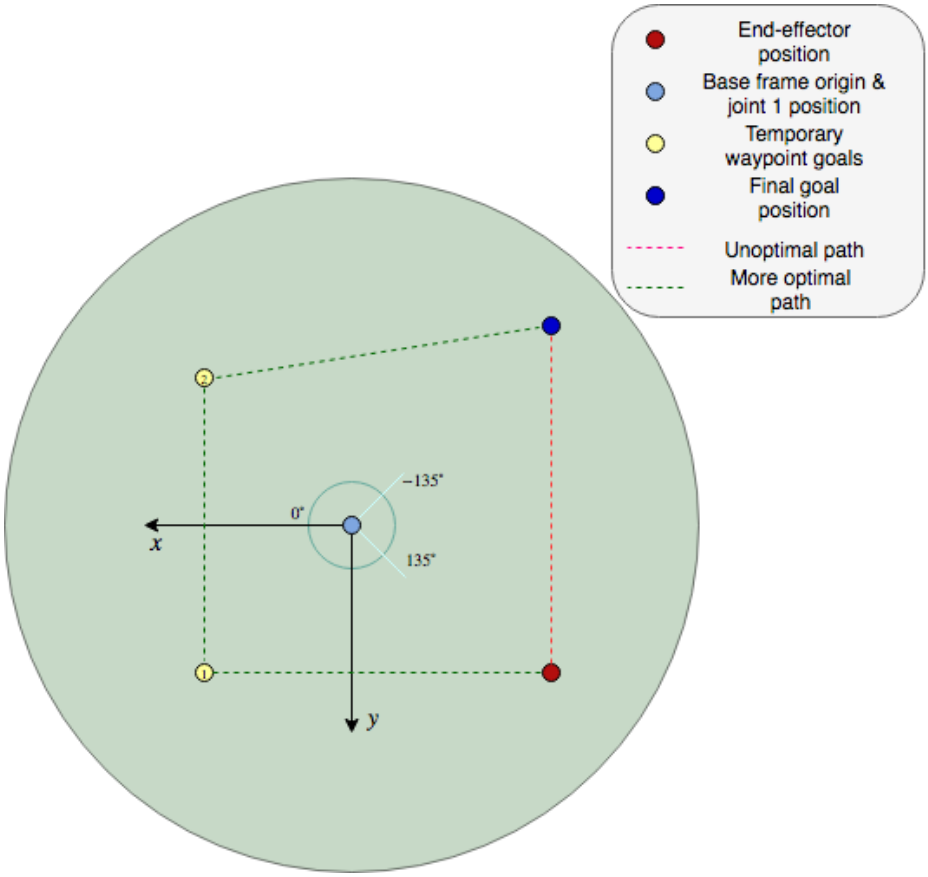


Figure 4.2: Task space of robot and coordinate axes of robot's base frame seen from above; the z-axis points upwards (out of paper).

4.2 Trajectory generation

The following sections frequently use terms that have been introduced and described in sections 2.5-2.6.

4.2.1 Motivation

During early development of the security net, a now obsolete motion guidance was created that used discrete waypoints as input, see appendix A. It quickly became apparent that the velocity guidance based solely on discrete waypoint positions made the task of ensuring smooth robot movements excessively cumbersome. If a sufficiently smooth parametrically-defined trajectory $C(s)$ is generated, where s is the curve parameter, the velocity guidance can be simplified to making the

end-effector follow a (desirably) continuous trajectory, where multiple techniques exist in the literature. Furthermore, the s parameter can be utilized to calculate the distance of the entire trajectory the end-effector has traveled, and the distance remaining.

4.2.2 B-splines with continuous acceleration profile

The trajectory generation technique implemented is inspired by Baker's lecture at UCLA[1] [7], where a C^2 continuous B-spline is generated by combining piecewise relaxed cubic Bézier curves. A cubic spline curve is *relaxed* if its second derivative is zero at each endpoint. The reason for imposing the relaxed end conditions on the Bézier curves is to ensure that the final B-spline satisfies C^2 continuity (i.e. continuous acceleration profile). In order to explain the technique, start by considering two cubic Bézier curves that can be glued together but otherwise are not well matched. Let the first and the second curve have control points P_{0-3} and Q_{0-3}, respectively. Moreover, let $P_3 = Q_0 := S$. The result is shown in Figure 4.3.

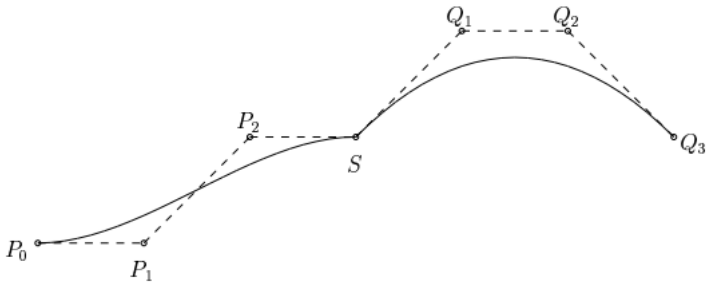


Figure 4.3: Matching endpoints: A coarse gluing. [7]

Now impose the condition that the first derivative match at the point of gluing. This is satisfied with the requirement $S - P_2 = Q_1 - S$, or equivalently, that S is the midpoint of the line segment P_2Q_1. The result is shown in Figure 4.4.

[1] University of California, Los Angeles.

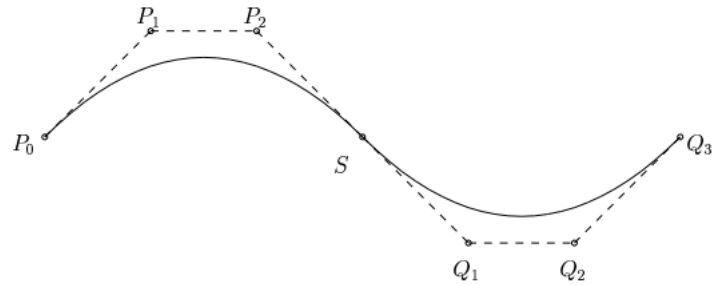


Figure 4.4: Matching endpoints and first derivative: An improved gluing. [7]

This certainly looks more smooth. On the other hand, to obtain an even smoother join, the curvature should be continuous. Because the curvature can be expressed in terms of the first and second derivatives, continuity of curvature can be achieved by matching second derivatives, as well as first derivatives, at the point of gluing. At S, the second derivatives of the Bézier curves are $6(P_1+2P_2+S)$ and $6(S-2Q_1+Q_2)$. Set the terms equal to each other, and negate both sides to obtain

$$2P_2 - P_1 = 2Q_1 - Q_2. \tag{4.3}$$

The left-hand side corresponds to a particular point A_+ on the line through P_1 and P_2. In fact, $A_+ = 2P_2 - P_1 = P_2 + (P_2 - P_1)$, as displayed in Figure 4.5. Similarly, $A_- = 2Q_1 - Q_2$. A_+ and A_- will be called the right- and the left apex, respectively.

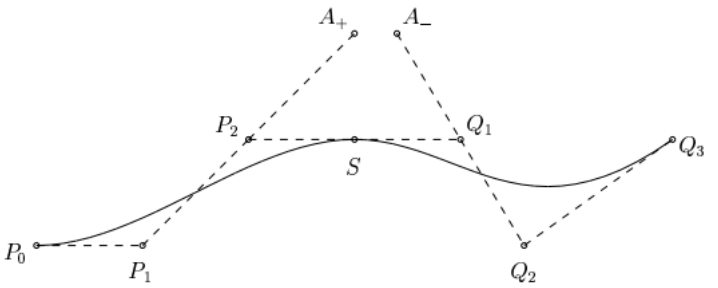


Figure 4.5: A gluing almost matching second derivatives. [7]

The Figure displays that the two apexes are not equal, and thus (4.3) is not satisfied. Conversely, Figure 4.6 shows the respective gluing where (4.3) is satisfied. Moreover, an $A-$frame is a figure with points as indicated, in which S is the midpoint of the line P_2Q_1, P_2 is the midpoint of the line P_1A, and Q_1 is the midpoint of Q_2A. This is the foundation for the observation: If two Bézier curves are joined at a point S, both their first and second derivatives match at S if and only if their control polygons fit an $A-$frame.

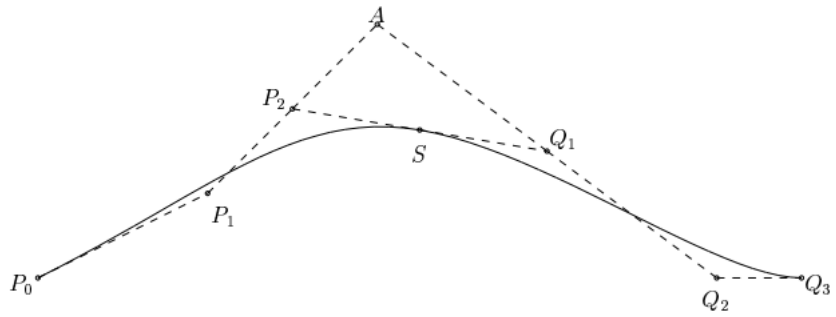


Figure 4.6: Matching endpoints, first- and second derivative: A satisfactory gluing. [7]

Now that the requirements for joining two cubic Bézier curves have been outlined, a general requirement for creating a B-spline that joins multiple relaxed cubic Bézier splines is necessary. The method is as follows: Specify a control polygon of $B_0, B_1, ..., B_n$. Divide each leg of the control polygon into thirds by marking two "division" points. At each B_i except the first and last, draw the line segment between the two nearest "division" points, and call the midpoint S_i. This creates an $A-$frame with B_i at the apex, see Figure 4.7. Finally, sketch a cubic Bézier curve from each point S_i to the next, using as Bézier control points the four points S_i, two "division" points", and S_{i+1}.

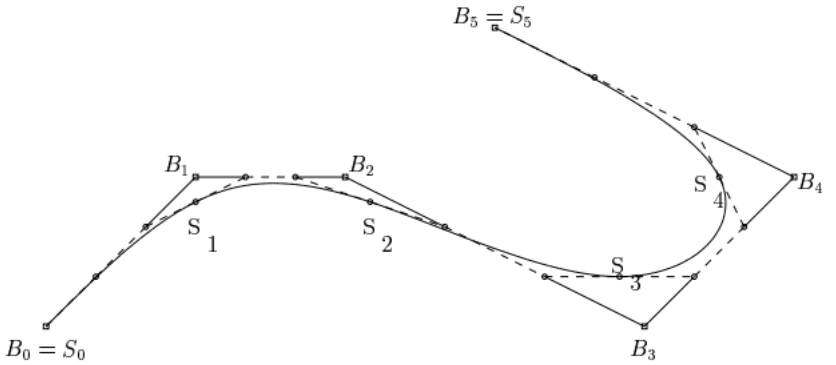


Figure 4.7: Control polygon with respective "division" points, S_i and relaxed cubic B-spline. [7]

A computer method can be summarized as follows:
Given $B - spline$ control points $B_0, ..., B_n$, calculate $S_i = \frac{1}{6}B_{i-1} + \frac{2}{3}B_i + \frac{1}{6}B_{i+1}$, for $i = 1, ..., n-1$, and let $S_0 = B_0,\ S_n = B_n$. There are n cubic Bézier curves to plot. Curve $\#i$ has control points $S_{i-1}, \frac{2}{3}B_{i-1} + \frac{1}{3}B_i, \frac{1}{3}B_{i-1} + \frac{2}{3}B_i$ and S_i. The resolution of the respective Bézier curves is determined by the user. Reasonable starting values could be $s = \{0, 0.05, 0.10, ..., 0.95, 1\}$.

Finally, let $p_i(s)$ be the ith Bézier curve ($0 \leq s \leq 1$). These N curves can be combined into a single curve $P(t)$ for $0 \leq s \leq N$ by letting

$$P(t) = p_i(s - (i-1)) \text{ for } i - 1 \leq s \leq i, \text{ where } i = 1, ..., N. \tag{4.4}$$

Then $P(t)$ is a relaxed cubic spline curve. Furthermore, $P(t)$ is called a *uniform* curve because its domain $0 \leq s \leq N$ was made from intervals all of length 1.

4.2.3 Closest point on a spline curve

In order to design proper velocity guidance based on the trajectory, it is essential to know which point on the B-spline curve is closest to the end-effector position at all times. Furthermore, the computation needs to be applicable in real-time, in addition to being robust. Obtaining an optimal s^* that minimizes the distance between an arbitrary *linear* function and the end-effector is straightforward. However, for the nonlinear cubic B-spline, this is an incredibly complex problem and the easiest solution is a numerical approach. Several methods exist in the literature; Kamermans [25] uses a binary search method, Chen et al. [26] use an algebraic method with a complex root-finding algorithm. Here, a solution inspired by Wang et al. in [27] is implemented due to its promising robustness and real-time design criteria. The procedure uses a two-step method that exploits the complementary strengths of two optimization techniques: Newton's method and quadratic minimization.

4.2.3.1 The problem

Let the cubic B-spline curve in three-dimensional space be expressed as

$$(x(s), y(s), z(s)), \ 0 \leq s \leq L, \tag{4.5}$$

where s denotes the arc length, L is the arc length of the entire spline curve, and $x(s)$, $y(s)$, and $z(s)$ are the cubic spline functions with equally spaced breakpoints $\{s_0, s_1, ..., s_n\}$ with $s_0 = 0$ and $s_n = L$. At each time step of the motion control, the end-effector attains a new position in Cartesian coordinates. Let $p_0 = (x_0, y_0, z_0)$ be the position of the end-effector. The square Euclidean distance between p_0 and position $(x(s), y(s), z(s))$ on the spline curve is

$$D(s) = (x(s) - x_0)^2 + (y(s) - y_0)^2 + (z(s) - z_0)^2. \tag{4.6}$$

The value s^* that minimizes $D(s)$ determines $p_1 = (x(s^*), y(s^*), z(s^*))$, the closest point to p_0 on the cubic spline curve.

4.2.3.2 Quadratic minimization method

Quadratic minimization uses quadratic interpolation to minimize a one-variable function. Suppose that $\tilde{s}_1$, $\tilde{s}_2$, and $\tilde{s}_3$ are given initial estimates of s^*. The quadratic polynomial that interpolates $D(s)$ at these estimates is given by

$$P(s) = \frac{(s-\tilde{s}_2)(s-\tilde{s}_3)}{(\tilde{s}_1-\tilde{s}_2)(\tilde{s}_1-\tilde{s}_3)}D(\tilde{s}_1) + \frac{(s-\tilde{s}_1)(s-\tilde{s}_3)}{(\tilde{s}_2-\tilde{s}_1)(\tilde{s}_2-\tilde{s}_3)}D(\tilde{s}_2) + \frac{(s-\tilde{s}_1)(s-\tilde{s}_2)}{(\tilde{s}_3-\tilde{s}_1)(\tilde{s}_3-\tilde{s}_2)}D(\tilde{s}_3). \tag{4.7}$$

The minimum of $P(s)$ is used to approximate the minimum of $D(s)$, and it is given by

$$s^{*,k} = \frac{1}{2}\frac{y_{23}D(\tilde{s}_1) + y_{31}D(\tilde{s}_2) + y_{12}D(\tilde{s}_3)}{s_{23}D(\tilde{s}_1) + s_{31}D(\tilde{s}_2) + s_{12}D(\tilde{s}_3)},\ k = 1, 2, 3, ..., \tag{4.8}$$

where $s_{ij} = \tilde{s}_i - \tilde{s}_j$ and $y_{ij} = \tilde{s}_i^2 - \tilde{s}_j^2$ for $i, j = \{1, 2, 3\}$. Three values are picked from $\tilde{s}_1$, $\tilde{s}_2$, $\tilde{s}_3$, and $s^{*,k}$ by eliminating the value which gives the largest $P(s)$ among the 4 values. In our implementations, the algorithm continues three iterations so that all initial estimates $\tilde{s}_1$, $\tilde{s}_2$, and $\tilde{s}_3$ have an opportunity to be replaced by a better estimate. Then the optimal estimate is used as initial estimate for Newton's method.

4.2.3.3 Newton's method

The value s^* that minimizes $D(s)$ in (4.6) satisfies

$$D'(s^*) = 0. \tag{4.9}$$

Newton's method can be used to find a root of this equation. This leads to the iteration formula

$$s^{*,m+1} = s^{*,m} - \frac{D'(s^{*,m})}{D''(s^{*,m})},\ m = 0, 1, 2, ... \tag{4.10}$$

As previously mentioned, the initial estimate for $s^{*,0}$ is based on the optimal value returned by the Quadratic minimization.

4.2.3.4 The combined method

Neither Newton's method nor quadratic minimization perform satisfactory for real-time simulation.[2] However, both methods possess individual strengths that compliment each other. Quadratic minimization is good at refining coarse estimates. Newton's method is good at converging to the optimal value with a good initial estimate. The composite algorithm begins with quadratic minimization method to find a rough estimate that serves as an initial guess for Newton's method.

Their method has undergone rigorous testing in a real-time ground vehicle simulator. Wang et al. argue that in 10 months of daily runs they have had no failures. This is used as a foundation for their robustness argument.

4.3 Motion guidance

In order to properly test the motion controller, there had to be established a way to generate reliable and executable reference signals to the controller. Furthermore, it is desirable to generate reference signals in such a way that the movement of the robot is sufficiently smooth. This avoids unnecessary jerk and acceleration for the robot. Moreover, it is safer, makes the robot movement more predictable and is less damaging to the robot joints. During early development, the trajectory generation was missing. Hence, there were only discrete waypoints to follow. Consequently, there were higher difficulties with creating suitable motion guidance, more precisely regarding the direction of the translational velocity, and the rotational velocity. This section presents the current version of the motion guidance in details. Obsolete versions are mentioned in a terse summary. For detailed description of the obsolete versions, visit appendix A.

4.3.1 Translational velocity

4.3.1.1 Direction

The direction of the translational velocity depends on the cubic B-spline described in the previous section. Moreover, the method developed here is inspired by the line-of-sight (LOS) guidance with a lookahead-based steering (see pages 257-262 in [28] for a detailed description). The translational velocity is given by

$$v_{ref} = c_1 v_e(s^*) + \hat{v}_t(s_{ahead}), \tag{4.11}$$

where v_e is the velocity given by the cross-track error, c_1 is a weighting constant, and $\hat{v}_t$ is the tangent directional vector where the hat indicates a unit vector[3].

[2] occurrence of divergence, slow behavior.

[3] a vector of length 1.

Furthermore, s^* is the trajectory parameter closest to the end-effector and s_{ahead} is the s-parameter defined a number of steps ahead of s^*. For clarity, Figure 4.8 displays the vectors in 2D space for $s_{ahead} = s^*$. The idea behind s_{ahead} is to ensure that the end-effector can follow high-curvature sections without excessive overshoot. Moreover, c_1 is used to ensure that the cross-track error stays within acceptable measures. After v_{ref} has been calculated through (4.11), it is transformed to a unit vector to let the velocity magnitude reference be the main deciding factor of the end-effector speed.

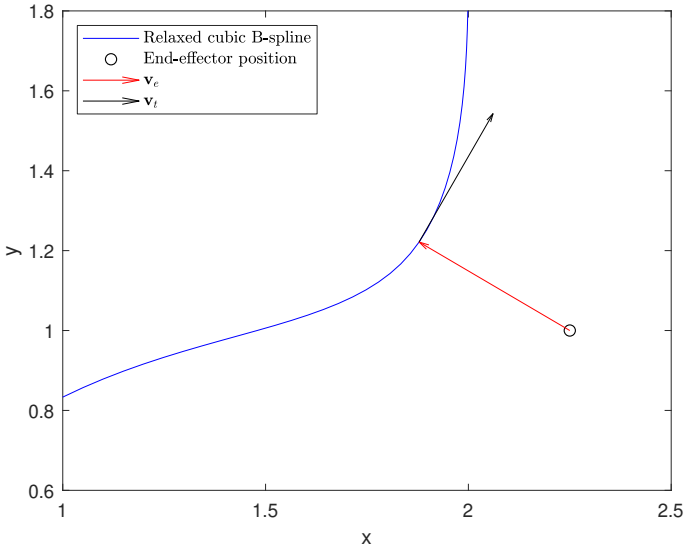


Figure 4.8: Cross-track error vector and tangent vector with trajectory. The tangent vector illustrated in this figure is not a unit vector.

4.3.1.2 Outdated implementations

The outdated version of this motion guidance was also inspired by the line-of-sight (LOS) guidance. It differs by having the vectors point to- and between discrete waypoints, instead of to a continuous spline curve. Furthermore, the waypoints were too close to each other to properly implement the procedure. The procedure worked to a certain extent, although it suffered from the relatively large discrete jump in the closest waypoint reference. A detailed description can be found in appendix A.

4.3.1.3 Magnitude

The magnitude of the translational velocity is directly given by the Euclidean distance transform of the security net. The idea is to allow movement at a standard

maximum operation speed v_{max} if the end-effector is further away from obstacles than a distance d_{safe}. If the distance to the closest obstacle is smaller than the safe distance, i.e. $d_{obj}(n) < d_{safe}$, the velocity magnitude will linearly decrease from v_{max} until the lowest allowed velocity v_{min} is reached. This is visualized in Figure 4.9 for a linear decrease. Future implementations should consider replacing the linear decrease with more a thoughtful reduction function, e.g. an exponential decrease.

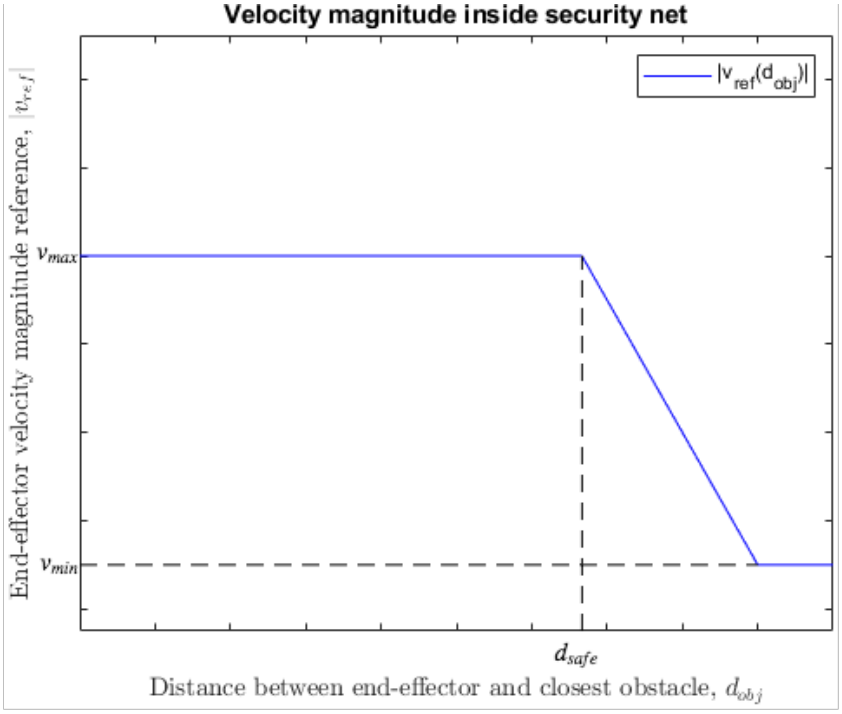


Figure 4.9: Velocity magnitude as a function of the Euclidean distance transform

4.3.2 Rotational velocity

The orientation of the end-effector is of huge importance. Based on the way the orientation guidance is performed, it may aid or counteract the robot's ability to move into cumbersome, ineffective joint configurations.

As previously mentioned, the goal orientation is set by the user or AI before the motion is initiated. Currently, a simple yet effective guidance for the rotational velocity is given by

$$\omega_{ref} = \kappa(\Theta - \Theta_{goal}), \tag{4.12}$$

where Θ is the current orientation of the end-effector, Θ_{goal} is the goal orientation, and κ is the tunable constant for the motion controller, defined in (3.1). Furthermore, all orientations are currently expressed using the Euler angle representation, i.e. roll, pitch, yaw $= [\phi, \theta, \psi]$.

The effectiveness of the proposed solution is obtained by specifying κ to be dependant on the trajectory parameter s. Moreover, it is given by

$$\kappa(s) = \sqrt{\frac{s^*}{\sum_{n=s_0}^{n=s_n} 1}}, \tag{4.13}$$

where s^* is the trajectory parameter closest to the end-effector, and $\sum_{n=s_0}^{n=s_n} 1$ denotes the total number of s used for the trajectory. Hence, $\kappa(s)$ is given by the square root of the percentage of the trajectory length traveled. Linear and quadratic exponents were also tested. However, it is more important that the orientation error is limited as s^* goes towards s_n, i.e. when the end-effector reaches the goal position. Hence, a more rapid increase in $\kappa(s)$ is desirable and the exponent $\frac{1}{2}$ is the better choice, illustrated in Figure 4.10.

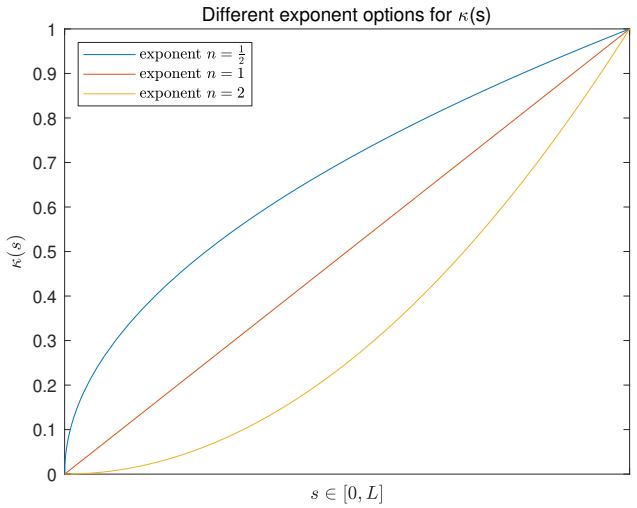


Figure 4.10: $\kappa(s)$ displayed for different exponent values.

4.3.2.1 The obsolete implementation

There exist two outdated versions for the rotational velocity. Method 1 was the first one, where the idea was to make the manipulator mimic the movements of a snake. Method 2 swapped between a rough and a fine path. During the rough path the orientation set to hold the roll and pitch zero while changing the yaw to make the end-effector point away from the base. During the fine path, the motion

guidance set to obtain the goal orientation before the end goal position was reached. Detailed descriptions can be found in A.

4.4 Motion control

Once the first versions of path planning and guidance system were implemented, making the end-effector follow the respective reference signals became the next challenge.

The following introductory paragraphs briefly explain the user design- requirements and possibilities for the Panda robot introduced in section 1.4.

In Franka Emika's provided $C++$ API[4], the robot can move by using a motion generator, where one of the four interfaces can be utilized:

- Joint position
- Joint velocity
- Cartesian position
- Cartesian velocity

The interfaces concerning the joints request the user to return a desired position or velocity for all joints. In the Cartesian interfaces, the user is requested to return a position and orientation (6 DOF) or a translational- and rotational velocity. Furthermore, all the control parameters are required to be fed to the robot every $1ms$. Hence, the definition

$$\Delta t = 0.001 \tag{4.14}$$

is made in advance to coming derivations.

The control parameter requirements are best explained by Franka Emika: "The control parameters fed into the robot should fulfill necessary and recommended conditions.[5] If necessary conditions are not met then the motion will be aborted. Recommended conditions should be fulfilled to ensure optimal operation of the robot.

The final robot trajectory is the result of processing the user-specified trajectory ensuring that recommended conditions are fulfilled. As long as the necessary conditions are met, the robot will try to follow the user-provided trajectory, but it will only match the desired trajectory if it also fulfills the recommended conditions. If the necessary conditions are violated, an error will abort the motion: if, for instance, the first point of the user defined joint trajectory is very different from q_{start}, a velocity limits violation error will abort the motion." [5]

[4] Application Programming Interface
[5] See section 1.4.

4.4.1 Constraint handling

Because the $C++$ library does not include a soft controller[6], the challenge with satisfying the constraints had to be addressed. The original goal was to use the Cartesian velocity interface. As previously mentioned, if the control parameters violate the constraints, the robot will stop. Consequently, the initial idea was to set a limitation on the subsequent velocity control parameters by requiring the acceleration between time steps to be within the requirements. Let the desired Cartesian velocity at time t be denoted v_t, and let the actual input to the robot at time t is denoted $\tilde{v}_t$. Then, the Cartesian acceleration can be approximated as

$$a_t = \frac{\|\tilde{v}_t - \tilde{v}_{t-1}\|}{\Delta t}. \tag{4.15}$$

Moreover, let the Cartesian input velocity be given by the equation

$$\tilde{v}_t = \tilde{v}_{t-1} + c(v_t - \tilde{v}_{t-1}). \tag{4.16}$$

Inserting (4.16) into (4.15), and squaring both sides yields

$$a_t^2 = \frac{c^2\|v_t - \tilde{v}_{t-1}\|^2}{(\Delta t)^2}. \tag{4.17}$$

Solving for c yields

$$c = \sqrt{\frac{(\Delta t)^2 a_t^2}{\|v_t - \tilde{v}_{t-1}\|^2}}, \tag{4.18}$$

and by setting $a_t < \ddot{p}_{max}$ the theory was that the former two equations in (1.10) were satisfied. It quickly became clear that although the Cartesian acceleration problem was fixed in theory, the Franka Emika robot continually aborted the motion due to violations; the debugging procedure was also unnecessary cumbersome. Furthermore, it was less intuitive to continue with a similar procedure to satisfy the constraints not yet addressed in (1.10). Consequently, using the Cartesian space interface was scrapped, and a proper constraint handling procedure was developed.

In addition to having the opportunity to minimize the joint velocity, a formulation which weighs this cost versus the respective errors in translational and rotational velocities is proposed. This yields an optimization problem on the Quadratic Programming (QP) form. To ensure that the kinematic relationship is satisfied, (1.2)

[6] A controller handling all constraints, regardless of input.

is included as an equality constraint for the problem. In addition, the respective joint constraints in (1.6) are included as inequality constraints. This yields

$$\begin{aligned} \text{minimize} \quad & J_c = \gamma \left\| v - v_{ref} \right\|_2^2 + \kappa \left\| w - w_{ref} \right\|_2^2 + \mu \dot{q}^T \dot{q} \\ \text{subject to} \quad & \begin{bmatrix} v \\ w \end{bmatrix} = J\dot{q}, \\ & q_{min} < q \ < q_{max}, \\ & -\dot{q}_{max} < \dot{q} < \dot{q}_{max}, \\ & -\ddot{q}_{max} < \ddot{q} < \ddot{q}_{max}, \\ & -\dddot{q}_{max} < \dddot{q} < \dddot{q}_{max}, \end{aligned} \tag{4.19}$$

where γ, κ and μ are weighting constants. Hence the idea is an adaptive soft controller where a trade-off between reducing errors in translational- or rotational velocity and reducing overall joint velocity can be dynamically adjusted in real-time operation. This is on top of the fact that, in theory, all constraints are satisfied. An immediate advantage with this cost function is that it linearly separates the different optimization criteria, where the optimization variable contains both Cartesian- and joint space. Hence, it provides an easy way to weigh between Cartesian- and joint space criteria.

4.4.2 QP standard form and boxed inequality constraints

In order to implement the QP problem, it needs to be transformed to canonical standard form. By defining the optimization variable as

$$z := \begin{bmatrix} v^T & \omega^T & \dot{q}^T \end{bmatrix}^T, \tag{4.20}$$

this can be achieved. The canonical standard form is

$$\begin{aligned} \underset{z}{\text{minimize}} \quad & z^T Q z + q_0 z \\ \text{subject to} \quad & A_{eq} z = 0, \\ & A_{ineq} z \leq b. \end{aligned} \tag{4.21}$$

Consequently, the joint- position, acceleration and jerk have to be approximated by using the joint velocity, $\dot{q}$.

Position
For the joint position, the approximation

$$q \approx q_{prev} + \Delta t \dot{q}_d \tag{4.22}$$

is used. Inserting that into the corresponding constraint in (4.19), and solving for $\dot{q}$ yields

$$\underbrace{\frac{q_{min} - q_{prev}}{\Delta t}}_{:=\dot{q}_{d1min}} < \dot{q}_d < \underbrace{\frac{q_{max} - q_{prev}}{\Delta t}}_{:=\dot{q}_{d1max}}. \tag{4.23}$$

Velocity
The joint velocities already satisfy the standard form. The following definitions are made

$$\dot{q}_{d2min} := -\dot{q}_{max} \tag{4.24}$$
$$\dot{q}_{d2max} := \dot{q}_{max}. \tag{4.25}$$

Acceleration
For the joint acceleration, the approximation

$$\ddot{q}_d \approx \frac{\dot{q}_{end} - \dot{q}_{beginning}}{\Delta t} = \frac{\dot{q}_d - \dot{q}_{prev}}{\Delta t} \tag{4.26}$$

is used. As before, it is inserted into the corresponding constraint in (4.19), and solved for $\dot{q}$. The resulting equation becomes

$$\underbrace{-\Delta t \ddot{q}_{max} + \dot{q}_{prev}}_{:=\dot{q}_{d3min}} < \dot{q}_d < \underbrace{\Delta t \ddot{q}_{max} + \dot{q}_{prev}}_{:=\dot{q}_{d3max}} \tag{4.27}$$

Jerk
For the joint jerk, the approximation becomes slightly more complicated. Jerk $\dddot{q}$ is the relative change in acceleration, and thus we have the relationship

$$\begin{aligned} \dddot{q}_d &\approx \frac{a_{end} - a_{beginning}}{\Delta t} = \frac{\frac{\dot{q}_d - \dot{q}_{prev}}{\Delta t} - \frac{\dot{q}_{prev} - \dot{q}_{prevprev}}{\Delta t}}{\Delta t} \\ &= \frac{\dot{q}_d - 2\dot{q}_{prev} + \dot{q}_{prevprev}}{(\Delta t)^2}, \end{aligned} \tag{4.28}$$

where (4.26) was used. Inserting (4.28) into the corresponding constraint in (4.19), and solving for $\dot{q}$ yields

$$\underbrace{-(\Delta t)^2 \dddot{q}_{max} + 2\dot{q}_{prev} - \dot{q}_{prevprev}}_{\dot{q}_{d4min}} < \dot{q}_d < \underbrace{(\Delta t)^2 \dddot{q}_{max} + 2\dot{q}_{prev} - \dot{q}_{prevprev}}_{\dot{q}_{d4max}}. \tag{4.29}$$

This will bring the inequality constraints on the form

$$\begin{aligned}\dot{q}_{d1min} < \dot{q}_d < \dot{q}_{d1max} \\ \dot{q}_{d2min} < \dot{q}_d < \dot{q}_{d2max} \\ \dot{q}_{d3min} < \dot{q}_d < \dot{q}_{d3max} \\ \dot{q}_{d4min} < \dot{q}_d < \dot{q}_{d4max}.\end{aligned} \tag{4.30}$$

An immediate benefit that can be exploited from this is that (4.30) can be boxed, effectively reducing the number of inequality constraints the QP problem has to satisfy from $8 \times 7 = 56$ (7 joints), to $2 \times 7 = 14$. This is achieved by defining

$$\dot{q}_{dmin} := max(\dot{q}_{d1min}, \dot{q}_{d2min}, \dot{q}_{d3min}, \dot{q}_{d4min}) \tag{4.31}$$

$$\dot{q}_{dmax} := min(\dot{q}_{d1max}, \dot{q}_{d2max}, \dot{q}_{d3max}, \dot{q}_{d4max}). \tag{4.32}$$

The final inequality becomes

$$\epsilon\dot{q}_{dmin} \leq \dot{q}_d \leq \epsilon\dot{q}_{dmax}, \tag{4.33}$$

where $0 < \epsilon < 1$ is included to handle approximation errors in addition to ensuring that the constraints satisfy the QP canonical form.

Using the boxed constraint formulations, transforming (4.19) to the canonical standard form in (4.21) using (4.20) as the optimization variable yields the following matrices

$$\begin{aligned} Q &= \begin{bmatrix} \gamma\mathcal{I}_{3x3} & 0 & 0 \\ 0 & \kappa\mathcal{I}_{3x3} & 0 \\ 0 & 0 & \mu\mathcal{I}_{7x7} \end{bmatrix}, \quad q_0 = \begin{bmatrix} -2\gamma v_{ref} & -2\kappa w_{ref} & 0 \end{bmatrix}, \\ A_{eq} &= \begin{bmatrix} \mathcal{I}_{6x6} & -J \end{bmatrix}, \\ A_{ineq} &= \begin{bmatrix} 0_{1x3} & 0_{1x3} & \mathcal{I}_{7x7} \\ 0_{1x3} & 0_{1x3} & -\mathcal{I}_{7x7} \end{bmatrix}, \qquad b = \begin{bmatrix} \epsilon\dot{q}_{dmax} \\ \epsilon\dot{q}_{dmin} \end{bmatrix}. \end{aligned} \tag{4.34}$$

4.4.3 Joint limit

It quickly became apparent that the joint limit in (4.23) does not work in practice. To understand why this is the case, consider the following illustration. Say joint number k is approaching its positive joint limit and $\dot{q}_k > 0$. The controller will in many cases continue with positive joint velocity until the joint limit constraint inhibits further increase in joint velocity. This happens approximately one iteration

before the violation occurs. At this point, the only way to avoid violating the joint limit would be to severely decrease $\dot{q}_k$. However, this would violate the joint acceleration constraint, unless $\dot{q}_k$ is unlikely small at that instant. Hence, either the QP formulation becomes infeasible or the Franka robot's fail-safe procedures abort the motion control. Nevertheless, this is a flaw that had to be addressed. To obtain a behavior that hinders joints from approaching their respective limits, a term that penalizes operations away from q_{mid} is suggested. q_{mid} will be defined as the joint configuration that is in the middle of q_{min} and q_{max} in (1.6). In mathematical terms, consider adding the following term to the cost function J_c

$$\|q - q_{mid}\|_2^2 \,. \tag{4.35}$$

The idea is to increase the cost of operations further from the joint's mid values q_{mid}. Intuitively, this could display the desired effect with correct tuning. To fit the QP formulation, the approximation (4.22) is used again. Inserting the approximation into (4.35) and removing constant terms yield

$$(q_{prev} + \Delta t\dot{q}_d - q_{mid})^T(q_{prev} + \Delta t\dot{q}_d - q_{mid}) = 2\Delta t q_{prev}^T\dot{q}_d - 2\Delta t q_{mid}^T\dot{q}_d + (\Delta t)^2\dot{q}_d^T\dot{q}_d.$$

This will alter the following matrices to

$$Q = \begin{bmatrix} \gamma\mathcal{I}_{3x3} & 0 & 0 \\ 0 & \kappa\mathcal{I}_{3x3} & 0 \\ 0 & 0 & (\mu + (\Delta t)^2)\mathcal{I}_{7x7} \end{bmatrix}, \quad q_0 = \begin{bmatrix} -2\gamma v_{ref} & -2\kappa w_{ref} & 2\Delta t(q_{prev}^T - q_{mid}^T) \end{bmatrix}.$$

It quickly became apparent that including (4.35) severely impaired the normal operation of the controller. Extensive tuning of the control parameters would perhaps yield the desired effect. However, that would probably require a trade-off where the other elements in the cost function become shunned. Additionally, tuning the controller with the new term was no longer intuitive. Hence, the idea was not further investigated and a more thought-through solution was pursued.

The main theory as to why (4.35) did not provide desired results is because it excessively affected the normal operation of the controller. Hence, a natural approach was to create a term that only has an impact when it is truly necessary, which is when a joint is close to either of its limits. Furthermore, the expression in (4.35) increases quadratically. Having a term with a slope that increases more rapidly would produce a quicker response from the soft controller. The natural exponential function e^x has a promising slope. In addition, the function is relatively easy to modify in order to obtain desired performance in terms of activation region, penalty weighting etc.

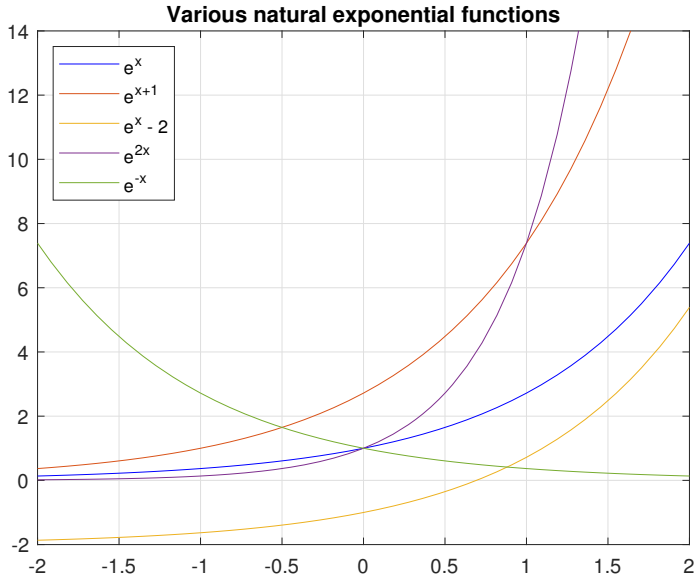


Figure 4.11: Various versions of exponential function

Observing Figure 4.11, it becomes clear that increasing the slope and shifting the curve on the x-axis is relatively easy. Consequently, consider the function

$$g(q) = c(e^{-\lambda(q-q_{min})} + e^{-\lambda(q_{max}-q)}), \tag{4.36}$$

where c and λ are constants used for tuning, e is the natural exponential function, q is the joint angle for the respective joints, q_{min} and q_{max} are from (1.6). Figure 4.12 shows the behavior of the function for different tuning values. With the exception of the $c = 1, \lambda = 1$ curve, the cost function term displays desired performance; it has zero impact during normal joint-safe operations.

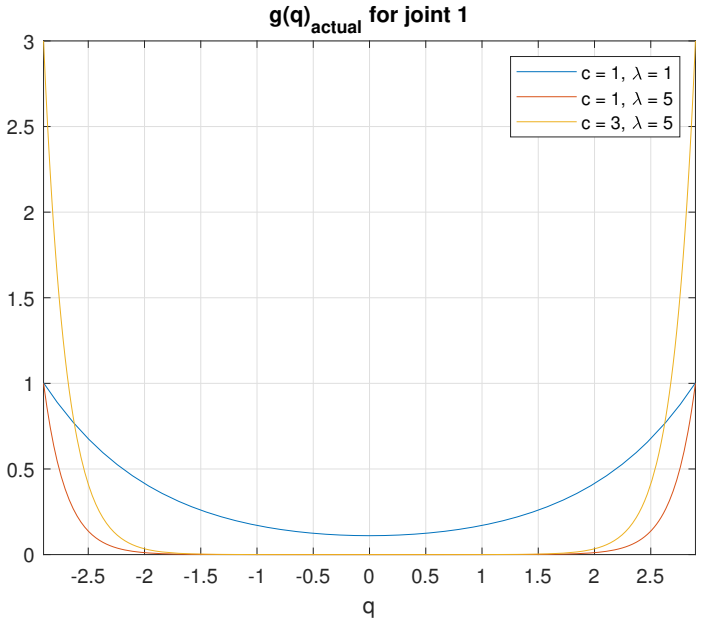


Figure 4.12: Actual $g(q)$ plotted for various c and λ

Now that a promising mathematical term has been derived, it is necessary to transform it to fit the QP problem formulation. For openers, the optimization variable contains $\dot{q}$ and not q. Hence, the approximation (4.22) is used again. This yields

$$g_1(q) = c(e^{-\lambda(q_{prev}+\Delta t\dot{q}-q_{min})} + e^{-\lambda(q_{max}-q_{prev}-\Delta t\dot{q})}). \tag{4.37}$$

Moreover, $\dot{q}$ must be represented in linear or quadratic terms only in order to satisfy the QP canonical form. The natural exponential function can be defined as the value of the power series

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{1}{2}x^2 + \frac{1}{3}x^3 + \frac{1}{4}x^4 + ... \tag{4.38}$$

If possible, it is desired to use only linear terms in the optimization variable z. This is because quadratic terms will affect the Q matrix, which complicates the tuning of the soft controller. This was seen to be the case with the q_{mid} formulation. Therefore, the approximation

$$e^x \approx 1 + x \tag{4.39}$$

is used, and the quadratic term $\frac{1}{2}x^2$ is omitted analogous to the larger nonlinear terms. Furthermore, the properties of the natural exponential function are directly related to properties of simple power series. Hence, the following property will be utilized

$$e^{x+y} = e^x e^y. \tag{4.40}$$

Utilizing (4.39) and (4.40) for $g_1(q)$ in (4.37) yields

$$g_2(q) = c((1 - \lambda\Delta t\dot{q})e^{-\lambda(q_{prev}-q_{min})} + (1 + \lambda\Delta t\dot{q})e^{-\lambda(q_{max}-q_{prev})}). \tag{4.41}$$

For the QP problem, constant terms are omitted from the cost function, because they do not have any effect on the optimal solution. Hence, the approximated function becomes

$$g(\dot{q}, q) = c(-\lambda\Delta t e^{-\lambda(q_{prev}-q_{min})} + \lambda\Delta t e^{-\lambda(q_{max}-q_{prev})})\dot{q}, \tag{4.42}$$

which is linear in $\dot{q}$. Thus, only the linear term matrix $q0$ in the cost function needs adjustment. The cost function matrices become

$$Q = \begin{bmatrix} \gamma\mathcal{I}_{3x3} & 0 & 0 \\ 0 & \kappa\mathcal{I}_{3x3} & 0 \\ 0 & 0 & \mu\mathcal{I}_{7x7} \end{bmatrix},$$
$$q_0 = \begin{bmatrix} -2\gamma v_{ref} & -2\kappa w_{ref} & c(-\lambda\Delta t e^{-\lambda(q_{prev}-q_{min})} + \lambda\Delta t e^{-\lambda(q_{max}-q_{prev})}) \end{bmatrix}.$$

The behavior of (4.42) needs to be analyzed to assert that the desired performance is preserved after the approximations have been made. There are two variable terms that dynamically and uncontrollably adjusts the value of $g(q, \dot{q})$, specifically q_{prev} and $\dot{q}$, which both can have positive and negative values. Hence, there are four cases that need to be investigated.

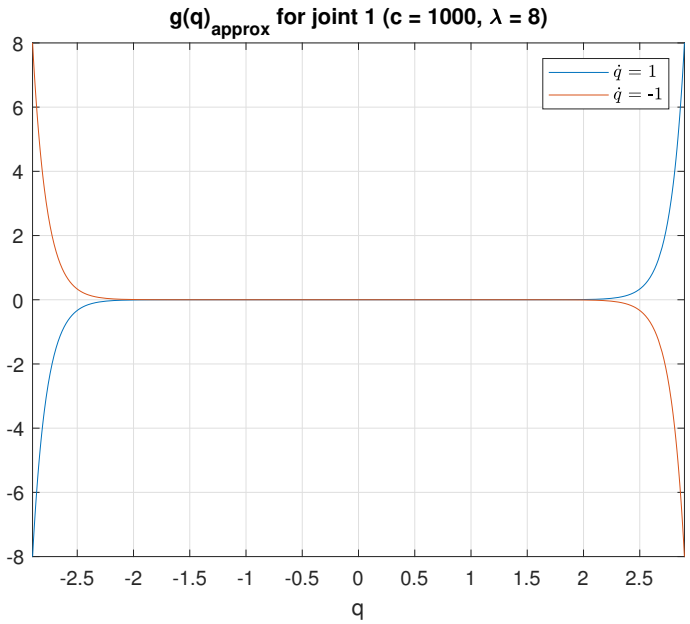


Figure 4.13: Approximated $g(q)$ plotted for positive and negative $\dot{q}$.

By observing Figure 4.13 we can discuss the four cases.

Case (i): $\dot{q} > 0 \wedge q_{prev} > 0$
Because $\dot{q} > 0$, the angle q will increase, and hence move further to the right and away from $q = 0$. At $q \approx 2$, the penalty term will increase exponentially until it accumulates a significant cost value. Then the QP solver will decrease $\dot{q}$ within the accepted acceleration limits until it surpasses $\dot{q} = 0$ and becomes negative. Consequently, the positive joint limit is avoided in theory.

Case (ii): $\dot{q} < 0 \wedge q_{prev} < 0$
By symmetry, the negative joint limit is also avoided.

Case (iii): $\dot{q} > 0 \wedge q_{prev} < 0$
If the joint angle $q < -2$, one might expect the QP solver to dislike having $\dot{q} > 0$, because this will increase the $g(q, \dot{q})$ term and thus might also increase the total cost function. Hence, local minima exist in theory, where the QP solver is switching rapidly between cases (i) and (iv) or cases (ii) and (iii), respectively.

Case (iv): $\dot{q} < 0 \wedge q_{prev} > 0$
By symmetry, see case (iii).

Several experiments in section 5.4 and the experiment in section 5.2 showcase the result of including $g(q, \dot{q})$ in the cost function.

Chapter 5

Experiments and results

This chapter presents four experiments. Section 5.1 displays the security net's ability to prevent obstacle collisions for the end-effector. Section 5.2 demonstrates the security net's ability to reach a goal pose when having an extraordinary cumbersome initial joint configuration. Furthermore, it is the most extensive experiments because most relevant parts of the security net are also presented. The additional plots are omitted from the other experiments to keep the thesis concise. Section 5.3 analyzes the end-effector's ability to follow a trajectory for different velocity magnitudes. Finally, section 5.4 analyzes the joint limit term $g(\dot{q}, q)$ derived in section 4.4.3 in several practical experiments.

Figure 5.1 displays the setup used for the experiments in sections 5.1, 5.2, and 5.3 with the exception of some obstacles in front of the robotic manipulator. Figure 5.2 displays the full environment seen by the voxel grid. Moreover, in sections 5.1, 5.2, and 5.3 the plots do not display all the relevant obstacles. This is done to only show the relevant parts. However, the objects and the manipulator base are mapped in the voxel grid for all experiments. The reason for mapping the manipulator as an obstacle is to prevent the path planning algorithm from creating paths that go through the manipulator's base. Additionally, the ground is mapped as an obstacle. However, it is not displayed in Figure 5.2.

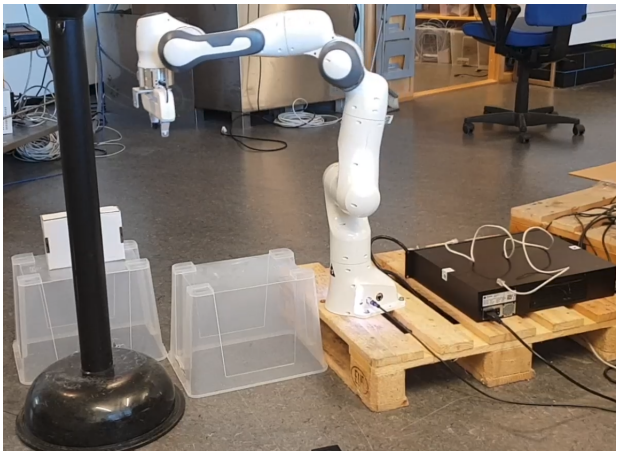


Figure 5.1: Real display of the setup used in several experiments.

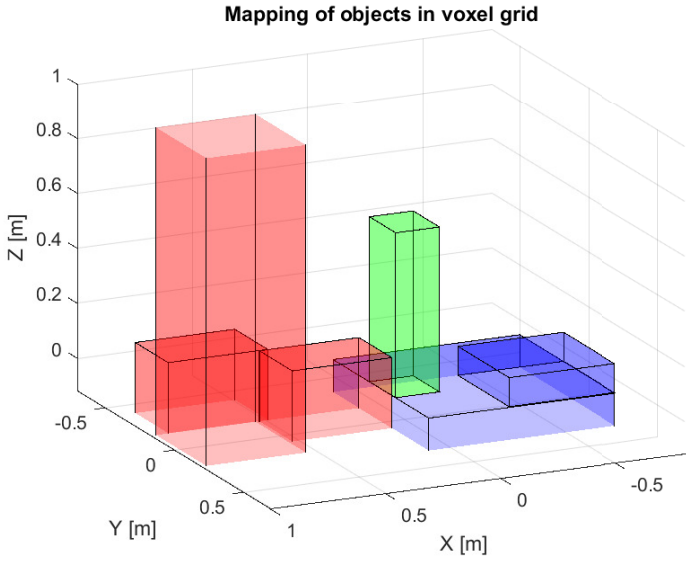


Figure 5.2: Voxel grid display of mapped obstacles used in several experiments. The blue objects represent a euro pallet and the robot control box, the green object represents the base of the robot, and the red objects represent two boxes of similar size and a pole. Axes are in robot coordinates.

The experiments in section 5.1 and 5.2 used the following parameter values for the motion controller: $\gamma = 1$, $\kappa(s)$ as in (4.13), $\mu = 0.002$, $\chi = 1$[1]. Moreover, the tuning parameters for $g(\dot{q}, q)$ in (4.23) satisfy $\lambda = 8$ and $c = 1000$. The maximum and minimum velocity magnitudes were set to $v_{max} = 0.15m/s$ and $v_{min} = 0.04m/s$, respectively. Section 5.3 used the same parameters, with the additional test using $v_{max} = 0.3m/s$. Finally, section 5.4 used the same parameter values with the exception of $v_{max} = 0.3m/s$, and $\kappa = 1$. Most of these values were chosen based on the comprehensive experience acquired from extensive testing of the security net during the development process.

5.1 Security net with- and without obstacles

This section compares two experiments. The first experiment demonstrates the security net's ability to reach an end-effector goal pose in an environment without significant obstacles. Following is a similar experiment with the same start- and goal pose, only with new obstacles present and mapped in the voxel grid. The new obstacles are placed in such a way that the path generated in the first experiment would result in a collision.

5.1.1 Experiment without significant obstacles

Figures 5.3 and 5.4 display the real life setup for this experiment.

(a) Start configuration

(b) Intermediate configuration

Figure 5.3: Start- and intermediate configurations of a pick and place operation in an environment without significant obstacles.

[1]Except for the final part of the trajectory, where it is set to zero. This is explained in details in section 6.4.1

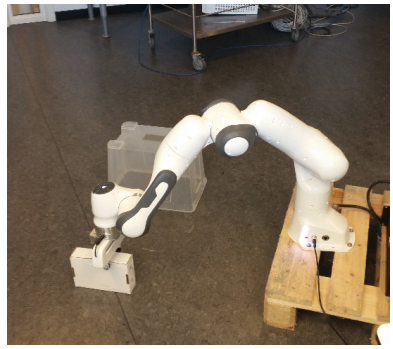
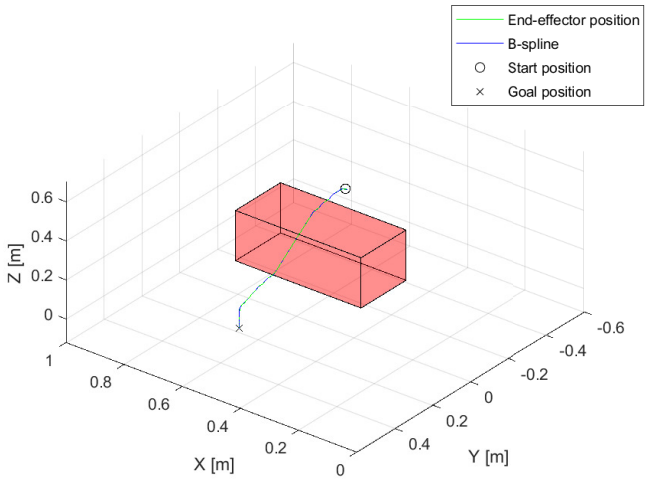
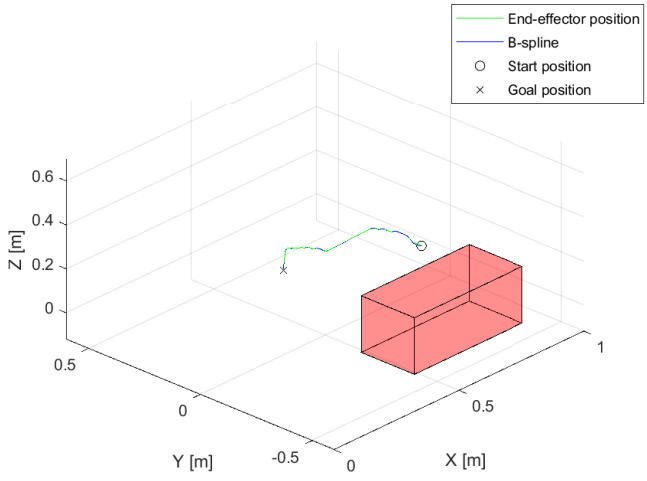


Figure 5.4: End configuration of a pick and place operation in an environment without significant obstacles.

Figure 5.5 displays that the security net creates and traverses a short and simple trajectory when the environment allows it. Moreover, the reference trajectory keeps a safe distance to all obstacles and the end-effector successfully tracks the trajectory.



(a) View from left side

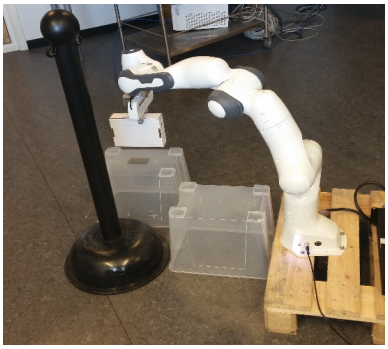


(b) View from right side

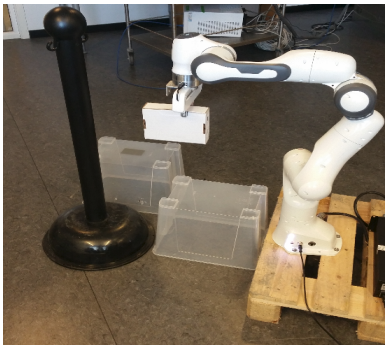
Figure 5.5: End-effector trajectory tracking of cubic B-spline without significant obstacles

5.1.2 Experiment with significant obstacles

Figures 5.6 and 5.7 display the real life setup for this experiment.



(a) Start configuration



(b) Intermediate configuration

Figure 5.6: Start- and intermediate configurations of a pick and place operation in an environment with significant obstacles.

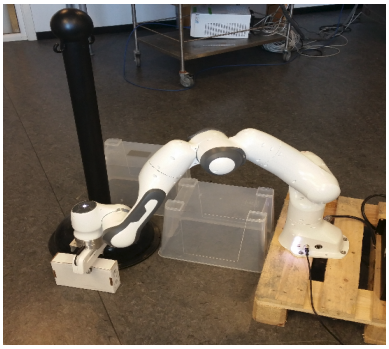
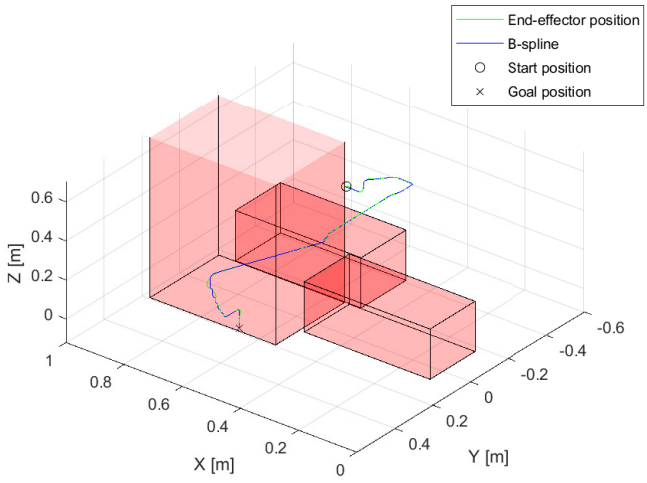
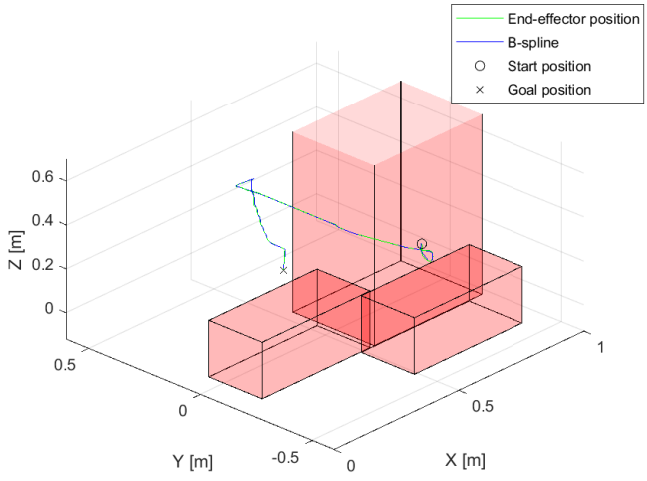


Figure 5.7: End configuration of a pick and place operation in an environment with significant obstacles.

By observing Figure 5.8 it is clear that when the additional obstacles are properly mapped in the voxel grid, the security net performs necessary adjustments to the trajectory and is still able to successfully execute the pick and place task. Furthermore, the reference trajectory keeps a safe distance to all obstacles in the cluttered environment, and the end-effector's cross-track errors relative to the reference trajectory are kept minimal.



(a) View from left side

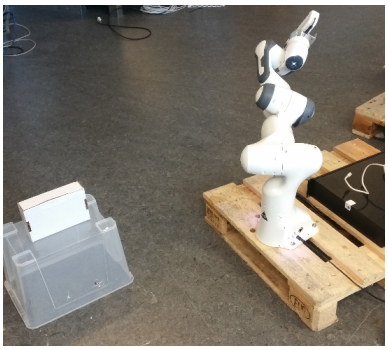


(b) View from right side

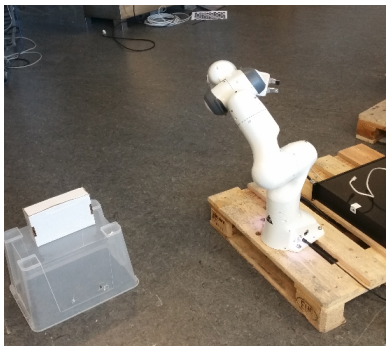
Figure 5.8: End-effector trajectory tracking of cubic B-spline with significant obstacles

5.2 Security net with cumbersome initial joint configuration

As previously mentioned, this section includes an experiment that demonstrates the security net's ability to perform a simple pick operation when the robot is manually positioned into a cumbersome initial joint configuration. Figures 5.9 and 5.10 display the real-life setup for this experiment.

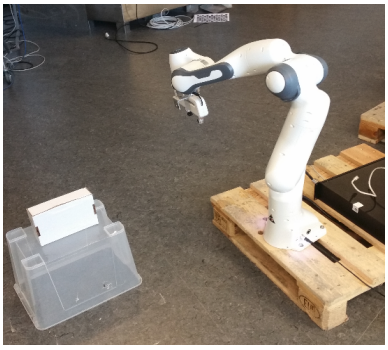


(a) Start configuration

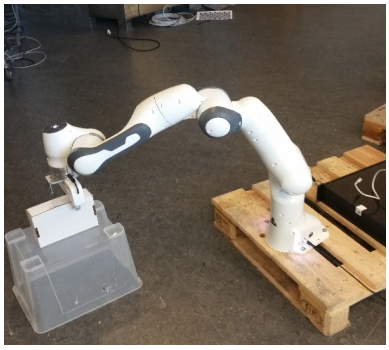


(b) Intermediate configuration 1

Figure 5.9: Start- and intermediate configurations of an experiment when the initial configuration is extraordinary cumbersome.



(a) Intermediate configuration 2

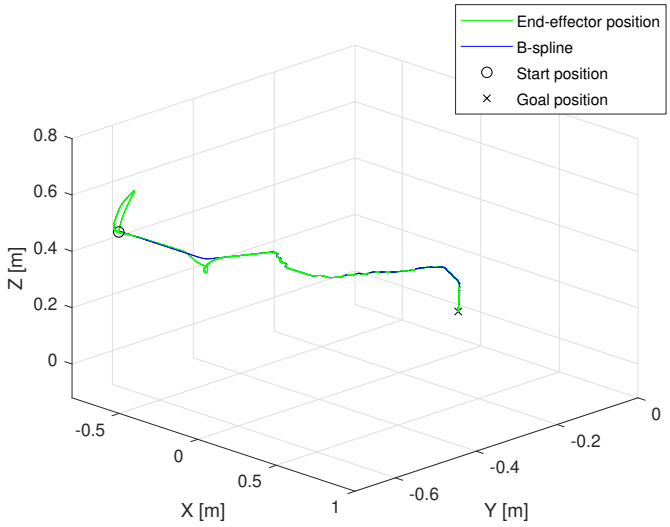


(b) End configuration

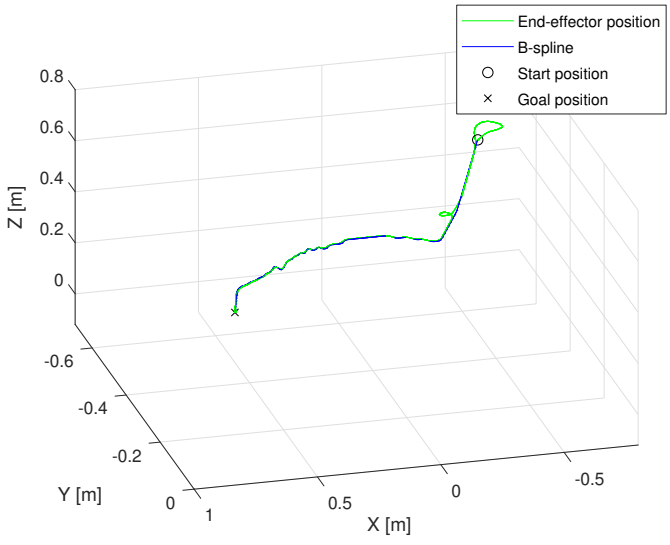
Figure 5.10: Intermediate- and end configurations of an experiment when the initial configuration is extraordinary cumbersome.

Figure 5.11 displays the security net's ability to follow the trajectory when an extraordinary cumbersome initial joint configuration is manually set in advance. The end-effector has two noticeable deviations from the reference trajectory: one at the start of the motion and a smaller one following soon after. The magnitude of these

deviations are displayed in detail in Figure 5.12 (a), with the largest deviation reaching almost 14 cm and the second deviation reaching almost 6 cm. Furthermore, these noticeable cross-track errors are neutralized within $2.5s$, respectively. The remaining parts of the end-effector trajectory satisfy approximately zero cross-track error. Figure 5.12 (b) suggests that the larger deviation at the beginning is caused by the joint correction term, and that the second deviation is not. Moreover, the joints at the highest risk of reaching their limits are joints 5 and 2.

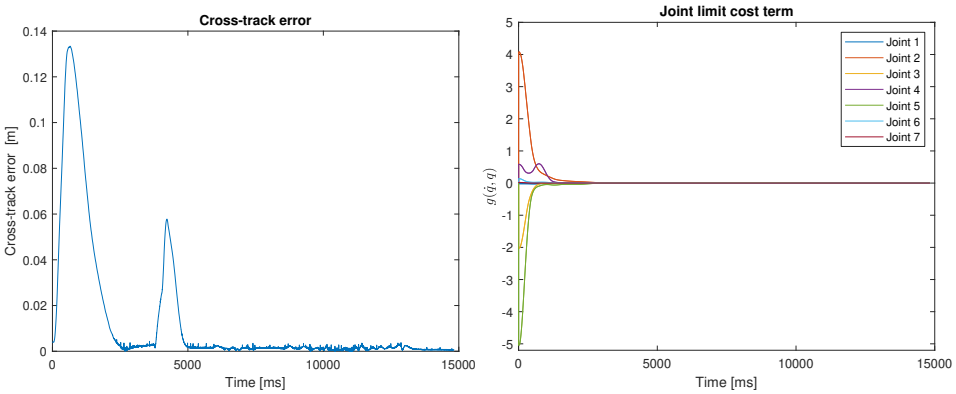


(a) Front view



(b) Side view

Figure 5.11: End-effector trajectory when the initial configuration is extraordinary cumbersome.

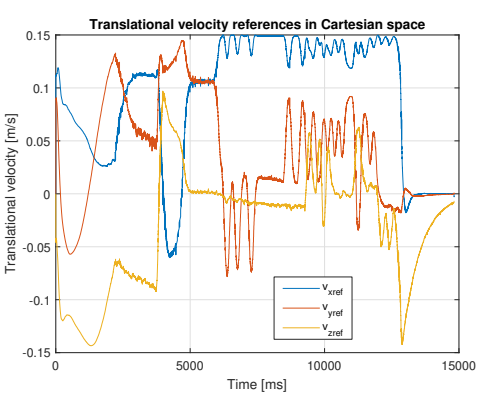


(a) Cross-track error

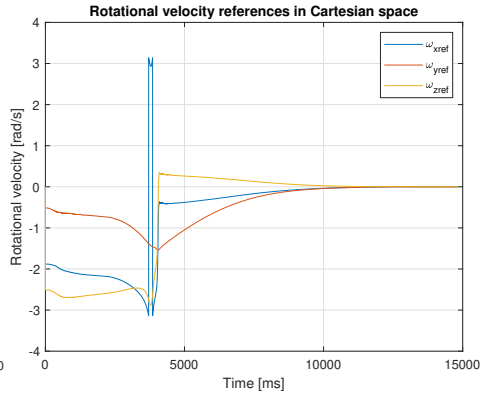
(b) Joint limit term

Figure 5.12: Cross-track error and joint limit term $g(\dot{q}, q)$ when the initial configuration is extraordinary cumbersome.

Figures 5.13 (a) and (b) display the translational- and rotational velocity references, respectively, generated by the motion guidance. The translational references are slightly oscillating with the exception of the first 2.5 seconds where they are smooth. The rotational references are continuous for the entire experiment with the exception that the rotational velocity reference in the $x-$direction has a critical discontinuity at $\approx 4s$. This discontinuity is likely to be the cause of the second tracking deviation. Moreover, a detailed description of the reason for the discontinuity is derived in Section 6.3.



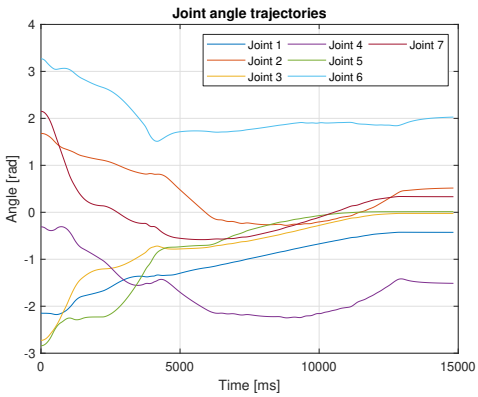
(a) Translational velocity references



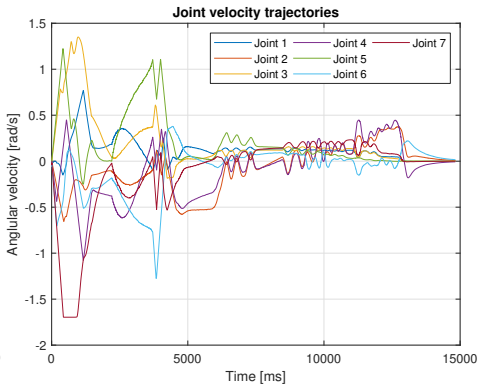
(b) Rotational velocity references

Figure 5.13: Translational- and rotational velocity references when the initial configuration is extraordinary cumbersome.

Figures 5.14 (a) and (b) display the joint- and joint velocity trajectories. The former demonstrates a high change in joint values at the beginning of the experiment, where most joints are moved away from their respective limits. The latter figure shows that high joint velocities are required during both the noticeable cross-track error incidents. Furthermore, when the cross-track errors are small the velocities oscillate.



(a) Joint trajectories

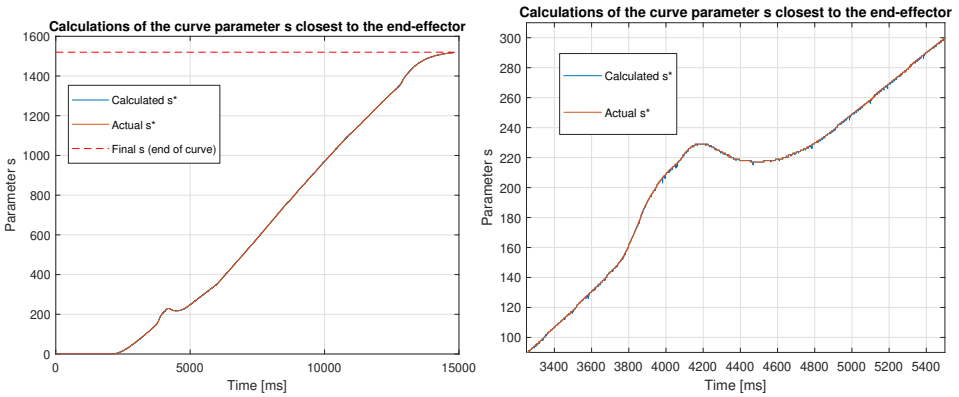


(b) Joint velocity trajectories

Figure 5.14: Joint- and joint velocity trajectories when the initial configuration is extraordinary cumbersome.

Figure 5.15 demonstrates the effectiveness of the algorithm used to calculate the curve parameter s^* that is closest to the end-effector (depicted in section 4.2.3). The

algorithm's output and the actual s^* are almost indistinguishable. Consequently, a zoomed view is necessary to display that there are minor differences between them. The actual s^* is calculated with the brute force method, which was run subsequent to the experiment.



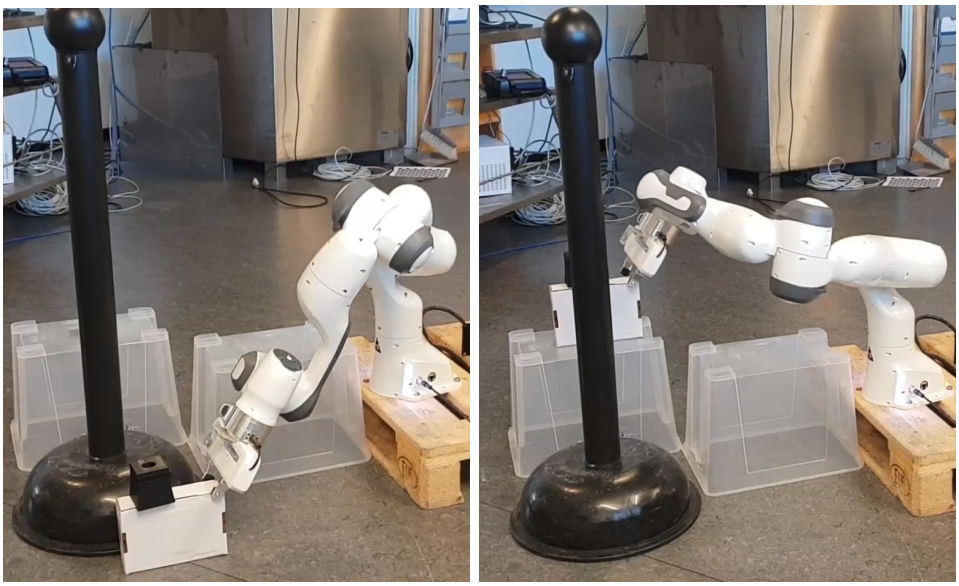
(a) View for all s^* calculated vs actual s^*

(b) Zoomed view

Figure 5.15: Calculated s^* versus actual s^* when the initial configuration is extraordinary cumbersome.

5.3 Robot trajectory tracking with different velocity magnitudes

The experiment in this section attempts to test the motion guidance's ability to follow the trajectory for different levels of velocity magnitude. Moreover, the plots presented in this section are from the final trajectory in the video attached to this thesis, i.e. when the manipulator carries two objects. For clarity, Figure 5.16 displays the start- and end configurations.

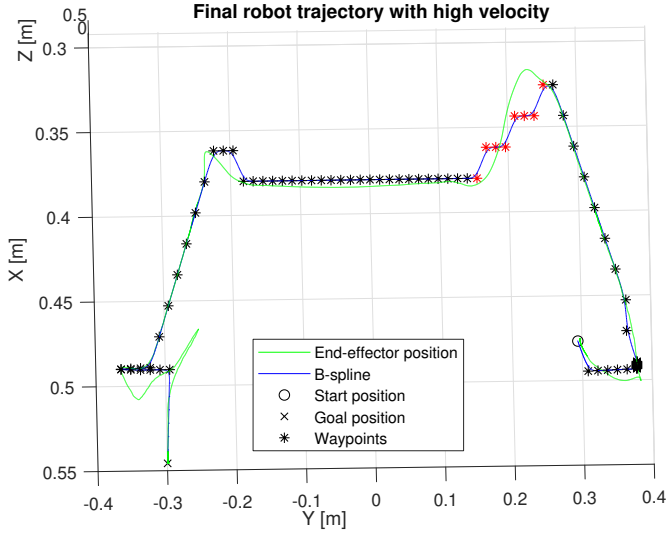


(a) Start configuration (b) End configuration

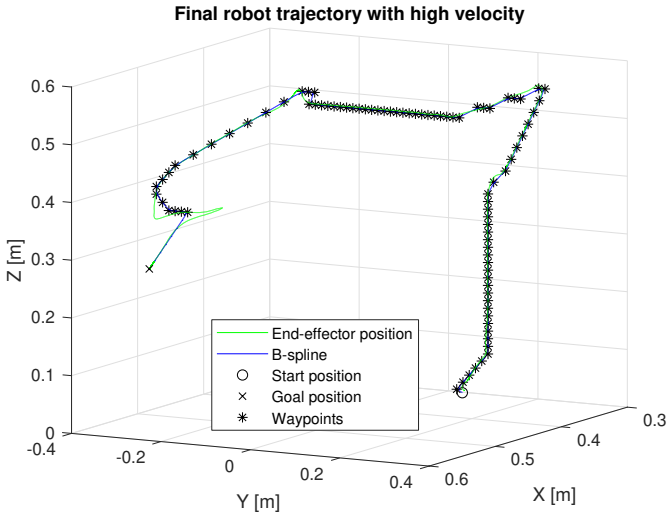
Figure 5.16: Start- and end configuration for this experiment.

Figure 5.17 presents the end-effector trajectory for $v_{max} = 0.3m/s$ with the corresponding reference trajectory as a B-spline, and waypoints. For straight line spline segments, the end-effector successfully follows the reference trajectory. On the other hand, it obtains noticeable cross-track errors when encountering high-curvature sections. Figure 5.18 presents the same plot, only with a lower velocity magnitude $v_{max} = 0.15m/s$. In comparison, it shows that all the major cross-track errors are overcome when using a lower velocity magnitude.

The observant reader might notice that the final part of the trajectories in the aforementioned figures have a shortage of waypoints. The explanation for this can be found in section 6.1.

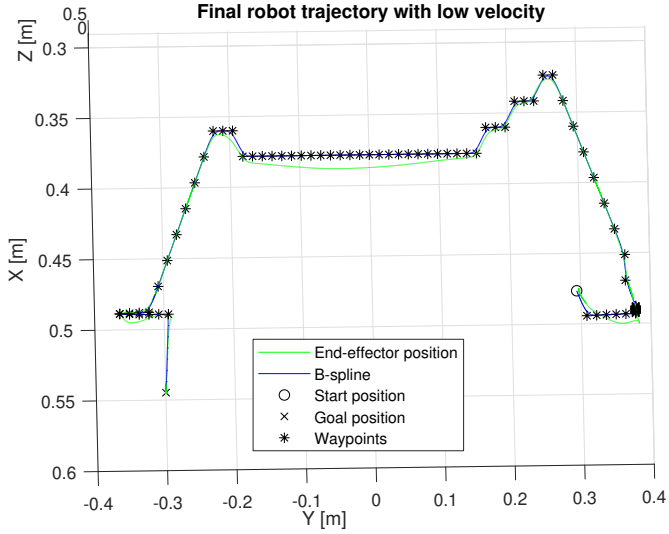


(a) Top view. Waypoints drawn in red are discussed in section 6.1

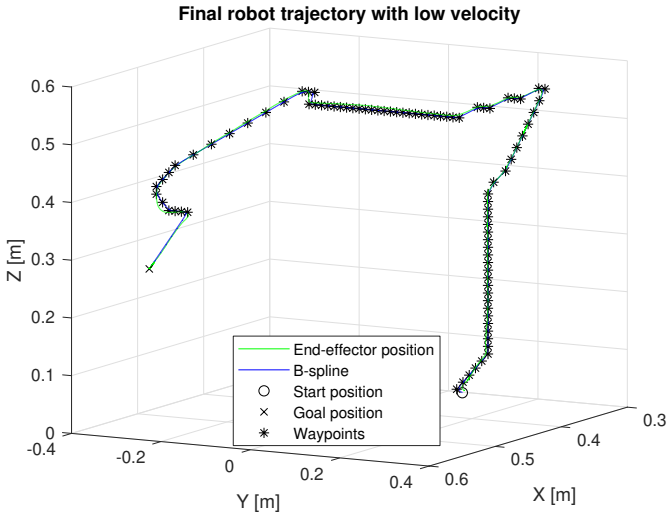


(b) Side view

Figure 5.17: End-effector trajectory with cubic B-spline and waypoints for a high velocity magnitude $v_{max} = 0.3m/s$.



(a) Top view



(b) Side view

Figure 5.18: End-effector trajectory with cubic B-spline and waypoints for a low velocity magnitude $v_{max} = 0.15m/s$.

5.4 Joint limit term

The experiments in this section were performed antecedent to the development of the trajectory generation in section 4.2. Hence, the obsolete methods described it appendices A.1.1 and A.2.2 were used as motion guidance.

The effect of adding $g_(\dot{q}, q)$ to the cost function needs to be analyzed. Forthcoming are two experiments where the robot is manually put in a start joint configuration, paired with a specific goal position for the end-effector. The pairs are designed in such a way that without the additional $g_(\dot{q}, q)$ term, the robot reaches a specific joint limit and the internal fail-safe mechanism of the robot inhibits further control. The idea is to observe if the joint limits are avoided the additional term is included. A discussion regarding the acquired results is derived in section 6.4.1. Moreover, two supplementary experiments are presented in Appendix B to keep the main report tidy.

5.4.1 Experiment - Negative limit joint 1

Comparing Figure 5.19 and Figure 5.20 it is clear that joint 1 avoids its negative joint limit when including (4.42) in the cost function. Furthermore, looking at time $t > 2000ms$ (because this is the time where joint 1 acquires a significant disparity between the respective figures), it seems that joints 3, 5, and 2 correct for most of the necessary adjustment in joint 1. Figures 5.21 and 5.22 show that (especially after time $t > 2000ms$) only minor adjustments are required with respect to joint velocities to avoid the joint limit.

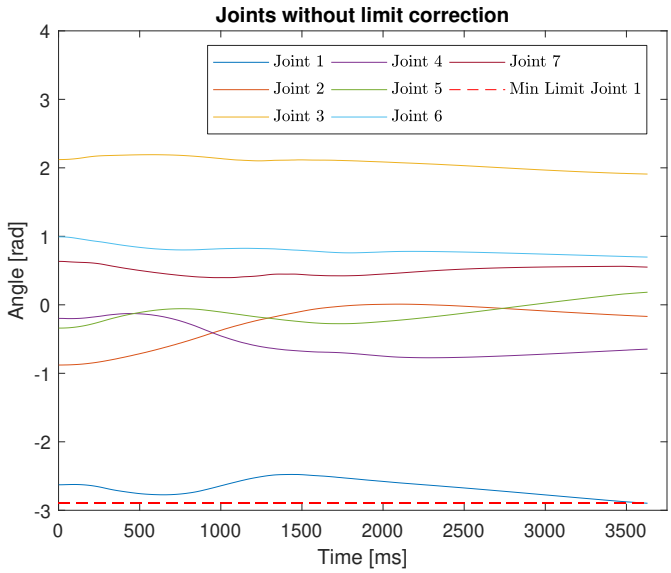


Figure 5.19: Joint trajectories where joint 1 reaches its limit.

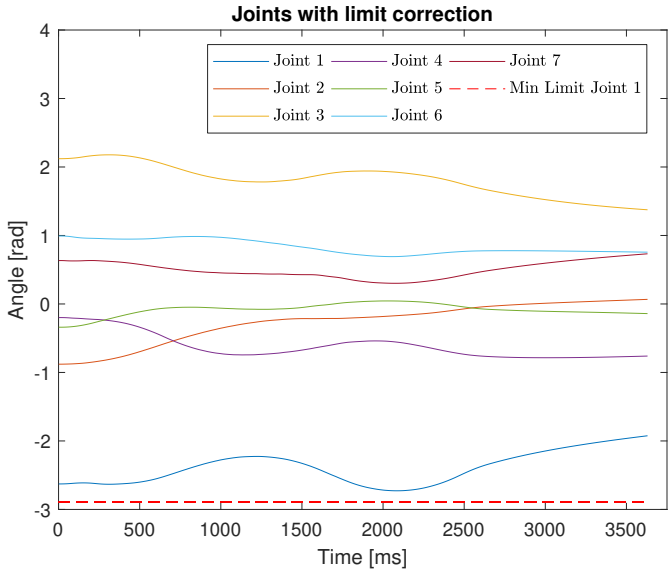


Figure 5.20: Joint trajectories where joint 1 avoids its limit.

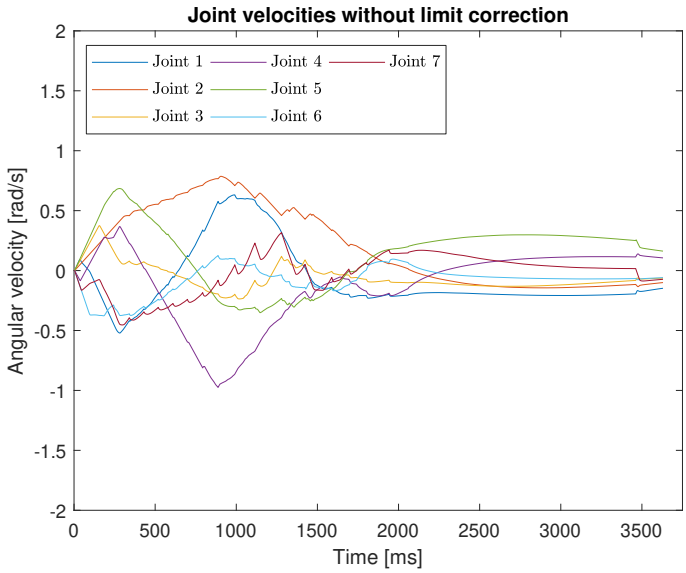


Figure 5.21: Joint velocity trajectories where joint 1 reaches its limit.

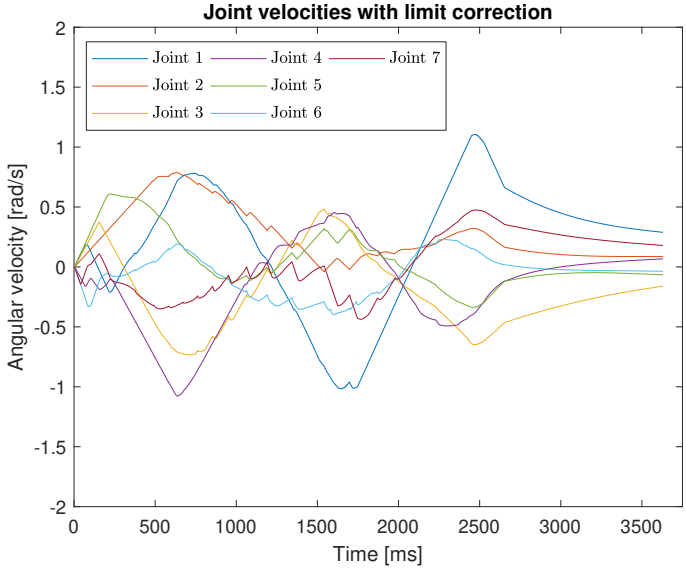


Figure 5.22: Joint velocity trajectories where joint 1 avoids its limit.

5.4.2 Experiment - Positive limit joint 1

Comparing Figure 5.23 and Figure 5.24 it is clear that joint 1 exemplary avoids it positive joint limit with the joint term $g(\dot{q}, q)$ added to the cost function. Furthermore, joints 3 and 4 perform the largest adjustments to achieve this avoidance. Figures 5.25 and 5.26 show somewhat large increases in joint velocities with the additional joint term in the cost function.

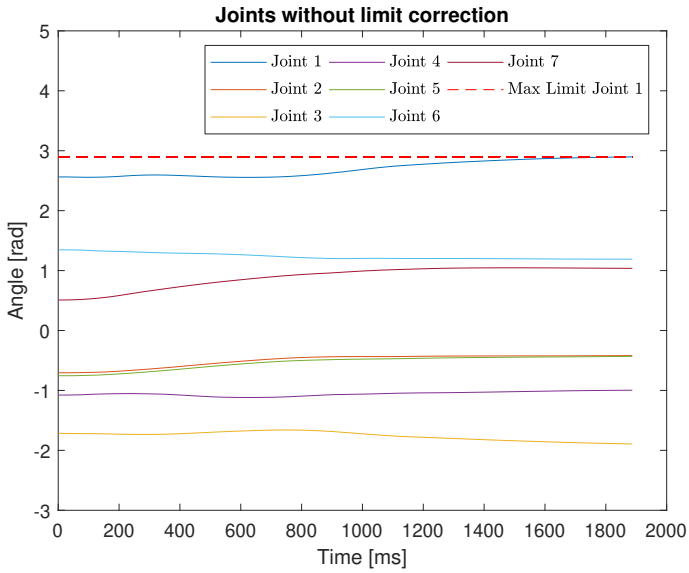


Figure 5.23: Joint trajectories where joint 1 reaches its limit.

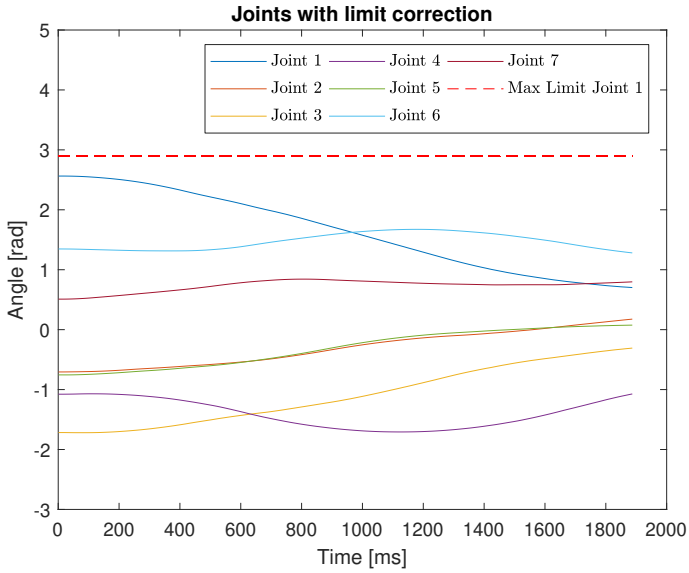


Figure 5.24: Joint trajectories where joint 1 avoids its limit.

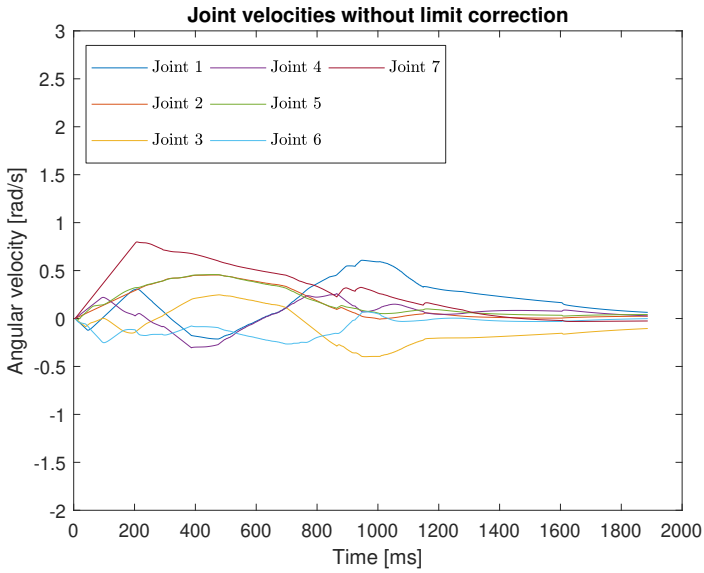


Figure 5.25: Joint velocity trajectories where joint 1 reaches its limit.

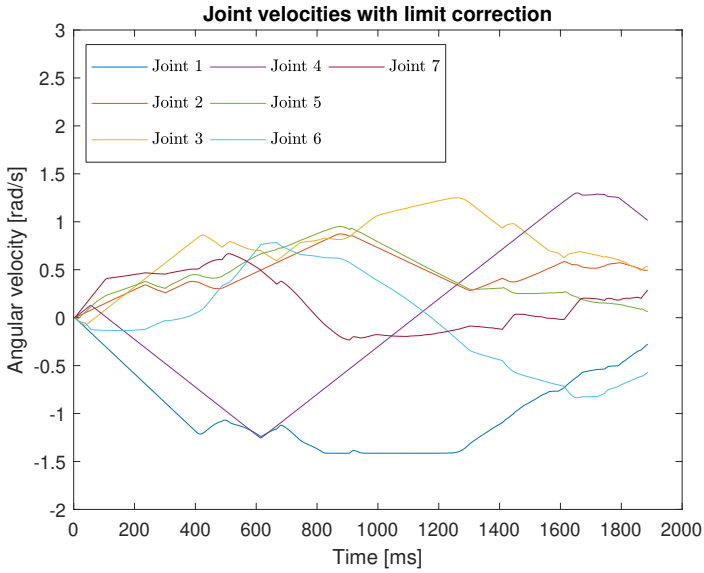


Figure 5.26: Joint velocity trajectories where joint 1 avoids its limit.

Chapter 6

Discussion

The following chapter presents a discussion for all main parts of the security net. Moreover, it attempts to explain all results from the preceding chapter that are not self-evident.

6.1 Path planning

By comparing Figure 5.5 with Figure 5.8 in section 5.1, it is indicated that the path planning successfully finds alternative waypoints that effectively allows the end-effector to execute the same place operation when newly present objects are impeding the shortest path. Nevertheless, the current implementation of the path planning is not without flaws. Figure 5.17 (a) in section 5.3 demonstrates an ineffective waypoint placement signified by the waypoints drawn in red. It is self-evident that a more optimal placement procedure would make the waypoints form a diagonal line, which would yield a straight line segment in the reference trajectory, which in return would make the tasks of the motion guidance and the motion controller significantly easier. The path planning procedure also supports modification of several parameters, i.e. c, K, v_{max} and v_{min} in (4.2), to weigh how far away from obstacles the waypoints should be [1].

It should also be clarified that the security net finds waypoints from the end-effector to a point A, placed at a certain distance away from the final goal. The operator of the security net specifies the Euclidean distance between point A and the goal position, in addition to the corresponding direction the end-effector should arrive from. Moreover, all experiments in the preceding chapter used a distance of $10cm$. The final path is then formed by temporary, evenly spaced waypoints on a line between point A and the goal position. A consequence of this mechanism is that it eliminates a conversion error between the voxel grid and the robot coordinates,

which allows the goal pose to be reached with minimal position errors[1].

6.2 Trajectory generation

The relaxed cubic B-spline successfully creates a C^2 continuous curve [7] based on the waypoints generated. Moreover, the algorithm that finds the closest curve parameter s^* shows effectiveness and robustness, as was promised in [27]. This makes the curve a reliable input to the motion guidance. Moreover, the curve parameter s turned out to be severely advantageous as it can be used to calculate the arc length of the trajectory, determine accurate cross-track errors, calculate distance traveled on the trajectory, and tune the motion controller.

6.3 Motion guidance

The motion guidance has significant room for improvement. For starters, the translational velocity references suffer from continual oscillations. A comparison between the translational velocity references in Figure 5.13 (a) and the cross-track error in Figure 5.12 (a) suggests that the references' oscillations seem to occur mostly when the cross-track error is minimal, i.e. when the linear combination between v_t and v_e in (4.11) has reached some sort of unstable equilibrium. Conversely, Figure 5.14 (b) displays the rotational velocity references as sufficiently smooth with the exception of the discrete jump in ω_{xref} from $-\pi$ to π at $\approx 4s$. For the clarity of the following explanation, keep in mind that the goal orientation Θ_{goal} was set to $[\phi, \theta, \psi] = [0, 0, 0]$ in that experiment. Hence, $\omega_{ref} = \Theta^2$ (see (4.12)) in Figure 5.14 (b). Consequently, it is clear that the measurements provided by the Panda robot regarding roll ϕ jump from $-\pi$ to π instead of continuing towards -2π, and vice versa. Nevertheless, with exceptional help from the redundant motion controller, the current motion guidance successfully moves the end-effector to its goal pose based on the B-spline trajectory. Furthermore, this thesis work has not conducted an extensive literature study on motion guidance and thus it should not be expected to be completely optimal.

6.4 Motion controller

For clarification, the *soft* motion controller in (3.1) is sometimes granted the word "soft" due to its ability to handle references that would, without limitation, violate one or more of the robot's constraints. Moreover, with the nonoptimal velocity references (depicted in Figure 5.14) as input, the motion controller is still able

[1] In the experiments the maximum position error was set to $1mm$. Smaller errors were not tested.

[2] Θ is the end-effector's orientation

to output joint velocities $\dot{q}$ that best follow the references without triggering the fail-safe mechanisms of the Panda robot. Figure 5.14 (b) in section 5.2 displays the output from the controller. Notice that for the second peak at $\approx 4s$, the one likely caused by the discrete jump in the rotational velocity reference, some joint trajectories seem moderately discrete. However, this is probably an effective way to return the robot to the trajectory while simultaneously satisfying a reduction in the rotational velocity error without violating the robotic manipulator's constraints. Moreover, oscillations in the controller output are most likely induced by the oscillations in the references from the motion guidance.

For the convenience of this discussion, the motion controller depicted in (3.1) is restated here with the tuning constants $\kappa(s)$ and $\chi(s)$ as functions of the curve parameter s:

$$
\begin{aligned}
\text{minimize} \quad & J_c = \gamma \left\| v - v_{ref} \right\|_2^2 + \kappa(s) \left\| \omega - \omega_{ref} \right\|_2^2 + \mu \dot{q}^T \dot{q} + \chi(s) g(\dot{q}, q) \\
\text{subject to} \quad & \begin{bmatrix} v \\ \omega \end{bmatrix} = J\dot{q}, \\
& -\dot{q}_{max} \leq \dot{q} \leq \dot{q}_{max}, \\
& -\ddot{q}_{max} \leq \ddot{q} \leq \ddot{q}_{max}, \\
& -\dddot{q}_{max} \leq \dddot{q} \leq \dddot{q}_{max}.
\end{aligned}
$$

Considering the individual terms of the cost function, they all fulfill their purpose and are weighed in a linear combination through their respective constants, which support real-time tuning. The first two terms regard minimizing the errors in Cartesian translational- and rotational velocity. They make the end-effector follow the trajectory with the output from the motion guidance. Furthermore, the constant $\kappa(s)$ is tuned real-time with a continuous and smooth (see Figure 4.10) increase in order to avoid large cross-track errors when the rotational velocity error is large.[3] The controller also avoids the computationally extensive calculation (here: approximation) of the Jacobian inverse, see (4.34) for clarity. In section 1.3.4 it was discovered that several methods benefit from the least square property of the pseudo-inverse, in that this yields the minimum joint velocity required to obtain the respective velocities v and ω. The motion controller in (3.1) can also achieve this by weighing μ high. Moreover, it can weigh each individual joint independently of the others.[4] The final term handling joint limits is granted a separate subsection due to its uniqueness and the compelling analysis it has received.

[3] Having a high, discrete increase in κ would cause noticeably large cross-track errors because the controller would weigh the rotational velocity errors over the translational velocity errors, due to how the motion guidance is constructed.

[4] If this is not clear consider the fact that $\dot{q}^T \dot{q}$ is a vector multiplication and thus μ should be set as a diagonal matrix between them, with each element in the diagonal weighing the individual joints. In mathematical terms, this looks like $\dot{q}^T M \dot{q}$.

6.4.1 Joint limit term

This discussion also relates to the supplementary experiments in Appendix B

By observing the cross-track error in Figure 5.12 (a) (from the experiment in Section 5.2) it is clear that the correction procedure caused by the joint limit term produces a noticeable cross-track error, reaching $\approx 13cm$ at its peak. On the other hand, the error is eliminated within $\approx 2.5s$. Hence, unless the area surrounding the end-effector at that particular moment is especially cluttered this is an acceptable trade-off. Furthermore, it is important to clarify that the initial joint configuration of the robot in that experiment was manually constructed to be particularly cumbersome with respect to the joint limits. Hence, the probability that the end-effector will have to deal with similar configurations without manual intervention is relatively small. The experiment could also be reconstructed with adjustments to the tuning parameters λ and c in (4.42). This might yield even better performance.

The figures comparing joint velocities in Section 5.4[5] display that in some cases a significant increase in joint velocities is necessary to perform the joint correction while still following the waypoints. On the other hand, to expect that no trade-off would be necessary when including the joint limit term would be a naive prediction. To limit the required increase in joint velocities, a helping factor might be to increase μ in the cost function, which will impose a larger cost of having large joint velocities when $g(\dot{q}, q)$ overrides the other terms. Furthermore, when the motion controller is governed mainly by $g(\dot{q}, q)$, i.e. when a joint correction maneuver is required, the security net does not take obstacles into account. Hence, the tuning constant $\chi(s)$ is set to zero at the final part of each trajectory. Although not optimal, this is done because the final parts of the robot trajectory usually involve a close encounter between the end-effector and arbitrary objects.

To summarize, the joint correction term fulfills its purpose by correcting the joints only when necessary. Furthermore, the increased tracking error required is quickly neutralized and the whole procedure satisfies a desired continuous behavior. Nevertheless, a safety check should run in advance of a joint correction procedure, in order to ensure that the correction does not produce a collision that would otherwise not happen.

[5] And Appendix B.

Chapter 7

Future work

The following chapter discusses several possible future improvements to this thesis' contributions in order to better realize a cooperative robotic manipulator.

As previously mentioned, the experiments presented in this thesis suggest that the motion guidance should be improved. For instance, the *LOS*-inspired method developed should receive further testing and be tuned to achieve more stable translational velocity references. One improvement to the translation velocity guidance would be to take the curvature of the spline into account. The curvature data can be computed for the entire spline when it is constructed, i.e. outside of the control loop. Additionally, there should be conducted an extensive literature study similar to the ones conducted in section 1.3, as they have proved extremely helpful for their respective implementations in this thesis.

An investigation regarding the Panda robot's ability to return orientation measurements for the end-effector should be conducted. Moreover, it should be verified that singularities are handled for the Euler angles. An alternative representation to the Euler angles would be to use unit quaternions. Unit quaternions use a four parameter representation to avoid the representation singularity of the Euler angles (chapter 2 in [28]).

The current implementation of the security net only considers obstacle avoidance for the end-effector. However, when the end-effector is carrying an object, this object's collision avoidance should also be considered. The object's size can be estimated using computer vision[1]. The problem can thus be described as temporarily having a larger end-effector size and form. A way to handle this problem might be to tune the parameters of the path planning algorithm, that can be weighed such that the waypoints form a path further away from obstacles.

The tuning of the motion controller deserves additional analysis. An interesting

[1] As previously mentioned, the Neodroid project will include computer vision for the robot, in order to properly map obstacles.

solution to test would be to weigh the optimization criteria of the cost function by combining a task priority strategy with fuzzy interference, similar to what was done by Bi et al.[20].

Collision avoidance for the end-effector is an important feature of this thesis's implementation. In order to satisfy the requirements of a robot that can cooperate with humans in the same configuration space, all 7 joints' respective positions and velocities should be considered. As previously mentioned, the redundancy feature of the manipulator allows infinite joint configurations for a given end-effector pose. This redundancy can be exploited to change the robot configuration (joint coordinates) to safely react to external forces while still following the desired end-effector trajectory. Two intuitive methods will be suggested here. The former method is to generate virtual, repulsive forces around the obstacles, creating a 3D potential field, and imposing these forces on the respective joints of the robot. This technique can be considered to be a global obstacle avoidance technique. The latter method is more tilted towards a response behavior when a collision has occurred for one of the robot- joints or links. The idea is to utilize the properties of the joint limit term $g(\dot{q}, q)$, defined in (4.42). Consider the case when joint 3 has a rotational velocity in the direction displayed in Figure 7.1, which leads to a collision with the human hand. For simplicity, assume all other joint velocities are 0. Moreover, assume that the collision occurs when $\theta_3 = 1.0rad$. Temporarily modifying $q_{max,soft}$ in table 1.1 for joint 3 from 2.8973 to a value below 1.0 rad might, with proper parameter tuning in the soft controller, enforce joint 3 to reduce in size, consequently moving the link/joint at impact away from the obstacle.[2] From this point, multiple correction procedures exist. The robot could recalculate the reference path with a larger penalty for traversing close to obstacles. This is easily achieved by altering certain parameters in the path planning algorithm [1]. Another possibility is to increase the magnitude of the repulsive forces the impacted obstacle emits, assuming that the aforementioned potential field method is implemented.

[2]This joint enforcement technique has been tested, and the robot obtains the new joint space configuration. On the other hand, the collision avoidance ability while the correction procedure is executing has not been sufficiently tested.

Figure 7.1: Robot collision with human hand. Original image taken from [4]

The current implementation sets the joint term's constant χ to zero for the final part of the trajectory. This is done because the joint correction procedure does not consider obstacle avoidance. This is not optimal and was made as a temporary safety mechanism until an appropriate procedure has been implemented. A better solution might be to use a passive joint limit term for the final part of the trajectory: if the cost term $g(\dot{q}, q)$ becomes too large during the final traversal part, the robot should reverse the movement to a safe spot, then perform the correction on the joint at risk, before it continues its original task.

Another important thing that should be addressed in future work would be to test the stability of the security net. Moreover, a detailed stability analysis should be conducted for the motion controller. This would also provide a basis for the tuning of the controller parameters, which is a desired alternative to simply using experience.

Chapter 8

Concluding remarks

The presented thesis has developed a complete system to control a robotic manipulator. Moreover, the system is applicable as a virtual security net that includes path planning and obstacle avoidance, trajectory generation on the generated waypoints combined with a velocity guidance, and a unique soft motion controller with high potential in the robotics field. Most features were developed based on extensive literature studies, where multiple popular methods were compared, and a choice concerning which method(s) to adapt followed. Furthermore, the thesis has presented several experiments and results of the security net that helped in forming a self-critical discussion. Nevertheless, several improvements can be made to the security net, which have been thoroughly discussed.

Appendices

Appendix A

Outdated motion guidance

As previously mentioned, the velocity guidances described in this appendix were developed antecedent to the development of the continuous spline trajectory described in section 4.2.

A.1 Translational velocity

A.1.1 Direction

First, assume that all waypoints are enclosed by spherical zones with a radius equal to the size of each voxel in the grid, see "Circle of acceptance" in Figure A.1. Let waypoint n be defined as the waypoint that has the smallest Euclidean distance from its waypoint center to the end-effector position. In Figure A.1 this would be waypoint $n = 1$. Furthermore, v_i is defined as the vector pointing from waypoint i to the subsequent waypoint center $i + 1$. If the reference waypoint for the end-effector is always defined as waypoint $n + 1$, and the vector v_e is pointing from the end-effector position towards the reference waypoint, the reference velocity for the end-effector is given by

$$v(t) = c_1(t)v_e + c_2(t)v_{n+1}, \tag{A.1}$$

where $t \in [0, 1]$ is a line parameter formed by a straight line between waypoints n and $n + 1$. The varying constants $c_1(t)$ and $c_2(t)$ have to satisfy the equality constraint

$$c_1(t) + c_2(t) = 1, \ \forall \ t. \tag{A.2}$$

If the end-effector position is outside the circle of acceptance of waypoint n, then $c_1(t) = 1$ and thus $c_2(t) = 0$. On the other hand, if the end-effector position is within the circle of acceptance of waypoint n, the values of the constants satisfy the linear relationship in Figure A.2. In theory, this would create relatively smooth reference signals for the velocity of the end-effector.

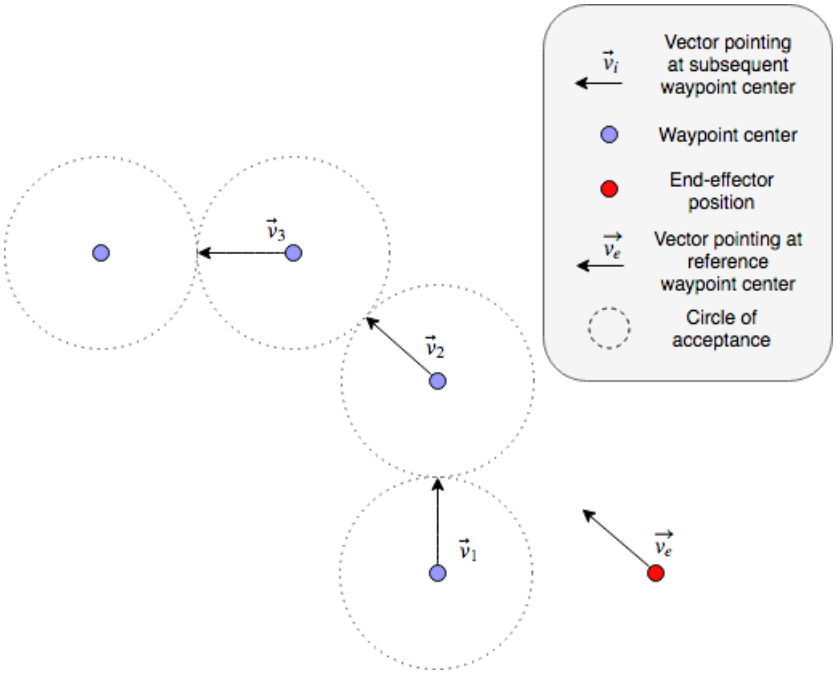


Figure A.1: Example of waypoints with corresponding vertices between the centers

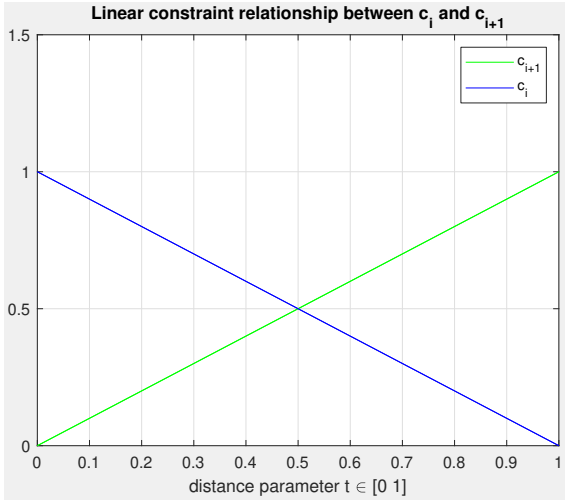


Figure A.2: Linear constraint relation between two consecutive waypoints

A.2 Rotational velocity

A.2.1 Method 1

During early testing of the security net, the workspace of the manipulator was restricted to a limited area in front of the manipulator for safety reasons. Furthermore, it was limited to a rectangular cuboid in front of the robot. An orientation guidance using biomimicry[1] was suggested. Moreover, the movement of a snake was the inspiration for the desired end-effector behavior. To obtain this behavior, the guidance separated between moving the end-effector in positive and negative x-direction. If the path moved in positive x-direction (away from the robot base), the end-effector should behave like the head of a snake. Conversely, for movements in the negative x-direction the end-effector should behave like the tail of a snake. Consequently, the desired roll is given by

$$\phi_d = 0. \tag{A.3}$$

Looking at Figure A.3, "θ_{neg}" implies that the showcased angle is in the negative rotation wrt. to the y-axis, implying that the y-axis is pointing out of the figure (right-hand-rule). Therefore, the desired pitch angle is given by

[1] Biomimicry is a "new" science that studies nature's best ideas and then imitates these designs and processes to solve human problems. Studying a leaf to invent a better solar cell is an example.[29]

$$\theta_d = \begin{cases} -arctan(\frac{\Delta z}{|\Delta x|}), & \text{if } \Delta x \neq 0 \\ -sgn(\Delta z)\frac{\pi}{2}, & \text{if } \Delta x = 0, \end{cases} \tag{A.4}$$

where $sgn(\cdot)$ is the signum function defined as

$$sgn(x) = \begin{cases} -1 & \text{for } x < 0, \\ 0 & \text{for } x = 0, \\ 1 & \text{for } x > 0. \end{cases} \tag{A.5}$$

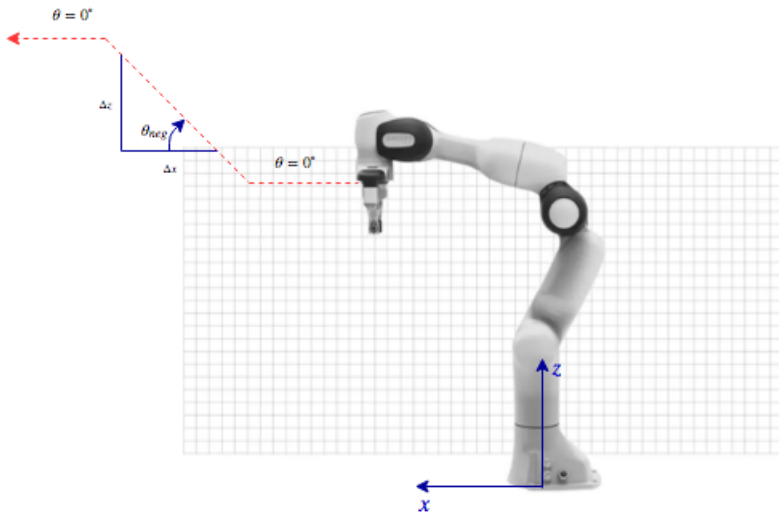


Figure A.3: Franka Emika Panda seen from side. The dashed red line represents a trajectory formed by creating linear segments between waypoints, arbitrarily chosen.

Looking at Figure A.4 and applying similar logic, the desired yaw angle is given by

$$\psi_d = \begin{cases} arctan(\frac{\Delta y}{|\Delta x|}), & \text{if } \Delta x \neq 0 \\ sgn(\Delta y)\frac{\pi}{2}, & \text{if } \Delta x = 0. \end{cases} \tag{A.6}$$

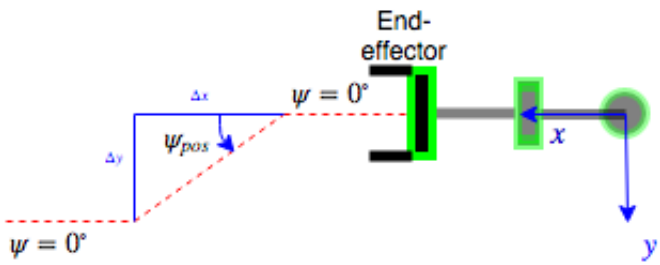


Figure A.4: Arbitrary robotic manipulator seen from top. The dashed red line represents a trajectory formed by creating linear segments between waypoints, arbitrarily chosen.

By utilizing the absolute value around Δx, the formulas work independently of which x-direction the end-effector is traveling.

A.2.2 Method 2

As previously mentioned, this method distinguished between a longer, rough path and a shorter, fine path. The longer part had the following Euler angle references

$$\phi_d = 0, \tag{A.7}$$
$$\theta_d = 0, \tag{A.8}$$
$$\psi_d = arctan(\frac{y}{x}), \tag{A.9}$$

where ϕ and θ are roll and pitch, respectively, and ψ is the yaw given by the x and y positions of the end-effector, see Figure A.4. Hence, the end-effector would ideally face away from the base of the robot. The idea was that this would help prevent excessive turning on the base joint. For the shorter path, the reference was similar to the current method, i.e. (4.12). The difference is that κ was a constant set to a larger number than γ in (3.1).

Appendix B

Joint term supplementary experiments

B.1 Experiment - Positive limit joint 3

Comparing Figure B.1 and Figure B.2, it is clear that joint 3 avoids its positive joint limit when including (4.42) in the cost function. The comparison between the joint trajectories implies that joint 5 might be compensating for joint 3, allowing joint 3 to reduce while simultaneously attempting to follow the reference path for the end-effector. Figure B.3 and Figure B.4 display that, at least in terms of magnitude, the joint limit avoidance procedure did not require large shifts in joint acceleration.

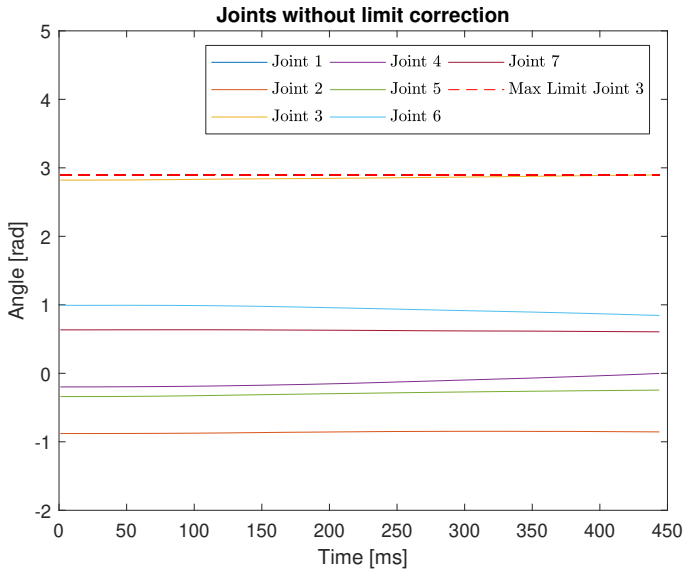


Figure B.1: Joint trajectories where joint 3 reaches its limit.

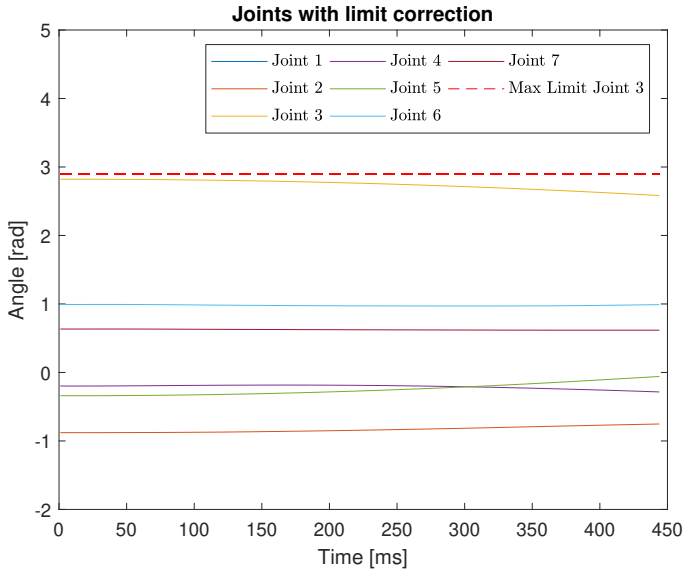


Figure B.2: Joint trajectories where joint 3 avoids its limit

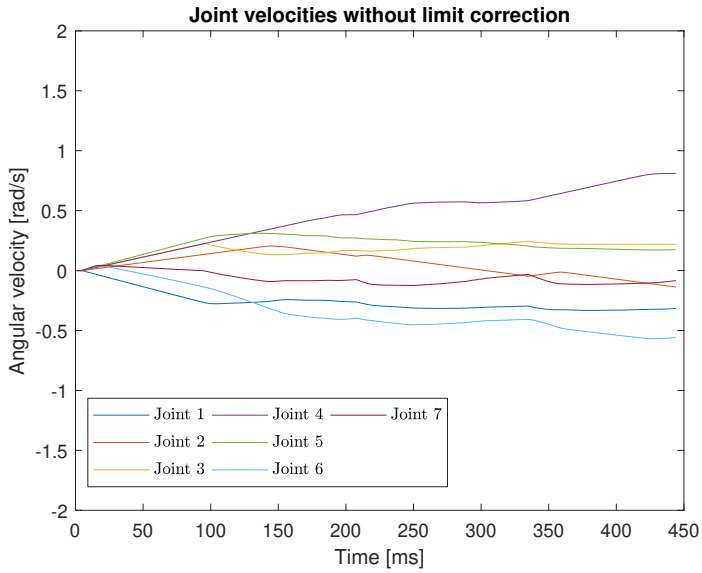


Figure B.3: Joint velocity trajectories where joint 3 reaches its limit.

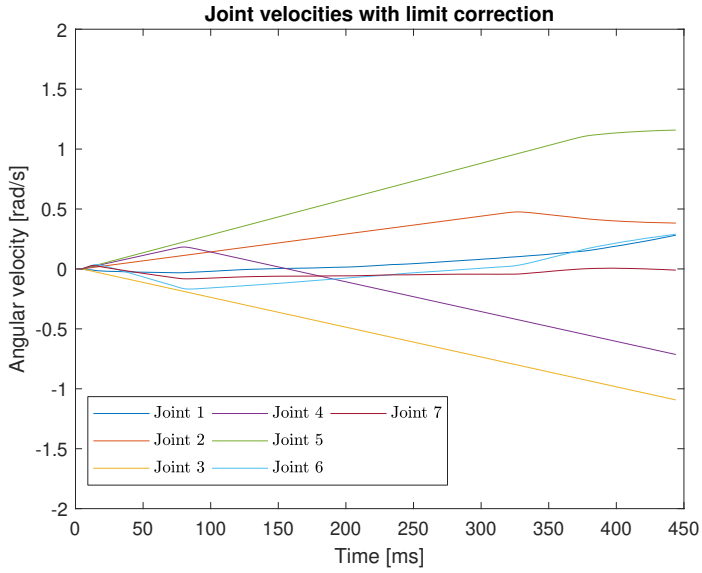


Figure B.4: Joint velocity trajectories where joint 3 avoids its limit.

B.2 Experiment - Positive limit joint 4

Comparing Figure B.5 and Figure B.6 joint 4 is seen to avoid its limit with the joint term in (4.42) added to the cost function. Moreover, it seems that all joints except joint 5 are adjusted to compensate for the necessary change. Figures B.7 and B.8 show that only minor velocity adjustments are required in terms of magnitude.

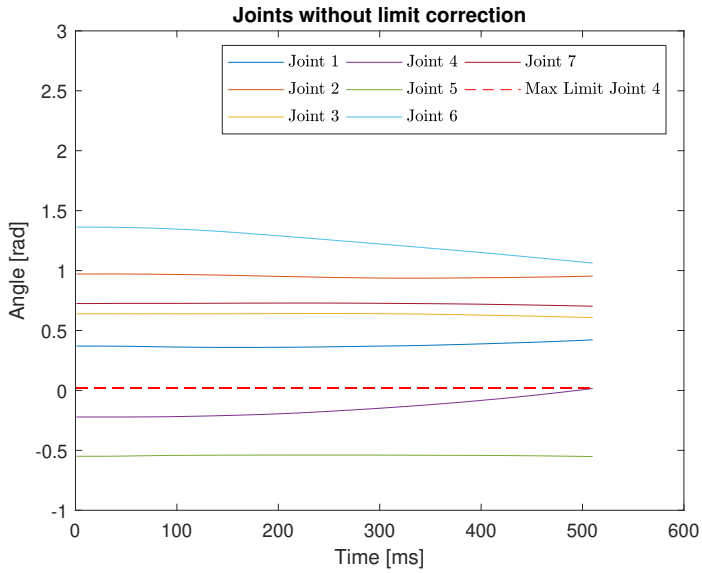


Figure B.5: Joint trajectories where joint 4 reaches its limit.

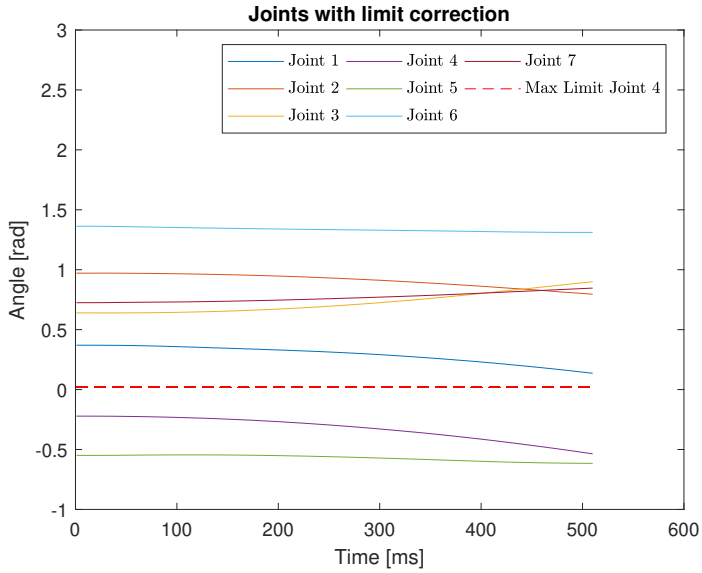


Figure B.6: Joint trajectories where joint 4 avoids its limit.

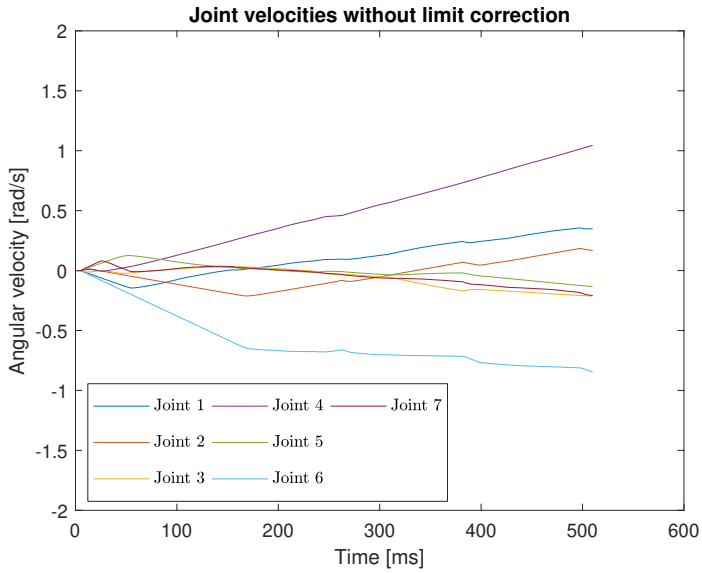


Figure B.7: Joint velocity trajectories where joint 4 reaches its limit.

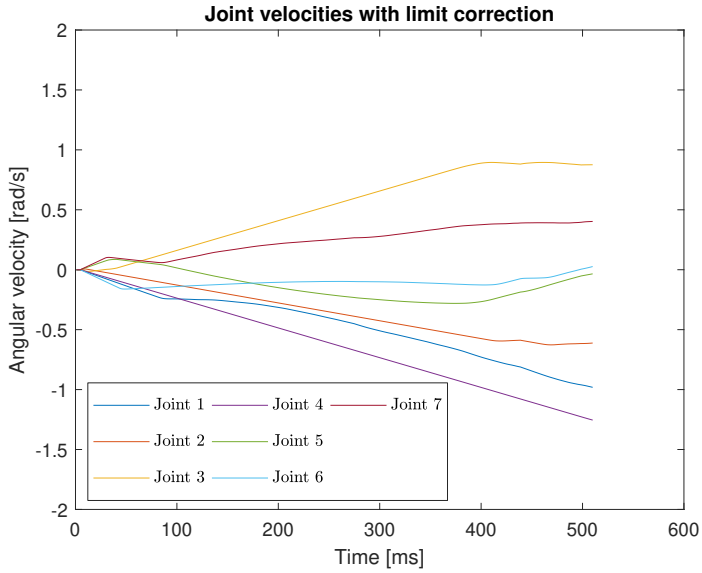


Figure B.8: Joint velocity trajectories where joint 4 avoids its limit.

Bibliography

[1] H. Grøtte, "Specialization project -Virtual security net for a learning robot," 2017.

[2] J. Mattingley and S. Boyd. (2013-) Cvxgen: Code generation for convex optimization. [Online]. Available: https://cvxgen.com/docs/index.html

[3] J. Chen and H. Y. Lau, "Inverse kinematics learning for redundant robot manipulators with blending of support vector regression machines," in *Proceedings of IEEE Workshop on Advanced Robotics and its Social Impacts, ARSO*, 2016.

[4] F. Emika. (2017-) Franka emika's official homepage. [Online]. Available: https://www.franka.de/

[5] F. Emika. (2017-) Franka control interface. [Online]. Available: https://frankaemika.github.io/docs/index.html

[6] CartouCHe. (2012) Computer graphics - bézier curve. [Online]. Available: http://www.e-cartouche.ch/content_reg/cartouche/graphics/en/html/Curves_learningObject1.html

[7] K. A. Baker, "Cubic spline curves," *University of California, Los Angeles, Mathematics, Math 149, W02, DD.*

[8] A. E. Foka and P. E. Trahanias, "Predictive Control of Robot Velocity to Avoid Obstacles in Dynamic Environments," 2003.

[9] S. Wardani, V. Halfiani, S. Munzir, and T. Usman, "Modified formula for velocity and acceleration setting in Obstacle Avoidance problem," in *Proceedings - 2016 12th International Conference on Mathematics, Statistics, and Their Applications, ICMSA 2016: In Conjunction with the 6th Annual International Conference of Syiah Kuala University*, 2017.

[10] F. Belkhouche, "Reactive path planning in a dynamic environment," *IEEE Transactions on Robotics*, 2009.

[11] E. Owen and L. Montano, "Motion planning in dynamic environments using the velocity space."

[12] A. Zelenak, C. Peterson, J. Thompson, and M. Pryor, "THE ADVANTAGES OF VELOCITY CONTROL FOR REACTIVE ROBOT MOTION."

[13] V. Hlaváč, "Robot trajectory generation," *Czech Technical University in Prague, Center for Machine Perception: Czech Institute of Informatics, Robotics and Cybernetics and Faculty of Electrical Engineering, Department of Cybernetics*.

[14] T. Horsch and B. Jüttler, "Cartesian spline interpolation for industrial robots," *CAD Computer Aided Design*, 1998.

[15] B. Sencer and E. Shamoto, "Curvature-continuous sharp corner smoothing scheme for Cartesian motion systems," in *International Workshop on Advanced Motion Control, AMC*, 2014.

[16] M. Švejda and T. Čechura, "Interpolation method for robot trajectory planning," in *Proceedings of the 2015 20th International Conference on Process Control, PC 2015*, 2015.

[17] A. Gasparetto and V. Zanotto, "Optimal trajectory planning for industrial robots," *Advances in Engineering Software*, 2010.

[18] R. M. Murray, Z. Li, and S. S. Sastry, "A Mathematical Introduction to Robotic Manipulation." [Online]. Available: http://www.cds.caltech.edu/ http://www.crcpress.com/product/isbn/9780849379819.

[19] B. Siciliano, "Kinematic Control of Redundant Robot Manipulators: A Tutorial," *Journal of lntelligent and Robotic Systems*, vol. 3, pp. 201–212, 1990.

[20] W. Bi, X.-J. Liu, F. Xie, and W. Ding, "Motion Control Strategy of Redundant Manipulators Based on Dynamic Task-Priority," 2017.

[21] C. Schuetz, J. Pfaff, F. Sygulla, D. Rixen, and H. Ulbrich, "Motion Planning for Redundant Manipulators in Uncertain Environments based on Tactile Feedback," *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2015.

[22] S. Chiaverini, "Singularity-robust task-priority redundancy resolution for real-time kinematic control of robot manipulators," *IEEE Transactions on Robotics and Automation*, 1997.

[23] T. Saito and J. I. Toriwaki, "New algorithms for euclidean distance transformations of an n-dimensional digitised picture with applications," *Pattern Recognition*, vol. 27, no. 11, pp. 1551 – 1565, 1994.

[24] W. H. H. A. Meijster, J. B. T. M. Roerdink, "A general algorithm for computing distance transforms in linear time," *Goutsias J., Vincent L., Bloomberg D.S. (eds) Mathematical Morphology and its Applications to Image and Signal Processing. Computational Imaging and Vision*, vol. 18, 2002.

[25] M. P. Kamermans. (2011-2017) A primer on bézier curves. [Online]. Available: https://pomax.github.io/bezierinfo/

[26] X. D. Chen, Y. Zhou, Z. Shu, H. Su, and J. C. Paul, "Improved algebraic algorithm on point projection for bézier curves," in *Proceedings - 2nd International Multi-Symposiums on Computer and Computational Sciences, IMSCCS'07*, 2007.

[27] H. Wang, J. Kearney, and K. Atkinson, "Robust and Efficient Computation of the Closest Point on a Spline Curve," *Proc. 5th Int. Conf. Curves Surf.*, 2002.

[28] T. Fossen, *Handbook of Marine Craft Hydrodynamics and Motion Control*. John Wiley & Sons, 2011.

[29] J. Campos. Biomimetic design. [Online]. Available: https://pages.stolaf.edu/bio-architecture/what-is-biomimicry/