Compliance of a Robot Arm using Torque-Based Cartesian Impedance Control

Oussama Chouman

LUND UNIVERSITY

Department of Automatic Control

MSc Thesis
TFRT-6147
ISSN 0280-5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

Abstract

Robots have become important for the development of today's society. They are known to obtain good accuracy and repeatability of tasks that either complement or replace human labour. Furthermore, as machine-learning is emerging in many technologies, it is important to consider the uncertainties that are introduced when this method is used in robotics. This work aims at developing and evaluating a torque-based control framework that allows a state-of-the-art robot arm to be seen as an impedance, and its environment as an admittance. Using said control strategy allows the robot to behave as a mass-spring-damper system, which allows it to act more compliantly. Moreover, the type of strategy is referred to as Cartesian impedance control.

The framework was developed both in a simulation environment called Dynamic Animation and Robotics Toolkit (`DART`), but also in the common robotics environment called Robot Operating System (ROS). The latter framework was in turn used to run the control strategy on a real robot arm.

The results showed compliant behaviour of the robot in Cartesian space, both in simulation and on the real system. Translational, rotational and nullspace compliances were tested and evaluated. For reasonably high stiffness values, these experiments showed the expected behaviour when the robot was subjected to external forces. Moreover, the behaviour of the robot in selected singularities was also studied, and the robot was stable in all the tested singular configurations with no apparent oscillations. The experiments also revealed some limitations of the real system, allowing the robot to behave sufficiently well only under certain bounds. That is, the inheritance of friction on the real system limited the tracking ability when lower stiffness values were used. A peg-in-hole assembly experiment was also done both on the real system and in simulation, in order to get a sense of how the robot performs in the intended contact-rich tasks. The results of these experiments showed that the robot could insert the attached peg into a box with a hole, after sliding along the

surface of the box for some time. The peg insertion in simulation was about twice as fast compared to the real robot. Moreover, the specific path of the end-effector in simulation compared to the real system did not match precisely due to uncertainties and limitations such as calibration errors and friction.

Acknowledgements

I would like to take the opportunity to express my uttermost gratitude to my supervisors, Dr. Björn Olofsson, Ph.D. students Matthias Mayr and Julian Salt Ducaju, for the continuous advice, explanations and motivation throughout the whole project. This work would certainly not have been possible without the guidance and support of my supervisors.

I would also like to thank Joel Holmesson for giving me insight into his M.Sc. Thesis work, which was performed in parallel and was similar to my thesis project in many aspects.

Contents

1 Introduction

Robots are becoming a more prevalent part of today's society [Statista Research Department, 2021]. They are suitable for performing a multitude of labour tasks in many types of industries. Some of the labour can be fulfilled solely by robots, while other types of tasks require interactions with a partially unknown environment, or human collaboration. In order to accomplish tasks, the robots are usually equipped with a peripheral device called the *end-effector*, on which different types of customizable tools can be mounted [Denning, 2017]. The end-effector is hence a crucial part in robotics, and it usually needs to be tracked in terms of position and orientation during robot tasks.

Contact-rich tasks usually require some sort of controller that allows the robot to become more compliant with the environment. One approach of dealing with this is by implementing a control strategy, which, in a sense, makes the manipulator "softer". More specifically, one such strategy is known as *impedance control* [Hogan, 1984]. This type of control makes the robot act as a virtual mass-spring-damper and applies said spring between the desired position and the actual position of the robot. Essentially, the robot is treated as a mechanical impedance, while the environment is treated as an admittance. Mechanical impedance is in fact the relationship between force and position. Therefore, damaging equipment through collisions becomes less likely as the contact forces would not be as elastic. This is especially favourable when the tasks are contact-rich with the environment.

Another important aspect in robotics is the requirement of obtaining accurate robot movements while doing tasks [Freidovich, 2013, p. 10]. More specifically, robot arms that are used in industry should follow planned trajectories sufficiently well. To obtain a precise movement, a reliable model of the robot needs to be developed. However, it is not always the case that the robot follows the planned plan sufficiently well [Płaczek and Piszczek, 2018]. This may be due to several reasons and varies for each robot model. Physical limitations and uncertainties of the control signals with respect to the actual robot dynamics are some examples, which may lead to exigent consequences. These in turn would bring on unexpected collisions or failure to perform the desired task.

While machine-learning is emerging in many fields of technology, robotics is certainly not excluded [Williams, 2018]. One aspect in machine-learning is based on automatically finding an optimal strategy by iterative trial-and-error interactions, namely reinforcement learning (RL) [Kaelbling et al., 1996]. Moreover, RL has the potential to deal with complex tasks, such as robot manipulation, which would otherwise be hard to implement using traditional programming. But as the name suggests, the manipulations would be subjected to deviations as the algorithm is learning from its mistakes. There are multiple ways of dealing with the effects of said uncertainties. A common way is by using some type of compliance control, for example, the previously mentioned *impedance control*.

1.1 Problem Statement

The main objective of this project is to develop and evaluate a control strategy that integrates with the open source package collection Robot Operating System (ROS) [Dattalo, 2018]. More specifically, the type of strategy that is of interest is a torque-driven impedance controller in Cartesian space. This should then be applied on a real 7 degree of freedom (DoF) state-of-the-art robot arm, namely the *LBR iiwa* by *KUKA AG* [KUKA, 2021] (see Figure 1.1), but also simulated using the Dynamic Animation and Robotics Toolkit (`DART`) [Lee et al., 2018]. Since `DART` is not a part of ROS, a *base library* that separates the control system from ROS should also be developed. Some important properties of the base library should be that it is not dependent on ROS, and that it should contain a set of functionalities that run the calculations of a Cartesian impedance controller. This is explained further in Chapters 3 and 4.

In addition to this, the following features are of interest:

- Being able to configure some parameters, like stiffness and damping of the controller, during run-time.
- Being able to command a force or torque along a specific direction, that is then exerted at the end-effector, during run-time.

The specific parameters that should become re-configurable during operation will be discussed in the following chapters, but essentially these make up for the overall impedance of the robot.

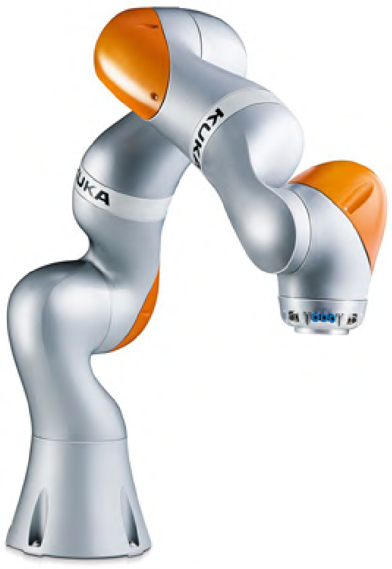


Figure 1.1: Robot arm used for this project [KUKA, 2021].

As of now, RL methods allow to master complex tasks in simulations [OpenAI, 2018]. However, in order to make them more trustworthy and reliable in real scenarios, a framework that works well both in simulation and on the real robot is desired.

Many tasks require a relocation of the end-effector. For this, a trajectory generator is needed and will therefore also be developed, during the course of the project.

Related Work

This project is, in many aspects, similar to the one of another student's thesis work. In particular, the thesis that is being referred to here is "Accurate Simulation of a Collaborative Robot Arm with Cartesian Impedance Control" by Joel Holmesson, where the main objective was to find a control strategy for the robot, so that the real robot and the simulations match well [Holmesson, 2021, pp. 10-11]. The control framework that was developed was, however, based on forward dynamics, and used feedback from force-torque measurements at the end-effector in the controller. Moreover, the related work used joint positions as outputs of the controller, rather than joint-torque control. There are advantages and disadvantages for both methods, but they will not be discussed further here.

Besides this, the simulation environment of focus was different, as this work focused more on the usage of `DART`, while the simulations in the related work were produced from a ROS-integrated 3D-simulator called `Gazebo` together with the Open Dynamics Engine (ODE) as physics engine [Gazebo, 2014]. The approach for this project is thus a different one from the work of Holmesson.

Scientific Basis In this work, the scientific basis in terms of impedance control is built upon the book by Ott (2008), which is mainly dedicated on the topic of torque-based Cartesian impedance control. As for the related work, the controller is built upon the presented paper by Scherzinger et al. (2017).

1.2 Aim and Limitations

The targeted application of the framework is mainly in a reinforcement-learning scenario, but also in combination with kinesthetic teaching of robots. The results will mainly contribute to the development of a safer environment for robots that work with contact-rich tasks, but also when the robots operate in collaboration with humans [Abu-Dakka and Saveriano, 2020]. By including an impedance controller in robots it is also possible to apply experimental methods more safely. Moreover, by developing the *base library* mentioned earlier for the control system, it would be possible to use the same controller in other independent robotic simulation software.

The thesis will be based upon existing material and solutions, that is, both a theoretical foundation, but also a skeleton framework of the control strategy. What this work is intended for is giving new insights of how the control strategy would perform on a practical system, and how it may differ from the theoretical expectations. Furthermore, the framework intends to set a foundation in which the Cartesian impedance controller becomes integrated with the `DART` environment. Thus, future works could use the framework as a starting point for kinesthetic teaching of state-of-the-art robots.

An issue may arise from intrinsic non-linearities in the robot dynamics, where, e.g., identifying counter-intuitive friction effects in the system can become a demanding and time-consuming task. Therefore, system identification and friction compensation are not included in the scope of this thesis.

1.3 Outline

The structure of the report is shaped as follows. In Chapter 2, a basic overview of the relevant theory is presented. Chapter 3 gives an overview of the specific robot and software interface that was used. The methodology of the work is described in Chapter 4. In Chapters 5 and 6, the experiments and results are presented and discussed. Lastly, Chapter 7 concludes the results and some ideas for future work are proposed.

2

Theory

In this chapter an introductory background to the relevant theory for the work is described. Firstly, a brief overview of the general concepts in robotics is presented in Sections 2.1 and 2.2. A short description of Cartesian trajectory planning is given in Section 2.3. In Section 2.4, a common modeling approach to the dynamics of a robot arm is presented. Lastly, Section 2.5 introduces the concept of Cartesian impedance control for a rigid manipulator.

2.1 Basics of Robotics

Robotics is a branch of science and technology which combines many other technology fields such as electromechanics, computer science and automation [Infineon, 2018]. Robots are mainly used as a substitution for, or complemented to human labor in hazardous environments or when the tasks at hand become too complicated or repetitive to perform manually. They are also commonly used in collaboration with humans to achieve shared goals. There are many types of robots which come in different shapes, sizes, etc. One of the most commonly used today are the articulated robots (also referred to as robot arms) [Guarana-DIY, 2020]. As this work is done using a robot arm, the following theory will be shaped solely for this type of robot. Furthermore, it will be assumed that the parts of the robot are rigid.

Joint Space

The mathematical descriptions and tools are based on a geometrical model-representation of the robot with simplifications. For instance, it is assumed that the robot is built by a serial chain of rigid links, in which the position and velocities for each joint are given. The joints are also assumed to be revolute and independent from the other joints and thus contribute to one DoF each, which makes it easier to manage the mathematical modeling of the components needed.

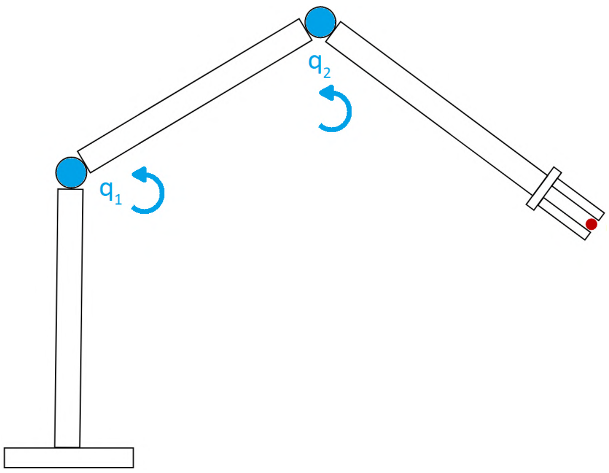


Figure 2.1: A simple robot model with two revolute joints, q_1 and q_2. A red dot (tool center point) is held in place by a gripper tool (end-effector).

From a chain of links the joint space is defined by a set of general coordinates, also known as the configuration,

$$q = \begin{pmatrix} q_1 & q_2 & \dots & q_n \end{pmatrix} \tag{2.1a}$$

$$\dot{q} = \begin{pmatrix} \dot{q}_1 & \dot{q}_2 & \dots & \dot{q}_n \end{pmatrix} \tag{2.1b}$$

where n is total number of independent joints, q_i is the i:th revolute joint angle and $\dot{q}_i$ is the joint velocity of said joint. A simple model of a robot arm is shown in Figure 2.1.

Coordinate Frames

In robotics it is important to keep track of where objects, sensors, robot joints and the end-effector are located in space. This is usually done with the help of coordinate frames. In order to describe the locations of the frames in space, they must be measured in relation to some other frame [Sprague, 2016]. Usually, the origin is set as the *base* frame. Furthermore, each frame is described as the composition of its position and orientation in space (also referred as its *pose*).

For the position, it is quite straight forward to use the Cartesian system with one axis per dimension. In three-dimensional space, the axes are usually denoted x, y and z, respectively. On the other hand, the orientation can be represented in several ways, for example using Euler angles, rotation matrices or quaternions.

The choice of which representation to use depends on the application, since they have advantages and disadvantages relative to one another. For instance, quaternions are less computationally demanding and more numerically stable, but on the other hand less intuitive to understand and visualize compared to Euler angles [Ben-Ari, 2014].

Transformations

In many applications, including robotics, the relationship between two arbitrary coordinate frames is often described using homogeneous transformations [Freidovich, 2013, Ch. 2]. The matrix representation of a homogeneous transformation is defined as follows,

$$H^0 = \begin{pmatrix} \mathbf{R}^0 & \mathbf{d}^0 \\ \mathbf{0}_{1\times 3} & 1 \end{pmatrix} \tag{2.2}$$

where the rotation matrix $\mathbf{R}^0 \in \mathbb{R}^{3\times 3}$ belongs to the group of rotations $(SO(3))$, describing the relative rotation, and $\mathbf{d}^0 \in \mathbb{R}^{3\times 1}$ describes the relative translation. Note that the lower row of (2.2) is only included so that multiplication of transformation matrices can be performed.

To illustrate how (2.2) is used, consider two arbitrary frames denoted as the 0-frame and 1-frame, respectively. Furthermore, some known position $P^0 = \begin{pmatrix} p^0 & 1 \end{pmatrix}^T$, $p^0 \in \mathbb{R}^{3\times 1}$ in the 0-frame is assumed to be known, where the last element in P^0 is, just as the second row of (2.2), introduced as an auxiliary element. Then, in order to represent the same point but in the 1-frame, the following formula can be used,

$$P^0 = H_1^0 P^1 \tag{2.3}$$

where

$$H_1^0 = \begin{pmatrix} \mathbf{R}_1^0 & \mathbf{d}_1^0 \\ \mathbf{0}_{1\times 3} & 1 \end{pmatrix} \tag{2.4}$$

is the homogeneous transformation between the 0-frame and 1-frame. The details of how rotational operations are performed is left out here, but is explained in more detail by Freidovich (2013).

2.2 Kinematics

The main idea of kinematics in robotics is to find a relationship between the joint configuration and the end-effector pose of a robot. Furthermore, in order to efficiently use a robot in different applications, the end-effector is a necessary component [Denning, 2017]. Consequently, kinematics needs to be used in the development of robotic control systems and applications.

Essentially, the desired relation is obtained through strategically attached coordinate frames along the joints of the robot. The translational and rotational relationships between the frames are, in turn, usually described by homogeneous transformation matrices for example using the Denavit-Hartenberg convention [Freidovich, 2013, Ch. 2]. In Figure 2.2, the serial chain for the robot arm used in this project is shown.

Forward Kinematics

The procedure of finding the pose of the end-effector, is carried out by non-linear equations which in turn are used to map the configuration space in (2.1a), to the Cartesian pose [Freidovich, 2013, Ch. 2]. This is also known as *forward kinematics* and can be defined mathematically as follows. Given a set of joint angles $q \in \mathbb{R}^n$, the pose of the end-effector $x \in \mathbb{R}^m$ (not to be mixed with one of the Cartesian dimensions denoted by x) is obtained by computing

$$\begin{aligned} &x = f(q) \\ &f : \mathbb{R}^n \to \mathbb{R}^m \end{aligned} \tag{2.5}$$

where f is the map-function and m is the number of variables of the end-effector. Depending on how the orientation is described, the value of m will vary. The minimum amount of variables needed for full representation of the end-effector is six, i.e., three dimensions for the position and three for the orientation.

Workspace All robots have a limit of the space that can be reached by the end-effector. This space can also be seen as a set of all reachable coordinate frames within the working range [Cao et al., 2011].

Inverse Kinematics

When the pose of the end-effector is known and the robot configuration is of need, the problem is then to solve the inverse of (2.5), i.e.,

$$q = f^{-1}(x). \tag{2.6}$$

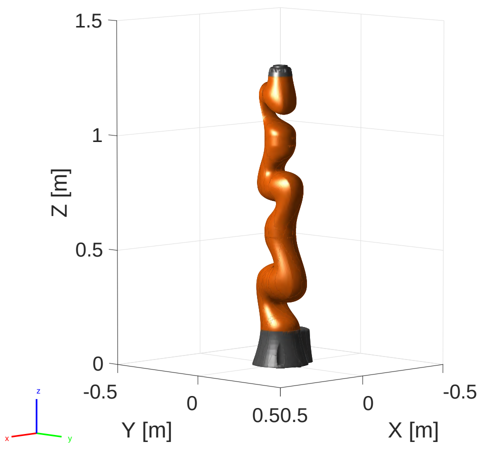
This is, however, often not as straight forward, since the non-linear equations can lead to tedious and time-consuming calculations [Spong et al., 2006]. Since the workspace is limited, no feasible solution would exist if one tries to use (2.6) for poses out of range. There may even exist an infinite amount of solutions, and this is typically a result of reaching a singularity or when using a redundant manipulator with more DoFs than the task requires. This is described in more detail in the following subsection.

Jacobian

To get a sense of how fast the robot is moving both in the configuration- and workspace, the first and second derivatives of (2.5) are needed. The velocity $\dot{x}$ and acceleration $\ddot{x}$ are given by

$$\dot{x} = J(q)\dot{q} \tag{2.7a}$$

$$\ddot{x} = J(q)\ddot{q} + \dot{J}(q)\dot{q} \tag{2.7b}$$



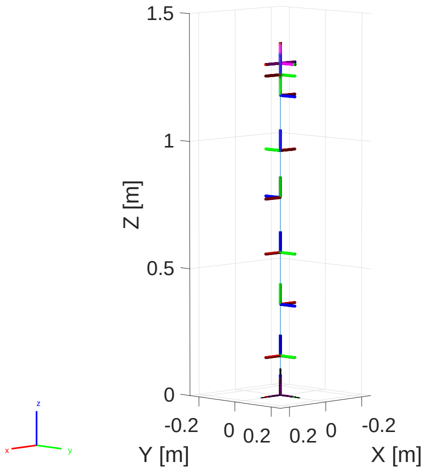


Figure 2.2: Robot arm (top) with its corresponding serial link of coordinate frames (bottom), attached at the respective joint. The base and end-effector frames are located at the bottom and top of the robot, respectively. Both frames are displayed in purple.

and

$$J(q) = \frac{\partial f(q)}{\partial q} \tag{2.8}$$

where $J(q) \in \mathbb{R}^{m \times n}$ is the analytical Jacobian matrix. There is also a way of directly finding the relation between the end-effector twist $\mathcal{V}$ and the joint velocities by using the following formula [Zhang, 2018],

$$\mathcal{V} = \begin{pmatrix} v \\ \omega \end{pmatrix} = J_g(q)\dot{q} \tag{2.9}$$

where $J_g(q)$ is the geometrical Jacobian matrix, v is the linear velocity and ω is the angular velocity. It is important to note that $\mathcal{V} \neq \dot{x}$, because the angular velocity is not necessarily equal to the rotational velocity in $\dot{x}$. Either way, only the analytical Jacobian will be considered from now on, and will be referred to as the Jacobian.

In (2.8) it is apparent that the velocity and acceleration profiles can be expressed either in the Cartesian space or the joint space, depending on what is needed. However, to get the representation in joint space one would need to find the inverse of the Jacobian, which is not necessarily a square matrix. The robot can have an arbitrarily high number of independent joints, while the end-effector is fully described with six dimensions. Therefore, using pseudo inverses instead can allow obtaining the joint velocities [Stewart, 1977], i.e.,

$$\dot{q} = J(q)^{+}\dot{x} \tag{2.10}$$

where $J(q)^{+}$ is the pseudo inverse of $J(q)$. There are several ways of computing the pseudo inverse, for example using singular value decomposition (SVD). By defining the SVD of the Jacobian as $J(q) = U\Sigma V^{*}$, the pseudo inverse is then given by

$$J(q)^{+} = V\Sigma^{+}U^{*} \tag{2.11}$$

where U and V are orthogonal, U^{*} and V^{*} are the conjugates and Σ is a real diagonal matrix consisting of singular values. Furthermore, Σ^{+} is shaped by taking the inverse of all the (non-zero) diagonal elements of Σ.

Singularities There are some limitations to the robot when performing motions and tasks. When two or more joints line up in a co-linear way, it may drive the robot to a state commonly known as a singularity [Spong et al., 2006]. When the robot reaches a singularity, several problems may occur. Some examples are:

- Certain motions may be unattainable.
- The end-effector may be blocked in certain directions.

- There may be an infinite amount of solutions to the inverse kinematics in singularities.
- The robot can result in unpredictable motions and oscillations due to implementation.

Reaching such a state is therefore undesirable and should preferably be avoided.

Furthermore, by inspection of the rank of the Jacobian in a certain configuration, one can identify singularities. If the robot reaches a certain configuration that decreases the rank of the matrix to a lower value than the maximum, a singularity has been reached.

2.3 Cartesian Trajectory Planning

The tasks of a robot are usually planned in advance, consisting of trajectories either in joint space or in terms of the end-effector pose [Freidovich, 2013, p. 10]. A trajectory embeds information that describes the motion with respect to time between an initial and a final pose (or configurations if the planning is done in the joint space). This means that the trajectories consist of a time plan for how to follow a path, subject to desired velocity and acceleration profiles.

There are different pros and cons with trajectory planning in joint or Cartesian space [Castro, 2019]. For example, joint-space trajectories have smooth actuator motion and have no problem with singularities. On the other hand, Cartesian-space trajectories are easier to visualize and the motion in task-space is more predictable, but at the expense of needing inverse kinematics, making it prone to singularities. Besides, actuator motion is more difficult to verify.

Only Cartesian trajectories will be considered in the following details. The trajectory planning can be formulated as follows. Given an initial end-effector pose x_i, the objective is to reach the final pose x_f as fast as possible subject to the velocity- and acceleration-constraints. Furthermore, separating the translational and rotational parts simplifies the calculations [Luca, 2014].

Translational Planning

The path between the initial and final position can generally be defined as any smooth geometric curve $s(t) \in [0,1]$, see Figure 2.3. $s = 0$ means that the trajectory plan is just starting and $s = 1$ means that the final pose is reached. The initial position, final position, maximum Cartesian velocity and maximum Cartesian acceleration are denoted as p_i, p_f, v_{max} and a_{max}. Moreover, the curve $s(t) = \frac{\lambda(t)}{L}$ is

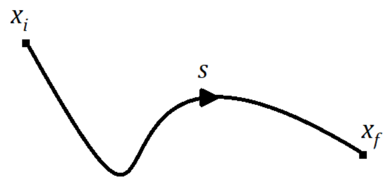


Figure 2.3: Path from the initial pose x_i to the final pose x_f.

a parameterization of the trajectory profile $\lambda(t) \in [0, L]$ with respect to the total arc length L ($L = ||p_f - p_i||$ for linear curves). Depending on the specific task of the robot, some types of paths are more suitable than others, but for the sake of simplicity, only linear paths are dealt with here. The linear interpolation between a start and final point is given by

$$p(t) = p_i + s(t)(p_f - p_i) \tag{2.12}$$

with $p(t) = \begin{pmatrix} x(t) & y(t) & z(t) \end{pmatrix}^T$. Furthermore, it is assumed that the robot is at rest at the start and at the end of the trajectory. By using classical mechanics and taking into consideration said constraints, the translational motion profile becomes [Luca, 2014]

$$\lambda(t) = \begin{cases} \dfrac{a_{max} t^2}{2}, \; t \in [0, t_s] \\[2ex] v_{max} t - \dfrac{v_{max}^2}{2 a_{max}}, \; t \in [t_s, t_f - t_s] \\[2ex] -\dfrac{a_{max}(t - t_f)^2}{2} + v_{max} t_f - \dfrac{v_{max}^2}{a_{max}}, \; t \in [t_f - t_s, t_f] \end{cases} \tag{2.13}$$

where t_s is the time at which the maximum velocity is reached and t_f is the final time. To illustrate further, the velocity profile of a typical trajectory generated from (2.13) is shown in Figure 2.4. If the trajectory is short enough, the maximum velocity may not be reached. In that case the trajectory is then given by the following formula instead [Luca, 2014]

$$\lambda(t) = \begin{cases} \dfrac{a_{max} t^2}{2}, \; t \in [0, t_f/2] \\[2ex] \dfrac{a_{max}(t_f/2)^2}{2} - \dfrac{a_{max}(t - t_f/2)^2}{2} + a_{max}\dfrac{t_f}{2}(t - \dfrac{t_f}{2}) t \in [t_f/2, t_f] \end{cases} \tag{2.14}$$

Rotational Planning

The rotational interpolation between an initial orientation and final orientation can be performed with different techniques. A common method used in robotics is

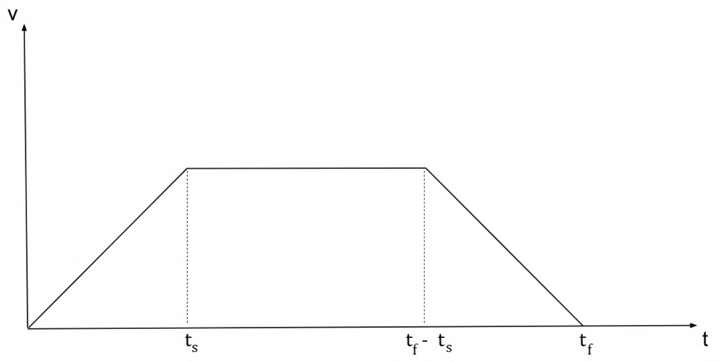


Figure 2.4: The velocity $v = \dot{\lambda}$ as a function of time for a linear translational trajectory.

called spherical linear interpolation (slerp) [Shoemake, 1985]. The technique is based on quaternions and uses a constant angular velocity about a fixed axis. In addition, the interpolation finds the shortest and most direct path for orientations. Given an initial orientation $\mathring{q}_i$ and a final orientation $\mathring{q}_f$[1], the interpolation becomes [Shoemake, 1985, p. 248]

$$\mathring{q}(s) = Slerp(\mathring{q}_i, \mathring{q}_f, s) \tag{2.15}$$

where it is assumed that the function $Slerp$ is a given operator.

The variable s can still be parameterized as $s(t) = \frac{\lambda(t)}{L}$, but λ needs to be modified and L would now represent the arc angle [Luca, 2014]. Since the kinematics describe rotational motion in this context, the constraints are bounded by the angular velocity ω_{max} and angular acceleration α_{max} instead. These parameters will then replace v_{max} and a_{max} in (2.13) and (2.14).

2.4 Dynamics

So far, the underlying theory for the kinematic aspects of the motions has been presented. But in order to be able to apply control strategies, a dynamic model of the robot is often needed. That is, a model which provides a relation between the generalized forces (input) acting on the robot and the motion of the robot (i.e., how the configuration q changes with time). The model is described using equations of motion of a rigid articulated manipulator. There are multiple approaches that lead to

[1] Note that $\mathring{q}_i$ and $\mathring{q}_f$ are not referring to robot joint configurations here, but to quaternions.

different formulations, all of which leads to equivalent equations [Deshpande and Verma, 2010].

The equations of motion for robots are commonly derived from Lagrangian mechanics and this procedure will therefore be explained briefly. Lagrangian mechanics can be used for many different types of systems in different applications when the equations of motion are of need. The Lagrangian L is defined as the difference between the kinetic and potential energies of a system [Freidovich, 2013, Ch. 5], i.e.,

$$L(q,\dot{q}) := T(q,\dot{q}) - V(q) \tag{2.16}$$

where T and V are the kinetic and potential energies, respectively. The next step is to rewrite Newton's second law in terms of the Lagrangian, which yields the following expression (also known as the Euler-Lagrange equations)

$$\frac{d}{dt}\left(\frac{\partial}{\partial \dot{q}_i}L\right) - \frac{\partial}{\partial q_i}L = \tau_i, \; i = 1,\ldots,n \tag{2.17}$$

where τ_i is the generalized force acting on link i. Equation (2.17) can in turn be used to derive the dynamics of a robot with an arbitrarily amount of links n.

Single-linked Manipulator

Consider the example with a single-linked robot, see Figure 2.5. In this case a DC motor with torque input u is coupled to a rigid link through a gear, with angles of the motor shaft and link being θ_m and θ_l, respectively. Thus $\theta_m = r\theta_l$ holds for a rigid-body robot, where r is the gear ratio. The energies are then given by

$$T = \frac{1}{2}J_m\dot{\theta}_m^2 + \frac{1}{2}J_l\dot{\theta}_l^2 = \frac{1}{2}(r^2J_m + J_l)\dot{\theta}_l^2 \tag{2.18a}$$

$$V = Mgl(1 - \cos\theta_l) \tag{2.18b}$$

where J_m and J_l are the rotational inertias of the motor and the link, respectively, M is the mass of the link, g is the gravitational acceleration constant and l is the distance between the joint and the center of mass of the link. Plugging in (2.18a) and (2.18b) into (2.17), the equations of motion eventually become

$$(r^2J_m + J_l)\ddot{\theta}_l + Mgl\sin\theta_l = ru \tag{2.19}$$

where $q_i, \; i = 1$ was replaced with θ_l and τ replaced with ru.

General Case

In practice it is much more convenient to rewrite the dynamic model of any robot with more than two links into matrix form, as the equations would quickly become

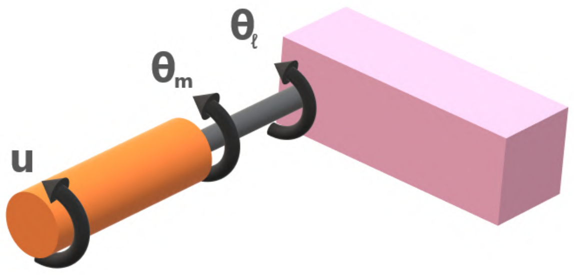


Figure 2.5: A single-linked manipulator, consisting of a DC motor (orange), gear (grey) and a rigid link (pink).

hefty. It can be shown that the link dynamics of a robot with n links can be compactly summarized as [Freidovich, 2013, Ch. 5]

$$M(q)\ddot{q}+C(q,\dot{q})\dot{q}+g(q) = \tau_d+\tau_{ext} \tag{2.20a}$$

$$\tau_{ext} = J^T(q)F_{ext} \tag{2.20b}$$

where $M(q) \in \mathbb{R}^{n\times n}$ is the inertia matrix, $C(q,\dot{q}) \in \mathbb{R}^{n\times n}$ is the Coriolis/Centrifugal matrix, $g(q) \in \mathbb{R}^n$ are the gravity terms and $\tau_d \in \mathbb{R}^n$ are the desired torques. Moreover, $\tau_{ext} \in \mathbb{R}^n$ are the external joint torques which are related to the generalized Cartesian forces $F_{ext} \in \mathbb{R}^m$ performing work on the end-effector. The generalized forces are in turn given by

$$F_{ext} = \begin{pmatrix} f_x & f_y & f_z & \tau_x & \tau_y & \tau_z \end{pmatrix} \tag{2.21}$$

where the first three and last three elements correspond to the forces and torques being felt at each of the spatial dimensions, respectively. With this laid out, the next step is to describe how the dynamics can be exploited to shape a Cartesian impedance controller.

2.5 Impedance Control

The concept of integrating an impedance controller (i.e., a virtual mass-spring-damper) on a manipulator was first proposed during the 80's [Hogan, 1984]. The idea is quite simple; to treat the robot as a mechanical impedance with its environment viewed as an admittance. In other words, the controller is essentially a framework that sets a dynamical relationship between the external forces and the motion of the robot.

Ultimately, the desired behaviour can be achieved by viewing external contact forces as disturbances. If (2.20a) is formulated in Cartesian space, the following is obtained [Ott, 2008, pp. 31-33]

$$M_x(x)\ddot{x} + C_x(x,\dot{x})\dot{x} + g_x(x) = F_d + F_{ext} \tag{2.22}$$

where $M_x(x) \in \mathbb{R}^{n\times n}$, $C_x(x,\dot{x}) \in \mathbb{R}^{n\times n}$ and $g_x(x) \in \mathbb{R}^n$ are the Cartesian inertia, Coriolis/centrifugal, gravity matrices, respectively, and F_d are the desired controlled forces. These matrices and vectors are in turn given by

$$\begin{aligned}
M_x(x) &= J^{-T}(q)M(q)J^{-1}(q)\\
C_x(x,\dot{x}) &= J^{-T}(q)(C(q,\dot{q}) - M(q)J^{-1}(q)\dot{J}(q))J^{-1}(q)\\
g_x(x) &= J^{-T}(q)g(q)\\
F_d &= J^{-T}(q)\tau_d.
\end{aligned} \tag{2.23}$$

Notice that the objective of the controlled forces F_d in (2.22) is similar to the controlled torques τ_d in (2.20a), but with respect to the Cartesian space instead. The state variables are now x and $\dot{x}$, hence the name Cartesian impedance control. Practically, the implementation would still employ the joint variables in (2.1a) and (2.1b) as they are directly measured and accessible. Recap that measurements are not directly obtained from the end-effector for most industrial robots, including the one used in this project. The external forces F_{ext} are not directly measured either (but could be estimated from sensed joint torques) and τ_d are the control inputs to the actuators. For these reasons, it is therefore easier to mix both dimensions in the actual implementation.

2.6 Control Shaping

The variables of a manipulator can be separated into two groups; redundant and non-redundant ones [Ott, 2008, Ch. 3-4]. If redundant variables are present, the robot would have more DoFs than needed ($n \geq 7$) to allow arbitrary pose of the end-effector. A result of this is the possibility to keep the tool center point (TCP) fixed while freely moving the robot along the redundant dimensions. This feature, which is also known as nullspace motion, is useful in several scenarios. For instance, the joints need to be in a specific configuration but maintain the same TCP in order to avoid collision. Another usage is reproducibility of motions, i.e., to achieve sufficient repetitional accuracy of the same task.

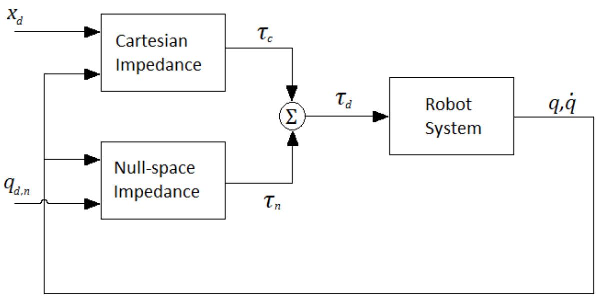


Figure 2.6: Block diagram of the control system.

In order to combine the Cartesian impedance and nullspace motion in a simple way the superposition principle is used. This results in the desired torque being separated as

$$\tau_d = \tau_c + \tau_n \tag{2.24}$$

where τ_c is responsible for the Cartesian impedance and τ_n is responsible for the nullspace impedance. To further illustrate the overall control system, a block-diagram is shown in Figure 2.6.

Cartesian Impedance

In order to make sense of a mechanical impedance in terms of control design, the external forces F_{ext} in the Cartesian space are defined as [Ott, 2008, pp. 30-31]

$$F_{ext} = M_d\ddot{x}_e + D_d\dot{x}_e + K_d x_e \tag{2.25a}$$

$$x_e = x - x_d \tag{2.25b}$$

where M_d, D_d and K_d are the desired inertia, damping and stiffness matrices, respectively. The deviation in position from the desired motion, x_d, is given by x_e[2]. Notice, however, that the error in orientation is not simply calculated by taking the linear "difference" between two orientations. Rather, Equation (2.25b) only shows a compact way of representing the deviation between two end-effector poses. To get the actual rotational error, one could for example use quaternions. Given two quaternions $\mathring{q}_i$ and $\mathring{q}_f$, the difference $\mathring{q}_e$ is given by [Baker, 2021]

$$\mathring{q}_e = \mathring{q}_f \mathring{q}_i^* \tag{2.26}$$

where $\mathring{q}_i^*$ is the conjugate of $\mathring{q}_i$.

[2] The deviation and desired motion can change with time, i.e., $x_e = x_e(t)$ and $x_d = x_d(t)$.

By inserting (2.25a) into (2.22) and rewriting the expression, the control law eventually becomes [Ott, 2008, p. 34]

$$\begin{aligned}\tau_c &= J^T(q)F_d \\ &= g(q) + J^T(q)(M_x(x)\ddot{x}_d + C_x(x,\dot{x})) - \\ &J^T(q)M_x(x)M_d^{-1}(K_d x_e + D_d \dot{x}_e) + \\ &J^T(q)(M_x(x)M_d^{-1} - I)F_{ext}\end{aligned} \tag{2.27}$$

where I is the identity matrix. Since the external forces F_{ext} can not be directly measured, the variable needs not be included in the control law. To deal with this issue, a simplification can be made without affecting performance significantly. That is, by setting the desired inertia equal to the robot inertia, i.e., $M_d = M_x$. The avoidance of inertia shaping also allows focusing on designing the desired stiffness and damping matrices instead. By taking into consideration said simplification, (2.27) is reduced to

$$\tau_c = g(q) + J^T(q)(M_x(x)\ddot{x}_d + C_x(x,\dot{x}) - K_d x_e - D_d \dot{x}_e). \tag{2.28}$$

Hence, the external forces are no longer a part of the control law.

Stiffness Designing an appropriate stiffness matrix depends mainly on the application. Consider a first scenario, where it is expected that there will be contact with the environment along a certain direction. Then the stiffness in the same direction can be set to a smaller value to reduce the impact. On the other hand, if a good trajectory tracking is desired, and no contact is expected along said direction, then the stiffness should be set to something larger. The choice of stiffness is thus a trade-off between contact force and accuracy.

The structure of the stiffness matrix is given by

$$K_d = \begin{pmatrix} K_t & K_c \\ K_c & K_r \end{pmatrix} \tag{2.29}$$

where K_t, K_r and K_c are the translational, rotational and coupling stiffness submatrices, respectively.

Damping Damping coefficients are designed to obtain desired transient behaviour. The damping matrix can be chosen with respect to the generalized eigenvalues of K_d [Ott, 2008, pp. 36-37],

$$D_d = 2Q^T D_\xi \sqrt{\lambda_K} Q \tag{2.30}$$

where $Q \in \mathbb{R}^{n\times n}$ is a non-singular matrix, $D_\xi \in \mathbb{R}^{n\times n}$ is a diagonal matrix where the i:th element ξ_i is a damping factor $\in [0,1]$ and $\lambda_K \in \mathbb{R}^{n\times n}$ is a diagonal matrix consisting of the generalized eigenvalues of K_d. If it is desirable to have a system that responds quickly but with minimal overshoot, the elements in D_ξ should be closer to the upper limit. On the other hand, if the damping factors are chosen to be on the lower end, the robot would dissipate energy at a lesser rate, allowing it to be more forgiving during contact with the environment.

Nullspace Impedance

There are several ways of modelling nullspace impedance to get the desired control law. One way is through the so called nullspace projection approach [Ott, 2008, pp. 48-51]. Firstly, one can define the joint space impedance τ_0 as

$$\tau_0 = -D_n\dot{q} - K_n(q - q_{d,n}) \tag{2.31}$$

where $D_n \in \mathbb{R}^{n\times n}$ and $K_n \in \mathbb{R}^{n\times n}$ are the desired damping and stiffness matrices in terms of nullspace motion. These matrices are designed in a similar way as the methods used for the Cartesian impedance. Moreover, $q_{d,n}$ is the desired nullspace configuration which satisfies $f(q_{d,n}) = x_d$. The desired nullspace controller is then given by

$$\tau_n = P(q)\tau_0 \tag{2.32}$$

where $P(q)$ is a projection matrix constructed such that the Cartesian impedance is dynamically decoupled from the joint-space impedance. This can be achieved by, e.g., projection in the operational space [Ott, 2008; Khatib, 1987]

$$P(q) = I - J^T(q)(J(q)M^{-1}(q)J^T(q))^{-1}J(q)M^{-1}(q). \tag{2.33}$$

The control law for the nullspace impedance can then be obtained by inserting (2.33) into (2.32).

3

Robot Setup and Software Interfaces

This chapter provides a brief overview of the robot setup and hardware as well as software interfaces used throughout the work. In Section 3.1, a description of the robot setup used for this work is layed out. In Section 3.2, a general overview of the open source program Robotic Operating System (ROS) is given, whereas in Section 3.3, a short introduction to the Dynamic Animation and Robotics Toolkit (`DART`) is given.

3.1 The Robot Setup

As mentioned earlier, the robot used in this project is a 7-DoF manipulator. The manipulator is called *LBR iiwa* and is manufactured by *KUKA AG* [KUKA, 2021], see Figure 1.1. It is a light-weight robot, especially designed for human–robot–collaboration tasks. With a weight of around 20 kg and a payload capacity of around 7 kg, the robot is suitable for human–robot–collaboration and complex assembly tasks. For insight of how the work-setup for this project looked like, see Appendix A.

The robot consists of actuators that manipulate the joint positions with specified input torques through the equations and relationships, as described in Chapter 2. These input torques are in turn obtained or manipulated, by reading or commanding torques to the motors that are mounted by the joints. There are two internal control modes for this robot, in which one of them needs to be used during operation. Namely, these are the position and the torque modes. Naturally, the torque-mode can be used here, as the framework is essentially implementing a force-controlled impedance.

Another feature that KUKA provides is a Fast Robot Interface (FRI) [KUKA, 2021], which allows implementing external controllers with up to 1000 Hz execution frequency. However, using this frequency range is only possible for implementations that can run as fast, or faster.

3.2 Robot Operating System

Robot Operating System, commonly referred to as ROS, is a collection of tools and libraries that aim at configuring the software of the robot such that the complexity of achieving desired behaviour is reduced [Dattalo, 2018]. The platform is not actually an operating system, but includes diverse features similar to one, such as hardware abstraction, low-level device control and communication between processes. Because of its powerful features, ROS is used by many in the robotics community, and has steadily been growing ever since it was released. The software can also be seen as the interface that allows communication and data flow between different applications, or more commonly known as the *middleware* [IBM Cloud Education, 2021].

ROS Control

One of the core packages for managing controllers is called `ros_control` [Chitta et al., 2017]. This extension can be seen as a safe layer of communication, which focuses both on real-time performance, but also on managing controllers in a robot-agnostic way. The way data flow from the controller, all the way to the actuators, is visualized in Figure 3.1. For instance, the `Controller Manager` is responsible for managing available controllers, whereas `hardware_interface::RobotHW` acts as an interface between the hardware and ROS. Furthermore, using the `ros_control` package allows to apply controllers on the real robot and in simulation, without needing to do any modifications in the implementation.

Hardware Interfaces and Inheritance In the package, there are multiple hardware interfaces that are responsible for sending or receiving commands to the hardware to choose from. The commands could, for instance, be effort-based (torque), position-based or velocity-based. Needless to say, the effort-joint-interface is of particular interest for this project, since the Cartesian impedance controller in this project uses torque inputs.

The interface requires an inheritance of four distinct methods, namely `init`, `starting`, `update` and `stopping` [Meeussen, 2016]. In the `init` method, the controller is loaded and initialized in non-realtime. Furthermore, the robot joints, sensors and actuators interfaces are also initialized here. The next function, `starting`, unlike the first one, is executed in real-time, and is called once every time a controller is started up. This is where the initial parameters of the robot are retrieved and

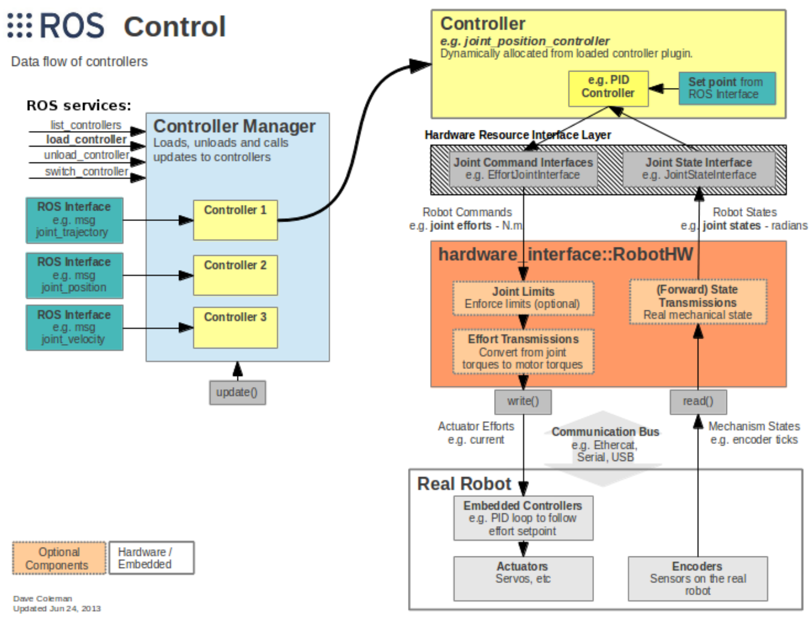

ROS Control
Data flow of controllers
ROS services:
list_controllers
load_controller
unload_controller
switch_controller
Controller Manager
Loads, unloads and calls updates to controllers
ROS Interface e.g. msg joint_trajectory
ROS Interface e.g. msg joint_position
ROS Interface e.g. msg joint_velocity
Controller 1
Controller 2
Controller 3
update()
Controller
e.g. joint_position_controller
Dynamically allocated from loaded controller plugin.
e.g. PID Controller
Set point from ROS Interface
Hardware Resource Interface Layer
Joint Command Interfaces e.g. EffortJointInterface
Joint State Interface e.g. JointStateInterface
Robot Commands e.g. joint efforts - N.m
Robot States e.g. joint states - radians
hardware_interface::RobotHW
Joint Limits Enforce limits (optional)
Effort Transmissions Convert from joint torques to motor torques
(Forward) State Transmissions Real mechanical state
write()
read()
Actuator Efforts e.g. current
Communication Bus e.g. Ethercat, Serial, USB
Mechanism States e.g. encoder ticks
Real Robot
Embedded Controllers e.g. PID loop to follow effort setpoint
Actuators Servos, etc
Encoders Sensors on the real robot
Optional Components
Hardware / Embedded
Dave Coleman
Updated Jun 24, 2013

Figure 3.1: Overview of the package `ros_control` [Chitta et al., 2017]. The figure is bounded by the *Attribution 4.0 International* license.

used to assign values such as initial desired pose, stiffness and more. It also allows the controller to be stopped and started without having to reinitialize the hardware. The function can also be seen as a middle-step between hardware-interface startup and the update-loop, `update`. In this latter function, the calculations that update the controller at every sampling instant are performed. The last method, `stopping`, is called when the controller is stopped.

Components of a ROS Program

A typical ROS system is easiest visualized by a graph, consisting mainly of *nodes*, *topics* and *messages* [Dattalo, 2018]. A *node* is in other words a running program or executable file, in which computations are performed. The developer can build multiple nodes and let them communicate with each other by sending information, i.e., messages, over some links named *topics*.

Each node can publish messages over multiple topics and also receive messages from multiple topics at the same time. A master node keeps track of the publications and subscriptions on all topics. Thus, developers need not worry about low-

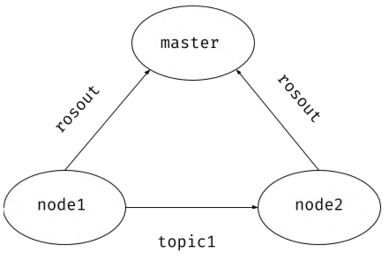


Figure 3.2: A simple ROS network consisting of three nodes, including the master node. In this case, `node1` is publishing on `topic1`, and `node2` is listening to this topic. Both of these nodes are also communicating with the master through the topic `rosout`.

level data flow, even if the system would consist of tens or hundreds of processes communicating with each other. A simple ROS network can be observed in Figure 3.2. The described software architecture is also known as the *publish-subscribe pattern* [Buck et al., 2018].

Simulations

Before applying any implementation on a real system, it is wise to simulate some virtual representative model beforehand [Hosseinpour and Hajihosseini, 2009]. Clearly, robotics is not excluded, as simulations not only help visualizing the dynamics of the robot, but also provide a safe way of testing new experimental methods and applications without risking damaging expensive equipment.

In ROS, the simulation is usually performed with the help of a virtual robot model described with, e.g., the Universal Robot Description Format (`URDF`), which is shaped using extensible markup language (XML) [Sucan and Kay, 2019]. `URDF` includes a physical description of an object, i.e., geometrical shape, joints, links, friction and more. A simplified example of how an `URDF`-file could look like is shown in Figure 3.3. There are also other types of formats that could be used for describing robots, but they are deliberately excluded here. The `URDF` format was not chosen for any particular reason here, but has historically been used within ROS, even if there might be better suited formats available today.

Graphical Visualization

To visualize the simulations in ROS, a graphical user interface (GUI) can be used. There are several GUIs that are integrated with ROS, each serving a different purpose. The ones mainly used for this project are called `Gazebo`, `rviz` and `rqt`.

Gazebo `Gazebo` is a 3D-simulator, which aims to display how the robot would behave in a real-world scenario, as accurately as possible [Gazebo, 2014]. The sim-

```
<?xml version="1.0"?>
<robot name="simple_example">
  <link name="link1">
    <visual>
      <geometry>
        <cylinder length="1" radius="0.2"/>
      </geometry>
    </visual>
  </link>
</robot>
```

Figure 3.3: A simple XML-block that defines a cylindrically shaped link with the length of 1 units and radius of 0.2 units.

ulator supports several physics engines, but uses a specific one by default. Depending on how accurate the model is described compared to the real system, one can achieve desired behaviour both with the real robot and in simulation. When the `URDF`-model of the iiwa from KUKA is launched in `Gazebo`, the environment in Figure B.1, Appendix B, is shown.

rviz Unlike `Gazebo`, `rviz` aims at aiding the developer with visualization tools in order to efficiently debug robot applications [Hershberger et al., 2018]. It can give insight to what information the robot is processing, by providing, e.g., the state of the robot, sensor values and coordinate frames. This GUI is therefore a suitable tool for visually debugging robot applications in ROS. In Figure B.2, Appendix B, the `URDF`-model of the robot in the `rviz` environment is shown.

rqt The `rqt`-package is a framework that utilizes tools and interfaces in the form of plugins [Thomas et al., 2016]. Over time there have emerged many useful plugins from `rqt`. For example, one could use `rqt_graph` to visualize a ROS-system consisting of nodes, topics and messages, similar to the system in Figure 3.2. There are also other useful topics that were used for this work. For instance, `rqt_reconfigure`, which allows dynamically configuring parameters in run-time, which is convenient for tweaking parameters in a debugging scenario.

Tweaking Parameters During Operation

As mentioned in the previous paragraphs, one could interact with the robot in real-time using `rqt` plugins. However, in practice, some parameters usually need to be varied with respect to the robot state, but also to the specific application, without needing to manually tweak parameters. In essence, this means that the variables need to have the ability to change in run-time, automatically. Firstly, the parameters that should become configurable are the stiffnesses and damping factors along each

Cartesian dimension. This is necessary in order to achieve certain tasks. Another variable that is also desired to change during run-time is the desired pose of the end-effector. In fact, it is needed for commanding new Cartesian locations of the end-effector within the workspace. Lastly, applying Cartesian wrenches at the end-effector is also something that should be a possibility. For instance, this might be useful in scenarios where the payload is heavy. One could then counter the weight by "pushing" in the opposite direction with a force with equal magnitude.

To achieve this, one could for instance setup ROS-subscribers that listen for incoming information over some topics, which in turn manipulate the robot/controller accordingly. With this approach, information could readily be published (commanded) from outside of the ROS-system through, e.g., Python or Bash scripts. This is explained further in Section 4.3.

3.3 Dynamic Animation and Robotics Toolkit

The Dynamic Animation and Robotics Toolkit (`DART`) is an open source library intended for robotic applications and simulations [Lee et al., 2018]. Fundamentally, the toolkit is a physics engine, making it possible to animate kinematic and dynamic applications in robotics.

Just like the simulators in ROS, `DART` allows simulating and visualizing robots that are described with `URDF`. Furthermore, the `DART`-engine can also be used in other robotic simulators, including `Gazebo`. However, `Gazebo` uses the Open Dynamics Engine by default [Gazebo, 2014].

RobotDART

`RobotDART` is a wrapper around the `DART` physics engine [Chatzilygeroudis and Mouret, 2021]. In Figure B.3, Appendix B, the `URDF`-model of the robot in the `RobotDART` environment is shown. Furthermore, this extension is mainly developed for robotics and machine-learning researchers, making it suitable for this project.

One could then ask why this simulator should be used instead of `Gazebo`, that would have the possibility to use the same physics engine. On the other hand, `RobotDART`'s implementation is said to have low overhead and delay, unlike `Gazebo`. An implication of this is that parallelization is easier to achieve using `RobotDART`. In fact, this is one of the main limitations when using `Gazebo`, as parallelization is difficult to achieve with the simulator.

4 Approach and Methodology

In the coming sections, the approach and methodology of the work are presented. First, a general overview of the work is presented in Section 4.1. In Sections 4.2–4.5, a deeper explanation of the methodology is given. Lastly, in Section 4.6 the licenses of the software tools used are mentioned.

4.1 General Overview

In order to setup a framework that is convenient both for the ROS and `DART` environments, a library with the base algorithms was developed separately. That is, an implementation of the Cartesian trajectory planner and Cartesian impedance controller described in Sections 2.3 and 2.6, respectively. While the base libraries take care of the underlying calculations, the ROS and `DART` frameworks implement the desired features presented in Section 1.1. That is, the ability to configure parameters as well as exerting Cartesian forces during run-time. The overall structure of the framework can also be visualized as in Figure 4.1.

The main framework of the base algorithms is separated into two libraries, mainly the *Cartesian Trajectory Generator* and the *Cartesian Impedance Controller*. However, since the latter library had a higher priority, a limited time was spent on the trajectory generator. At a later occasion during the project, it was further developed by one of the supervisors to be more generic and user-friendly[1]. In fact, it also included the possibility to generate overlay motions that replicate a circular search motion. Nevertheless, its features was mainly used to perform simple trajectories and motions, replicating applications such as a peg insertion.

[1] I am thankful that one of my advisors, Matthias Mayr, contributed with this.

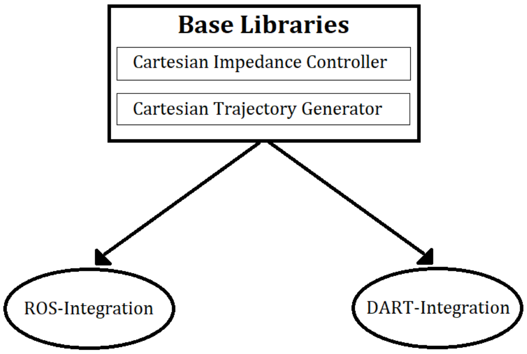

Base Libraries
Cartesian Impedance Controller
Cartesian Trajectory Generator
ROS-Integration
DART-Integration

Figure 4.1: The overall structure of the framework.

Table 4.1: Example parameters that are given to the trajectory generator.

Initial Position (m)	x=0, y=0, z=0
Initial Orientation (rad)	roll=0, pitch=0, yaw=0
Final Position (m)	x=1, y=1, z=1
Final Orientation (rad)	roll=$\pi/2$, $\text{pitch} = 0, \text{yaw} = \pi$
$v_{\max}$ (m/s)	0.3
$a_{\max}$ (m/s^2)	0.05
$\omega_{\max}$ (rad/s)	0.3
$\alpha_{\max}$ (rad/s^2)	0.05
Synced?	yes

Cartesian Trajectory Generation

The algorithm of the Cartesian trajectory planner implements the formulas presented in Section 2.3, that is, a generator that can take translational and rotational reference changes independently from each other. The algorithm also allows for planning synchronized trajectories, i.e., making the translational and rotational plans to start and finish at the same time.

Consider the case where a new goal-pose is requested some distance away from the initial pose, subject to velocity and acceleration constraints. More specifically, assuming that the information in Table 4.1 is provided to the algorithm, the calculations would result in the trajectory plan shown in Figures 4.2–4.4. For better visualization of the orientation, the Euler representation is used here. The Euler angles are in turn described with the XYZ-representation, also known as roll, pitch and yaw.

As seen in Figure 4.2, the initial acceleration jumps to its maximum value, resulting in second-order increase in terms of position. When the maximum velocity is reached, the acceleration becomes zero, hence the linear progression in position. When the distance to the goal pose decreases to a certain amount, the retardation phase is started and has (in this case) an identical behaviour to the initial acceleration phase. Taking the derivative of the generated position trajectory would result in a velocity profile similar to the one shown in Figure 2.4.

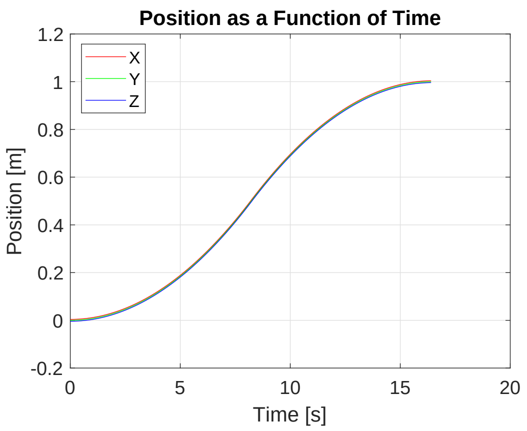


Figure 4.2: Translational plan generated from the parameters in Table 4.1. As the initial and final poses are the same for each axis, the individual plans overlap each other.

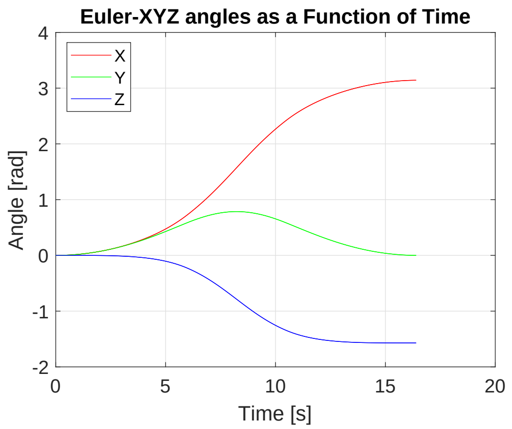


Figure 4.3: Rotational plan generated from the parameters in Table 4.1.

In the same manner as mentioned in Section 2.3, a rotation can be performed with

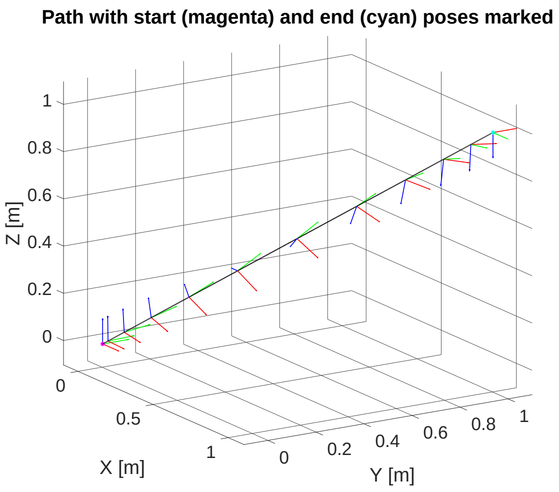


Figure 4.4: Trajectory plan in 3D-space, generated from the parameters in Table 4.1. To get an idea of how the plan is progressing, selected intermediate pose frames are scattered along the trajectory. Red, green and blue directions of the frames represent the *x*, *y* and *z*-axis, respectively.

similar behaviour given angular velocity and acceleration profiles. This can be visualized by studying the rotational change in Figure 4.3. Notice, however, that the rotation along the *y*-axis starts to deviate from the desired angle progressively, before eventually reaching back to the desired angle along said direction. As a matter of fact, this is a result from using quaternions along with (2.15) in the actual algorithm. The rotational representations shown in Figure 4.3 are hence converted from quaternions. This is because, as discussed before, the Euler angles give an intuitive way of visualizing rotations, while quaternions are more suited for calculations and numerical stability.

Limitations The trajectory generator inherits some limitations that are important to consider. Firstly, the algorithm has no knowledge of the reachable points of the end-effector of a robot. If one commands an unreachable point, the robot may stretch out and end up in a singularity. Another limitation is that the trajectory generator does not get any feedback from either the environment or an existing non-zero velocity of the end-effector. This may, consequently, lead to collisions or undesired motions if one is not careful enough.

4.2 Control Implementation

For the Cartesian impedance controller, an example implementation is already given in [Emika, 2017], which is based upon the package `ros_control`[2]. However, the base algorithm was integrated within the ROS environment and needed to be separated. By moving the base algorithms from the ROS environment into a library that only requires `C++` and a template library for linear algebra[3], it became possible to integrate it into other frameworks, e.g., the `DART` environment.

The control law in (2.28) was modified in the implementation to the following equation instead,

$$\tau_c = J^T(q)(-K_d x_e - D_d J(q)\dot{q}). \tag{4.1}$$

where $\dot{x}_e$ was rightfully replaced with $J(q)\dot{q}$ using (2.7a). This simplification may seem absurd at first, but can be justified assuming a number of things. Firstly, the gravitational term is already accounted for internally in some robots as it is in the KUKA iiwa. Secondly, the Coriolis and inertia terms can be neglected for this work, because the targeted area for the controller of this thesis is in kinesthetic teaching and slower and contact-rich motions. This also allows focusing on the shaping of good stiffness and damping matrices instead.

Another simplification can be done by letting the stiffness matrix have a specific shape. The parameters K_t and K_r are often chosen as diagonal matrices [Ott, 2008, pp. 40-41]. Furthermore, it was assumed that the coupling stiffness was insignificant, i.e., $K_c = 0$. The resulting matrix was then a diagonal one, where each element represents the stiffness along a specific translational or rotational axis. Hence, this allowed the stiffness along each direction to be specified without interfering with the other directions. Consequently, it also followed that the damping matrix became a diagonal, with each element describing the damping behaviour in a certain dimension.

Just like for the Cartesian impedance control law, the projection matrix in the nullspace term in (2.32) can be simplified. That is, by setting the inertia matrix equal to identity in (2.33), the following relation is obtained

$$\tau_n = (I - J^T(q)J^{-T}(q))(-D_n\dot{q} - K_n(q - q_{d,n})) \tag{4.2}$$

Although this substitution may have some consequences in terms of dynamic behaviour, one can, once again, focus on shaping good nullspace stiffness and damp-

[2] A new implementation based on only ROS and RBDyn was created, where the latter is a dependency-package that provides a set of classes and functions which allows modelling the dynamics of rigid body systems. Further information about the latter library is found through the following link: `https://github.com/jrl-umi3218/RBDyn`

[3] The specific library here is called `Eigen`, and further information about the library can be found in the given link: `https://eigen.tuxfamily.org/index.php?title=Main_Page`

ing values instead. Needless to say, these simplifications result in fewer computations, and therefore, a faster controller on the real system.

As mentioned in Chapter 3, the robot can respond at a rate of up to 1000 Hz. However, even with said simplifications taken into account, it turned out that the controller needed around 1 millisecond per loop for the computations[4], see Appendix C. Consequently, for safety reasons, the FRI was setup to run at 500 Hz.

Main Functionalities

The base framework was shaped as follows. The main function, when provided with the state of the robot[5], returns the desired torques. These torques are then commanded, either to the real robot or to the virtual model in simulation.

The specific parameters that were desired to become re-configurable are mentioned in Section 3.2. This was achieved by implementing separate functions that receives said parameters and update the control law accordingly.

Applying Cartesian Wrenches

Furthermore, applying a Cartesian wrench F_{ext} during run-time was also one of the desired features of the framework. To facilitate this, they were added through (2.20b), which would then be added to the overall control law given by (2.24). Nevertheless, applying forces in this fashion would technically mean that they are not external. Rather, they only demonstrate how the robot would react, should *similar* external forces be applied at the end-effector.

Safety Features

An important aspect to consider when working with robots, or any other solid and heavy system for that matter, is that sudden reference changes may lead to rapid maneuvers, which could result in dangerous situations as well as oscillations in the robot structure. More specifically, consider the case where the stiffness is initially zero, and is located some distance away from the desired pose. A sudden increase to some high stiffness would result in movement that could be faster than the average response time of a human. Similarly, by applying a sudden virtual wrench while the robot has little or no stiffness, the situation could once again become dangerous.

[4] That is, for the computational machine used in this project. It is a high-performance machine, dedicated just for the robot arm. It was configured to prioritize controller computations and used a real-time kernel. Further information about the processor can be found through the following link: `https://linux-hardware.org/index.php?id=pci:8086-1901-1462-127d`

[5] For easier annotation, this term is used to combinedly refer to the configuration, its derivative, the end-effector pose and the Jacobian.

A way of dealing with the issue described in the previous paragraph is by applying some type of filtering that "smoothens" the transitions between reference changes. Hence, a first order low-pass filter was used both for stiffness-transitions but also for applying Cartesian wrenches. Sudden forces exerted on the robot could also lead to volatile transients in the output of the control law. To delimit the effects of this, a torque-rate limiter was also embedded and used before sending the commands to the robot.

Another important consideration is to saturate the stiffness, damping factors and Cartesian wrenches to reasonable limits. For instance, if a user accidentally requested a negative stiffness or damping value, the controller could face potential problems and become unstable. Moreover, applying too big of a wrench could drive the robot to instability. For these reasons, saturation of said parameters was also embedded in the framework.

Usage of the Base Implementations

To summarize how the base libraries would be utilized in either of the ROS or `DART` frameworks, consider Figure 4.5. In the figure, the `User Interface` block contains a collection of the configurable parameters, which would in turn be commanded either through ROS topics or the `rqt_reconfigure` plugin. Then, after passing through some safety functions, the desired parameters are sent to the `Controller` block. In addition, the state of the robot is also sent repeatedly to the `Controller` block from the `Robot` block, allowing the desired torques to be generated and updated. The `Robot` block is in turn either representing the real robot, or the model.

4.3 Working Environment

Most of the functionality was implemented using the programming language `C++`. This includes both the ROS and `DART` integrations. Aside from the actual implementations, the work was conducted through a Linux environment. For example, to launch a ROS system, one would either need to run every ROS node, including the master node in a new terminal, or join the relevant nodes in a so called launch-file, which, in turn, runs the collected processes subsequently.

When nodes are started up and running, information such as topics, subscribers, publishers and parameters becomes accessible within the ROS framework via terminals. Consequently, one could send or receive messages on dedicated topics and allow configuration of parameters during run-time. As a matter of fact, this feature allowed automating tasks with the help of Python and Bash scripts. This in turn made it possible to run identical experiments both in simulation and on the robot.

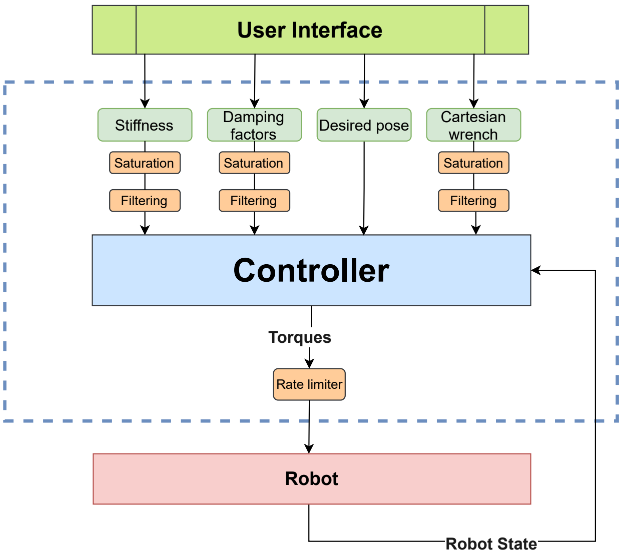


Figure 4.5: Flowchart of the overall control strategy. The (blue) dotted rectangle shows what is included in the base framework.

4.4 ROS Framework

The framework in ROS was established using the `ros_control` package described in Section 3.2. There are several other dependencies of the framework that make it possible to utilize the FRI together with `ros_control`, in order to connect to the robot. Moreover, a `URDF` file, which describes the LBR iiwa robot and its basic setup, was provided by Virga et al. (2019). This model was in turn used along with `Gazebo` and `rviz` to graphically visualize trajectories, control output, robot behaviour and other experiments.

Updating Spatial Frames

As discussed in Sections 2.1 and 2.2, the serial chain of the robot frames would change during tasks. Fortunately, ROS comes with a package called `tf` [Foote et al., 2017], that lets the user keep track of the frames as time progresses. The only input that needed to be provided when using this package was the joint configuration along with the `URDF`-models. The remaining transformations and calculations would hence lead to available and up-to-date spatial frames, including the end-effector.

Configuring Parameters

As stated earlier, there are several parameters that need to be configurable during operation. As shown in Figure 4.5, the way the controller receives inputs is through the `User Interface` block. The details of what this blocks represents will be discussed in the coming subsections.

Parameter Server In Section 4.3, it was stated that information about the nodes would become available when nodes are started and running, Whilst some information is available through topics, other might be available through a shared dictionary, namely a *Parameter Server* [Miller, 2018]. This is where one could specify parameters that are also retrievable during operation. In many cases the dictionary can become quite big, and is usually specified in so called `YAML`-files, which in short is a collection of parameters that is associated with some node(s). As a matter of fact, the *Cartesian Trajectory Generator* uses a `YAML`-file with specifications such as maximum velocities, accelerations, publishing rate and more.

Dedicated Topics In Figure 4.5, the controller receives inputs from the `User Interface` block, which is mainly through topics, where each output from said block is paired with a topic.

rqt Plugin Another way of specifying the parameters is through the `rqt_reconfigure` plugin. As discussed earlier, this is primarily convenient in debugging scenarios, but also for doing more supervised experiments and tasks. For each of the outputs from the `User Interface` block in Figure 4.5, new nodes were created. Each of the nodes were in turn paired with a so called dynamic server, which made it possible to use the plugin to configure the desired parameters. The resulting GUI is shown in Appendix D.

Applying Cartesian Wrenches at the End-Effector Frame When applying forces through (2.20b), it is also important to consider in what coordinate frame the wrench is being exerted at. The general forces, F_{ext}, will by default be applied at the end-effector frame (translational) , but with respect to the orientation of the base frame. Consider the case where the z-axis of the end-effector is pointing in the opposite direction relative to the base frame. Then, if a positive force is applied, the end-effector frame would be subjected to a force in the same direction as the z-axis of the base frame and hence move along said direction. However, in many applications it is often desired to apply a force with respect to the end-effector frame.

To solve the issue described in the previous paragraph, the generalized forces would need to be rotated such that the exertion is applied at the end-effector. Firstly, (2.21) was split into two 3D-vectors, each corresponding to the forces and torques along the spatial axes, respectively. Consequently, the vectors were transformed into the desired frame using the previously mentioned package `tf`, which also inherits the ability to transform vectors from one frame into another.

Measuring Cartesian Wrenches

A way of verifying that the applied forces are exerted correctly is by using a force-torque (FT) -sensor, mounted at the end-effector. A physical FT-sensor is, however, not available on the LBR iiwa robot. Instead, the robot embeds an estimator of the external joint torques, which would then be transformed to the spatial exertion at the end-effector through (2.20b).

Simulation Tools in ROS

The introduced simulation tools in ROS were not explicitly used for evaluation and comparison with the performance of the real robot. Rather, they were mainly used as a way of making sure that the general behaviour was expected and non-chaotic. `DART` was the target simulation platform for reasons stated in Section 3.2.

4.5 DART Framework

Sequentially, the `DART` integration was separated into two smaller libraries. The first one was dedicated for integrating the base functionalities into the `RobotDART` environment. More specifically, an interface class in `RobotDART` was available by Chatzilygeroudis and Mouret (2021) and thus inherited. This library is a wrapper that already implements functionalities often needed in robotics. In essence, it would act as an intermediate library that passes through the available functions in the base library, into another class that inherits the interface.

The second library implements a class that was shaped as a ROS node using similar structure as the `ros_control` package. More specifically, two functions in this library were developed, namely `init` and `update`[6]. These functions would then use the integrated functionalities in the first library to run the controller in the 3D-simulator. This not only allowed to exploit the features and tools of ROS, but also made it possible to use topics to configure parameters during run-time.

Aside from this, the features in this framework were, to a large extent, implemented with identical features to the ones in Section 4.4. However, a few features were excluded in the `DART` framework due to the time-restriction of the project. These are dynamic reconfiguration of parameters using `rqt_reconfigure` and an FT-estimator.

4.6 Licenses

All of the tools and packages that were used for the implementation are open source software. Essentially, this means that the packages are free to use, distribute and modify, as long as the licenses are mentioned and honoured. As far as ROS is of

[6] Using a `starting` function like for the `ros_control` package is not needed here.

concern, the standard BSD license covers the majority of the system. The BSD license allows for reuse both in commercial and closed-source products[7]. `DART` is also licensed under the BSD contract, whereas `RobotDART` is bounded by the CeCILL license[8].

[7] Further information about the BSD license can be found through the following link: `https://opensource.org/licenses/BSD-3-Clause`

[8] Further information about the CeCILL license can be found through the following link: `http://www.cecill.info/index.en.html`

5

Experiments and Results

This chapter contains results from experiments both on the robot and simulation using `DART`, which were performed using the control strategy presented in Chapter 4. In Section 5.1, the performance of the robot for different stiffness and damping values is evaluated. Moreover, the results of exerting external Cartesian wrenches are also presented here. In Section 5.2, different experiments where singularities are involved are shown. Lastly, Section 5.3 shows the results of performing a contact-rich assembly task using said control strategy.

5.1 General Performance

This section shows the general performance of the control strategy through strategically designed experiments. Selected experiments were done both on the robot and in simulation, whilst others were performed on the robot only. This is because the FT-observer was only available in the ROS environment. Therefore, experiments that are partly validated through the estimated forces could not be shown in the simulation environment. The reason why this was not included in the `DART` environment was because of the time-restriction of the project.

Linear Trajectories

The trajectory generator described in Section 2.3 was used to generate a linear path from one pose to another. More specifically, the generated trajectories were executed with different stiffness values and damping factors. Moreover, the orientation of the end-effector was kept constant, and only the reference of the end-effector position along the y-direction was changed. The resulting experiments in `DART` and on the real robot are summarized in Figures 5.1 and 5.2, respectively.

Firstly, consider Figures 5.1a and 5.2a. In the top plots of these figures, the position of the end-effector along y-direction is shown. The same trajectory was repeated for three different stiffness values and the bottom plots show how the stiffness was varied during the experiment. Furthermore, the whole experiment was repeated with the damping factors (along all dimensions) 1.0, 0.85 and 0.7, respectively.

In Figures 5.1b and 5.2b, the positional errors (along all dimensions) for each repeated experiment are shown. The top, middle and bottom subplots represent the experiment using damping factors 0.7, 0.85 and 1.0, respectively. Furthermore, the error was derived using (2.25b).

Low Stiffness

In order to test how the robot performs for lower stiffnesses, the robot was once again commanded to track a linear positional path along one direction, but with lower translational stiffnesses. It turned out that using stiffnesses less than 200 N/m on the real robot led to significant deviations along all dimensions. On the other hand, no significant deviations appeared in the simulations. The results for these experiments in simulation and on the real system can be seen in Figures 5.3 and 5.4, respectively.

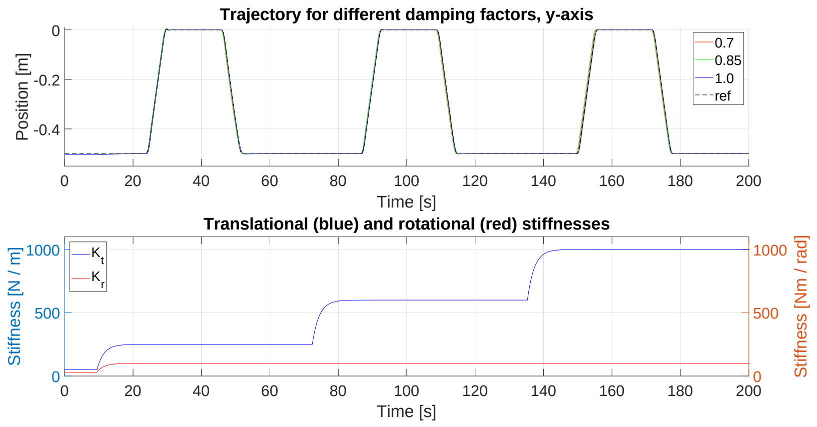
In Figures 5.3a and 5.4a, the top plots show how the desired end-effector position along the y-axis was being tracked and the bottom plots show the positional deviation along all directions. Furthermore, in these figures, the translational stiffnesses were set to 200 N/m. In Figures 5.3b and 5.4b, corresponding plots are shown for a similar experiment using a translational stiffness of 100 N/m instead.

Sliding Motion

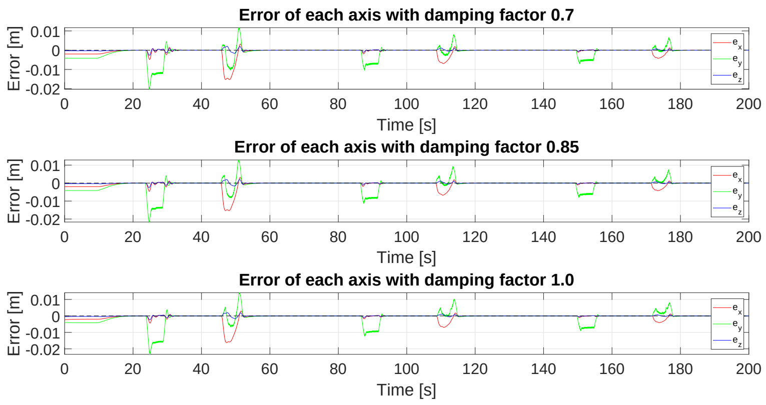
Cartesian forces were evaluated by setting low stiffness in the direction perpendicular to the black table seen in Figure A.1, and then commanding the robot to move along the other directions whilst applying a Cartesian force along the direction with low stiffness, resulting in a sliding motion on the surface of the table. Four different magnitudes of forces were exerted along said direction, namely 5 N, 8 N, 10 N and 12.5 N, and the results of applying these forces are shown in Figure 5.5.

In Figure 5.5, sliding motion with different force magnitudes applied at the end-effector is displayed in the base-frame orientation (z-axis of the end-effector is pointing in the opposite direction of the z-axis in the base-frame). The top figures show the commanded, along with the estimated (using the FT-estimator) forces. In the bottom figures, the position along the z-direction is shown. Moreover, the noisy parts of the FT-observer signals show when the robot is sliding along the surface.

Further illustration of the experiments can be visualized through Figures E.1–E.2 in Appendix E.



(a) Position of the end-effector along *y*-axis (top plot) and stiffness values (bottom plot). In the top plot, the reference position is black (dashed), while the experiments with damping values 0.7, 0.85 and 1.0 are shown in red, green and blue, respectively. In the bottom plot, the blue graph represents the translational stiffnesses, and the red graph represents the rotational stiffnesses.

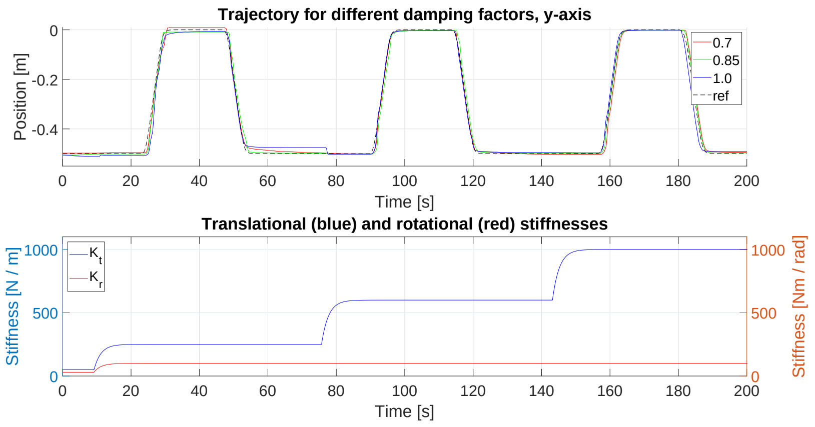


(b) Positional error for each repeated experiment. For each repeated experiment, the positional errors along the *x*, *y* and *z*-directions are shown in red, green and blue, respectively.

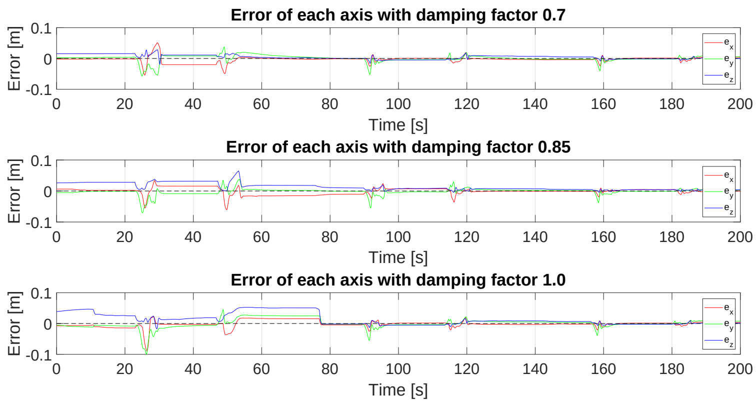
Figure 5.1: Linear trajectories for different stiffness values and damping factors, performed in `DART`. The values are identical for all dimensions.

Compliance

Some interesting cases occur when the translational stiffnesses are high and rotational stiffnesses low, or vice versa. Then, by perturbing the robot, one could verify that the robot was behaving as expected. More specifically, the expected behaviour



(a) Position of the end-effector along y-axis (top plot) and stiffness values (bottom plot). In the top plot, the reference position is black (dashed), while the experiments with damping values 0.7, 0.85 and 1.0 are shown in red, green and blue, respectively. In the bottom plot, the blue graph represents the translational stiffnesses, and the red graph represents the rotational stiffnesses.

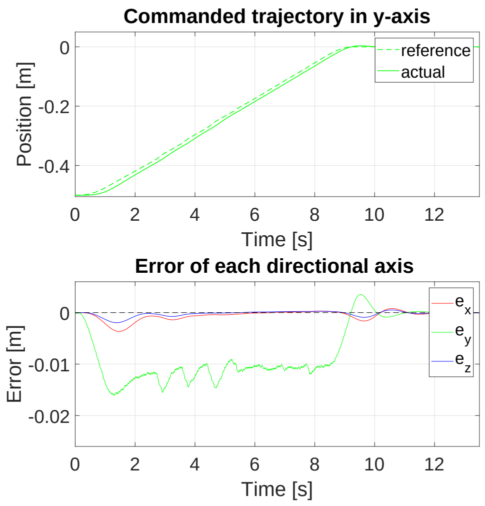


(b) Positional error for each repeated experiment. For each repeated experiment, the positional errors along the x, y and z-directions are shown in red, green and blue, respectively.

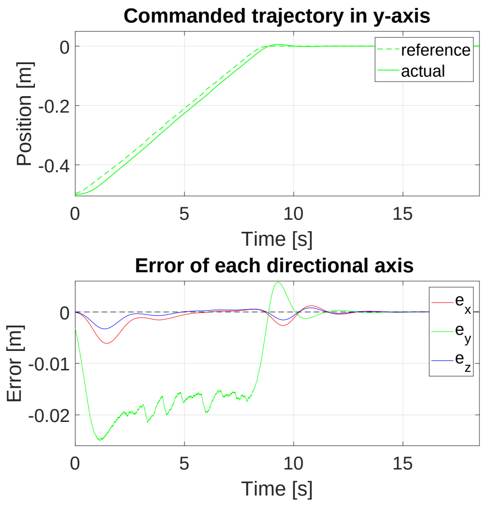
Figure 5.2: Linear trajectories for different stiffness values and damping factors, performed on the real robot. The varied parameters are identical for all dimensions.

for high translational stiffness is that the end-effector should not be able to deviate more than a few centimeters in terms of position, when disturbed externally by a human[1]. Similarly, high rotational stiffness should not allow the end-effector to

[1] This assumes that the applied disturbances are not abnormally high.

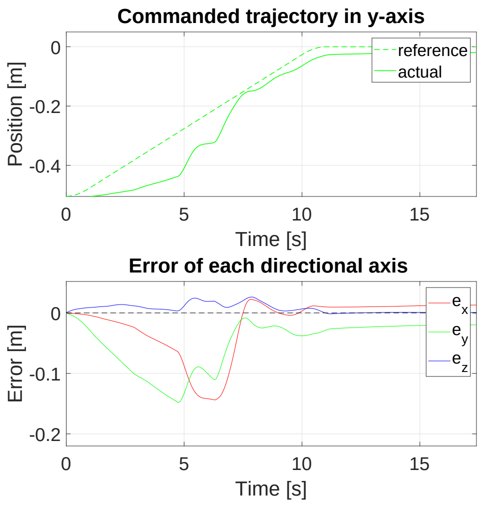


(a) The translational stiffness was set to 200 N/m for all directions.

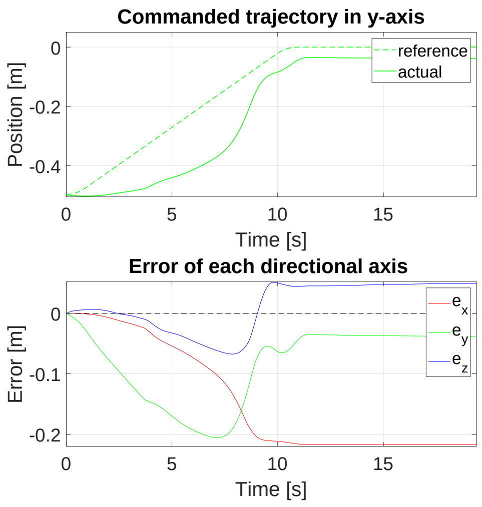


(b) The translational stiffness was set to 100 N/m for all directions.

Figure 5.3: Linear trajectory tracking for lower stiffness values, performed in `DART`. The damping factors were chosen to 1.0 for all directions.

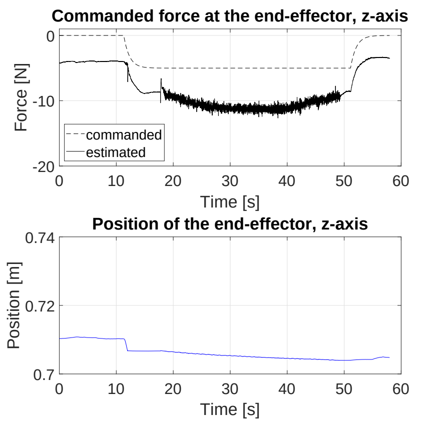


(a) The translational stiffness was set to 200 N/m for all directions.

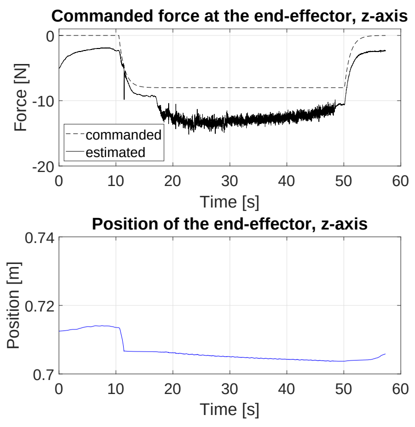


(b) The translational stiffness was set to 100 N/m for all directions.

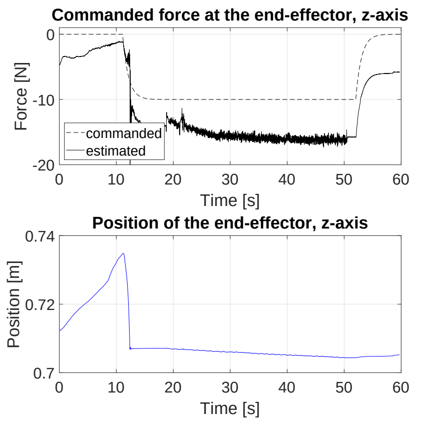
Figure 5.4: Linear trajectory tracking for lower stiffness values, performed on the real robot. The damping factors were chosen to 1.0 for all directions.



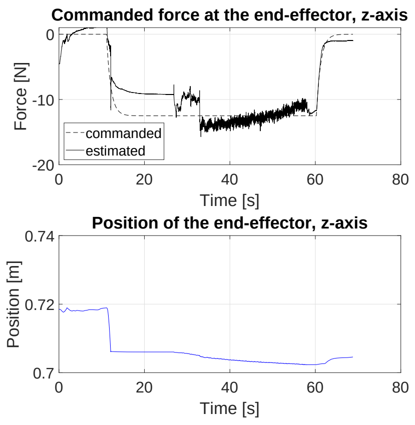
(a) Force applied: 5 N.



(b) Force applied: 8 N.



(c) Force applied: 10 N.



(d) Force applied: 12.5 N.

Figure 5.5: Sliding motion with different force magnitudes applied at the end-effector, z-axis.

rotate much when perturbed.

Another interesting case is when both the translational and rotational stiffnesses are high, while perturbing the robot along the nullspace dimensions. This allowed verifying that nullspace motion was possible while keeping the end-effector at the desired pose with minimal deviation.

These experiments were, however, only tested on the real robot as it is much easier to verify the behaviour that way, that is, both visually, but also applying disturbances. Besides, the FT signal estimation was, as mentioned before, only available in the ROS environment for this work, and the experiment in the simulation environment was therefore excluded.

Translational Compliance Translational compliance, i.e., low translational stiffnesses (close to zero), allowed moving the robot around freely in space without much perturbation. However, as the rotational stiffnesses were high, rotating the end-effector was hard and therefore the rotational error would stay relatively low throughout the experiment. The results of said experiment are shown in Figure 5.6.

In Figure 5.6, the top-left and bottom-left plots show the estimated (using the FT-observer) Cartesian forces and torques through interaction, respectively. The top-right and bottom-right plots show the end-effector positional and rotational errors (using Euler-XYZ representation), respectively. As the translational stiffnesses were low, the end-effector position would move away from the desired position with little effort, hence increasing the translational error noticeably. On the other hand, the rotational error would stay relatively small throughout the experiment.

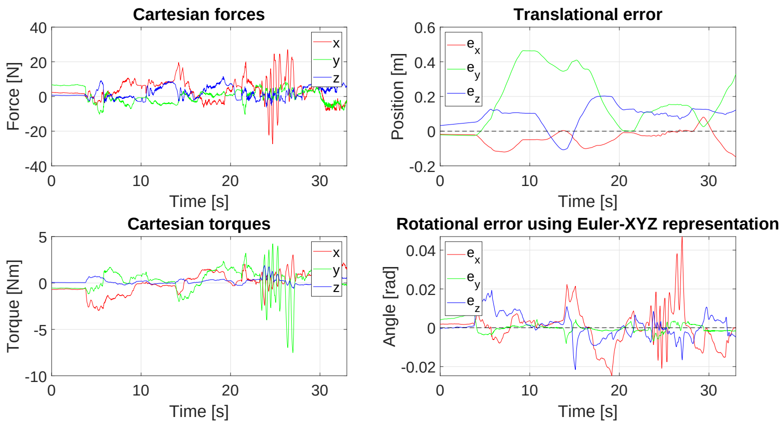


Figure 5.6: Physical perturbation on the robot with translational compliance. Around the time interval 23 s to 27 s, the robot end-effector was manually shaken a little along a specific direction, hence the rapid oscillations.

Rotational Compliance On the other end, rotational compliance allowed to rotate the end-effector freely when perturbed, but it was much harder to move around in terms of position. Thus, the positional error would stay relatively low, and the overall experiment is summarized in Figure 5.7.

In Figure 5.7, the top-left and bottom-left plots show the estimated (using the FT-observer) Cartesian forces and torques, respectively. The top-right and bottom-right plots show the end-effector positional and rotational errors (using Euler-XYZ representation), respectively. As the rotational stiffnesses were low, the end-effector orientation would deviate from the desired orientation through little effort. On the other hand, the translational error would stay relatively small throughout the experiment.

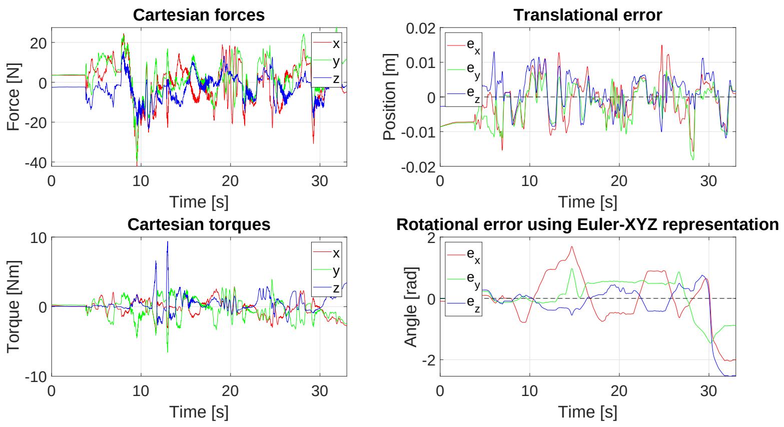
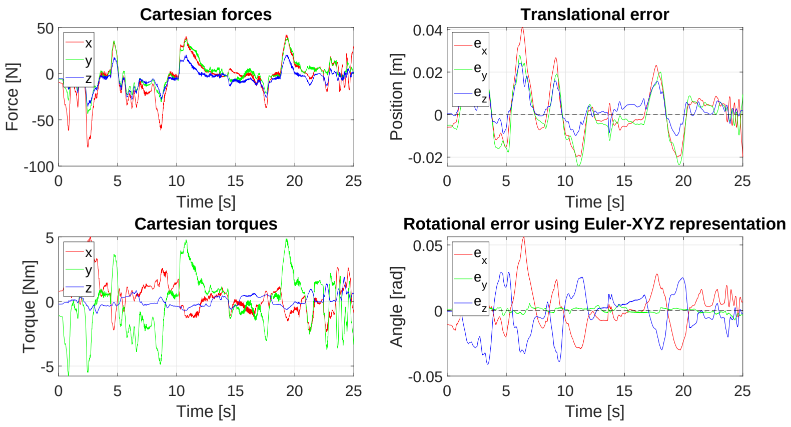


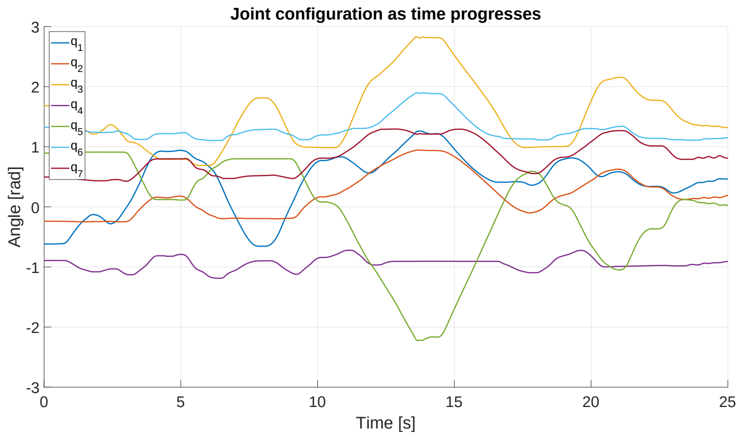
Figure 5.7: Physical perturbation on the robot with rotational compliance.

Nullspace Control Using none or small nullspace stiffness while perturbing the robot, along with high translational and rotational stiffnesses, allows the joint configuration to change considerably while keeping the end-effector close to the desired pose. A translational stiffness of 2000 N/m along all directions and a rotational stiffness of 200 Nm/rad along all directions were used. Furthermore, the nullspace stiffness was set to zero, which allowed free nullspace motion. The results of this experiment can be studied in Figure 5.8.

In Figure 5.8a, the top-left and bottom-left plots show the estimated (using the FT-observer) Cartesian forces and torques through interaction, respectively. The top-right and bottom-right plots show the end-effector positional and rotational errors (using Euler-XYZ representation), respectively. Moreover, in Figure 5.8b, the joint-configuration of the robot is presented. It shows how the robot was able to change configuration noticeably and continuously, while keeping the end-effector deviation small.



(a) The top-left and bottom-left plots show the estimated (using the FT-observer) Cartesian forces and torques through interaction, respectively. The top-right and bottom-right plots show the end-effector positional and rotational errors (using Euler-XYZ representation), respectively.



(b) Joint configuration of the robot throughout the whole experiment.

Figure 5.8: Illustration of nullspace motion. As the robot was perturbed throughout the experiment, the joint configuration would change considerably while at the same time keeping the end-effector deviation relatively small.

5.2 Singularities

This subsection shows the behaviour of the robot ccccwhen approaching singularities in different ways. Some of the experiments were done both on the real robot and in simulation, but only the experiments on the real robot will be shown here, as the output of the FT-observer is important to consider, but, as stated before, was only available in the ROS environment.

Going In and Out of a Singularity

The following experiment was performed in order to test how the robot would behave, should the robot for some reason reach a singularity during a task. The robot was first driven into a singularity, by using nullspace stiffness along with a strategically chosen nullspace configuration, allowing two or more joints to be co-linearly aligned. When the robot reached the chosen singular configuration, it would deliberately stay there for some time until eventually be driven out using the trajectory generator, to a non-singular pose. The results from this experiment are summarized in Figure 5.9.

In Figure 5.9, the top and middle plots show the Cartesian effort in terms of forces and torques, respectively. The bottom figure shows how the nullspace stiffness and translational stiffness were varied during the experiment. During the time interval 12.5 s to 28.5 s, the robot was in a singular configuration. By 30 s, the robot was moved out of the singular configuration. At the same time, the nullspace stiffness was commanded to zero, and the translational stiffness was quadrupled. For more detailed visualization of the experiment, see Figure E.5 in Appendix E.

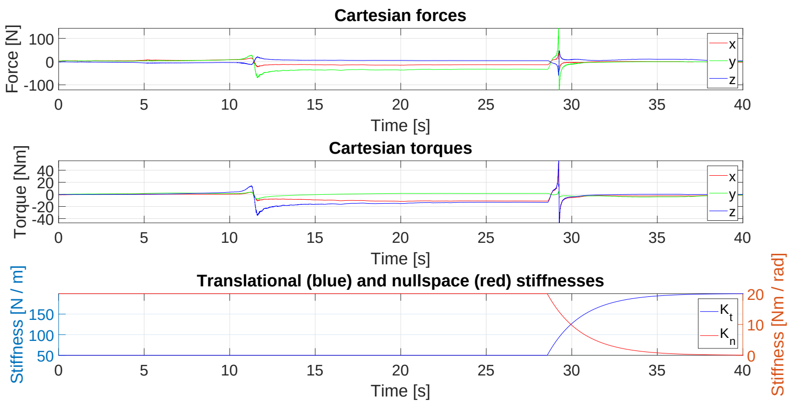


Figure 5.9: Driving the robot into a singularity, with two of the joints co-linearly aligned.

Commanding Out-Of-Reach Poses

Another way of driving the robot into a singularity is by commanding a position that is outside the workspace, hence allowing the robot to stretch out. The results of commanding a pose outside the workspace are shown in Figure 5.10.

In Figure 5.10, the top plot shows the position of the end-effector, along with the desired position. After around 18 s into the experiment, the robot was stretched out and not moving anymore. This can also be seen in the middle plot, where the translational deviation is shown. As the target pose is mostly out-of-reach along the x-axis, the error along said axis became the largest, about 18.4 cm. In the third plot, the Cartesian forces that were estimated by the FT-observer, are shown. For this particular experiment, a stiffness of 500 N/m was used for all translational directions. Further illustration of the experiment can be seen in Figures E.3–E.4 in Appendix E.

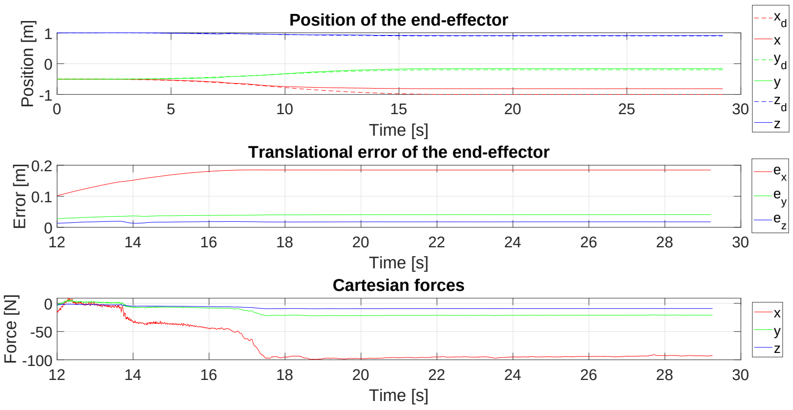


Figure 5.10: Commanding a position outside the workspace.

5.3 Peg-in-Hole

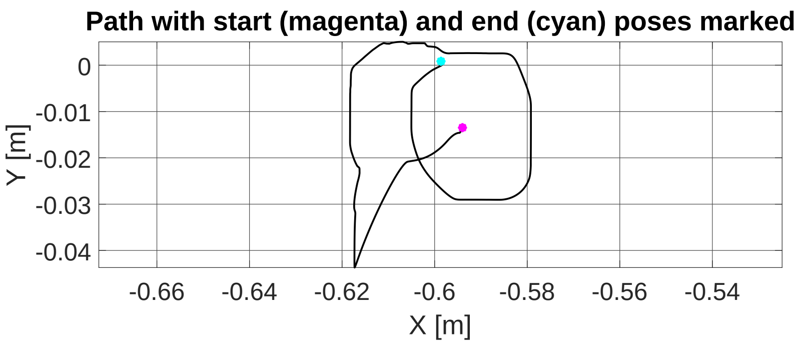
Considering what has been presented so far, the robot would in general behave in a stable and decent way, as far as the configurable parameters would stay within reasonable limits. However, to get a better idea of how the framework performs in intended application tasks, that is, contact-rich tasks, assembly experiments were conducted. More specifically, a peg-in-hole assembly task was performed both in simulation and on the robot.

The procedure was done as follows: Firstly, a box with hole with a diameter slightly bigger (about 5 mm) than the peg attached at the end-effector was provided, and placed on the black table that is visible in Figure A.1. That is, both in simulation (as a `URDF`-model), but also a physical box for the real experiments. The robot end-effector was then commanded to a pose which aligned the peg with the hole in the box, slightly above it. Then, the stiffness along the vertical direction (z-axis) was set to a value close to zero, whilst the stiffnesses along the other directions (x and y axes) were empirically set to 300 N/m, such that the target pose could be tracked reasonably well (empirically), while keeping the forces as minimal as possible. As it was desired to keep the peg perpendicular to the hole throughout the experiment, the rotational stiffnesses (x and y axes) along the stiff directions were set to reasonably high values of 100 Nm/rad, and the remaining axis was instead set to a smaller value of 5 Nm/rad. This of course assumes that the mounted tool, i.e., peg, is symmetric along the z-axis (the attack direction of the end-effector). Moreover, the nullspace stiffness was set to zero.

With this in place, the next step was to command a search motion[2] using the Cartesian trajectory generator, along the stiff translational directions, while applying a force towards the box, resulting in a sliding motion along the surface of the box. The shaping of the search motion was configured, such that the peg would always be inserted in the hole after some movement on the surface of the box. When the peg was inserted in the hole, the experiment was considered to be completed. The results from the experiment in simulation and on the real system are shown in Figures 5.11 and 5.12, respectively.

The upper plots of Figures 5.11 and 5.12 show the path of the end-effector along the directions (x and y axes) with higher stiffness (300 N/m) whilst the peg was in contact with the surface of the box. At the start (magenta dot), the peg was slightly above the box, and at the end (cyan dot), the peg was inserted in the hole. This can also be seen in the lower plots of the figures, where the height (z-axis) of the end-effector is shown. At the start, (0 s – 2 s), the peg was still not in contact with the box. During the search motion (roughly 2 s – 18 s) the height was constant, until the peg became aligned with the hole, resulting in decline in height. The sequence of the experiment can also be visualized by studying Figure E.6 in Appendix E.

[2] The shape of the search motion was circular with increasing radius, and was shaped to move along a path on the surface that eventually led to peg insertion.



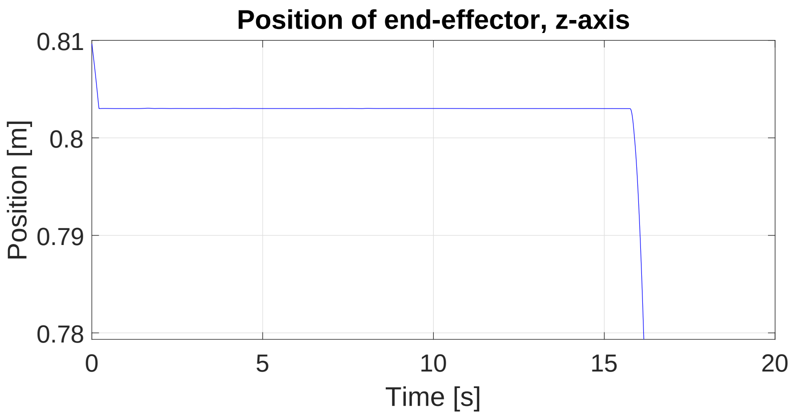
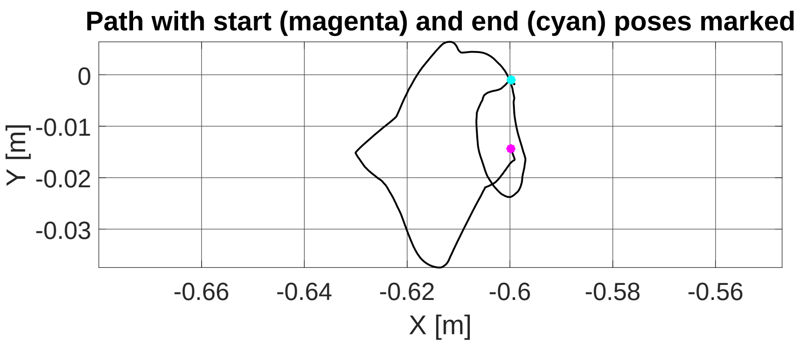


Figure 5.11: Peg-in-hole task performed in `DART`. In the upper figure, the path starts at the pose marked as a magenta dot, and ends at the pose marked as a cyan dot.



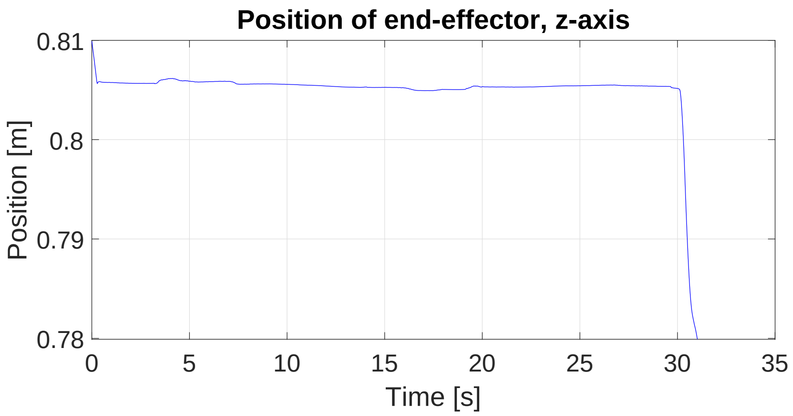


Figure 5.12: Peg-in-hole task performed on the real robot. In the upper figure, the path starts at the pose marked as a magenta dot, and ends at the pose marked as a cyan dot.

6 Discussion

This chapter aims at giving a critical review of the obtained results.

6.1 Discussion of the Results

General Performance

Overall, the robot behaved reasonable and could track desired trajectories as long as the stiffnesses were sufficiently high. As presented in Chapter 5, the value of the stiffness stimulated the trajectory-tracking deviation for the end-effector in, arguably, a proportional way. The results in simulation and on the real robot showed that the deviation in position would decrease as the stiffness increase (see Figures 5.1 and 5.2). In simulation, however, the deviation would not reach a dimensional error of more than 2 cm during a trajectory when using a stiffness of 250 N/m, as seen in Figure 5.1b. On the other hand, the corresponding experiment on the real robot showed that the deviation could exceed 5 cm. Besides this, there would often appear stationary errors on the real system (unlike in the simulations), especially for lower stiffness values. This is most probably a result of the inherent friction in the robot joints, which were not accounted for. In fact, the model in simulation did not include any friction, and could therefore track trajectories with virtually no stationary positional errors, even for stiffness values of 100 N/m or lower (see Figure 5.3b). Using any value lower than 200 N/m on the real robot, however, showed that the deviation could be off about 2 cm (see Figure 5.4).

The adjustment of damping factors did not seem to affect the results in simulation significantly. In fact, the overall behaviour in terms of deviation, as seen in Figure 5.1b, seemed to be, apart from the small tweaking in magnitudes, identical (for the tested damping factors). On the other hand, the real experiments showed that a combination of low stiffnesses and high damping factors (250 N/m and 1.0, respectively) gave rise to a noticeable stationary error, as seen in Figure 5.2. One must, however, be careful in drawing any conclusions about the consistency of these results. As the experiments were only tested a few times (at most) each, it is hard to say whether these outcomes are reliable in terms of consistency and repeatability.

The sliding-motion experiments showed that the exerted forces would most probably be exerted correctly. Although it is apparent in the top plots in Figure 5.5 that the estimated forces were not perfectly aligned with the commanded wrench, they still prove that the force is applied along the correct direction with somewhat correct (but offset) magnitudes. After all, when no wrench was applied, the estimated forces would still be offset by a non-zero value. This was probably because of calibration errors, as well as uncertainties in the FT observer, but it must be further investigated for validity. Furthermore, the decline in height in the lower plots in Figure 5.5 was probably due to the surfaces not being completely flat and smooth. The decline would, however, not exceed more than a few millimeters. Either way, the main idea with this experiment was to get a sense of validation for commanding Cartesian wrenches.

The tested compliance experiments on the real system yielded expected results. The robot would show expected behaviour during translational, rotational and nullspace compliance, as seen in Figures 5.6, 5.7 and 5.8, respectively.

Singularities

The first singularity experiment, i.e., going in and out of a singular configuration, showed that the controller and the robot would not indicate any hints of oscillations or instability. This was the case even though the experiment appeared to produce spiky estimated Cartesian forces and torques around 12.5 s in Figure 5.9 whilst reaching the singularity. However, this did not appear visually during the experiment, and the spikes could therefore simply be a result from how the FT-observer is estimating the forces. Moreover, the spikes also seemed to correlate to stiffness changes around 30 s. But, as stated before, one cannot draw any solid conclusions about the consistency of said results, as extensive repeatability of the experiments were not tested.

The other singularity that was tested, i.e., commanding out-of-reach poses, also showed that the robot would behave in a stable way. As the robot would stretch out to try and reach the desired pose, a Cartesian force would obviously build up the further the pose deviated from the desired one (up until a certain point), as seen in Figure 5.10. One could also verify the magnitude of the stiffness along a direction, by simply dividing the Cartesian force along the same direction, with the corresponding deviation along the same direction. For instance, consider the stationary values of Cartesian force and translational stiffness along the x-axis in Figure 5.10, respectively. The estimated magnitude of the translational stiffness along the x-axis would then be roughly 95 N / 0.185 m $\approx$ 514 N/m, which is not that far away from the actual set stiffness of 500 N/m. However, in order to verify if this is a coincidence, one could try out the same experiment using different stiffness values and observe whether the results are consistent or not.

Peg-In-Hole

The last experiment, i.e., the peg-in-hole assembly task, was performed in order to evaluate how well the robot would behave in a contact-rich task. As shown in Figures 5.11–5.12, the attached peg on the robot would eventually get inserted into a box with a hole, after sliding along the surface of the box for some time. This motion was, as stated earlier, intended to replicate a type of circular search motion. However, neither the timing nor the shape of the search-paths were accurately similar, comparing the two cases. In fact, the search motion lasted for about twice as long on the real robot compared to the simulations.

There are several factors that may have a significant role in determining said accuracy. Firstly, the inherent stiction in the real robot resulted in the movement being more jerky compared to the simulations. Secondly, the friction between the surface of the box and the surface of the peg was not the same in simulation as on the real system. In fact, an adjustable friction parameter of the surface of the box with hole was given, but only a limited time was spent on trying to find the optimal friction value. On the other hand, the intention of this experiment was not to try and match the simulation and robot behaviours optimally. Rather, the outcomes can help indicate how well the control framework might perform in contact-rich tasks.

7 Conclusions and Future Work

As stated in the problem formulation, the main objective of this work was to develop and evaluate a torque-based Cartesian impedance controller in a ROS environment as well as in a DART environment. Thus, said control strategy was implemented, which allows treating a state-of-the-art light-weight robot arm as an impedance, and its environment as an admittance. With regards to the implementation of the framework, it can be seen as a composition of three main building blocks. The first block consisted of the actual control implementation with calculations and such, along with other embedded features as well. In the second block, a ROS environment that allowed running the control implementation on the real robot was developed. In the third and last block, a similar implementation was used, but in a DART environment instead.

On top of this, the desired feature of being able to configure parameters that shapes the impedance of the robot was desired and hence implemented. The framework also includes the desired feature of exerting Cartesian wrenches on command, during run-time. Furthermore, it was also desired to have a strategy that works well in simulation as well as on the real robot. In both cases, the results have shown that the controller allows the robot arm to become compliant in terms of position and orientation, but also in terms of nullspace motion. On the other hand, they have also showed what limitations the control strategy inherits from, e.g., stiction and inaccurate calibration, allowing accuracy to be achieved only under certain bounds. Nevertheless, the overall performance showed promising results, which may be used for future work.

7.1 Future Work

Some aspects that may be interesting to investigate further in future work are:

- Investigate and implement control strategies that overcome the situation of the real system. That is, strategies that reduce the limitations of the system, allowing a more sophisticated, compliant and expected behaviour.
- Identifying the friction of the real robot so that it matches the simulations better.
- Including an FT-observer in the `DART` environment, so that the simulations become more comparable.
- Investigate how computational time affects the stability and performance of the controller. An interesting follow-up would then be to look into how one could decrease the computational time.
- Investigate the repeatability and consistency of tasks using the Cartesian impedance controller.

References

Abu-Dakka, F. J. and M. Saveriano (2020). “Variable Impedance Control and Learning—A Review”. *Frontiers in Robotics and AI* **7**. ISSN: 2296-9144. DOI: 10.3389/frobt.2020.590681.

Baker, M. J. (2021). *Maths - Angle between vectors*. EuclideanSpace - Mathematics and Computing. URL: https://www.euclideanspace.com/maths/algebra/vectors/angleBetween/ (visited on 2021-05-09).

Ben-Ari, M. (2014). *A Tutorial on Euler Angles and Quaternions*. Department of Science Teaching, Weizmann Institute of Science. URL: https://www.weizmann.ac.il/sci-tea/benari/sites/sci-tea.benari/files/uploads/softwareAndLearningMaterials/quaternion-tutorial-2-0-1.pdf (visited on 2021-04-05).

Buck, A., E. Price, R. Park, N. Peterson, D. Kshirsagar, D. Coulter, D. Stanford, K. Furbush, N. Schonning, and M. Wilson (2018). *Publisher-Subscriber pattern*. Microsoft. URL: https://docs.microsoft.com/en-us/azure/architecture/patterns/publisher-subscriber (visited on 2021-07-14).

Cao, Y., K. Lu, X. Li, and Y. Zang (2011). “Accurate Numerical Methods for Computing 2D and 3D Robot Workspace”. *International Journal of Advanced Robotic Systems* **8**, p. 1. DOI: 10.5772/45686.

Castro, S. (2019). *Trajectory Planning for Robot Manipulators*. Mathworks. URL: https://medium.com/mathworks/trajectory-planning-for-robot-manipulators-522404efb6f0 (visited on 2021-04-15).

Chatzilygeroudis, K. and J.-B. Mouret (2021). *RobotDART*. URL: http://www.resibots.eu/robot_dart/ (visited on 2021-02-16).

Chitta, S., E. Marder-Eppstein, W. Meeussen, V. Pradeep, A. R. Tsouroukdissian, J. Bohren, D. Coleman, B. Magyar, G. Raiola, M. Lüdtke, and E. F. Perdomo (2017). “ros_control: A generic and simple control framework for ROS”. *Journal of Open Source Software* **2**:20. DOI: 10.21105/joss.00456.

Dattalo, A. (2018). *ROS/Introduction*. ROS. URL: `http://wiki.ros.org/ROS/Introduction` (visited on 2021-02-16).

Denning, T. (2017). *End Effector Accessories*. Design For Making. URL: `https://www.designformaking.com/m10` (visited on 2021-02-19).

Deshpande, V. and A. Verma (2010). "Dynamics of Robot Manipulators: A Review". *International Journal of Engineering Research and Technology* **3**, pp. 603–606.

Emika, F. (2017). "franka_ros". *GitHub repository*. URL: `https://github.com/frankaemika/franka_ros`.

Foote, T., E. Marder-Eppstein, and W. Meeussen (2017). *tf; Package Summary*. ROS. URL: `http://wiki.ros.org/tfe` (visited on 2021-05-13).

Freidovich, L. B. (2013). *Control Methods for Robotic Applications: Lecture Notes*. Saint Petersburg National Research University of Information Technologies Mechanics and Optics, St. Petersburg, Russia. Umeå University. URL: `http://umu.diva-portal.org/smash/record.jsf?pid=diva2%3A663949&dswid=9505f` (visited on 2021-02-15).

Gazebo (2014). *Gazebo; Robot simulation made easy*. Open Source Robotics Foundation. URL: `http://gazebosim.org/` (visited on 2021-05-13).

Guarana-DIY (2020). "The Top Six Types of Industrial Robots in 2020". URL: `https://diy-robotics.com/blog/top-six-types-industrial-robots-2020/` (visited on 2021-03-04).

Hershberger, D., D. Gossow, J. Faust, and W. Woodall (2018). *rviz; Package Summary*. ROS. URL: `http://wiki.ros.org/rviz` (visited on 2021-05-13).

Hogan, N. (1984). *Impedance Control: An Approach to Manipulation*. DOI: `10.23919/ACC.1984.4788393`.

Holmesson, J. (2021). "Accurate Simulation of a Collaborative Robot Arm with Cartesian Impedance Control". M.Sc. TFRT-6142. ISSN: 0280–5316.

Hosseinpour, F. and H. Hajihosseini (2009). "Importance of Simulation in Manufacturing". *International Journal of Economics and Management Engineering* **3**:3, 229–232. ISSN: 1307-6892.

IBM Cloud Education (2021). *Middleware*. IBM Cloud Education. URL: `https://www.ibm.com/cloud/learn/middleware` (visited on 2021-07-19).

Infineon (2018). "Fundamentals of robotics". URL: `https://www.infineon.com/cms/en/discoveries/fundamentals-robotics/` (visited on 2021-03-02).

Kaelbling, L. P., M. L. Littman, and A. W. Moore (1996). "Reinforcement Learning: A Survey". *Journal of Artificial Intelligence Research* **4**, pp. 237–285. DOI: `10.1613/jair.301`.

Khatib, O. (1987). “A unified approach for motion and force control of robot manipulators: The operational space formulation”. *IEEE Journal on Robotics and Automation* **3**:1, pp. 43–53. DOI: 10.1109/JRA.1987.1087068.

KUKA (2021). *LBR iiwa*. URL: https://www.kuka.com/en-se/products/robotics-systems/industrial-robots/lbr-iiwa (visited on 2021-05-09).

Lee, J., M. X. Grey, S. Ha, T. Kunz, S. Jain, Y. Ye, S. S. Srinivasa, M. Stilman, and C. K. Liu (2018). “DART: Dynamic Animation and Robotics Toolkit”. *Journal of Open Source Software* **3**:22, p. 500. DOI: 10.21105/joss.00500.

Luca, A. D. (2014). *Trajectory planning in Cartesian space*. Course material: Robotics 1. Department of Computer, Control, and Management Engineering Antonio Ruberti, Sapienza University of Rome. URL: http://www.diag.uniroma1.it/~deluca/rob1_en/14_TrajectoryPlanningCartesian.pdf (visited on 2021-04-17).

Meeussen, W. (2016). *pr2_controller_interface; Package Summary*. ROS. URL: http://wiki.ros.org/pr2_controller_interface (visited on 2021-06-01).

Miller, B. (2018). *Parameter Server*. ROS. URL: http://wiki.ros.org/Parameter%20Server (visited on 2021-05-13).

OpenAI (2018). *Learning Dexterity*. OpenAI. URL: https://openai.com/blog/learning-dexterity/ (visited on 2021-02-16).

Ott, C. (2008). *Cartesian Impedance Control of Redundant and Flexible-Joint Robots*. Vol. 49. Springer Tracts in Advanced Robotics. Springer, Berlin, Heidelberg. ISBN: 978-3-540-69253-9. DOI: 10.1007/978-3-540-69255-3.

Płaczek, M. and L. Piszczek (2018). “Testing of an industrial robot’s accuracy and repeatability in off and online environment”. *Eksploatacja i Niezawodnosc - Maintenance and Reliability* **20**, pp. 455–464. DOI: 10.17531/ein.2018.3.15.

Scherzinger, S., A. Roennau, and R. Dillmann (2017). “Forward Dynamics Compliance Control (FDCC): A new approach to cartesian compliance for robotic manipulators”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4568–4575. DOI: 10.1109/IROS.2017.8206325.

Shoemake, K. (1985). “Animating Rotation with Quaternion Curves”. *SIGGRAPH Comput. Graph.* **19**:3, pp. 245–254. ISSN: 0097-8930. DOI: 10.1145/325165.325242.

Spong, M., S. Hutchinson, and M. Vidyasagar (2006). “Robot modeling and control”. *IEEE Control Systems* **26**:6, pp. 19–21. ISSN: 1066-033X. DOI: 10.1109/MCS.2006.252815.

Sprague, N. (2016). *Coordinate Frames*. URL: https://www.academia.edu/38172552/Frames (visited on 2021-06-13).

Statista Research Department (2021). *Size of the global market for industrial and non-industrial robots between 2018 and 2025*. Statista Research Department. URL: `https://www.statista.com/statistics/760190/worldwide-robotics-market-revenue/` (visited on 2021-07-28).

Stewart, G. W. (1977). "On the Perturbation of Pseudo-Inverses, Projections and Linear Least Squares Problems". *SIAM Review* **19**:4, pp. 634–662. ISSN: 00361445. DOI: `10.1137/1019104`.

Sucan, I. and J. Kay (2019). *urdf; Package Summary*. ROS. URL: `http://wiki.ros.org/urdf/` (visited on 2021-05-13).

Thomas, D., D. Scholz, and A. Blasdel (2016). *rqt; Package Summary*. ROS. URL: `http://wiki.ros.org/rqt` (visited on 2021-05-13).

Virga, S., M. Esposito, and D. Niewinski (2019). "IIWA STACK". URL: `https://github.com/IFL-CAMP/iiwa_stack` (visited on 2021-02-15).

Williams, A. (2018). *Applying Machine Learning to Robotics*. Robotics Business Review. URL: `https://www.roboticsbusinessreview.com/wp-content/uploads/2018/04/RBR_MachineLearningRobots_WP_Final.pdf` (visited on 2021-02-16).

Zhang, W. (2018). *Velocity Kinematics and Jacobian*. Course material: Introduction to Robotics ECE5643, Lecture Note 7. Department of Electrical and Computer Engineering Ohio State University Columbus, Ohio, USA. URL: `http://www2.ece.ohio-state.edu/~zhang/RoboticsClass/docs/LN7_VelocityKinematics_a.pdf` (visited on 2021-05-10).

A

Appendix — Robot Setup

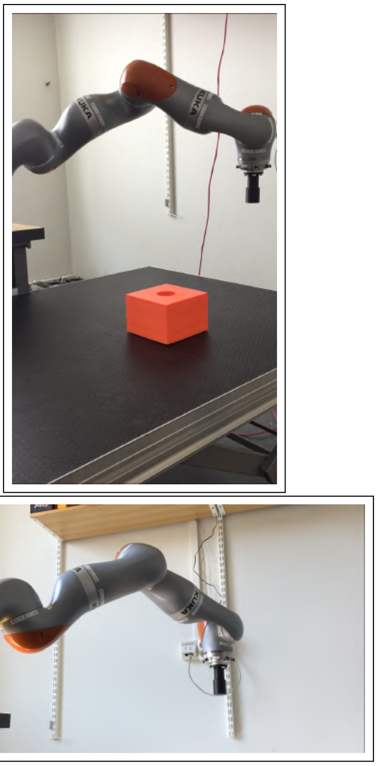


Figure A.1: The LBR iiwa robot setup. A peg-tool is attached at the end-effector, intended for peg-in-hole experiments. For these experiments, a box with a hole was provided.

B

Appendix — Simulation Environments

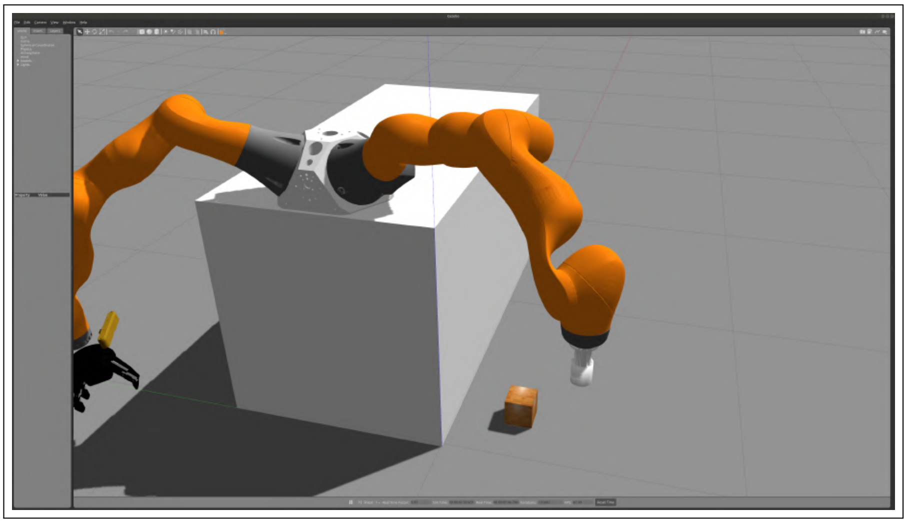


Figure B.1: URDF model of LBR iiwa setup in the Gazebo environment.

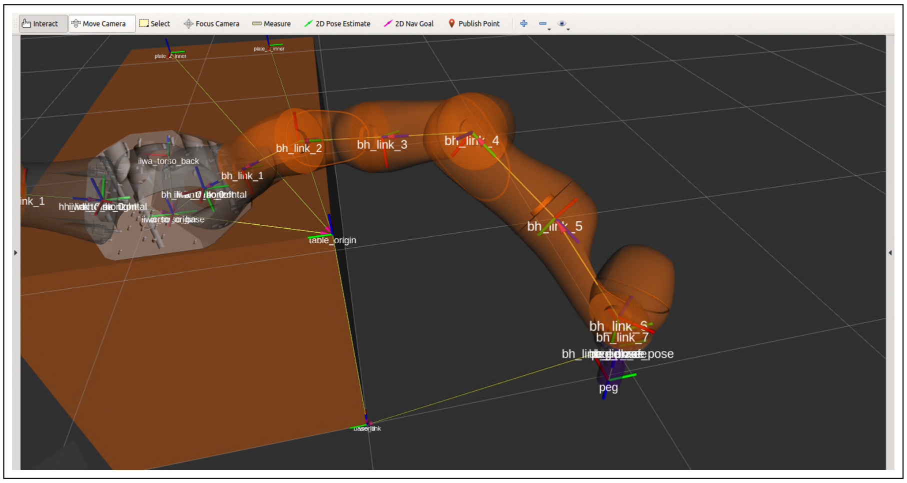


Figure B.2: URDF model of LBR iiwa setup in the rviz environment. The robot is half transparent, making it possible to see the spatial chain of frames. Moreover, the base frame is located at the bottom right corner of the table.

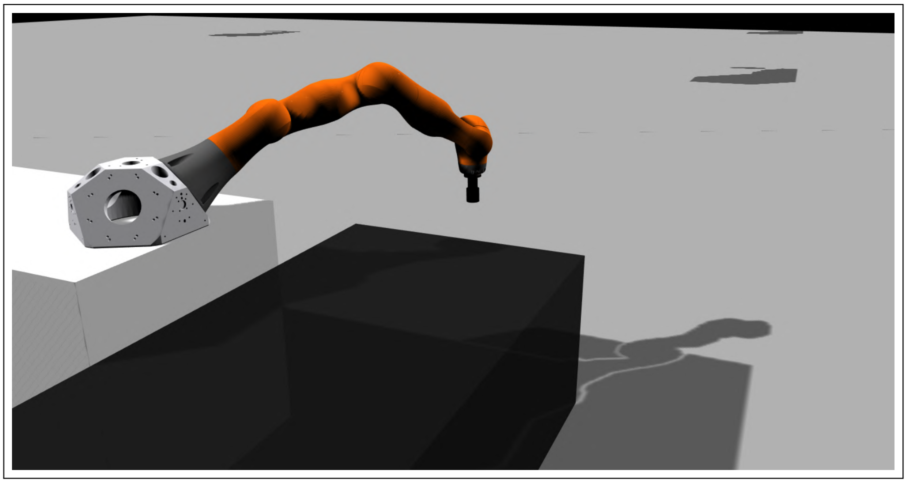


Figure B.3: URDF model of LBR iiwa setup in the RobotDART environment. In comparison to the other simulation environments, there is also a black box added, replicating the black table of the real robot setup (compare with Figure A.1).

C

Appendix — Computational Time

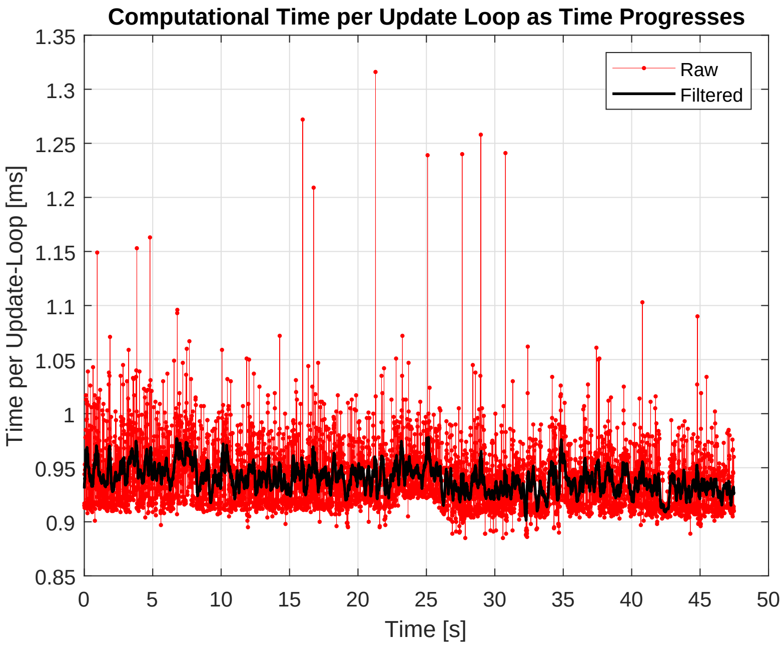
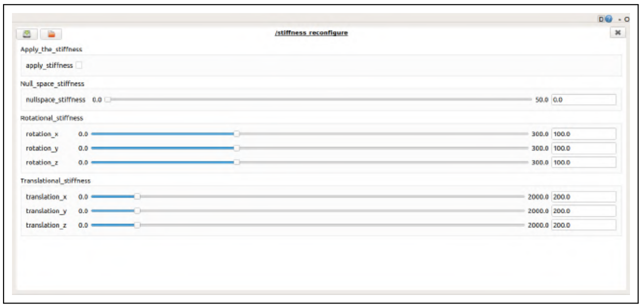


Figure C.1: Computational time per update loop as time progresses. The raw (red) data show how the computational time would vary from just under 0.9 ms to more than 1.25 ms at some occasions. The filtered (black) data are obtained using moving average with a window of 20 data points. It shows that the computational time would, on average, vary between 0.9 – 1 ms per update loop.

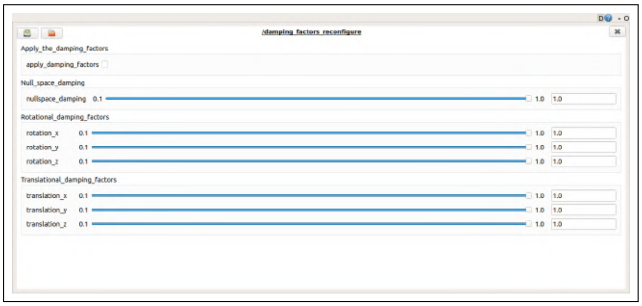
D

Appendix — rqt_reconfigure Plugin



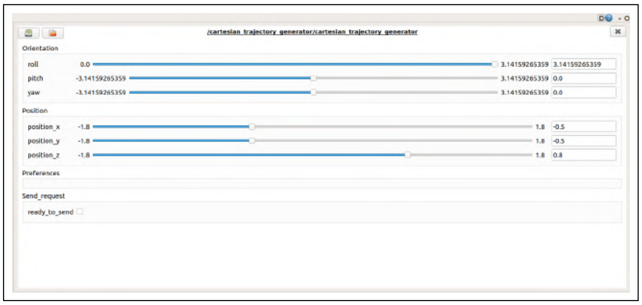
```
/stiffness_reconfigure
Apply_the_stiffness
  apply_stiffness ☐
Null_space_stiffness
  nullspace_stiffness  0.0   50.0    0.0
Rotational_stiffness
  rotation_x     0.0   300.0   100.0
  rotation_y     0.0   300.0   100.0
  rotation_z     0.0   300.0   100.0
Translational_stiffness
  translation_x  0.0   2000.0  200.0
  translation_y  0.0   2000.0  200.0
  translation_z  0.0   2000.0  200.0
```

Figure D.1: Dynamic reconfiguration of stiffness values using `rqt_reconfigure`.



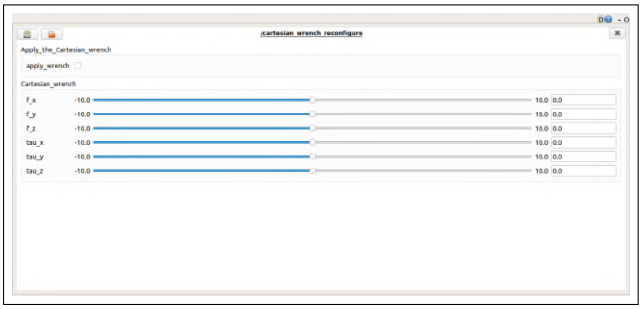
```
/damping_factors_reconfigure
Apply_the_damping_factors
  apply_damping_factors ☐
Null_space_damping
  nullspace_damping  0.1   1.0  1.0
Rotational_damping_factors
  rotation_x     0.1   1.0  1.0
  rotation_y     0.1   1.0  1.0
  rotation_z     0.1   1.0  1.0
Translational_damping_factors
  translation_x  0.1   1.0  1.0
  translation_y  0.1   1.0  1.0
  translation_z  0.1   1.0  1.0
```

Figure D.2: Dynamic reconfiguration of damping factors using `rqt_reconfigure`.



```
/cartesian_trajectory_generator/cartesian_trajectory_generator
Orientation
  roll        0.0            3.14159265359  3.14159265359
  pitch       -3.14159265359 3.14159265359  0.0
  yaw         -3.14159265359 3.14159265359  0.0
Position
  position_x  -1.8   1.8  -0.5
  position_y  -1.8   1.8  -0.5
  position_z  -1.8   1.8  0.8
Preferences
Send_request
  ready_to_send ☐
```

Figure D.3: Dynamic reconfiguration of the desired pose using `rqt_reconfigure`.



```
/cartesian_wrench_reconfigure
Apply_the_Cartesian_wrench
  apply_wrench ☐
Cartesian_wrench
  f_x    -10.0  10.0  0.0
  f_y    -10.0  10.0  0.0
  f_z    -10.0  10.0  0.0
  tau_x  -10.0  10.0  0.0
  tau_y  -10.0  10.0  0.0
  tau_z  -10.0  10.0  0.0
```

Figure D.4: Dynamic reconfiguration of Cartesian wrenches using `rqt_reconfigure`.

E

Appendix — Experiments

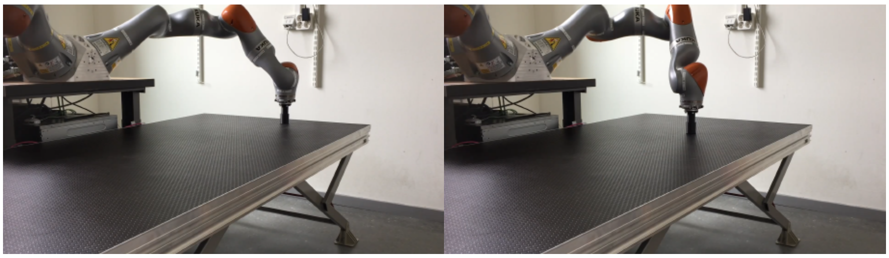


Figure E.1: Sliding motion along the surface of the table with initial position (left picture) and final position (right picture) shown.

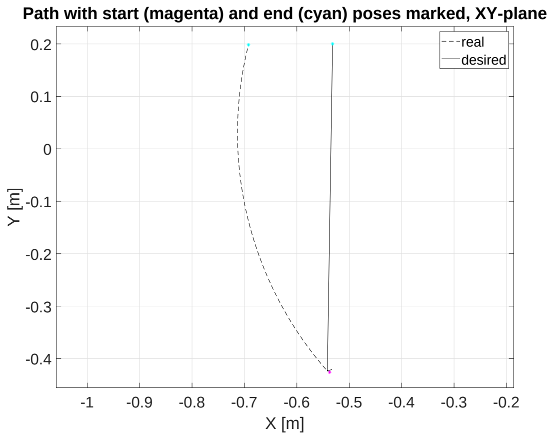


Figure E.2: Sliding motion along the surface of the table. The translational stiffness along the *x*-direction was set to a value lower than 10 N/m, hence the noticeable deviation along said direction. On the other hand, the stiffness along the *y*-direction was set to 1000 N/m and the deviation was therefore not as big as along the *x*-direction. The combination of a high stiffness value along one direction and a low stiffness value along another allows the robot to slide on the table evenly while reducing the risk of reaching joint limits (but at the expense of not achieving good tracking along the direction with low stiffness).

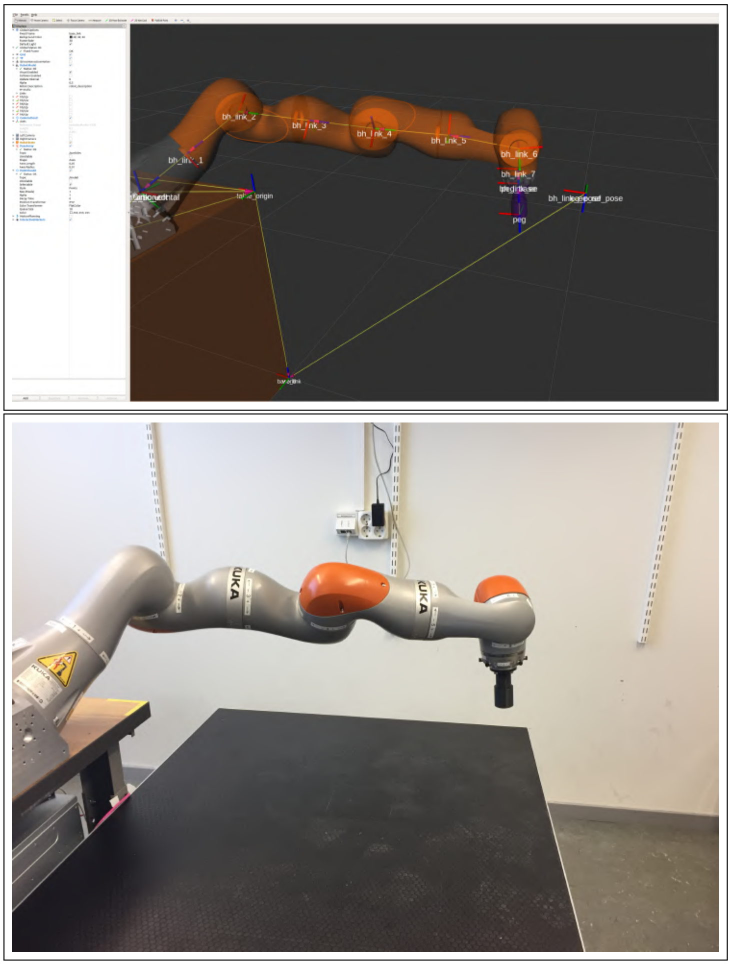


Figure E.3: Commanding out-of-reach pose (the right-most frame on the top figure), driving the robot into a singularity, where joints 3 and 5 are aligned co-linearly.

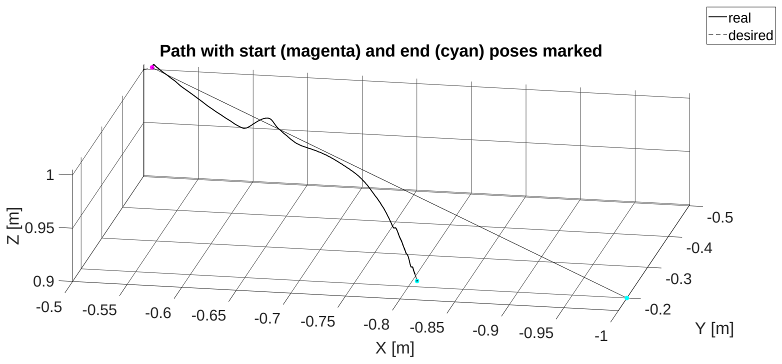


Figure E.4: Path of the end-effector position in 3D space, as the robot is commanded to move the end-effector pose outside of the workspace. By the final pose of the real path, the robot is stretched out.

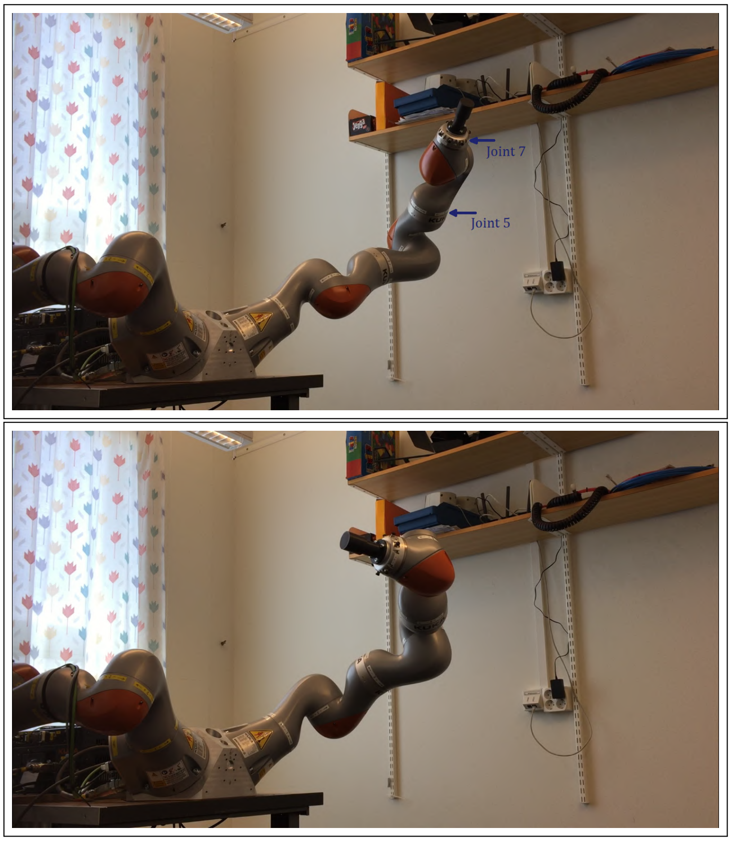


Figure E.5: Robot in a singularity (top picture) with joints 5 and 7 aligned, and robot in a non-singular configuration (bottom picture). Chronologically, the top picture was taken first, and the bottom picture was taken shortly after.

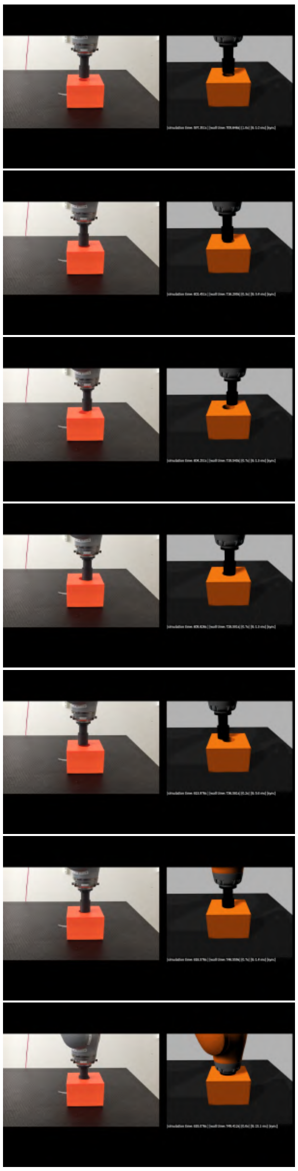


Figure E.6: Peg-in-hole task displayed in sequence. The left and right pictures represent the experiment on the real robot and in simulation, respectively.

Lund University **Department of Automatic Control** **Box 118** **SE-221 00 Lund Sweden**	*Document name* MASTER'S THESIS *Date of issue* November 2021 *Document Number* TFRT-6147
Author(s) Oussama Chouman	*Supervisor* Julian Salt Ducaju, Dept. of Automatic Control, Lund University, Sweden Matthias Mayr, Dept. of Computer Science, Lund University, Sweden Björn Olofsson, Dept. of Automatic Control, Lund University, Sweden Anders Robertsson, Dept. of Automatic Control, Lund University, Sweden (examiner)

Title and subtitle

Compliance of a Robot Arm using Torque-Based Cartesian Impedance Control

Abstract

Robots have become important for the development of today's society. They are known to obtain good accuracy and repeatability of tasks that either complement or replace human labour. Furthermore, as machine-learning is emerging in many technologies, it is important to consider the uncertainties that are introduced when this method is used in robotics. This work aims at developing and evaluating a torque-based control framework that allows a state-of-the-art robot arm to be seen as an impedance, and its environment as an admittance. Using said control strategy allows the robot to behave as a mass-spring-damper system, which allows it to act more compliantly. Moreover, the type of strategy is referred to as Cartesian impedance control.

The framework was developed both in a simulation environment called Dynamic Animation and Robotics Toolkit (DART), but also in the common robotics environment called Robot Operating System (ROS). The latter framework was in turn used to run the control strategy on a real robot arm.

The results showed compliant behaviour of the robot in Cartesian space, both in simulation and on the real system. Translational, rotational and nullspace compliances were tested and evaluated. For reasonably high stiffness values, these experiments showed the expected behaviour when the robot was subjected to external forces. Moreover, the behaviour of the robot in selected singularities was also studied, and the robot was stable in all the tested singular configurations with no apparent oscillations. The experiments also revealed some limitations of the real system, allowing the robot to behave sufficiently well only under certain bounds. That is, the inheritance of friction on the real system limited the tracking ability when lower stiffness values were used. A peg-in-hole assembly experiment was also done both on the real system and in simulation, in order to get a sense of how the robot performs in the intended contact-rich tasks. The results of these experiments showed that the robot could insert the attached peg into a box with a hole, after sliding along the surface of the box for some time. The peg insertion in simulation was about twice as fast compared to the real robot. Moreover, the specific path of the end-effector in simulation compared to the real system did not match precisely due to uncertainties and limitations such as calibration errors and friction.

Keywords		
Classification system and/or index terms (if any)		
Supplementary bibliographical information		
ISSN and key title 0280-5316		*ISBN*
Language English	*Number of pages* 1-83	*Recipient's notes*
Security classification		

http://www.control.lth.se/publications/