KU LEUVEN

FACULTEIT INDUSTRIËLE
INGENIEURSWETENSCHAPPEN

CAMPUS BRUGGE

Development of a compliant controller for a generic 6DOF robot in Python

Maarten DETAILLEUR

Promotor: prof. dr. ir. M. Versteyhe

Begeleider: ing. M. De Ryck

Masterproef ingediend tot het behalen van de graad van master of Science in de industriële wetenschappen: energie , automatisering

Academiejaar 2019 - 2020

KU LEUVEN

FACULTEIT INDUSTRIËLE
INGENIEURSWETENSCHAPPEN

CAMPUS BRUGGE

Development of a compliant controller for a generic 6DOF robot in Python

Maarten DETAILLEUR

Promotor: prof. dr. ir. M. Versteyhe

Begeleider: ing. M. De Ryck

Masterproef ingediend tot het behalen van de graad van master of Science in de industriële wetenschappen: energie , automatisering

Academiejaar 2019 - 2020

Acknowledgements

Throughout this master's thesis, several people have assisted me and helped me to successfully complete this assignment. First I would like to thank my promotor Mr Versteyhe and co-promoter Mr De Ryck for the interesting thesis subject. Further I would like to thank Mr De Ryck once more for the good guidance. He was always available and gave constructive feedback and tips at the two weekly meetings. In addition, I could always reach him with any question by e-mail. I would like to thank my parents for being able to follow these studies and to support me continuously. They also provided the ideal balance between taking my lessons and investing the necessary time in my master thesis. My parents and my sister checked my thesis for language and typing errors for which I am also grateful to them. I am also grateful to all those who have written literature that I have used on this subject. Finally, I would like to thank my girlfriend, the rest of my family and my dearest friends for the mental support.

Abstract

Het hoofddoel van deze thesis is het ontwerpen van een krachtgeregelde controller in Python voor een zes vrijheidsgraden robotarm en het uitwerken van enkele toepassingen met deze controller. Daarvoor moet kennis van kinematica en dynamica van een robotsysteem aanwezig zijn en is kennis van de krachtregeltechnieken noodzakelijk. In deze thesis wordt admittantie krachtregeling gebruikt om een vooraf bepaald traject te volgen. Wanneer een externe kracht optreedt, kan een aanpassing worden gedaan aan het vooraf bepaalde traject aan de hand van een virtuele veer, een demper en een massasysteem. Deze applicatie is ontworpen voor een UR3 CB-series robot. Om de werking van het systeem te testen, werden verschillende parameters, zoals verwachte gewrichtsposities of Cartesiaanse posities, de reële gewrichtsposities of Cartesiaanse posities bijgehouden en uitgezet in functie van de tijd. De krachten die door de omgeving op het systeem worden uitgeoefend, worden vergeleken met de eerder verkregen resultaten. Hierdoor kan aan de hand van controleberekeningen nagegaan worden of het systeem naar de wens van het ontwerp functioneert. Er wordt geen rekening gehouden met wrijving omdat dit niet gekend is, alsook met de inverse dynamica wordt geen rekening gehouden wegens verwaarloosbaar en een te trage werking in realtime processing van Python.

Om deze applicatie te bekomen werden simulaties in MATLAB gemaakt voor toepassingen van zowel twee vrijheidsgraden als zes vrijheidsgraden. Uiteindelijk werd de kennis die hierbij werd verkregen in een Python controller verwerkt waarbij de uiteindelijke applicatie, path tracking with disturbance, werd ontworpen voor de UR3 CB-series robot. Dit is enkel getest in simulatieomgeving vanwege onvoorziene omstandigheden. Als extra werd de dynamica omschreven aan de hand van de Euler-Lagrange methode. Hiervoor werd een algemeen MATLAB script geschreven dat functies genereert in MATLAB die de dynamische parameters van het systeem omvatten, om deze daarna verder te kunnen gebruiken in de Python controller.

Om meer waarde te geven aan deze applicatie werd een tweede applicatie theoretisch benaderd. Het gaat hierbij om een applicatie waarbij een bout in een niet exact geboord gat moet geleid worden aan de hand van de admittance controle die eerder werd gebruikt. Deze applicatie werd niet in praktijk uitgewerkt.

Keywords: admittance controle, Universal Robots, Python, interactieve robot, Matlab

Summary

This thesis is a study within the domain of robotics and control engineering. It is created to serve future research and educational purposes. The goal is to design a force compliant controller for a six degrees of freedom robot through a Python interface. The Universal Robots 3 CB-series collaborative robot is used for this. Two applications have been designed for this purpose: the first application involves path tracking with disturbance and the second application involvles turning in a bolt using force-compliant control. Unfortunately, this second application was only treated in theory.

The thesis contains a gradual rapprochement to the problem starting with an extensive literature study. This literature study contains all the necessary information to understand the theory on this topic, namely: (1) kinematics and dynamics of a robot arm, (2) general position control of joints, (3) compliant force control, (4) trajectory generation, (5) Universal Robots properties, (6) TCP/IP protocol and (7) Python pros and cons. In these sections, the general principles are clarified and reference is often made to literature where a detailed explanation can be found.

The included theory is gradually incorporated into a number of partial experiments. For this purpose MATLAB simulations are used with an interactive GUI to display the first properties of admittance control. The dynamic behavior of an admittance control is presented and worked out in a two degrees of freedom application. First, an application is created in which a fixed target is taken. For this purpose, the dynamic behavior of the manipulator and properties of the admittance control has been examined. Secondly, a variable target is added. For this purpose, it became clear that with admittance control and with extension impedance control, a devitation occurs when a variable target is given. This result has been checked for correctness in SIMULINK. Then the same application is reworked from Cartesian space to joint space to observe the influence of this. As a result of the comparison between Cartesian and joint space, it is decided to develop the following applications in Cartesian space. During these first applications it already has become clear that the calculations of inverse kinematics via iterative methods were too slow. As for the final applications in Python, it was assumed that a frequency of 125Hz would be achieved. This means that slow calculations are not desirable. For this purpose, the various solver parameters that can be set, are discussed in more detail.

After the two degrees of freedom applications have been made and discussed, a six degrees of freedom application is made. The two degrees of freedom application with fixed target is redesigned in MATLAB for six degrees of freedom, using the Peter Corke robotics toolbox. This application leans closer to the first purpose that had to be achieved, namely the path tracking with disturbance application. The general control used is an extension of the two degrees of freedom application and uses the same control technique, i.e. admittance control. The inputs here are: the current Cartesian positions, velocities and the external force acting on the manipulator. In this experiment it has been experienced that several calculations such as the inverse kinematics, are processed too

slowly. No solution has been found for this, so the simulations do run in slow motion, but produce representative real time data.

At this point of the thesis the transition has been made to the Python environment. To start with, it was first foreseen that communication with the Universal Robots would be possible. The communication consists of two classes: (1) a broadcasting class and (2) a monitor class. Later classes have been created and obtained to: (1) get a robot description, (2) generate a trajectory and (3) create a GUI that was needed for simulation purposes.

Due to the COVID-19 pandemia, it was not possible to test the obtained controller in practice. The trajectory generation and the force reading of the robot have been tested in practice, after which there was no longer access to the campus. The controller is further finished for the first application, namely path tracking with disturbance, but is never tested in practice. The application is only tested by means of the URSim simulation environment in a virtual linux machine.

As a result, an externally applied force creates a response in the robot's end-effector position that can be compared to the response of a second order spring, damper and mass system. As this was the goal of this application, it is considered successful. However, for path tracking a PID controller can be built in to eliminate the error during path tracking. As a result, the second order system response will no longer be able to be detected, which is why this is not executed.

Finally, a theoretical approach to the second application, turn in a bolt using force compliant control, is included in the experiments. In addition, an approach of the dynamic parameters is made using the Lagrange equations quoted in the literature study for a two degrees of freedom application, but elaborated for a six degrees of freedom application. This determination cannot be found in this thesis, but can be found on my GitHub page [49].

Contents

List of Figures

List of Tables

List of Symbols

- ${}_sT^b$: Transformation of the b-frame relative to the s-frame
- ${}_sR^b$: Rotation of the b-frame relative to the s-frame
- ${}_sP^b$: Translation of the b-frame relative to the s-frame
- $\theta, \dot{\theta}, \ddot{\theta}$: Joint configurations
- $\theta_d, \dot{\theta}_d, \ddot{\theta}_d$: Desired joint configurations
- $x, \dot{x}, \ddot{x}$: Cartesian configurations
- $x_d, \dot{x}_d, \ddot{x}_d$: Desired Cartesian configurations
- α: Fixed joint angle
- $J(\theta)$: Jacobian matrix
- $\dot{J}(\theta)$: Derivative of the Jacobian matrix
- τ: Joint torques
- τ_{ext}: Joint torques caused by an external force
- F_{tip}: Force at the end-effector
- F_{ext}: The external applied force
- g: Gravitational constant
- L_i: Length of link i
- m_i: Mass of link i
- $\mathcal{L}$: Lagrange total energy
- $\mathcal{K}$: Kinetic energy
- $\mathcal{P}$: Potential engery
- $\mathcal{V}_d$: Desired twist matrix
- $M(\theta)$: The mass matrix in joint space
- $c(\theta, \dot{\theta})$: Coriolis and centripetal torques matrix

- $g(\theta)$: Gravity compensation matrix in joint space
- $b(\dot{\theta})$: Friction force matrix in joint space
- $h(\theta, \dot{\theta})$: Lumped forces matrix in joint space
- $\Lambda(x)$: The mass matrix calculated in Cartesian space
- $\mu(x, \dot{x})$: Coriolis forces matrix in Cartesian space
- $\gamma(x)$: Gravity compensation matrix in Cartesian space
- $\eta(\dot{x})$: Friction force matrix in Cartesian space
- ω_m: Mechanical rotation speed
- ψ: Flux
- U: Input voltage
- T: Torque
- R: Resistor
- E: Elektromotive force
- J: Inertia DC motor
- ω_n: Natural frequency
- ζ: Damping ratio
- D: Mechanical damping DC motor
- K_P: Virtual stiffness parameter
- K_D: Virtual damping parameter
- M: Virtual mass
- τ: Time constant

List of Abbreviations

- DA: Direct Adaptive
- DOF: Degrees Of Freedom
- ERSP: Evolution Robotics Software Platform
- GUI: Graphical User Interface
- IA: Indirect Adaptive
- IOS: Internetwork Operating System
- IT: Information Technology
- ITEA: Integrated Time-multiplied Absolute Error
- LIDAR: LIght Detection And Ranging or Laser Imaging Detection And Ranging
- LTS: Long-Term Support
- MSRDS: Microsoft Robotics Developer Studio
- NAOqiOS: NAO humanoid robot Operation System
- OS: Operating System
- OpenRTM: Open Robotics Technology Middleware
- OPROS: Open Platform for RObotics Services
- OROCOS: Open RObot COntrol Software
- OSI: Open Systems Interconnection
- PC: Personal Computer
- PhD: Doctor of Philosophy
- PID: Proportional Integral Differential
- ROS: Robot Operating System
- UR3: Universal Robots 3 CB-series
- TCP/IP: Transmission Control Protocol/Internet Protocol

1 Introduction

1.1 Motivation

This master thesis was chosen as a continuation of my bachelor thesis which also took place in the robotics field. I fully committed myself and built up the fascination for robotics. Since there was only limited contact with robots in the bachelor's thesis and because there are no courses about robotics in this four-year course, I decided to continue with this subject.

First a master thesis in the industry was considered. However, it was established that the pace of communication did not meet my expectations. My supervisor from my bachelor thesis, Mr De Ryck had prepared some theses for his doctorate together with Mr Versteyhe. Since the topic industry 4.0 is well represented on our campus with 'The Ultimate Factory', I also decided to make my master thesis on the campus. In addition to that, communication with my promoters would certainly go well.

1.2 Purpose of the thesis

The purpose of this thesis was to develop a force compliant controller in Python. Therefore, it was supposed to be able to describe the kinematics and dynamics of a robot arm. In addition, it was expected to get a feeling for Python and to write code myself that causes a force compliance on an UR3 robot. An application for this force compliant controller had to be worked out. As additional application there was chosen to let the robot arm turn in a bolt in a hole that is not drilled in the exact location. Mr De Ryck has experienced experimentally that screwing in a bolt, with a robot arm, is not always going smoothly. That is why this thesis will be attempted on the basis of force compliance.

1.3 Goal of the thesis

The goal of this thesis is to develop and implement a compliant controller in Python for a general six degrees of freedom robot. To enhance the applicability of the controller, two applications were worked out in this thesis: (1) path tracking with a disturbance and (2) screw in a bolt. The first application needs to work when started up, in regime. This last application needs to have a high success ratio, as 6-sigma is often used as a target in industry. A minimum reliability of 4-sigma will be achieved in this thesis. This means that only 0.62% of the actions will go wrong.
Therefore some partial objectives were made. The first goal was to have a robot that can go from A to B and be disrupted along the way. Later, the previously discussed application had to be worked out. Unfortunately due to time constraints this has only been discussed theoretically. The main task was to learn Python and apply knowledge of robot kinematics and dynamics to a compliant controller.

1.4 Thesis structure

In chapter 2, the literature study, a short explanation is given of all subjects that will be used. The subjects include: the kinematics and dynamics of a robot arm, followed by a description of the control techniques and finally it discusses trajectory generation, the specifications of the UR3 robot, pros and cons of Python and the connection through TCP/IP. The method of elaboration is described in chapter 3 and finally an overview of the experiments performed with the corresponding results is given in chapters 4 and 5. Finally, chapter 6 concludes this thesis.

2
Literature study

The aim of the literature study is to incorporate all the necessary knowledge into related chapters. Both the kinematics and dynamics of a robot with multiple degrees of freedom are discussed in section 2.1. In addition the internal joint position control loop is discussed in section 2.2. Hereafter, in section 2.3, the different force control methods are discussed with their properties, followed by the theory of trajectory generation in section 2.4. Finally, some topics are discussed that are necessary to achieve the final goal. In section 2.5 an overview is given of the UR3 properties and sections 2.6 and 2.7 discuss the used communication protocol and the pros and cons of using Python.

2.1 Kinematics and dynamics of a robot arm

2.1.1 Introduction

In this thesis a six degree of freedom robot was used. Mastering the kinematics and dynamics of an arm-type robot was therefore necessary. Two books were used as a basis for this study: 'Robotics, Vision and Control' and 'Modern Robotics' [30] [1]. The book 'Modern Robotics' comes with video lessons on YouTube, suitable for self-study and other educative purposes. In this literature study you can find some of the fundamentals about the kinematics of a robot arm.
The following subsections are a summary of the book. Taking into account the knowledge of specialists, only the most fundamental content is described.

2.1.2 Rigid-body motions

To describe a movement of any rigid-body in a second or third dimensional space, from the linear algebra, it is known that translations and rotations are used. As Figure 2.1 shows in two dimensional space, a vector can be defined to move the s-frame to another location in space. This is called a translation. If a rotation and a translation took place, all coordinates in the s-frame from Figure 2.1 are first multiplied by a rotation matrix and afterwards translated to a new point in space. As a result the b-frame in Figure 2.1 is defined.

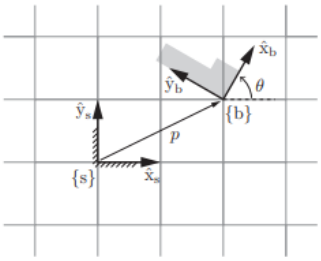


Figure 2.1: Translation and rotation of a frame

In a two dimensional space this could be described by using a column vector with two elements used for the translation $[{}^sP_b]$ and a two by two matrix used for the rotation part $[{}^sR_b]$.

$$ {}^sP_b = \begin{bmatrix} {}^sP_{bx} \\ {}^sP_{by} \end{bmatrix} = \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \tag{2.1.1} $$

$$ {}^sR_b = \begin{bmatrix} {}^sX_{bx} & {}^sY_{bx} \\ {}^sX_{by} & {}^sY_{by} \end{bmatrix} = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix} \tag{2.1.2} $$

If both equations 2.1.1 and 2.1.2 are combined, then equation 2.1.3 applies.

$$ {}^sT_b = {}^s P_b * {}^s R_b = \begin{bmatrix} {}^sX_{bx} & {}^sY_{bx} & 0 \\ {}^sX_{by} & {}^sY_{by} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & {}^sP_{bx} \\ 0 & 1 & {}^sP_{by} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} {}^sX_{bx} & {}^sY_{bx} & {}^sP_{bx} \\ {}^sX_{by} & {}^sY_{by} & {}^sP_{by} \\ 0 & 0 & 1 \end{bmatrix} \tag{2.1.3} $$

In this example, work was done in a plane. Only three variables are required to describe a displacement: the angular displacement and the distance for displacement in x- and y-coordinates

based on the s-frame. However, when this is transferred to a three dimensional space, six variables are required, which means that one translation axis and two rotation axes are added.

$$ {}^{s}T_b = \begin{bmatrix} {}^{s}R_b & {}^{s}P_b \\ 0 & 1 \end{bmatrix} \tag{2.1.4} $$

$$ {}^{s}R_b = \begin{bmatrix} {}^{s}X_{bx} & {}^{s}Y_{bx} & {}^{s}Z_{bx} \\ {}^{s}X_{by} & {}^{s}Y_{by} & {}^{s}Z_{by} \\ {}^{s}X_{bz} & {}^{s}Y_{bz} & {}^{s}Z_{bz} \end{bmatrix} \tag{2.1.5} $$

$$ {}^{s}P_b = \begin{bmatrix} {}^{s}P_{bx} \\ {}^{s}P_{by} \\ {}^{s}P_{bz} \end{bmatrix} \tag{2.1.6} $$

2.1.3 Forward kinematics

In this subsection, forward kinematics are discussed. Forward kinematics in robotics refers to the use of the robot's kinematics to compute the position of the end-effector using the joint positions. In other words, finding a configuration of the base frame relative to the end-effector frame given the joint angles (θ) and the necessary robot kinematics data. Figure 2.2 shows a visual representation of the multiple robot parameters needed for this.

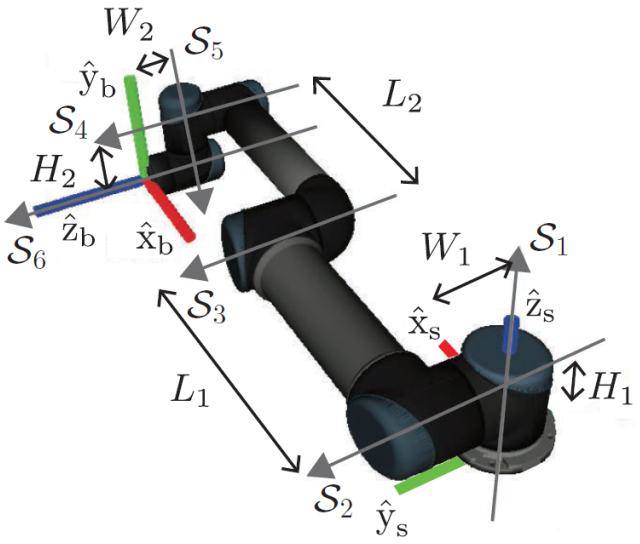


Figure 2.2: UR robot kinematics [1]

As rigid-bodies are discussed in section 2.1.2, each link of a robot is a rigid-body. The different bodies are linked by means of joints. However, this also means that the position of the end-effector can be determined by multiple transformation matrices. Each transformation matrix is built up in the same way as in section 2.1.2 and uses the previous joint as s-frame (begin of a link) and the next joint represents the b-frame (end of the link), this definition is taken from Figure 2.1. When the transformation of the end-effector must be known, this can be represented as the multiplication of all partial transformation matrices, as given in equation 2.1.7.

$$ {}^{0}T_6[\theta_1,\theta_2,\theta_3,\theta_4,\theta_5,\theta_6] = {}^{0}T_1[\theta_1] \cdot {}^{1}T_2[\theta_2] \cdot {}^{2}T_3[\theta_3] \cdot {}^{3}T_4[\theta_4] \cdot {}^{4}T_5[\theta_5] \cdot {}^{5}T_6[\theta_6] \tag{2.1.7} $$

$$ {}^{i-1}T_i = \begin{bmatrix} cos(\theta_i) & -sin(\theta_i) & 0 & a_{i-1} \\ sin(\theta_i)cos(\alpha_{i-1}) & cos(\theta_i)cos(\alpha_{i-1}) & sin(\alpha_{i-1}) & sin(\alpha_{i-1})d_i \\ sin(\theta_i)sin(\alpha_{i-1}) & cos(\theta_i)sin(\alpha_{i-1}) & cos(\alpha_{i-1}) & cos(\alpha_{i-1})d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.1.8} $$

Because it's not always clear how to build up each partial transformation matrix, the Denavit-Hartenberg parameters are used. These DH parameters represent a standardized form of how the kinematics of the robot are built up, depending on the joint positions. For example Table 2.1 shows the DH parameters of the UR3 robot used in this thesis. These standarized parameters ensure that a transformation matrix of a specified link can be easily set up using equation 2.1.8.

Kinematics	θ [rad]	a [m]	d [m]	α [rad]
Joint 1	0	0	0.1519	π/2
Joint 2	0	-0.24365	0	0
Joint 3	0	-0.21325	0	0
Joint 4	0	0	0.11235	π/2
Joint 5	0	0	0.08535	-π/2
Joint 6	0	0	0.0819	0

Table 2.1: DH parameters UR3 robot [28]

As a small example, forward kinematics can be used for a two-link manipulator. In this literature study this is done to get knowledge of the principles. In practice this is used to create a simple force controlled simulation in MATLAB.

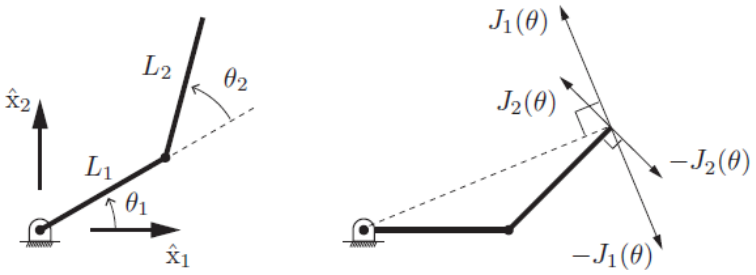


Figure 2.3: Two link manipulator[1]

If the transformation matrices in equations 2.1.9, 2.1.10 and 2.1.11 are multiplied, we obtain the general result of the forward kinematics of the two-link manipulator. Therefore the manipulators DH parameters must be defined. In Table 2.2 these are given.

Kinematics	θ [rad]	a [m]	d [m]	α [rad]
Joint 1	θ_1	0	0	0
Joint 2	θ_2	L_1	0	0
End-effector	0	L_2	0	0

Table 2.2: DH parameters of the two-link manipulator [28]

$$ {}^{0}T_1 = \begin{bmatrix} cos(\theta_1) & -sin(\theta_1) & 0 & 0 \\ sin(\theta_1) & cos(\theta_1) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.1.9} $$

$$ {}^{1}T_2 = \begin{bmatrix} cos(\theta_2) & -sin(\theta_2) & 0 & L_1 \\ sin(\theta_2) & cos(\theta_2) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.1.10} $$

$$ {}^{2}T_{end} = \begin{bmatrix} 1 & 0 & 0 & L_2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.1.11} $$

$$ {}^{0}T_{2} = {}^{0}T_{1}^{1}T_{2}^{2}T_{end} = \begin{bmatrix} cos(\theta_1+\theta_2) & sin(\theta_1+\theta_2) & 0 & L_1cos(\theta_1)+L_2cos(\theta_1+\theta_2) \\ sin(\theta_1+\theta_2) & cos(\theta_1+\theta_2) & 0 & L_1sin(\theta_1)+L_2sin(\theta_1+\theta_2) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.1.12} $$

$$ \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} L_1cos(\theta_1)+L_2cos(\theta_1+\theta_2) \\ L_1sin(\theta_1)+L_2sin(\theta_1+\theta_2) \end{bmatrix} \tag{2.1.13} $$

Equation 2.1.13 is the translation part of the transformation matrix of the two-link manipulator. The other elements of the transformation matrix in equation 2.1.12 are the rotation values of the end-effector. Conclusion, the forward kinematics give us the end-effector coordinates and rotations, given the joint angles.

2.1.4 Inverse kinematics

The forward kinematics were relatively easy to describe on the basis of a few transformation matrices. Completing the inverse is a lot harder due to the nonlinearity. In most cases there are several options to achieve the same configuration of the robot arm. There are two solution methods, the analytic closed-form and the iterative numerical method. The latter is most used because for an analytic closed-form equation we need an exact geometry and the equation has multiple solutions. In the alternative, there is only one solution, when starting from a guess. The Newton-Raphson algorithm is used as an iterative method in this example, but there are many more algorithms as discussed on page 90.

First an explanation of the analytic method is given. Therefore a course document of a PhD student from Aalborg University is used [2]. Besides this course document, a paper about the inverse kinematics of a six degrees of freedom manipulator and the book 'Modern Robotics' are used [31] [1]. As shown in Figure 2.2 there are multiple frames, according to multiple joints. In Figure 2.4 the frames of each joint are represented.

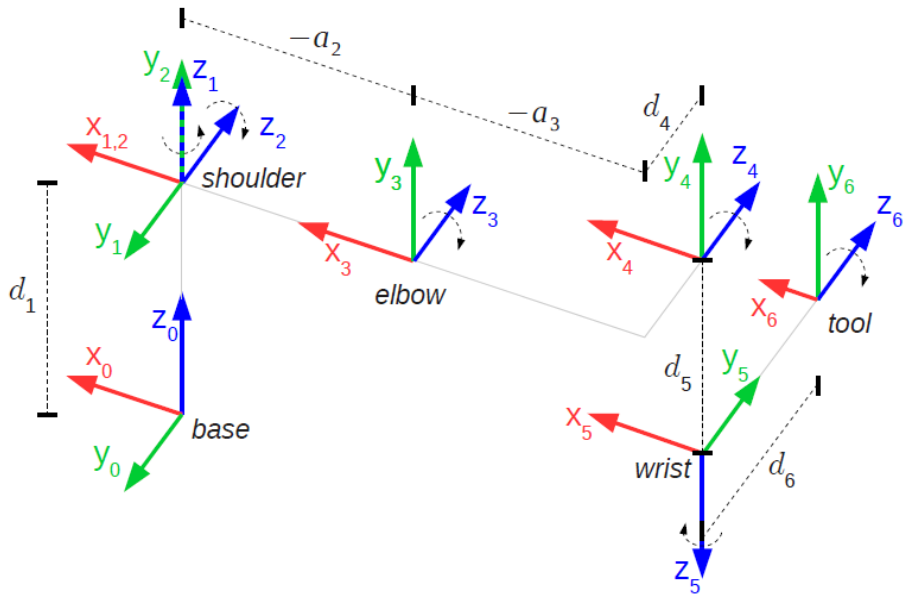


Figure 2.4: Frames of a 6DOF robot arm[2]

For this derivation, a robot from Universal Robots will be used to clear up the method. To start, the location of the fifth frame in relation with the base frame will be determined. In names of Universal Robots, the fifth frame is wrist number two. Based on Figure 2.4 can be written that:

$$ {}^0P_5 = {}^0P6 - d_6 * {}^0Z_6 = {}^0_6T \begin{bmatrix} 0 \\ 0 \\ -d_6 \\ 1 \end{bmatrix} \tag{2.1.14} $$

In words: the position of the wrist frame is a length $-d_6$ in the z-axis of the tool frame, removed from the tool frame.

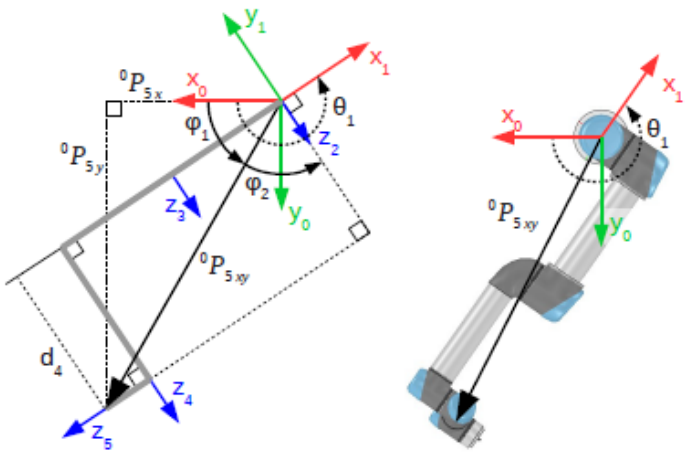


Figure 2.5: θ_1 sketch [2]

In Figure 2.5 a view from above is taken. The following equations can be derived.

$$\theta_1 = \phi_1 + \phi_2 + \frac{\pi}{2} \tag{2.1.15}$$

$$\phi_1 = atan\left(\frac{{}^0P_{5y}}{{}^0P_{5x}}\right) \tag{2.1.16}$$

$$\phi_2 = \pm acos\left(\frac{d_4}{|{}^0P_{5xy}|}\right) = \pm acos\left(\frac{d_4}{\sqrt{{}^0P_{5x}^2 + {}^0P_{5y}^2}}\right) \tag{2.1.17}$$

Substituting 2.1.16 and 2.1.17 in 2.1.15 we obtain equation 2.1.18 for the angle θ_1.

$$\theta_1 = arctan\left(\frac{{}^0P_{5y}}{{}^0P_{5x}}\right) \pm arccos\left(\frac{d_4}{\sqrt{{}^0P_{5x}^2 + {}^0P_{5y}^2}}\right) + \frac{\pi}{2} \tag{2.1.18}$$

After this, a second angle can be derived, namely θ_5. Here too, the top view is used in Figure 2.6 with a derivation following.

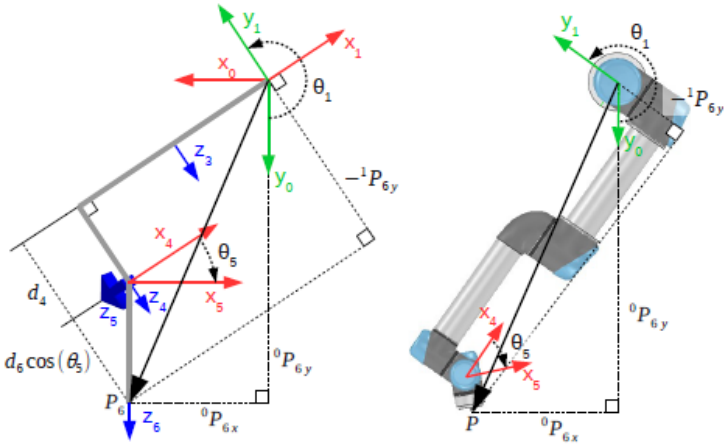


Figure 2.6: θ_5 sketch[2]

$$^1P_{6y} = -(d_4 + d_6 cos(\theta_5)) \tag{2.1.19}$$

In equation 2.1.19, $^1P_{6y}$ can be described by $^0P_{6x}$, $^0P_{6y}$ and the first joint configuration θ_1.

$$^0P_6 = {}^0_1R * {}^1P_6 \Leftrightarrow {}^1P_6 = ({}^0_1R)^\top * {}^0P_6 = \begin{bmatrix} cos(\theta_1) & sin(\theta_1) & 0 \\ -sin(\theta_1) & cos(\theta_1) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} ^0P_{6x} \\ ^0P_{6y} \\ ^0P_{6z} \end{bmatrix} \tag{2.1.20}$$

$$^1P_6 = {}^0P_{6x}(-sin(\theta_1)) + {}^0P_{6y}cos(\theta_1) \tag{2.1.21}$$

If equation 2.1.21 is substituted in equation 2.1.19, equation 2.1.22 is obtained, so that two angles already are described.

$$\theta_5 = \pm arccos\left(\frac{^0P_{6x}sin(\theta_1) - {}^0P_{6y}cos(\theta_1) - d_4}{d_6}\right) \tag{2.1.22}$$

Now θ_5 is defined, θ_6 is next. Therefore y_1 is examined seen from end-effector frame, this is noted as: $^6Y_1$. This axis will always be parallel to the z-axis from frame two, three and four. According to this, it turns out that $^6Y_1$ can in fact be described using spherical coordinates, where azimuth is $-\theta_6$ and the polar angle is θ_5, see Figure 2.7.

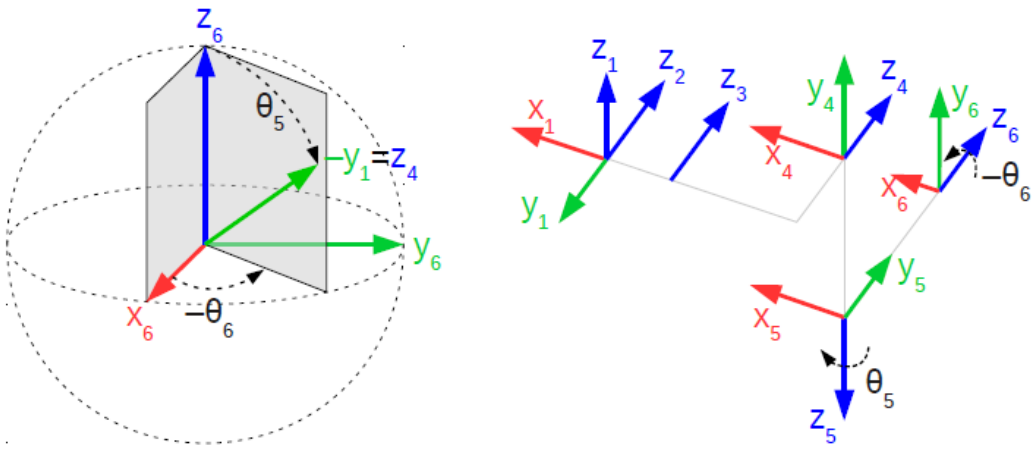


Figure 2.7: θ_6 sketch [2]

Converting from spherical space in Figure 2.7 to Cartesian coordinates gives equation 2.1.23 .

$$ {}^{6}Y_1 = \begin{bmatrix} -sin(\theta_5)cos(\theta_6) \\ sin(\theta_5)sin(\theta_6) \\ -cos(\theta_5) \end{bmatrix} \tag{2.1.23} $$

From equation 2.1.23, θ_6 could be determined in relation to the transformation matrix between the frame number one and the end-effector frame. To get an expression of θ_6 all the way from the base, ${}^{6}Y_1$ can be determined in another way using rotation of the base frame.

$$ {}^{6}Y_1 = \begin{bmatrix} -{}^{6}X_{0x}sin(\theta_1) + {}^{6}Y_{0x}cos(\theta_1) \\ -{}^{6}X_{0y}sin(\theta_1) + {}^{6}Y_{0y}cos(\theta_1) \\ -{}^{6}X_{0z}sin(\theta_1) + {}^{6}Y_{0z}cos(\theta_1) \end{bmatrix} \tag{2.1.24} $$

If equation 2.1.23 is equated with equation 2.1.24 we obtain equation 2.1.25 . Here, by means of substitution, the angle θ_6 which is given in equation 2.1.25 can be obtained.

$$ \theta_6 = arctan\left(\frac{-{}^{6}X_{0y}sin(\theta_1) + {}^{6}Y_{0y}cos(\theta_1)}{sin(\theta_5)} / \frac{{}^{6}X_{0y}sin(\theta_1) - {}^{6}Y_{0y}cos(\theta_1)}{sin(\theta_5)} \right) \tag{2.1.25} $$

The last three joint angles are easier to obtain. Therefore the side view in Figure 2.8 is used.

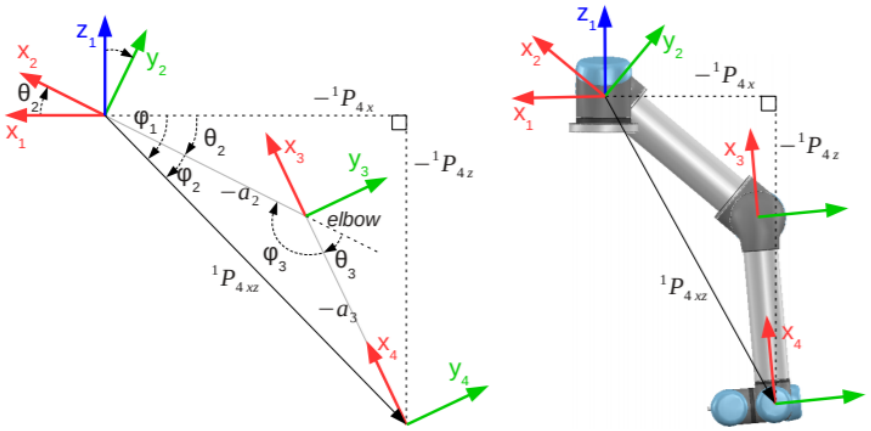


Figure 2.8: Sketch to determine $\theta_3, \theta_2$and θ_4 [2]

To get the third joint configuration, we look at the transformation matrix between frame one and four. On Figure 2.8 the angle ϕ_3 is defined as $\pi - \theta_3$. In the same figure it can be stated that equation 2.1.26 holds.

$$cos(\theta_3) = -\frac{a_2^2 + a_3^2 - \left|{}^1P_{4xz}\right|^2}{2a_2a_3} \tag{2.1.26}$$

$$\theta_3 = \pm acos\left(\frac{\left|{}^1P_{4xz}\right| - {}^2\, a_2^2 - a_3^2}{2a_2a_3}\right) \tag{2.1.27}$$

The angle θ_2 can be found as $\phi_1 - \phi_2$. Each of these can be found by inspecting Figure 2.8 and using tangent and sine relations.

$$\phi_1 = atan\left(\frac{{}^1P_{4z}}{{}^1P_{4x}}\right) \tag{2.1.28}$$

$$\phi_2 = asin\left(\frac{-a_3 sin(\phi_3)}{\left|{}^1P_{4xz}\right|}\right) \tag{2.1.29}$$

$$\theta_2 = \phi_1 - \phi_2 = atan\left(\frac{{}^1P_{4z}}{{}^1P_{4x}}\right) - asin\left(\frac{-a_3 sin(\phi_3)}{\left|{}^1P_{4xz}\right|}\right) \tag{2.1.30}$$

Finally, the last remaining joint angle to determine is θ_4. Due to the definition of the measured joint angle in Figure 2.4. The angle between the two frames is equal to the angle between X_{i-1} and X_i. It can be deduced that:

$$\theta_4 = atan\left(\frac{{}^3X_{4y}}{{}^3X_{4x}}\right) \tag{2.1.31}$$

To sum up, a total of nine solutions exist: two in the first joint, one in the second joint, two in the third joint, one in the fourth joint, two in the fifth joint and one in the sixth joint.

In general, this must be checked for each robot to ensure that the method is correct, based on the kinematic configuration. If an initial state is given, it can be chosen to, for example, take the closest joint angle and neglect the other possible angles. As a result, a certain solution can be obtained quickly.

Secondly, the interative method is explained. In the previous approach multiple solutions occur. The iterative method is nearly always based on a guess of the current situation. For this guess the closest new solution will be sought for the various joint angles. There are many algorithms that usually use the Newton-Raphson method or a least square method as a basis. In this part of the literature study, the Newton-Raphson method will be discussed, based on the book 'Modern Robotics' form the University of Cambridge [1]. The general idea is to guess the position and use the derivative to calculate a more accurate solution. Then the new solution is used as new guess to start over until a certain precision is conducted. This is shown in Figure 2.9. Equation 2.1.32 gives a formula that is built up using a first order Taylor expansion, to approximate a cost function g.

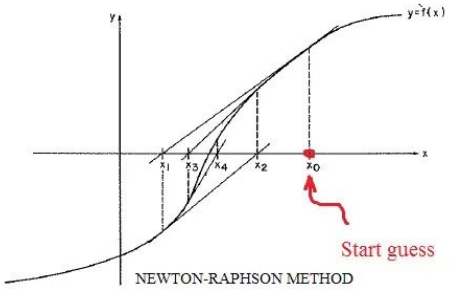


Figure 2.9: Outline of the working principle of the Newton-Raphson method [3]

$$g(\theta) = g(\theta^0) + \frac{\partial g}{\partial \theta}(\theta^0)(\theta - \theta^0) \tag{2.1.32}$$

In order to get a better guess the formula can be written as:

$$\theta^{k+1} = \theta^k - \left(\frac{\partial g}{\partial \theta}(\theta^k)\right)^{-1} g(\theta^k) \tag{2.1.33}$$

A two degrees of freedom example is worked out as an example. In Figure 2.10 the example is shown.

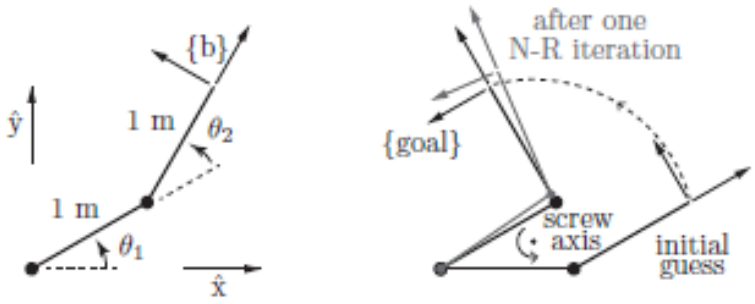


Figure 2.10: Newton-Raphson method [1]

The transformation matrix with angles $\theta_1 = 30^\circ$ and $\theta_2 = 90^\circ$ is shown in 2.1.34.

$$ {}_sT^b = \begin{bmatrix} -0.5 & -0.866 & 0 & 0.366 \\ 0.866 & -0.5 & 0 & 1.366 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.1.34}$$

In Table 2.3 an example is worked out. It starts with a guess that $\theta_1 = 0°$ and $\theta_2 = 30°$ [1]. After three iterations the true value is reached. For more degrees of freedom the example would be more complex.

i	(θ_1, θ_2)	(x, y)	$\mathcal{V}_b = (\omega_{zb}, v_{xb}, v_{yb})$	$\|\omega_b\|$	$\|v_b\|$
0	$(0.00, 30.00°)$	$(1.866, 0.500)$	$(1.571, 0.498, 1.858)$	1.571	1.924
1	$(34.23°, 79.18°)$	$(0.429, 1.480)$	$(0.115, -0.074, 0.108)$	0.115	0.131
2	$(29.98°, 90.22°)$	$(0.363, 1.364)$	$(-0.004, 0.000, -0.004)$	0.004	0.004
3	$(30.00°, 90.00°)$	$(0.366, 1.366)$	$(0.000, 0.000, 0.000)$	0.000	0.000

Table 2.3: Example Newton-Raphson method two degrees of freedom [1]

Looking at the iteration values of the joint angles, it can be said that they have an expected course. In the first iteration, too great a value is taken for θ_1 and the more iterations took place, the less deviation from the final value is obtained. The same can be said about all other variables in Table 2.3.

To apply this in practice, the algorithm to calculate inverse kinematics was taken from the robopy Python library and the Robotics toolbox in Matlab from Peter Corke [32]. Both toolboxes have different methods to achieve an end result. It has been experimentally investigated which algorithm is used for which application.

2.1.5 Velocity kinematics

In this chapter we will take a look at the velocity kinematics. In comparison to the forward kinematics, the velocity of the end-effector can be described in function of the joint velocities.

$$\dot{x} = J(\theta)\dot{\theta} \tag{2.1.35}$$

$J(\theta)$ is called the Jacobian and is the derivative of the transformation matrix T. This matrix presents the linear sensitivity of the end-effector velocity to the joint velocity. In Figure 2.11 a two degrees of freedom example is shown. In equations 2.1.36 and 2.1.37 the kinematics are given. As we want to obtain velocities, the kinematic equations derived in section 2.1.3 can be derived. The matrix we obtain is the Jacobian which is dependent on the configuration of the manipulator. In equation 2.1.38 the Jacobian is therefore processed in formula form for the two-link example of Figure 2.11.

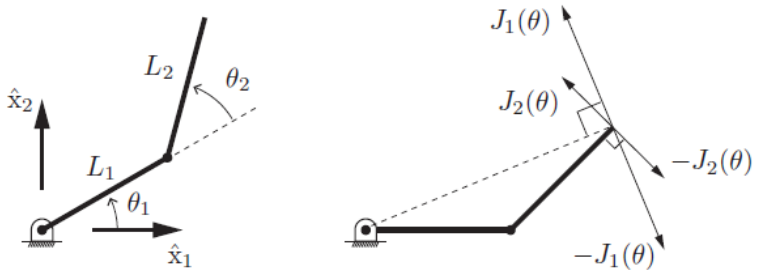


Figure 2.11: Two-link example [1]

$$x_1 = L_1 cos\theta_1 + L_2 cos(\theta_1 + \theta_2) \tag{2.1.36}$$

$$x_2 = L_1 sin\theta_1 + L_2 sin(\theta_1 + \theta_2) \tag{2.1.37}$$

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} -L_1 sin\theta_1 - L_2 sin(\theta_1 + \theta_2) & -L_2 sin(\theta_1 + \theta_2) \\ L_1 cos\theta_1 + L_2 cos(\theta_1 + \theta_2) & L_2 cos(\theta_1 + \theta_2) \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix} \tag{2.1.38}$$

This can be further expanded to a six degree of freedom example. This is not fully discussed in this literature study because of the complexity. For more information I would like to refer to a paper that has been made by Reza Yazdanpanah A. on this subject [4]. As an example of the geometry of the space Jacobian, a 6x6 matrix as shown in Figure 2.12. In general the Jacobian is a 6xn matrix, where n is the number of degrees of freedom.

$$\begin{bmatrix} \omega_{s1x} & \omega_{s2x} & \omega_{s3x} & \omega_{s4x} & \omega_{s5x} & \omega_{s6x} \\ \omega_{s1y} & \omega_{s2y} & \omega_{s3y} & \omega_{s4y} & \omega_{s5y} & \omega_{s6y} \\ \omega_{s1z} & \omega_{s2z} & \omega_{s3z} & \omega_{s4z} & \omega_{s5z} & \omega_{s6z} \\ \omega_{s1y}q_{1z} & \omega_{s2y}q_{2z} & \omega_{s3y}q_{3z} & \omega_{s4y}q_{4z} & \omega_{s5y}q_{5z} & \omega_{s6y}q_{6z} \\ -\omega_{s1x}q_{1z} & -\omega_{s2x}q_{2z} & -\omega_{s3x}q_{3z} & -\omega_{s4x}q_{4z} & -\omega_{s5x}q_{5z} & -\omega_{s6x}q_{6z} \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 2.12: Six revolute joint space Jacobian [4]

An other type of Jacobian, the body Jacobian corresponds to a screw axis expressed in the end-effector frame. In this thesis the space Jacobian will be used. As earlier described, the inverse kinematics could be done with a Newton-Raphson algoritm. If the desired twist and the Jacobian are described in the same frame the general solution could be written as:

$$\dot{\theta} = J^{\dagger}(\theta)\mathcal{V}_d \tag{2.1.39}$$

In equation 2.1.39, $J^{\dagger}(\theta)$ is the pseudoinverse matrix of $J(\theta)$ and $\mathcal{V}_d$ is the desired twist or velocity matrix. The pseudoinverse Jacobian is used, because the Jacobian is not square or it's singular so J^{-1} does not exist. Because the inverse does not exist, the Moore-Penrose pseudoinverse approach is used. In equation 2.1.40 a general way to find the pseudoinverse is given, when the Jacobian has more rows than columns or when there are singularities [1].

$$J^{\dagger} = (J^T J)^{-1} J^T \tag{2.1.40}$$

$$J^{\dagger} = J^T (J^T J)^{-1} \tag{2.1.41}$$

2.1.6 Dynamics of open chains

In the previous sections, we went through the kinematics of the system. In this chapter the general dynamics are delineated. To describe these dynamics, a second order differential equation is used, this is shown in 2.1.42.

$$\tau = M(\theta)\ddot{\theta} + c(\theta,\dot{\theta}) + g(\theta) + b(\dot{\theta}) + J^T(\theta)F_{tip} \tag{2.1.42}$$

In equation 2.1.42, τ is the vector of joint torques, $M(\theta)$ is the mass matrix, which includes each link's inertia. The torques that lump together cetripetal and Coriolis term are given by $c(\theta,\dot{\theta})$, while $g(\theta)$ is included to compensate the gravity force on each link. The viscous friction torques are described by $b(\dot{\theta})$ and $J^T(\theta)F_{tip}$ are the additional torques due to an external force on the end-effector.

In general the viscous friction term is neglected due to ignorance, because certain specifications such as the inertia and viscous friction of the joints are usually not released by the manufacturer. For the other terms in this equation, a list was made to clarify the specific properties of each term based on a PowerPoint of a lecture given by Dr Surya Singh [33].

- $M(\theta)$: The mass matrix could be determined using the Jacobian. By recalculating the Jacobian to both translations and rotations, each in a separate Jacobian, respectively J_{vi} and $J_{\omega i}$. In a six degree of freedom, these two Jacobians are determined for each link and have three rows and six columns. In total the mass matrix is calculated using equation 2.1.43, where m_i is the mass of a specified link and I_{C_i} contains all inertias of a link in a 6x6 matrix. The mass matrix is symmetric and positive definite [33].

$$M(\theta) = \sum_{n=1}^{\infty} (m_i J_{vi}^T(\theta) J_{vi}(\theta) + J_{\omega_i}(\theta) I_{C_i} J_{\omega_i}^T(\theta)) \tag{2.1.43}$$

- $c(\theta,\dot{\theta})$: The Coriolis and centripetal component can also be written as $C(\theta,\dot{\theta})\dot{\theta}$. The off-diagonal terms of the Coriolis matrix $C(\theta,\dot{\theta})$, $C_{i,j}$ represent coupling of joint j velocity to the generalized force action on joint i and is just like the mass matrix a square matrix [30].

- $g(\theta)$: The gravity matrix takes the masses of the different links into account, in order to obtain torque compensation. These torques are calculated on the center of gravity of each joint.

- $J^T F_{tip}$: To confirm whether this expression is indeed correct, we can make a derivation:

$$\delta X * F = \delta\theta * \tau \tag{2.1.44}$$
$$F^T * \delta X = \tau^T * \delta\theta \tag{2.1.45}$$
$$\delta X = J * \delta\theta \tag{2.1.46}$$
$$F^T * J = \tau^T \tag{2.1.47}$$
$$J^T F = \tau \tag{2.1.48}$$

  From this we can conclude that the Jacobian can also be used to convert the force on the end-effector to joint torques or reverse.

To determine these dynamic parameters I refer to chapter eight in the book 'Modern Robotics' [1]. Two options are discussed here. First, a general equation is drawn up using the Lagrage equation, given in equation 2.1.49. The aim of this comparison is to primarily determine the torques and to derive the various parameters from the torque equations. Secondly, Newton-Euler's method is explained, in which all parameters can be determined in a closed form.

$$\tau_i = \frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{\theta}_i} - \frac{\partial \mathcal{L}}{\partial \theta_i} \tag{2.1.49}$$

According to the Lagrangian formulation, each system can be defined as in equation 2.1.50. This is applied as an example shown in Figure 2.13 [1].

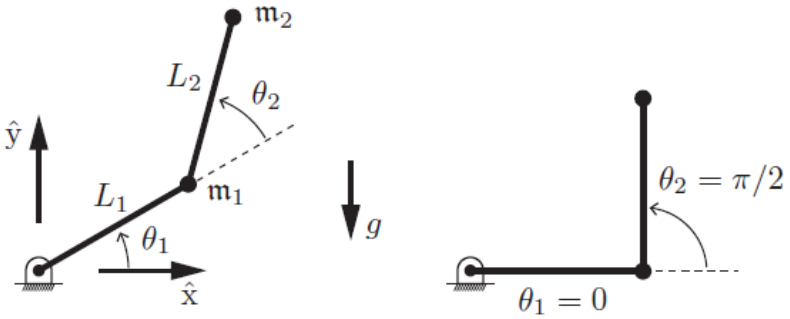


Figure 2.13: 2DOF example [1]

$$\mathcal{L}(x,\dot{x}) = \mathcal{K}(x,\dot{x}) - \mathcal{P}(x) \tag{2.1.50}$$

In equation 2.1.50, $\mathcal{K}(x,\dot{x})$ stands for the overall system's kinetic energy and $\mathcal{P}(x)$ stands for the overall potential energy. If Newton's second law is applied, we get equation 2.1.51

$$\mathcal{L}(x,\dot{x}) = \mathcal{K}(x,\dot{x}) - \mathcal{P}(x) = \frac{1}{2}m\dot{x}^2 - mgx \tag{2.1.51}$$

First, for each link of the two-DOF example, the kinetic energy parameter can be obtained, using the following equations.

$$\mathcal{K}_1 = \frac{1}{2}m_1(\dot{x}_1^2 + \dot{y}_1^2) = \frac{1}{2}m_1 L_1^2 \dot{\theta}_1^2 \tag{2.1.52}$$

$$\mathcal{K}_2 = \frac{1}{2}m_2(\dot{x}_2^2 + \dot{y}_2^2) = \frac{1}{2}m_2\left((L_1^2 + 2L_1L_2cos\theta_2 + L_2^2)\dot{\theta}_1^2 + 2(L_2^2 + L_1L_2cos\theta_2)\dot{\theta}_1\dot{\theta}_2 + L_2^2\dot{\theta}_2^2\right) \tag{2.1.53}$$

Secondly, the potential energy of each link can be obtained.

$$\mathcal{P}(x)_1 = m_1 g y_1 = m_1 g L_1 sin\theta_1 \tag{2.1.54}$$

$$\mathcal{P}(x)_2 = m_2 g y_2 = m_2 g(L_1 sin\theta_1 + L_2 sin(\theta_1 + \theta_2)) \tag{2.1.55}$$

From the Euler-Lagrange equation in 2.1.49, the partial torques can be defined for this two degrees of freedom example.

$$\begin{aligned}\tau_1 = (m_1 L_1^2 + m_2(L_1^2 + 2L_1 L_2 cos\theta_2 + L_2^2))\ddot{\theta} + m_2(L_1 L_2 cos\theta_2 + L_{)}^2)\ddot{\theta}_2 \\ -m_2 L_1 L_2 sin\theta_2(2\dot{\theta}_1\dot{\theta}_2 + \dot{\theta}_2^2) + (m_1 + m_2)L_1 g cos\theta_1 + m_2 g L_2 cos(\theta_1 + \theta_2)\end{aligned} \tag{2.1.56}$$

$$\tau_2 = m_2(L_1 L_2 cos\theta_2 + L_2^2)\ddot{\theta}_1 + m_2 L_2^2\ddot{\theta}_2 + m_2 L_1 L_2\dot{\theta}_1^2 sin\theta_2 + m_2 g L_2 cos(\theta_1 + \theta_2) \tag{2.1.57}$$

These torque equations can then be split up into the form of the general equation 2.1.42. The different parameters can be extracted from this, by isolating the angular acceleration, velocities and positions.

$$M(\theta) = \begin{bmatrix} m_1 L_1^2 + m_2(L_1^2 + 2L_1 L_2 cos\theta_2 + L_2^2) & m_2(L_1 L_2 cos\theta_2 + L_2^2) \\ m_2(L_1 L_2 cos\theta_2 + L_2^2) & m_2 L_2^2 \end{bmatrix} \tag{2.1.58}$$

$$c(\theta, \dot{\theta}) = \begin{bmatrix} -m_2 L_1 L_2 sin\theta_2(2\dot{\theta_1}\dot{\theta}_2 + \dot{\theta}_2^2) \\ m_2 L_1 L_2\dot{\theta}_1^2 sin\theta_2 \end{bmatrix} \tag{2.1.59}$$

$$g(\theta) = \begin{bmatrix} (m_1 + m_2)L_1 g cos\theta_1 + m_2 g L_2 cos(\theta_1 + \theta_2) \\ m_2 g L_2 cos(\theta_1 + \theta_2) \end{bmatrix} \tag{2.1.60}$$

This chapter cited how the dynamic model of the robot can be build up. Due to the complexity of, for example, the mass matrix and the Coriolis and centripetal torques, these have not been worked out in detail for a six degree of freedom example. During the experiments, the Peter Corke toolbox could be used in MATLAB. In the end, there was no need to determine the dynamic parameters in Python. Why these are neglected, will be discussed in section 2.3.5.

2.2 General position control of joints

In the previous section, we mainly discussed how the kinematics of a robot are described. It was also explained how the dynamic equations in a manipulator with multiple degrees of freedom are determined. This section focuses on how the positions that are ultimately obtained are processed by the internal robot controller. This section mainly uses knowledge gained in a course from Mr Vanoost on electrical drives and control techniques [6]. In general the dynamic response of a DC motor can be described by using multiple equations based on Figure 2.14:

$$EMF = k * \psi_0 * \omega_m \tag{2.2.1}$$

$$T_{el} = k * \psi_0 * I_a \tag{2.2.2}$$

$$U_a = R_a * I_a + L_a \frac{d\omega_m}{dt} + E \tag{2.2.3}$$

$$T_m = T_{el} - T_L = J \frac{d\omega_m}{dt} + D * \omega_m \tag{2.2.4}$$

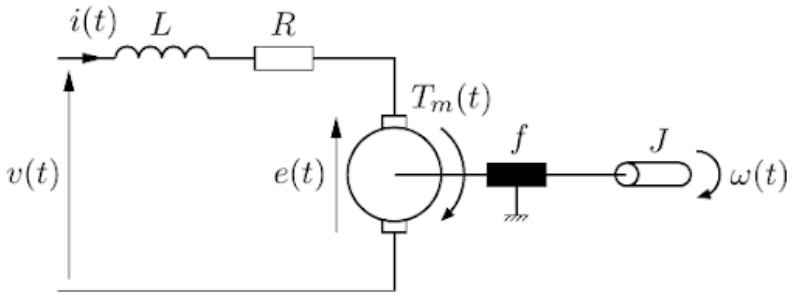


Figure 2.14: DC electrical scheme [5]

In Figure 2.15, the general second order system is shown. To get a certain rotational speed of the DC motor a certain input voltage must be given.

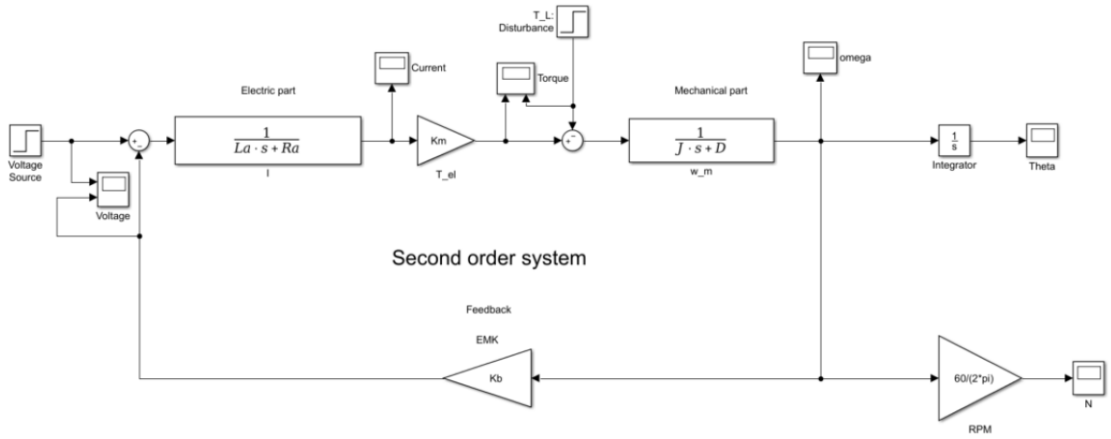


Figure 2.15: DC motor system [6]

This previous figure can be converted to a current control system by using the power converter, corresponding delay τ_s. In Figure 2.16 is shown which system blocks must be added to get a current controller with EMF compensation.

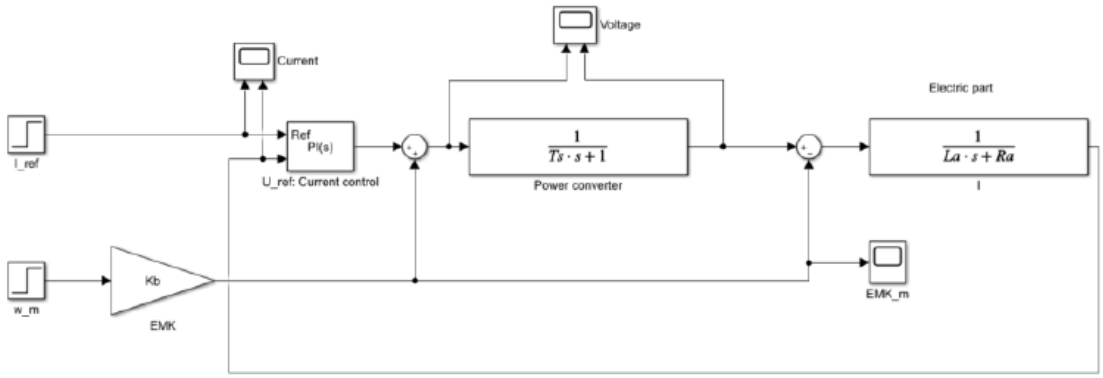


Figure 2.16: Current controller [6]

In this controller, K_i equals $\frac{R_a \tau_a}{2\tau_s}$ and $\tau_a = \tau_i = \frac{L_a}{R_a}$. The derivation of these parameters can be found in a book named 'Electrical Drives and Control Techniques' written by Gerd Terörde [34]. To control the torque an extra gain must be applied on the input step, equal to $\frac{1}{k\psi_0}$, due to equation 2.2.2.

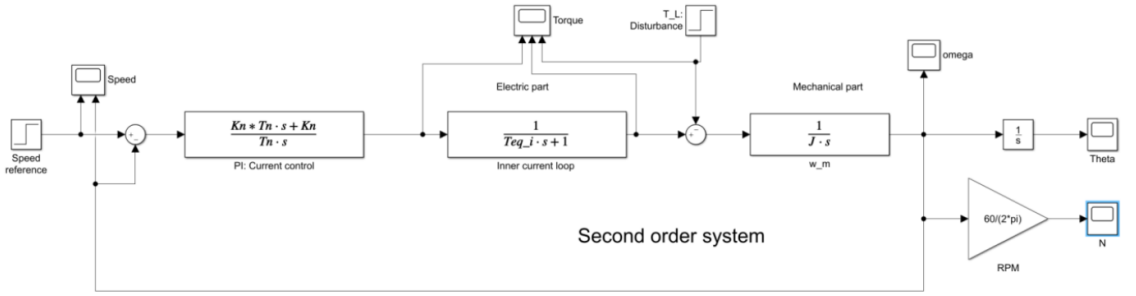


Figure 2.17: Speed controller [6]

In Figure 2.17, the current controller is simplified from a second order system to a first order system with $\tau_{eq_i} = 2\sqrt{2}\tau_s$. For the derivation I would also like to refer to the book about electrical drives again [34]. In the PI controller does $\tau_n = 20\tau_{eq_i}$ and $K_n = \frac{J}{\sqrt{\tau_{eq_i}\tau_n}}$ [6].

Finally, a position controller was added using a P controller. Therefore the speed control system is also simplified to a first order system, for determining the position controllers gain coefficient.

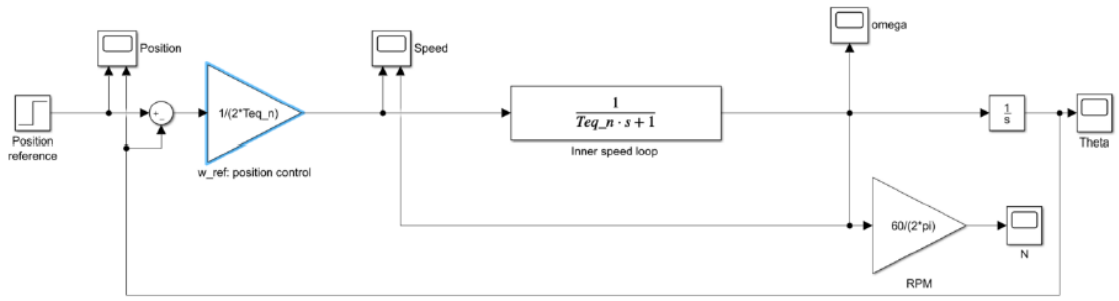


Figure 2.18: Position controller [6]

The new time constant τ_{eq_n} equals $2\sqrt{2}\tau_{eq_i}$ and is used to determine the gain of the P controller that's added for the position control, $K_\theta = \frac{2}{\tau_{eqn}}$ [34]. This derivation was made to get an overview of the internal operation of DC motors located in the robot.

2.3 Compliant force control

2.3.1 Introduction

The control technolgy to produce a compliant motion is called compliant control. Historically compliant motion has been defined as: ”Any robot motion where the end-effector trajectory is changed or even generated based on online sensor information”. Today the interest in compliant control is related to approaches where the controller forms the mechanical impedance of the robot system. In other words compliant control is related to dynamic relationships between robot position or velocity and external forces. By shaping the mechanical impedance, the unknown environment could be safely handled because, instead of controlling a position or force, the force transmitted to the environment through the energetic pair force-velocity is controlled [10].

At this point in my research some articles were found with an overview of compliant force control techniques [13] [10]. Furthermore, matters such as: the obtained response by the compliant force controller, definitions in the field of compliant force control, an overview of compliant force control laws and the implementation choice for this thesis are discussed in this chapter.

General response in compliant force control

The response required in compliant force control is described using a second order system. The best known second order system is a spring, damper and mass system. First, the general equation 2.3.1 is applied, to describe the dynamics of the spring, damper and mass system.

$$F = M\frac{d^2x(t)}{dt^2} + D\frac{dx(t)}{dt} + Kx(t) \tag{2.3.1}$$

Secondly the transition is made to the Laplace domain in equation 2.3.2.

$$F(s) = KY(s) = MX(s)s^2 + DX(s)s + KX(s) \tag{2.3.2}$$

Finally, the equation was transformed into a transfer function in equation 2.3.3.

$$\frac{X(s)}{Y(s)} = \frac{K}{Ms^2 + Ds + K} \tag{2.3.3}$$

In this mechanical system, two parameters are often considered, namely the natural frequency (ω_n) and the damping coefficient (ζ). The natural frequency indicates at what frequency the system will oscillate when no damper is applied. The damping is described by a number greater than zero and has an influence on the course of the response. For example, a damping coefficient of zero ensures osscillations with a step response. A damping of, for example, two ensures a gradual progression to a desired value. As the definitions say: a damping coefficient of zero provides an undamped system, a damping coefficient between zero and one provides an underdamped system, if ζ equals one there is a critically damped system and a damping coefficient greater than one, the system is overdamped.

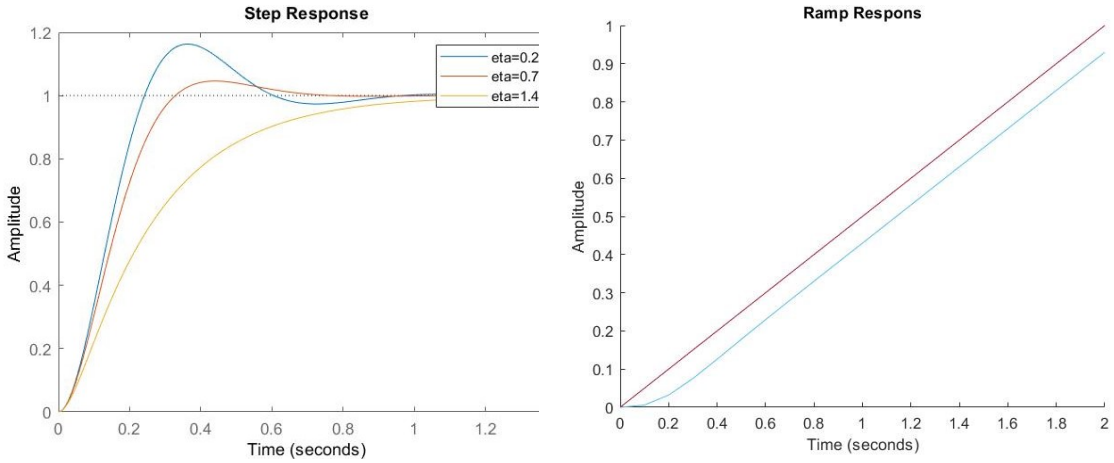
$$\frac{O(s)}{I(s)} = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} \tag{2.3.4}$$

In comparison with the general second order system given in equation 2.3.4, the natural frequency (ω_n) en damping coefficient (ζ) can be written as:

$$\omega_n = \sqrt{\frac{K}{M}} \tag{2.3.5}$$

$$\zeta = \frac{D}{2M\omega_n} \tag{2.3.6}$$

Some parameters were chosen as an example for the response, namely the spring equal to 500N/m and the mass equal to 5kg. No oscillatory behaviour is desired, therefore a non-zero damping coefficient is necessary. Figure 2.19a shows the step response for a damping coefficient of 0.2, 0.7 and 1.4. According to the ITEA (Integrated Time-multiplied Absolute Error) criterion yields the optimum damping of second order systems at a fixed natural frequency: $\zeta = \frac{1}{\sqrt{2}} = \pm 0.7$ [34].



(a) Step response with varying damping coefficient

(b) Ramp response

The goal of this thesis is not to produce and respond to a step. As a result, a ramp was taken to simulate a continuous changing desired position in Figure 2.19b.

As a conclusion, the response of a sudden force impact to the system causes the system to respond the way we want it to. However, there is always a steady state error on the response obtained, if a continuous displacement occurs. If a less steep ramp is chosen, the error is also smaller.

2.3.2 Passive versus active compliance

In order to get started with the multiple control laws, some definitions are given. Starting with a comparison between passive and active compliance. This comparison is especially about whether a system must be controlled by another system or whether it will be controlled by nature.

Passive control

A first example of passive control is the use of soft joints as in Figure 2.20. Soft joints are joints which are flexible and elastic, such as found in the human body. They already have some elasticity and damping so they are rather been controlled by nature, a natural compliance. The opposite, stiff joints, doesn't have this natural compliance. They need to be controlled in order to respond like a spring damper system. Typical examples are arm type robots, delta robots and humanoids [10].

Figure 2.20: Soft joint examples [7] [8]

A second example of passive control is the backdrivability of interactive transmissions. In other words, external forces can be pushed back to the actuator if the control force is smaller than the applied external force without breaking. Backdrivability provide actuators with high force sensitivity and high impact resistance which adapts to quick external force, mechanically [35].

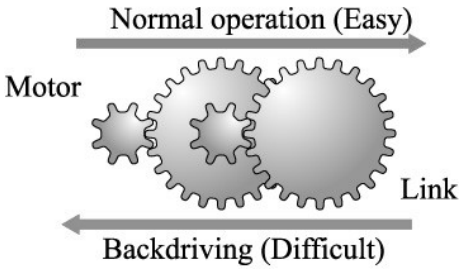


Figure 2.21: Backdrivability [9]

Active control

A more used and studied method is the active controlling of the desired parameters. This is done by measuring external parameters. It means that, for example, the environment can be scanned with a LIDAR and adjustments are made to the operations the robot will perform. In the example of this thesis, active compliance is used by measuring an external force. The previously discussed external parameters then become damping and stiffness of the system. In the remainder of the text, a few of the techniques are discussed that use this. In table 2.4 an overview of pros and cons of passive and active compliance is shown.

Active compliance	**Passive compliance**
Mainly software achievable.	Mainly hardware achievable.
Easy to regulate and compute and can be of general use.	Hard to regulate, compute and usually dedicated to particular applications.
Compliance centre can be moved easily.	Compliance centre is usually fixed.
Dynamic compliance.	Static/quasi-static compliance.
Easy to incorporate the system dynamics and force feedback information into the system so that the mechanical impedance of the robot end-effector can be controlled; possible to achieve negative compliance in the principal axis.	Hard but possible to consider the system dynamics by special design with stiffness, damping and inertia factors taken into account; only the compliance matrix with positive diagonal elements can be achieved.
Suffering from the kinematic singularity.	Not applicable.
Because the transformation Jacobian matrix is involved in the position and force transformation between the joint frames and end-effector frame, force and torque control are difficult in certain postures.	
Instability is often observed in active compliance system. Careful attention is needed.	Guaranteed overall stability due to its passive nature.
For position control, there is an upper limit on the desired stiffness to avoid the oscillation or instability.	Passive compliance can be set with a very high stiffness.
Relatively expensive in most applications.	Relatively cheap in most applications.
Limited response rate.	Fast response rate.
For any digital control system, the sampling rate determines the dynamic response of the active compliance and cannot be too fast. The response rate also depends on the control law used.	Passive compliance is inherent in the structure so can respond fast.

Table 2.4: A comparison between active and passive compliances [29]

2.3.3 Direct versus indirect force control

Secondly, the difference between direct force control and indirect force control is considered. A description is given on the basis of the definition of the respective force control method.

Direct force control

In active compliance a classification is made between indirect and direct force control, shown in Figure 2.22.

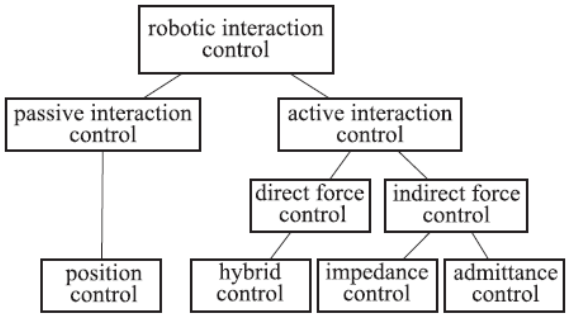


Figure 2.22: Classifications in active and passive compliance [10]

Direct force control schemes are developed to achieve force control when the end effector is in contact with a compliant environment. Due to the adoption of an integral action on the force error generated by an outer force loop, motion control capabilities along the unconstrained task directions are recovered using a parallel composition of the force and motion control actions. Force tracking along the constrained task direction is achieved by adapting the estimate of the contact stiffness [17].

Indirect force control

With indirect force control, the interaction force can be regulated indirectly by acting on the desired pose of the end effector assigned to the motion control system. Interaction between manipulator and the environment is in any case directly influenced by compliance environment and by compliance with manipulator impedance [17]. As shown in Figure 2.23, the indirect force controller is a controller that has two loops. The external one is force controlled, and the internal one is position-controlled. An indirect force controller is often built on top of a typical industrial manipulator controller, which makes it easier to develop than a direct force controller. A direct force controller also has two loops, but in contrast to the indirect force controller the internal loop is torque controlled. Therefore instead of achieving the desired position and velocity, the internal loop makes sure that each joint achieves its desired torque [11].

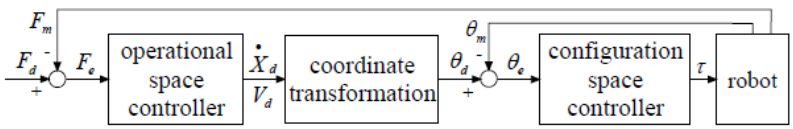


Figure 2.23: Indirect force control with internal position control [11]

2.3.4 Types of compliant force control

The purpose of compliance control is to achieve a suitable active compliance that can be easily modified acting on the control software so as to satisfy the requirements of different interaction tasks. First an overview is given on the different types of compliant force control. It only discusses the main lines, followed by a description of the most applied principles.
In equation 2.3.7 the general dynamic model is given in the joint space.

$$\tau = M(\theta)\ddot{\theta} + c(\theta,\dot{\theta})\dot{\theta} + g(\theta) + b(\dot{\theta}) + J^T(\theta)F_{tip} \tag{2.3.7}$$

As said before, in this model $M(\theta)\ddot{\theta}$ is the inertia parameter, $c(\theta,\dot{\theta})$ is the term accounting for coriolis and centripetal forces, F_{tip} stands for the forces applied on the end effector while $g(\theta)$ is the gravity compensation. The τ-matrix gives the joint torques and the final factor $b(\dot{\theta})$ represents the influence from friction losses in the joints [17]. All following control principles are built on this model.

Overview

Before the pros and cons of each system can be formulated, the existing methods must be known just as the different implementation conditions. In the tables below the algorithm classification is given with the workspace, measured variables, modified variables and modulated objectives.

Algorithm classification		Workspace	Measured variables	Modified variables	Modulated objectives
Active stiffness control	1. Version one	Joint space	Position force	Joint displacement, contact force	Joint stiffness matrix
	2. Version two	Task space[a]		Position error, contact force	Stiffness matrix
Impedance control	1. Basic impedance control	Task Space	Position, velocity, force,	Position and velocity error, contact force	Impedance
	2. Position-based impedance control			Modified desired trajectory, contact force	
Admittance control			force	force error	Admittance
Hybrid control	1. Hybrid position/force	$\{P\}^b$	Position	Position error	Position
		$\{F\}^c$	Force	Force error	Force
	2. Hybrid impedance	$\{P\}$	Force	Velocity error	$Z_{mp}{}^d$
		$\{F\}$		force error	$Z_{mf}{}^e$
Explicit force control	*PI, PD, PID,* etc.	Task space	Force	Force error	Desired force F_D
Implicit force control		Task space	Position	Position error	Predefined stiffness

Table 2.5: Algorithm overview [13]

More complicated methods are the result of these basic principles.

Algorithm		*Control objective*	*Control techniques*	*Applied theory*	*Applied domain*
Adaptive control	Indirect	parameter error convergence	parameter estimate	Adaptive principle	Having unknown parameters
	direct	tracking error convergence	Adaptation scheme		
Robust		Tracking error convergence	Robust control law	Sliding mode control, Bounded input-output stability	Having model uncertainties
Learning		Complete compensation to unknown	feedforward compensation	Learning principle	Being a period task

Table 2.6: Advanced algorithms [13]

In this thesis, an UR robot with build-in force sensors was used. The choice was to measure the tip force relative to the base frame. The other variables that can be measured are position, velocity and acceleration. The purpose of the assignment was to follow a trajectory and in the meantime to be able to be disturbed by external forces. With this goal in mind, position-based impedance control seemed to be the most obtainable. In combination with the adaptive principle it could be a perfect combination. In the following sections some of these principles are viewed in more detail.

Impedance control

Impedance control is a way to control the dynamic relationship between the environment and the end-effector. This method is the most general used for a compliant controller. As shown in Figure 2.24 a system can be represented by a spring, damper and mass system.

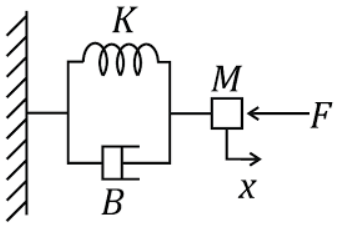


Figure 2.24: Spring/damper system [12]

Hogan N. came up with the idea that, although both position and force could not be controlled at the same time, the contact forces can be used to control the manipulators impedance [36] [37].

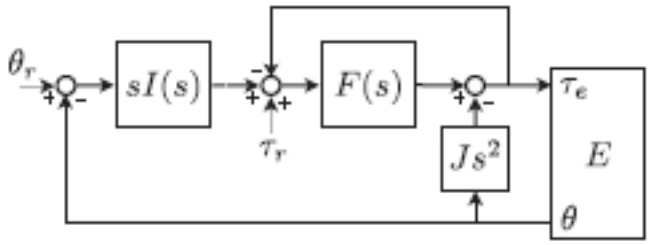


Figure 2.25: Impedance control schema [10]

Impedance control uses an inner loop to control the force and an outer loop to control position as shown in Figure 2.25. F(s) is the force controller and sI(s) implements the desired damping and stiffness [10]. Js^2 on the other hand stands for the inertia of the system.
Equation 2.3.7 could be modified to this approach by inserting extra stiffness and damping parameters.

$$\tau_{ext} = K_P(\theta_d - \theta) + K_D(\dot{\theta}_d - \dot{\theta}) + M(\theta)(\ddot{\theta}_d - \ddot{\theta}) \tag{2.3.8}$$

In equation 2.3.8, K_P and K_D are the stiffness and damping parameter of the controller.

$$\tau = M(\theta)\ddot{\theta} + C(\theta,\dot{\theta})\dot{\theta} + K_p(\theta_d - \theta) + K_D(\dot{\theta}_d - \dot{\theta}) + M(\theta)(\ddot{\theta}_d - \ddot{\theta}) + g(\theta) + h(\theta,\dot{\theta}) \tag{2.3.9}$$

In 2.3.9 θ_d stands for the desired joint configuration.
This equation can be placed in the task space instead of the joint space.

$$F = \Lambda(x)\ddot{x} + \mu(x,\dot{x})\dot{x} + K_p(x_d - x) + K_D(\dot{x}_d - \dot{x}) + \Lambda(x)(\ddot{x}_d - \ddot{x}) + \gamma(x) + \eta(x,\dot{x}) \tag{2.3.10}$$

With $\Lambda(x) = J^{-T}M(\theta)J^{-1}$ en $\eta(x,\dot{x}) = J^{-T}h(\theta,\dot{\theta}) - \Lambda(\theta)\dot{J}\dot{\theta}$ [1]

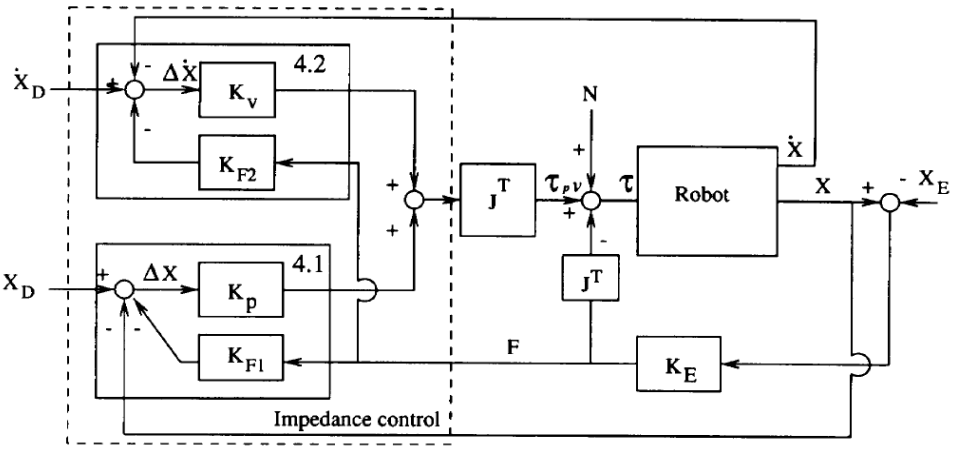


Figure 2.26: Impedance control schema [13]

The impedance control has no need of backdrivability and can be very accurate at the same time thanks to the inner force loop [10]. On the other hand, it is difficult to reach a high impedance because it requires a high gain for $K_D(\dot{x}_d - \dot{x})$, as the impedance can be described by $\frac{F}{\dot{x}}$. In some articles it has been reported that there is a tradeoff between accuracy and the ability to make high impedances. In impedance control, the inner loop has a softening action, while the outer loop has a stiffening action. In the case of high desired impedance, this leads to an antagonistic behaviour between the inner and the outer loops. As a final remark, there are issues related to the high performance requirement for the inner force loop. In fact, depending on the environment dynamics, a low-gain tuning can lead to poor performance, whereas a high-gain tuning can cause instability when touching a hard surface. Using more advanced force controllors or increasing the actuators bandwidth could solve this problem [10].

To conclude, impedance control is good if there are big changes in the sensored values or to slow down accelerations. It's also used to reduce the deviations from the reference path, which in this case is useful [38], but it's not good when a exact position needs to be reached. An other disadvantage of an impedance controller is that it needs a good system model to tune its gains.

In normal impedance control, the inverse dynamics must be taken into account. Therefore they are provided in the experiments. In a course document from Dr V. Krovi is said that these are negligible because the previous and the new configuration are close together. So this has little influence on the end result [39]. A small one-DOF example can be made such as the rotation around a single point, shown in Figure 2.27.

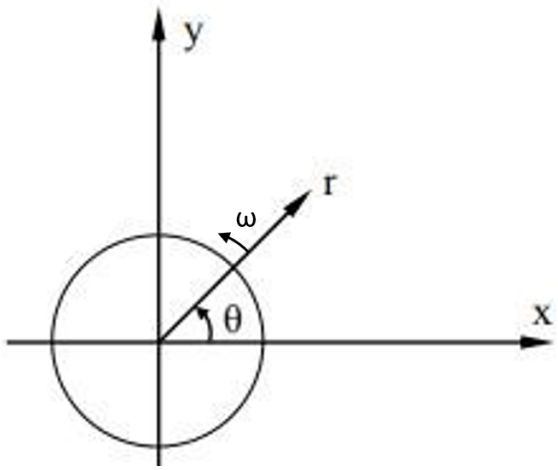


Figure 2.27: 1DOF example

The general formula can be reformulated:

$$\tau_{ext} = K_p(\theta_d - \theta) + K_D(\dot{\theta}_d - \dot{\theta}) + M(\ddot{\theta}_d - \ddot{\theta}) \tag{2.3.11}$$

With normal operation, the joint torques are equal to the dynamics of the system, $\tau_{ext} = 0$. So τ of equation 2.3.9 is equal to the inertia, coriolis, gravity and friction terms. When an external force is applied on the end-effector in the opposite direction to the speed, the rotating link will not follow the desired path, it will chase the desired position but never reach it. The term $K_p(\theta_d - \theta)$ can be seen as a mechanical spring, $K_D(\dot{\theta}_d - \dot{\theta})$ can be seen as a damper and $M(\ddot{\theta}_d - \ddot{\theta})$ can be seen as an extra virtual mass.

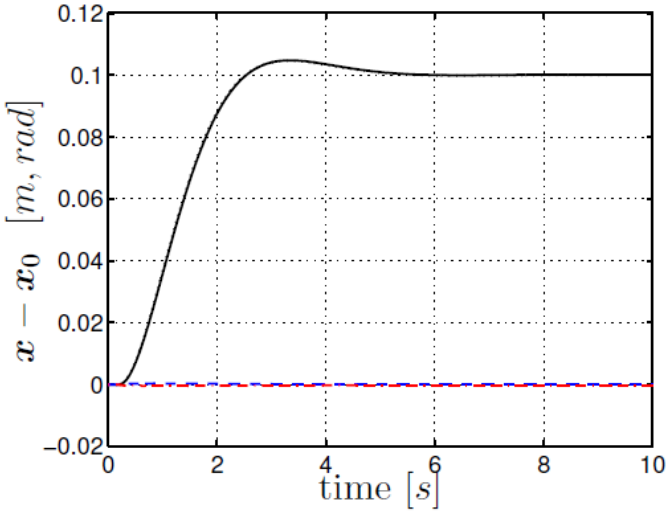


Figure 2.28: Simulation results 1DOF example [14]

Figure 2.28 shows an example response of a step input. When the external force is removed, the system will go back to zero as a response of a second order system, due to the virtual spring, damper and mass system. During the experiments, a two degrees of freedom example was elaborated on how this works exactly.

Admittance control

Admittance control is the inverse of impedance control. In this case, however, the configuration of the robot arm is controlled and not the force it exerts. Figure 2.29 shows a general admittance control scheme. Here $\frac{A}{s} = \frac{1}{ds+k}$ with d and k the desired damping and stiffness. The P(s) replaces the position control law and $\frac{1}{Js^2}$ represents the systems inertia [10].

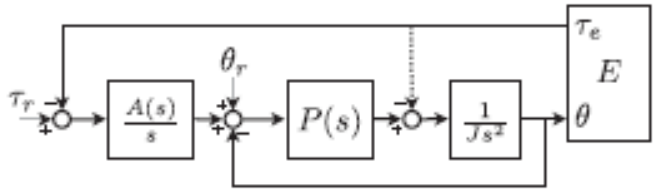


Figure 2.29: Admittance control schema [10]

The general purpose of compliant force control using admittance is to take a position-controlled robot as baseline and make the necessary modifications of the admittance to this system. From the definition it's known that the admittance is the inverse of the impedance. Compared to impedance control, admittance control focuses more on the desired force tracking control [13]. A more detailed scheme of admittance control is given in Figure 2.30.

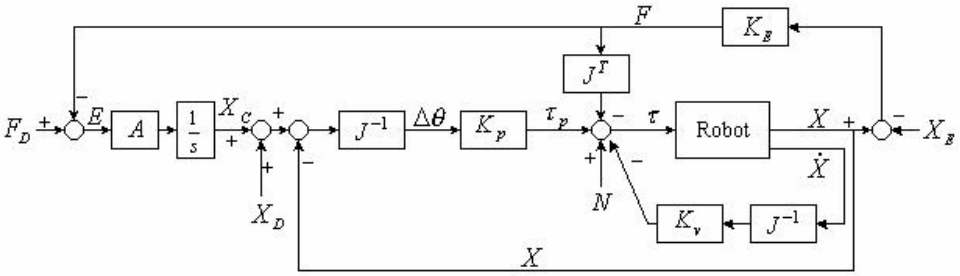


Figure 2.30: Detailed admittance control schema [15]

The admittance matrix A relates the force error vector to the end-effector velocity. Effective and precise admittance control can be achieved by choosing a suitable admittance matrix for the known stiffness of the environment [15].

Hybrid control

There are different types of hybrid control. Overall the purpose of hybride control is to take the advantages from impedance control or admittance control and position control without force control technique in a single system. Therefore the configuration and the forces or torques are combined as shown in Figure 2.31. In impedance and admittance control the configuration and external forces are coupled. In this case both are separately controlled. The hybrid controller is not a control law, it's an architecture where the different control laws might be applied. There are two general concepts of hybrid control: position/force control and hybrid impedance control.

Hybrid position/force control combines force and torque information with positional data. The position control and force control can be separately considered.

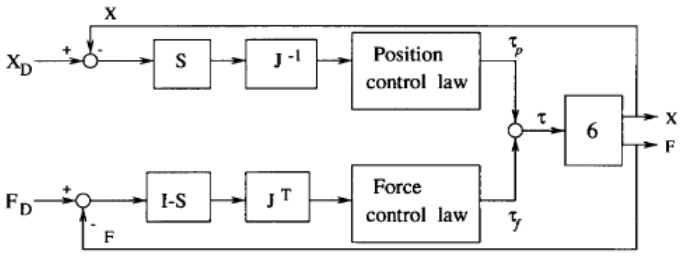


Figure 2.31: Hybrid position/force control schema[13]

The matrix S in Figure 2.31 determines the subspaces for which force of position are to be controlled. When S_j element is zero, the j^{th}-DOF must be force controlled, otherwise it's position controlled [13]. Figure 2.32 depicts a robot, its environment and the gravity compensation.

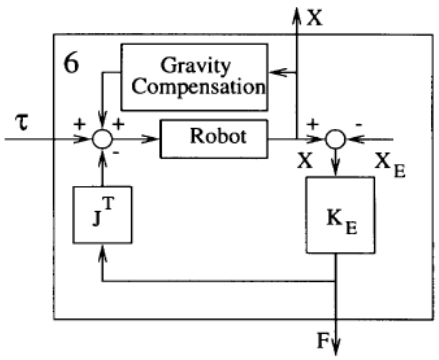


Figure 2.32: Robot environment and gravity compensation [13]

On the other hand, there is hybrid impedance control. It's developed by Anderson and Spong, who combined impedance control a hybrid control into one strategy. This gives the designer more flexibility in choosing the desired impedance of the system [13].
Hybrid impedance control is a more sophisticated algorithm than the hybrid position/force control. In contrast, impedance values of the manipulator should be reiterated as the manipulator encounters different environments. This also could be achieved by using adaptive control algorithms which are discussed in the next section [15].

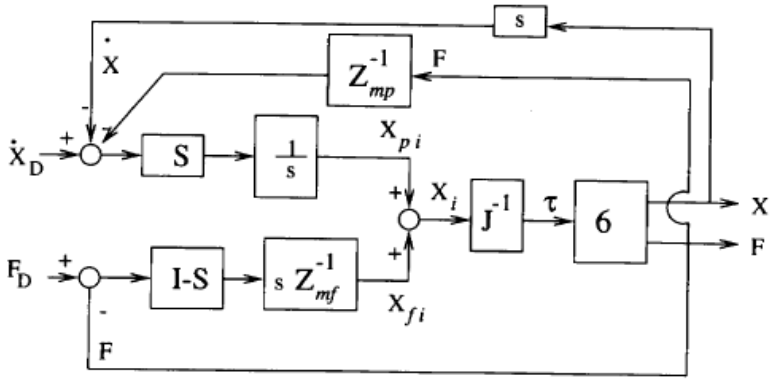


Figure 2.33: Hybrid impedance control schema [13]

In Figure 2.33, Z_{mp} and Z_{mf} are the desired impedance terms that can be selected by the user. If they are diagonal matrices, the impedance by each degree of freedom is represented by an element of these matrices. S is the compliance selection matrix, the same as the one in the hybrid position/force control. The modified desired trajectory $X_i = X_{pi} + X_{fi}$ where X_{pi} and X_{fi} are the modified position and force trajectories expressed in position and force subspace respectively [13].

Adaptive force control

The basic principle of adaptive force control is to adjust the control parameters during operation. This takes into account the performance and the environment to achieve an optimal effect. This principle can be applied to all fundamental control techniques already discussed.

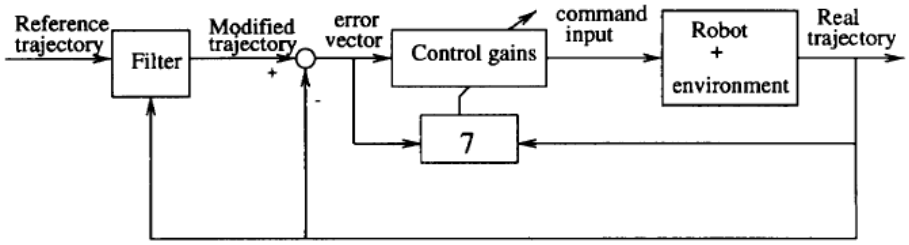


Figure 2.34: Adaptive control schema [13]

Adaptive control contains two types, direct or indirect. In the indirect method there is an explicit parameter estimation for unknown parameters of the dynamic model of the controlled robotic system. This parameter estimation is used in control gains. The existing indirect adaptive force control techniques are indirect adaptive (simply called IA), IA explicit force control and IA impedance control. The objective of an indirect adaptive force control is to make parameter error converge to zero. For this reason, its design requires a precise knowledge of structure of the entire robot and environment. In practice, the robot structure can be almost exactly described, but the environment is variable [13].

Because of this constraint, direct adaptive strategy is more and more used in robot force control. In direct adaptive force control, an adaptation scheme is applied so that the control gains are self-adjusting. The objective of this self-adjustment is to make the tracking error vector converge to zero. Figure 2.34 shows the general structure of adaptive force control. In Figure 2.34, the box 7 is an adaptation scheme or a parameter estimator. The former represents a direct adaptive force control and the latter stands for an indirect adaptive method. The existing direct adaptive force control techniques are direct adaptive (DA) admittance control, DA impedance control, DA position and force control and so on [13]. In summary, the objective of an adaptive force control is to design a command input for a robot, so that either of the following control aims are achieved in the presence of unknown parameters of robot and environment:

- Stiffness control: $\lim_{t\to\infty}(X_D - X) = -K_F F$
- Impedance control: $\lim_{t\to\infty}(X_D - X) = -[Ms^2 + Ds + K]^{-1} F$
- Hybrid control: $\lim_{t\to\infty} S(X_D - X) = 0, in\{P\}$ and $\lim_{t\to\infty}(I - S)(X_D - X) = 0, in\{F\}$
- Hybrid impedance control: $\lim_{t\to\infty} S(\dot{X}_D - \dot{X}) = -Z_{mp}^{-1} F, in\{P\}$ and $\lim_{t\to\infty}(I - S)(F_D - F) = 0, in\{F\}$

Where X_D is the desired position vector, X and $\dot{X}$ are position and velocity vectors, K_F is the compliance matrix for modifying position command, F is the resulting contact force vector, $-[Ms^2 + Ds + K]^{-1}$ represent the desired inertia, damping and stiffness values, Z_{mp} is the desired impedance term that is selected by the user and S determines the subspace for which force or position are to be controlled[13].

Robust force control

The objective of robust force control is to achieve the target dynamics such as the target impedance, etc. and to preserve the stability robustness in the presence of bounded model uncertainties (alternatively called modeling errors) in robot and environment. Figure 2.35 shows the structure of robust force control. In Figure 2.35 the command input includes two parts: robust control law and feedback control law. The feedback control normally uses PI, PD or PID, etc. The difficulty is to design a good robust control law. For this reason, the design concept of sliding mode is widely used. In terms of error information and output feedback, robust control law is usually built by employing Lyapunov's direct method [40]. Finally, the designed robust and feedback control laws guarantee the achievement of the predefined target dynamics, while preserving stability in the presence of modeling errors. Until now, the two main categories of robust strategies for robotic force control are: robust hybrid position/force control and robust impedance control [13].

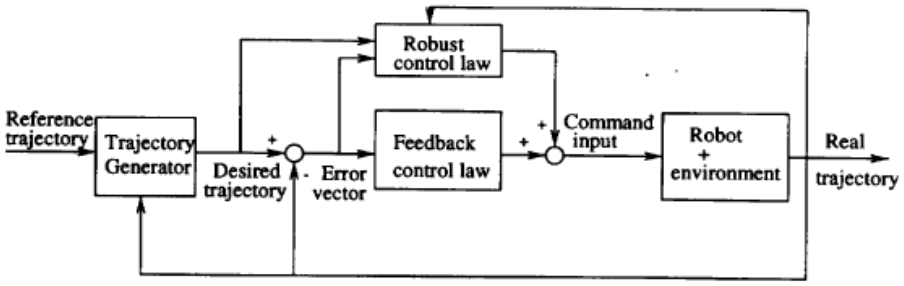


Figure 2.35: Robust force control schema [13]

Learning algorithm force control

In modern robotics, learning algorithms for force control have been independently introduced. Learning algorithms are mostly used in hybrid position/force control when a robot performs the same task repeatedly, like a pick-and-place operation [1]. It can enhance the performance of the controlled robotic system significantly. The algorithm utilizes position, velocity and acceleration errors or force error for learning the command input required to perform tasks. It guarantees the convergence of both position and force tracking errors, as well as robustness, for sufficiently small parameter uncertainties and disturbances [13]. This type of learning control differs from adaptive control in the sense that the learned information is generally nonparametric and useful only for a single trajectory. However, iterative learning control can account for effects that have not been parametrized in a particular model [1].

2.3.5 Implementation choice

Since the different types were discribed, the choice had to be made by combining the specifications due to the application. Various simplifications were made in this application. We neglected all friction and other losses like centripetal and Coriolis forces and the gravity compensation. This is because impedance control must take place in Cartesian space and the influence of the dynamics was neglected. This is said in a course document written by Dr V. Krovi on this subject and it's corroborated by a small experiment [39]. In the UR3, a PID controller is incorporated, that controls the joint angles. In turn, they ensure that the maximum torque is not exceeded by controlling the maximum joint accelerations.

The admittance force control has been chosen because of its general properties such as: easy to implement on top of the robots controller, it's usable in a path tracking environment and it can be applied in joint and task space. In the overview table, Table 2.5 we see that impedance control could be used when we need a desired performed force on a surface. In this application a path needs to be tracked so there is no desired force. When the system is completely known, the desired force could be recalculated and impedance control could be used. Therefore the system must be well defined, which is not the case and the torques must be controllable in the robot's controller.

Another approach could be the use of hybrid control. Therefore the desired force on a surface could be used and controlled by impedance control while the position on the surface could be controlled by admittance control. For example, sanding a surface can be used.

Other more advanced purposes such as direct adaptive control could also be used. For this the basics must be known first, because this technique adjusts the stiffness and damper parameters in the equation for maximum performance. Other more advanced algorithms such as learning algorithms are not applicable since there is no sequential task to perform.

2.3.6 Implementation

As we chose admittance control for this application the general equations were used. In this section all parameters and assumptions will be described. In the previous sections, the torques were always used. In the actual implementation the Cartesian coordinates are used. In most of the equations the joint angles were used, this is because the formulas are more understandable than in task space.

$$\tau = M(\theta)\ddot{\theta} + c(\theta,\dot{\theta}) + K_p(\theta_d - \theta) + K_D(\dot{\theta}_d - \dot{\theta}) + M(\theta)(\ddot{\theta}_d - \ddot{\theta}) + g(\theta) + h(\theta,\dot{\theta}) + J^T F_{ext} \quad (2.3.12)$$

Equation 2.3.12 can be simplified because coriolis forces, gravity compensation, torques caused by friction and the inertia were neglected. This is because it was previously stated that the dynamics could be omitted because there was only a minimal difference between the previous point and the next point that must be reached [39].

In general, it can therefore be said that admittance control is applied in Cartesian coordinates. Equation 2.3.13 is the equation to be applied in all applications. We assume that the forces are measured in base frame. As a result, there is a fixed reference and the admittance control can be observed.

$$0 = F_{ext} - [K_p(x_d - x) + K_d(\dot{x}_d - \dot{x}) + M(\ddot{x}_d - \ddot{x})] \quad (2.3.13)$$

This equation could be transformed to a control system in combination with the robot and its sensors. In Figure 2.36 the control scheme is given.

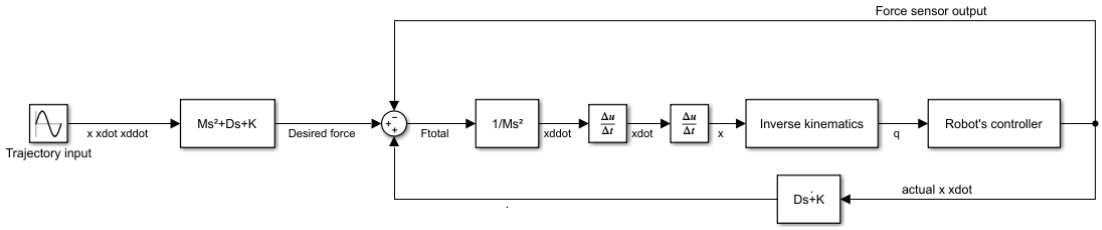


Figure 2.36: Simplified overall control loop

We know from previous chapters that the K_P and K_D parameters for stiffness and damping are always diagonal matrices in joint space and M is a virtual mass. In examples found from various sources, stiffnesses of 20N/m to 3000N/m have been used. In most of the lecture, the damping parameters are not described. These parameters formulas are not defined because the desired stiffness and damping are linked to the type of application. In this application there is some freedom in choosing them. The ideal parameters must not be known and will be experimentally recorded [41].

In addition, in the simulations, the torque limits were compared with the calculated torques to make corrections, where necessary using the inverse dynamics of the robot. This way the maximum torque would never be exceeded and there would be no damage to the joints of the UR3 robot. This can be said if the joints were to be controlled directly. Since URScript commands were used, this was unnecessary, because the controller of the robot itself takes this into account as shown in the control scheme in Figure 2.36.

2.4 Trajectory generation

For the robot to follow a smooth path, a trajectory had to be generated. Because there are different options and various algorithms, it was decided to briefly summarize these in this chapter of the literature study.

2.4.1 Definitions

Trajectory generation is used to create a path between two points with several end and begin constraints such as position and velocity constrains, which are time dependent. The main goal of trajectory generation is to find a path in function of time, between preset points, where the robots movements are smooth, rather than fast accelerations and decelerations. This is very energy inefficient and it also leads also to faster degradation of the lifetime of the engines in the joints. In Figure 2.37 a proposal of two paths is given to illustrate the path generation of the point to point path with discontinuities.

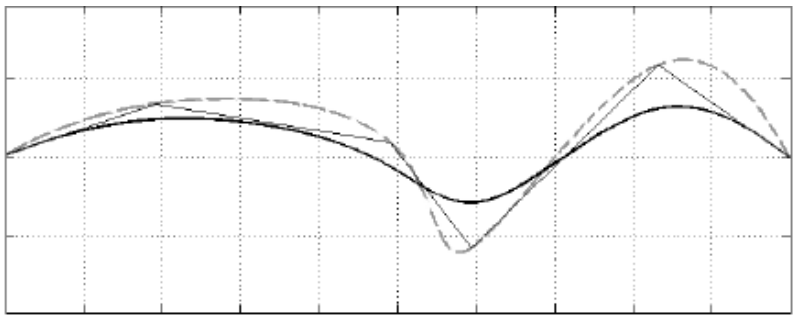


Figure 2.37: Smooth interpolation between several points [16]

A general trajectory consists of several trajectories between different points. In Figure 2.37 we see that at the second and third point the speed should not equal zero, while in the discontinuous trajectory, the velocity must always go to zero due to inertias. At the beginning and end of a path, the robots velocity goes to zero.

To do the trajectory generation, there is a need to be able to map out a path in which the progress is continuous. For this, path planning can be done in joint and Cartesian space. The joint space approach could be used when the robot's joint limits are important. As an example, a straight line can be taken between two points. In Cartesian space this will also form a straight line visually. However, if a trajectory is generated in joint space, the Cartesian trajectory will not have a predefined continuation, so not in a straight line, due to the robots kinematics. Figure 2.38 shows a normal trajectory with comparing joint velocities and accelerations. This trajectory uses a trapezoidal velocity profile and is most generally used in simple motor drives.

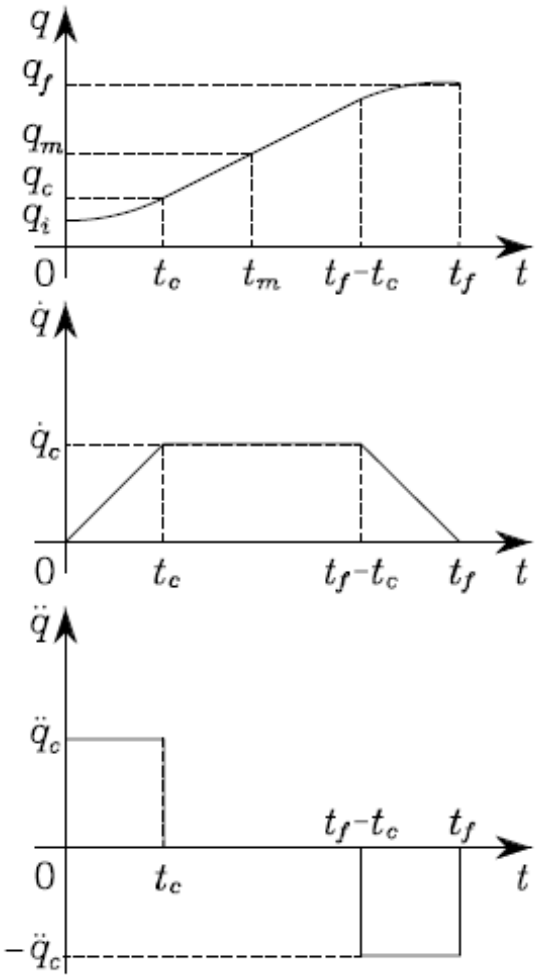


Figure 2.38: Trapezoidal velocity profile in terms of position, velocity and acceleration [17]

When path planning in Cartesian space is done, the motion between two points is always known exactly and can be plotted in space. Here the problem is that the singularities are not taken into account [17].

2.4.2 Types of trajectory generation

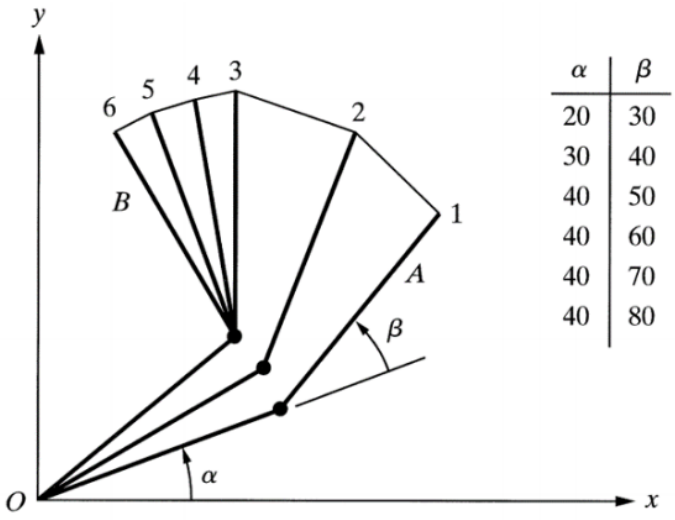
For trajectory generation there are multiple algorithms. In this section some of the most popular are described. Most of the information has been collected from a book about robotics modelling, planning and control and a PowerPoint about trajectory generation made by Václav Hlaváč [17] [18].

Straight line trajectory

A straight line can be made by using the trapezoidal velocity profile in Cartesian space. That is why the joint positions and velocities must be recalculated to the Cartesian space. This ensures that more computing power is required to perform the inverse kinematics.
There are two approaches to get a straight line trajectory. First a line can be interpolated between two points. For this reason the line must be divided in several segments where the joint angles do not change uniformly. On the other hand the trapezoidal approach can be used, where first there is an acceleration, then a moment of constant speed and a deceleration to the end of the movement, such as in Figure 2.38.

Non-normalized trajectory

The aim of this strategy is to bring the different joints to their maximum speed and thereby reach the final configuration as quickly as possible. The trapezoidal velocity profile timing law is used for each different joint.

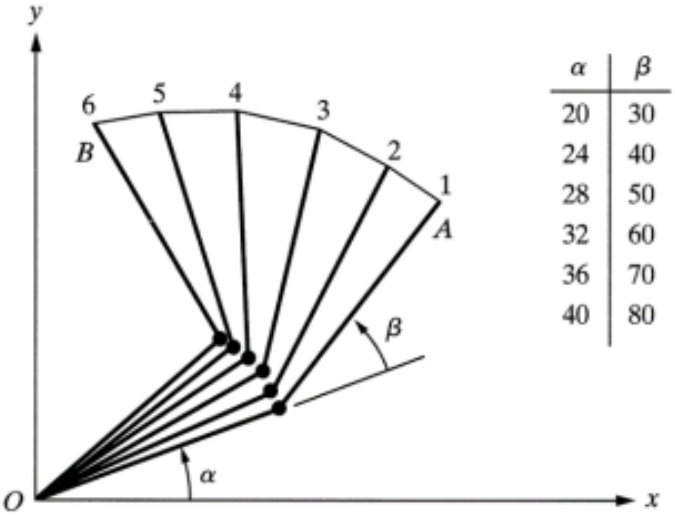


α	β
20	30
30	40
40	50
40	60
40	70
40	80

Figure 2.39: Non-normalized trajectory [18]

Normalized trajectory

This way is different from the non-standardized method. The longest period in which a joint reaches its end position is used here. This is then used as a reference to determine the speeds of the other joints. The different joints reach their final configuration at the same time, shown in Figure 2.40 .



α	β
20	30
24	40
28	50
32	60
36	70
40	80

Figure 2.40: Normalized trajectory [18]

Cubic interpolation

A cubic polynomial uses four boundary constraints, the begin and end position and the begin and end velocities. The parameters are set on the basis of a scaling, tau, which represents the position in the range of a number between 0 and 1. $\tau = t/T$ where t is the time when a certain position is reached and T is the time over which the robot performs its trajectory.
In equations 2.4.1 to 2.4.3 , the algorithm is given to calculate a trajectory from point to point with given boundary constraints.

$$\Delta\theta = \theta_{end} - \theta_{begin} \tag{2.4.1}$$

$$\tau = t/T, \tau \in [0,1] \tag{2.4.2}$$

$$\theta(\tau) = \theta_{begin} + \Delta\theta(a\tau^3 + b\tau^2 + c\tau + d) \tag{2.4.3}$$

There are four unknown parameters and four boundary conditions so this equation can be solved. Figure 2.41 shows a possible solution for the calculated trajectory. The speed and acceleration can always be calculated on the basis of integration.

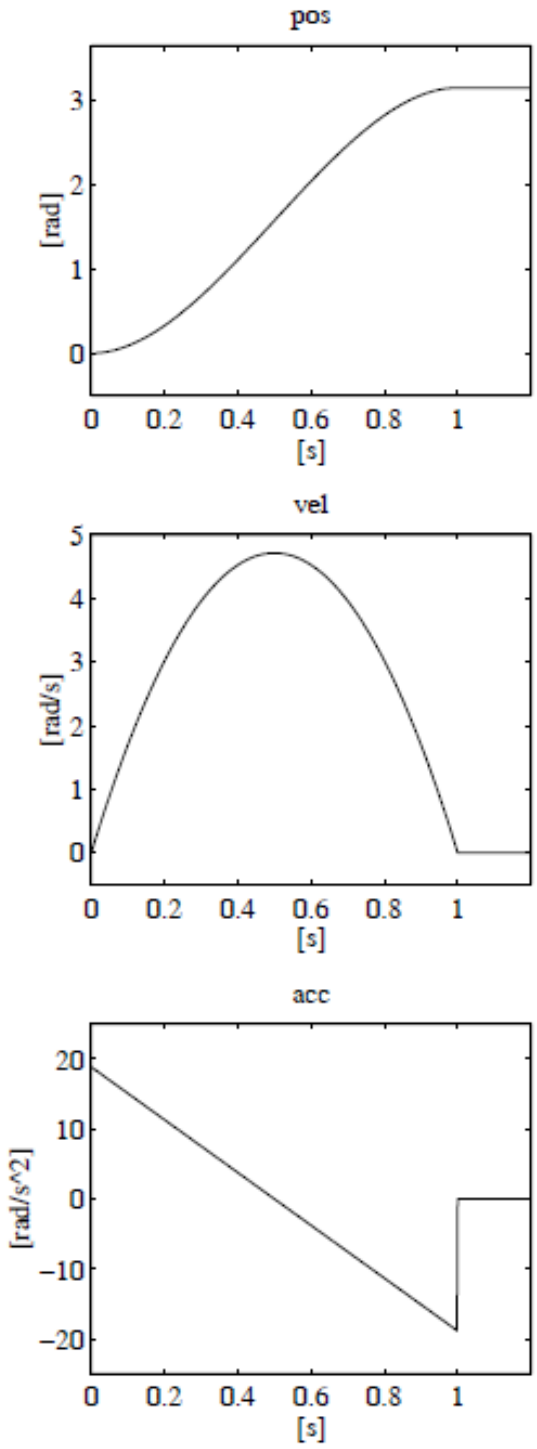


Figure 2.41: Cubic polynomial timing law, position, velocity and acceleration [17]

Quintic interpolation

This is roughly the same as the cubic polynomial. It is more extensive since the acceleration is now also integrated and a fifth order polynomial can be calculated. In most cases, the acceleration is equal to zero, giving us comparison 2.4.7.

$$\theta(0) = \theta_0; \theta(1) = \theta_1; \dot{\theta}(0) = \theta_0 T; \dot{\theta}(1) = \theta_1 T; \ddot{\theta}(0) = \theta_0 T^2; \ddot{\theta}(1) = \theta_1 T^2 \tag{2.4.4}$$

$$\tau = t/T, \tau \in [0,1] \tag{2.4.5}$$

$$\theta(\tau) = a\tau^5 + b\tau^4 + c\tau^3 + d\tau^2 + e\tau + f \tag{2.4.6}$$

$$\theta(\tau) = \theta_0 + \Delta\theta(6\tau^5 - 15\tau^4 + 10\tau^3); \Delta\theta = \theta_1 - \theta_0 \tag{2.4.7}$$

Higher order polynomials

A certain polynomial can always be drawn by a certain number of points. For example, a specific line can always be drawn through two points. Only a second order polynomial can be drawn through three points.

Higher order polynomials provide a suitable solution class for satisfying symmetric boundary conditions in a point-to-point motion that imposes values on higher order derivatives. Here the interpolating polynomial is always of the odd degree and the coefficients are always integers, alternate in sign, sum up to unity and are zero for all term up to the power $\frac{degree-1}{2}$ [18].

Path generation

The previous examples of traject generation can be used to travel from point to point. However, if several points have to be reached, these different line segments must be joined together as a whole to make a smooth path. This is called a spline. As an example, a cubic spline curve is a piecewise cubic curve with continuous second derivation. More examples and algorithms can be found in a course PowerPoint made by Václav Hlaváč and a book named 'Robotics Modelling, Planning and Control' [18] [17].

2.5 Universal Robots properties

In the user manual, all necessary specifications of the UR3 robot were found. Among other things, it discusses the hardware installation and the polyscope manual.

In addition, the technical manual contains numerous features that could be further related to this assignment such as: weight, payload, reach, joint ranges and speeds, repeatability, degrees of freedom ...
Some necessary parameters, such as the link's inertia, viscous friction coëfficients and motor inertia, were not found in manuals of the UR3 robot, these were requested from various distributors in Belgium and The Netherlands. But unfortunately these were not released.

Finally the Universal Robot URScript Programming Language manual was reviewed. It describes all possible commands.
In the following subsections there is a more detailed approach of the UR3 specifications in kinematics, dynamics and programming.

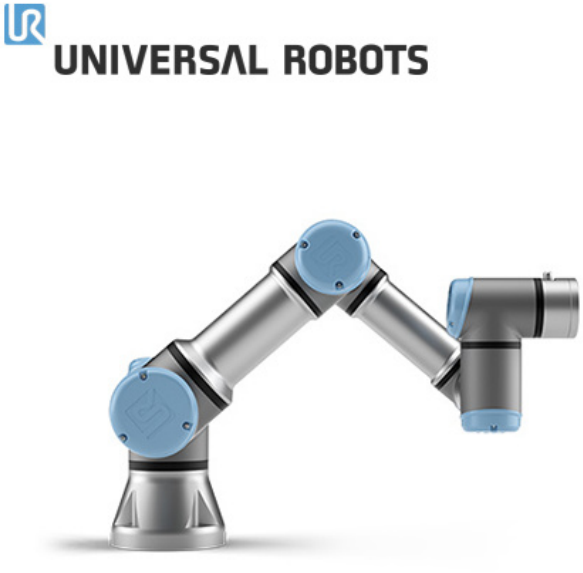


Figure 2.42: UR3 CB-series collaborative Robot [19]

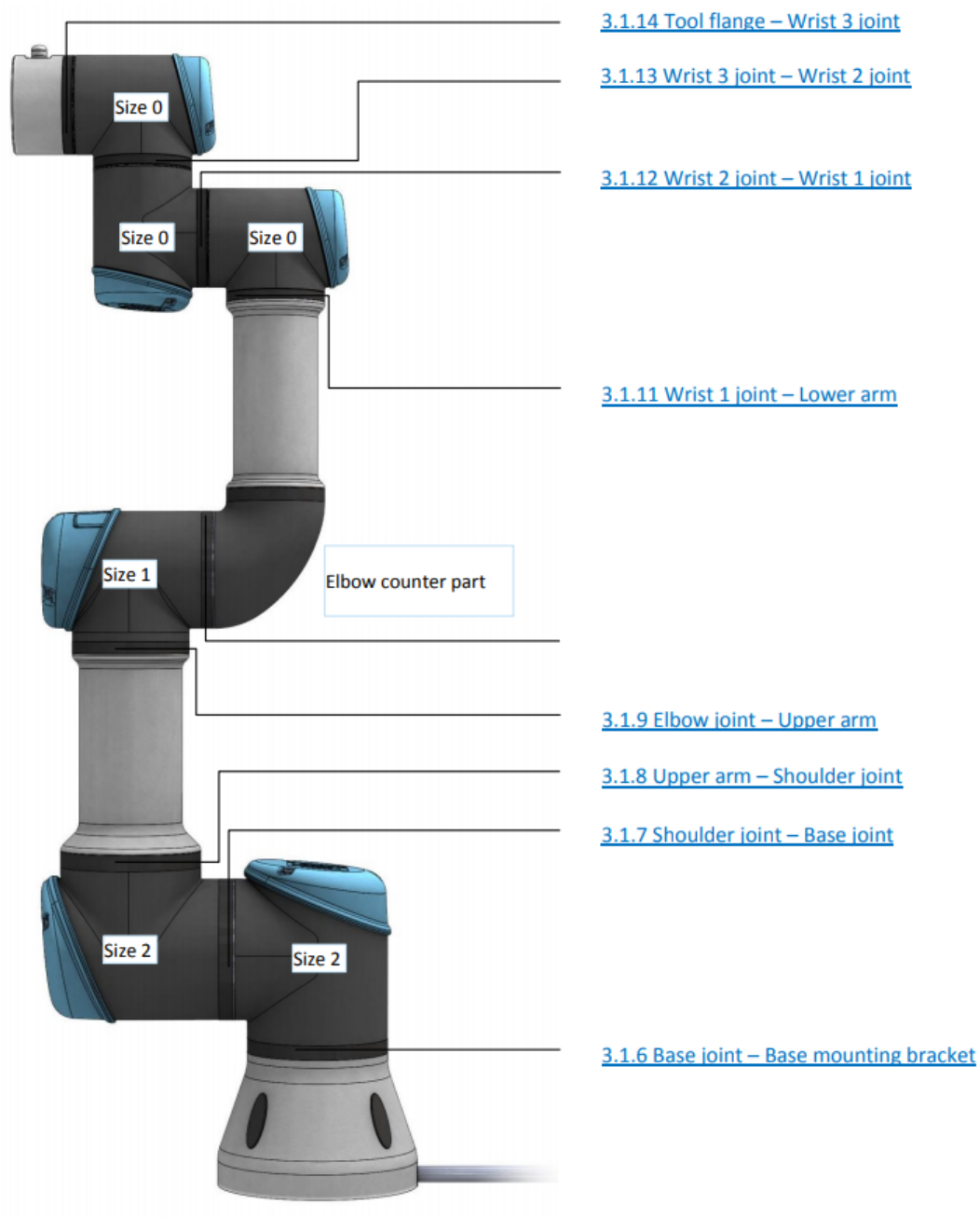


Figure 2.43: Joint names [20]

2.5.1 UR3 kinematics and dynamics

Because the maximum joint torques were not found in either of the two previously mentioned manuals, it's been looked up on the Universal Robots site [20] [42] [21]. In the following table the joint torques of each joint are given.

JOINT SIZE	TORQUE
Size 0	12 Nm
Size 1	28 Nm
Size 2	56 Nm
Size 3	150 Nm
Size 4	330 Nm

Joint	UR3/UR3e	UR5/UR5e	UR10/UR10e	UR16e
Wrist 3	Size 0	Size 1	Size 2	Size 2
Wrist 2	Size 0	Size 1	Size 2	Size 2
Wrist 1	Size 0	Size 1	Size 2	Size 2
Elbow	Size 1	Size 3	Size 3	Size 3
Shoulder	Size 2	Size 3	Size 4	Size 4
Base	Size 2	Size 3	Size 4	Size 4

Figure 2.44: Maximum joint torques [21]

The names of each joint are given in Figure 2.43. These maximum torques could be taken into account in the controller when a force is send directly into the controller. In this application there is no need to send data straight into the UR controller since the URScript commands are used. So the robot itself takes care of its maximum joint limits and gives an error when reached.

2.5.2 UR3 programming

To program the UR3 robot, Python was used. The communication with the robot was done with a TCP/IP protocol. This way multiple examples could be found to send data to the universal robot.
In this application there were multiple ports that could be used. As the controller must perform quickly and had to follow a smooth path, the fastest communication port of Figure 2.45 was used. In case of the CB-series robot, real-time ports 30003 and 30013 were used.

	CB-Series					
	Primary		Secondary		Real-time	
Port no.	30001	30011	30002	30012	30003	30013
Frequency [Hz]	10	10	10	10	125	125
Receive	URScript commands	-	URScript commands	-	URScript commands	-

Figure 2.45: Port information for TCP/IP control [22]

The precise way of communicating with the robot is discussed further in this thesis.

2.6 TCP/IP protocol

As previously mentioned in this thesis, the robot was controlled through the TCP/IP interface. This section discusses what TCP/IP is and how it was applied in the Python application.

2.6.1 Definition TCP/IP

TCP/IP or transmission control protocol over the internet protocol, is a series of network protocols that are used in network communication between computers. The internet is the best-known example of TCP/IP. TCP/IP is a packet switched protocol in which the data is sent independently in small packages. The communication software puts the packets back in the correct order, detects any errors in the reception and, if necessary, requests certain packets again until all packets are received. TCP service is obtained by both the sender and receiver creating end points, called sockets. Each socket has a socket number (address) consisting of the IP-address of the host and a 16-bit number local to that host, called a port. For TCP service to be obtained, a connection must be explicitly established between a socket on the sending machine and a socket on the receiving machine. A socket may be used for multiple connections at the same time. In other words, two or more connections may terminate at the same socket [23]. In this thesis there is no need to know how the communication exactly works because in Python there are preformed libraries that take care of the exact communication syntax.

2.6.2 TCP/IP in OSI model

OSI-model or Open System Interconnection model, is an ISO-standardized reference model for data communication standards, promoting interoperability between heterogeneous network topologies. The OSI model has somewhat lost its meaning, as the data communication world has become de facto standardized on Ethernet as a network topology and TCP/IP as a communication protocol. In Figure 2.46 the different layers of the OSI model are shown, each visualized with an animation and subscription of where the layer stands for. For more information about each layer and a more defined description, I would like to refer to a book named 'ComputerNetworks' written by Andrew S. Tanenbaum and David J. Wetherall [23].

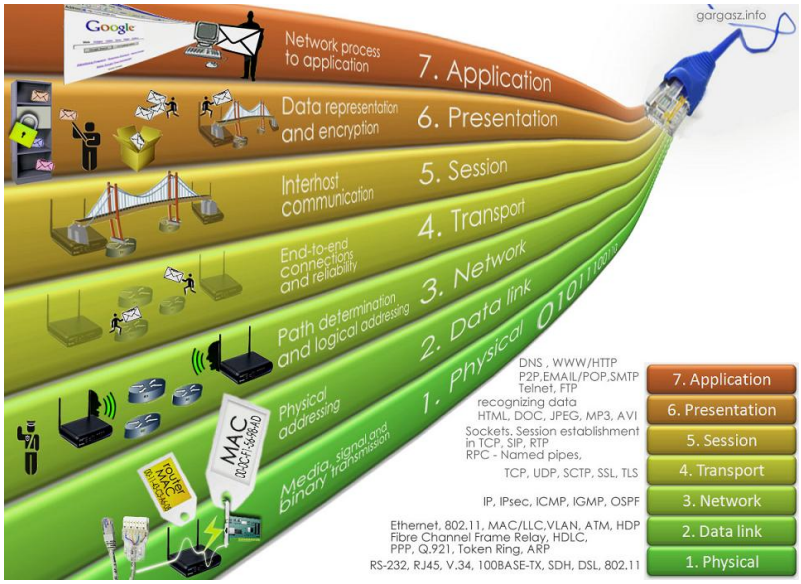


Figure 2.46: OSI model layers [23]

The TCP/IP protocol can be fitted in the seven layers OSI model. TCP/IP protocol uses all these seven layers, but some of them are put toghether in one layer of TCP/IP protocol. For example, the application, presentation and session layer are put together into one application layer. The transport layer is TCP (Transmission Control Protocol), a IP based network protocol that ensures a stable connection. The use of IP adresses is located in the network layer while the data link layer and the physical layer are represented by Ethernet and UTP cables [23] [24]. All of this is shown visually in Figure 2.47.

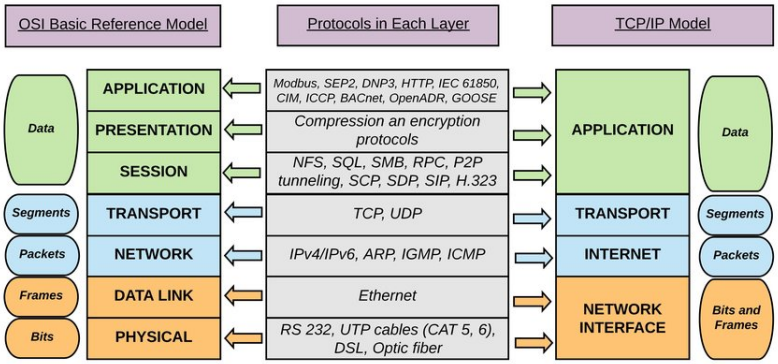


Figure 2.47: TCP/IP in OSI model [24]

2.6.3 TCP/IP with UR3 robot in Python

As the manual with technical specifications describes, the universal robot can be connected through a TCP/IP interface [42]. The manual shows the following communication specs: TCP/IP 100 Mbit: IEEE 802.3u, 100BASE-TX, Ethernet socket & Modbus TCP. In a book about computer networks was found that this ethernet connection has 100Mbps full-duplex when a cat. 5 UTP is used [23]. In addition to that the length of the UTP cable could be up to 100 meters.

For this application, the UR3 robot could also use various other interfaces such as profinet, modbus and various others given in Figure 2.48.

Port Number	Interface
80	Reserved by UR
502	Modbus TCP
2222	Ethernet/IP
7827	Internally used
7828	Internally used
8080	Not recommended since many old URCaps default to this port
29919	Internally used
29998	Internally used
29999	Dashboard server
30001	Primary
30002	Secondary
30003	Real-time
30004	RTDE
30011	Primary read only
30012	Secondary read only
30013	Real-time read only
34964	Profinet
40000	Ethernet/IP
40002	Profinet
44818	Ethernet/IP
49152	Profinet

Figure 2.48: An overview of client interfaces [25]

As already mentioned in section 2.5.2 the realtime sockets were used. The 30003 port was used to send data to the UR3 robot and the 30013 port was used to read data sent by the robot. For this connection the Python socket library was used [43]. The connection was made by the following sequence:

```
import socket
HOST = "192.168.0.9"  #IP adress robot
PORT = 30003 #Realtime port
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((HOST, PORT))
s.send ("set_digital_out(2,True)" + "\n") #URScript string
s.close()
```

2.7 Python

For this thesis Python was used, because it's a worldwide used language and it is easy to learn. Often, programs used with ROS (Robot Operating System) are also written in Python so when the switch must be made between the serial port and ROS, this would be easier. For more information on what ROS is and what its properties are, please visit 'ROS Robot Programming' a book written by the TurtleBot3 developers or 'Robot Operating Systems (ROS) for Absolute Beginners' a book written by Lentin Joseph [44] [45].

Python code could be written very readable and maintainable, so there is a good overview. Its language features support various concepts in functional and aspect-oriented programming. At the same time, Python also features a dynamic type system and automatic memory management.

Next to that Python has robust libraries, it's compatible with a lot of platforms and systems and there are a lot of applications regarding to robot arms.

In Figure 2.49 a small overview of the features of Python is given.

Figure 2.49: Python features overview [26]

There are many advantages, but Python certainly also has its disadvantages. Python is in comparison with C or C++, a lot slower because of the high-level programming. Other disadvantages are: weak for mobile development, high memory consumption, more runtime errors, ...

3
Methodology

I started my master's thesis in September 2019. First of all, I met with Mr De Ryck so he could explain the details of my master's dissertation. After some research, the actual purpose became clear: design a force compliant controller for the UR3 robot using Python and make an application. Collecting papers, books and other lecture that could help me to succeed my dissertation was the next step.

Thereafter, the literature study was created chapter by chapter using the formerly collected information. This was done from September until December 2019. In my schedule it was provided that a first part of the programming assignment would already have been concluded by then. Unfortunately, this was postponed to the second semester due to tasks and lessons. As a result of the postponement, I was able to do everything directly one after the other. However, this also meant that a servo test setup such as requested by Mr De Ryck was not executed. The necessary parts still had to be looked for, ordered and delivered and from my experience, this takes a lot of time often with little result.

Moving on to the second semester, research on previous projects was started using an impedance controller in MATLAB as a replacement for the servo application. An example was found for impedance control on path tracking, i.e. pure position [46]. This example was stripped and some parts were reused to create a well-organized program that could further be used to provide insights into impedance control.

First, an application was created simulating a two-link manipulator. Initially, no path was followed and the impedance control was done on a fixed target. As an extension to this, a circular target was added for simplicity. During these two experiments, the intention was to test the influence of the inverse dynamics and whether the control could possibly take place in joint space. In the end, it was decided to neglect the influence of the dynamics and to do the regulation in Cartesian coordinates. However, this meant that inverse kinematics had to be applied to obtain the joint angles. For this mainly a book by the University of Cambridge on modern robotics, written by Kevin M. Lynch

and Frank C. Park was used [1]. This book about robotics tells in detail how the kinematics and dynamics of rigid bodies are structured, including chapters about force control and motion control.

After the experiments with the two degrees of freedom simulation in MATLAB, a six degrees of freedom example was made. Only an impedance controller was made for a fixed target such as in two degrees of freedom, because MATLAB did work too slow and it was not the goal to design a complete controller in MATLAB that peers with the final application. Eventually, a performance was created that had to work in slow motion to obtain representative graphs. For this part the Peter Corke toolbox was used [32].

Finally, a Python controller was designed. The first part of this assignment was to provide communication between the robot and a Python program. To monitor the robot's data, a class was found on the GitHub page of SintefManufacturing, but no stand-alone class was found to send commands to the robot [47]. Since this wasn't that much work and it had to be straightforward and easy to use, it was written from scratch. To test all this, an opportunity was sought for simulation programs. There was a possibility to do the simulation in Python itself, but this would have given no added value when testing the connection. We first looked at RoboDK. This is software that simulates a conversion and adds different objects with various robots that could be programmed in different languages. It would have been ideal if this had been free software, which unfortunately was not the case. The second possible solution was to look at URSim. This is an option to write a program without a physical robot and later transfer it to the robot via USB. This software runs in a Linux virtual machine and can be obtained free of charge from Univeral Robots [48]. After the connection with the simulated robot was finalized, this was also done with the physical robot. After this, functions from both classes were tested and small programs were made to correct errors. Combined with the interpolator, received from Mr De Ryck, a program was put together before the open day so that the robot followed a certain path for the entire day. A second part of the Python controller consisted of writing a program in which impedance control is applied to the continuous trajectory of the robot. For this, it was originally thought to simply read and process the force that the robot measures itself. Because testing the inverse kinematics already showed that the application is relatively slow and that the desired frequency of 125Hz would not be achievable, it was decided to try an external force sensor. The force emitted by the robot also contains the force of movement or, in other words, the torque of movement. An Axia80 force sensor of Schunk was available but no data was read from the sensor after several attempts. As a result, it was decided to use the force from the robot and to subtract the dynamic force such as coriolis and inertia. Due to the COVID-19 pandemic this was postponed for a while and only continued later when all bugs were out of the program. Because there was always an error on the final result, an attempt was made to provide a controller that allows this error to converge to zero. Looking into the theory and simulations made in the literature study about the second order response, the obtained response was expected.

The Euler-Lagrange representation was used to calculate the dynamic parameters. This was reported in MATLAB and symbolically solved. This resulted in functions that could be used in the Python class which describes the robot. However, these calculations are very time consuming and the results cannot be converted to a Python syntax. This was solved by using a library which allows the use of MATLAB functions. The dynamic functions are also written specifically for the UR3 robot since obtaining the functions with only the joint positions and velocities took about three days. Finally, all programs were cleaned up so that only the essential code remained. It is available on my GitHub repository [49].

4
Experiments

To achieve the ultimate goal of this thesis, which is to write an force compliant controller for the UR3 robot in Python, several partial experiments were done first. The main aim of these experiments was to gain insight into the kinematics and dynamics of robot arms. Since Python was not yet known and there was a better knowledge of MATLAB, some MATLAB simulations were made to get started. The aim was to test various matters with both a two degrees of freedom example and a six degrees of freedom example. Because MATLAB is often used to simulate things, there already are some pre-made toolboxes for this that accelerate the design. Ultimately, a Python controller was created that has a similar functionality. To finish a theoretical example is given of how this application and / or insights can be used in other applications.

4.1 MATLAB simulations

First of all the simulations in MATLAB will be explained. As told in the methodology, this is used to get a first example of how the impedance control works. Further the goal of this part is to make a piece of code for educational use. The simulations could be used to get future students knowledge of this kind of robot control in a simple example. First of all two degrees of freedom examples were made. This is made based on a book by the University of Cambridge on modern robotics, written by Kevin M. Lynch and Frank C. Park [1].

4.1.1 Two degrees of freedom fixed point

This is a first experiment in which the general idea of impedance control has been worked out. The goal is to take a fixed target in Cartesian space. Here, an external force must act on the manipulator and as a result of this force the manipulator must make a deviation from the target. At the same time, the manipulator dynamics must be taken into account.

Working method

This specific section discusses how to perform impedance control with a fixed point as the target. It is therefore assumed that in steady state the speed and acceleration of the manipulator is zero. In this application, equation 4.1.1 is used. Because the target is fixed to a certain point, this equation can be simplified to equation 4.1.2 by substituting the target velocity and acceleration.

$$Fext = M(a_{tar} - a) + K_d(v_{tar} - v) + K_p(x_{tar} - x) \tag{4.1.1}$$

$$Fext = -M(a) - K_d(v) + K_p(x_{tar} - x) \tag{4.1.2}$$

For all applications that were made, the damping coefficient was taken equal to 0,7 and the natural frequency was calculated with the spring (K_p) and mass (M) values equal to respectively 20 N/m and 10 kg. So $\omega_n = \sqrt{Kp/M} = 1.41 rad/s$ and the damper coefficient $K_d = 2 * \xi * \omega_n * M = 19.74 Ns/m$.
In equation 4.1.2 there are still three unknown variables. Since the velocity is the derivative of the position as in equation 4.1.3 and the acceleration in turn the derivative of the velocity, equation 4.1.4, both can be described in function of the position x, which then remains as the only unknown parameter.

$$v = \frac{x - x_{old}}{dt} \tag{4.1.3}$$

$$a = \frac{v - v_{old}}{dt} \tag{4.1.4}$$

Equations 4.1.3 and 4.1.4 substituted in equation 4.1.1 finally gives us equation 4.1.5 which has been elaborated into Cartesian coordinates.

$$x = \frac{M * (a_{tar} + (v_{old} + x_{old}/dt)/dt) + Kd * (v_{tar} + x_{old}/dt) + Kp * x_{tar} - F_{ext}}{Kp + Kd/dt + M/dt^2} \tag{4.1.5}$$

If the speed or acceleration as a reference is better for further calculations, these can also be used. Equation 4.1.6 gives us the solution for the acceleration.

$$a = \frac{M * a_{tar} - Kd * (v_{old} - v_{tar}) - Kp * (x_{old} - x_{tar} + dt * v_{old}) - F_{ext}}{Kp * dt^2 + Kd * dt + M} \tag{4.1.6}$$

After the new position, velocity and accelerations are calculated, the dynamic equation could be used to add the dynamic model of the robot. For this we know that the torque of the desired position is not much different from the actual torque, so the dynamic parameters will not differ much.

$$\tau_d = M(\theta_d)\ddot{\theta}_d + C(\theta_d, \dot{\theta}_d)\dot{\theta}_d + G(\theta_d) \tag{4.1.7}$$

In equation 4.1.7 the torque to the desired position was calculated. Because this was in joint position, the earlier calculated Cartesian position, velocity and acceleration must be transformed to joint space. In equation 4.1.8 to 4.1.10 is given how these relate to each other.

$$a = J\ddot{\theta} + \dot{J}\dot{\theta} \tag{4.1.8}$$

$$v = J\dot{\theta} \tag{4.1.9}$$

$$x = T(\theta) \tag{4.1.10}$$

To take the dynamics into account, we needed to equalize the current value of the torque to the desired value and thereby determine the acceleration in joint space. This is shown in equation 4.1.11 .

$$\ddot{\theta} = M(\theta)^{-1}(\tau_d - C(\theta, \dot{\theta})\dot{\theta} - G(\theta)) \tag{4.1.11}$$

In a course document of Dr V. Krovi named 'Kinematic and dynamic control of a two-link manipulator', was found that the influence of the dynamics of the robot on the result is negligible [39]. But this was tested in this first application before it is omitted in the following applications.

Structure of the program

To start quickly, a program was sought where the main frame was already present. This was found on the MathWorks site but did not have the functionality applicable to this application [46]. As a result, the program was completely gutted and rebuilt. Some parts such as the main and the plotting functions were retained. Others such as the calculation of the new coordinates, the inverse kinematics, the forward kinematics and the torque calculations are completely self-made from the book 'Modern Robotics' made by the University of Cambridge and the information gained about the robotic toolbox from MATLAB [1] [32]. The following list shows the main functions used throughout this application.

- Main

  In the main program, the initial parameters are passed over and created a global variable that can be accessed anywhere in the program. In addition, the two-link is created and the spring damper mass parameters are given.

- Plotter

  The plotter is used to: initialize the figure, keep track of the elapsed time, update the coordinates, display the data on the screen, read the force created by dragging the mouse and finally collecting the data to save in a CSV file.

- NewCoAndTorques

  This function is used to determine the new coordinates. Here the new joint positions as well as Cartesain coordinates are recalculated. Therefore, a function was used that calculates the torques and the inverse kinematics. To perform the dynamics test, an additional function was created that calculated inertia, coriolis and gravity matrices. This was done based on the book 'Modern Robotics' [1].

4.1.2 Two degrees of freedom circle path

Working method

In this experiment there were two approaches. On the one hand, we used the same approach as the previous example with a fixed target. This meant that the impedance control was done in Cartesian space by using equation 4.1.5. The same method as in section 4.1.1 was used. Only in this case, a varying target position was integrated. On the other hand, the same thing was done in joint space. Because the UR robot could be easily controlled in joint space by using the servoj command from the URScript commands. In this command, a joint configuration must be given instead of the Cartesian coordinates that were used in the first approach. For this second approach the same impedance control method was used with only the joint configuration instead of Cartesian coordinates. In this case, the motivation of using one or another method will be substantiated in the results on page 79.

Structure of the program

The main structure of the program remained the same, but an extra varying target was provided. For simplicity, a circular path was chosen here. In Python, trajectory generation was used to generate a path between specific points. The circle target was generated as follows.

$$x = Acos(2\pi ft) \qquad y = Asin(2\pi ft) \tag{4.1.12}$$

$$vx = -2\pi fAsin(2\pi ft) \qquad vy = 2\pi fAcos(2\pi ft) \tag{4.1.13}$$

$$ax = -4\pi^2 f^2 Acos(2\pi ft) \qquad ay = -4\pi^2 f^2 Asin(2\pi ft) \tag{4.1.14}$$

To do the check in joint space, a new equation was transformed instead of equation 4.1.5 from section 4.1.1. This meant that the external force had to be transformed into joint space creating a torque. Equation 4.1.15 gives the general equation that must be solved to joint positions, velocities or accelerations.

$$J^T F = M(\ddot{\theta}_d - \ddot{\theta}) + K_d(\dot{\theta}_d - \dot{\theta}) + K_p(\theta_d - \theta) \tag{4.1.15}$$

In equation 4.1.15 the desired position, velocity and accelerations are given in joint space. The target is only defined in Cartesian space equations. The following equations show how they are related to each other.

$$\theta = InverseKinematics(x, y, manipulator) \tag{4.1.16}$$

$$\dot{\theta} = J^{-1} v \tag{4.1.17}$$

$$\ddot{\theta} = J^{-1}(v - \dot{J}a) \tag{4.1.18}$$

In equation 4.1.18 the derivative of the Jacobian is needed. Due to equation 2.1.38 the first row of the Jacobian is calculated by derivating the equation for the x-coordinate for both joint angles and the second row is calculated by derivating the equation for the y-coordinate for both joint angles, as the definition of the Jacobian tells. The derivative of the Jacobian can be obtained by repeating the previous calculations.

$$J = \begin{bmatrix} -L_1 sin\theta_1 - L_2 sin(\theta_1 + \theta_2) & -L_2 sin(\theta_1 + \theta_2) \\ L_1 cos\theta_1 + L_2 cos(\theta_1 + \theta_2) & L_2 cos(\theta_1 + \theta_2) \end{bmatrix} \tag{4.1.19}$$

$$\dot{J} = \begin{bmatrix} -L_1 cos\theta_1 - L_2 cos(\theta_1 + \theta_2) & -L_2 cos(\theta_1 + \theta_2) \\ -L_1 sin\theta_1 - L_2 sin(\theta_1 + \theta_2) & -L_2 sin(\theta_1 + \theta_2) \end{bmatrix} \tag{4.1.20}$$

4.1.3 Six degrees of freedom

Working method

The previous section, two degree of freedom examples were made to clarify the principle and to test some things such as the influence of the dynamics and in which space would be worked. After this, the insights were processed into a six degrees of freedom example. Only in the first approach, the fixed target in two degrees of freedom was converted to the six degrees of freedom application, because the application 'path tracking with disturbance' was made in Python to save time and work. In Figure 4.1 the simplified control system is given. In the impedance control block, equation 4.1.21 is solved for the three translation and the three rotation axises.

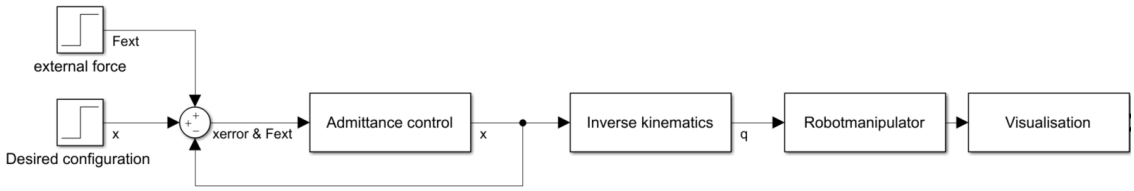


Figure 4.1: Overview of the used control method

$$x = \frac{M * (a_{tar} + (v_{old} + x_{old}/dt)/dt) + Kd * (v_{tar} + x_{old}/dt) + Kp * x_{tar} - F_{ext}}{Kp + Kd/dt + M/dt^2} \tag{4.1.21}$$

The following parameters are used in this experiment: $K_p = 200$ and $M = 10$kg. The damping is determined based on a damping coefficient that equals 0.7 and amounts to 62.58Ns/m. As a result, the response shown in Figure 4.2 is expected to be achieved.

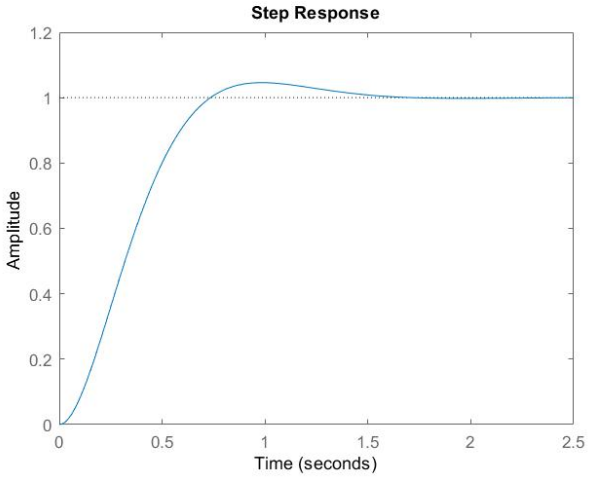


Figure 4.2: Step response

The response in Figure 4.2 has a rise time equal to 477 ms. The used rise time is defined as the time it takes to move from 10 percent of the final position to 90 percent of the final position. Because the same damping coefficient (ζ=0.7) is used for each experiment, the rise time can therefore serve as a reference.

Structure of the program

Due to the complexity of some functions such as the inverse kinematics and torque calculations, a new toolbox described in the 'Robotics Vision and Control' written by Peter Corke, was implimented [30]. The main reasons for the use of this toolbox were the fact that all functions were clearly explained in the previously mentioned book and the majority of the reviews of the toolbox were positive.

Despite the use of this toolbox, the outline of the program remained the same. A main function was used to initialize all variables, getting the figure in order with the plotter function and a separate function to calculate the new coordinates. Some functions such as the inverse kinematics, the forward kinematics and torque calculations were replaced by the functions from the toolbox. These had not been examined and tested in detail. The toolbox was believed to work correctly as it was created for educational purposes and was eagerly used when simulations were performed with a robot.

All functions of the used toolbox were straightforward except for the inverse kinematics function. For this function, some research was done on how to implement it with the given input parameters. In the previous application there was a choice of two algorithms. In this case, there was no choice and no algorithm parameters that could be finetuned. This led to a slow operation of the inverse kinematics calculations. Unfortunately, no solution was found to make this work faster, so the simulations are in slow motion. The generated graphs were better to follow the path as they were accurate with the ultimate value in function of time.

4.2 Python controller

This part of the thesis discusses how exactly the Python controller is built. The code is made available to the supervisor and co-supervisor. It is not the intention to explain all lines of code, only the main structure and the build-up of some classes are discussed here.

4.2.1 Communication with UR3 robot

First, communication with the UR robot was made. Due to the research that is done about the communication with UR3, two classes had to be written: one class where commands could be sent and another class that processes the data returned by the robot. Research was conducted to find out whether this already existed. To process the received data, a library was found on a GitHub repository of SintefManufacturing [47]. This library is old and not up to date, so it needed to be updated. No other library was found for sending data to the UR3 robot. Hence, the decision was made to write this from scratch.

Working method

First the urx library was disassembled in order to separate the necessary piece from the ones that turned out to be unnecessary. Because only a few parameters of the robot were read and many more parameters could be read, the class was extended with more attributes. According to an Excel file on the Universal Robots support page, the data coming out of the robot with software version 3.11 contains 1060 bytes [25]. This library was written for an older software version where only 86 bytes could be read. It is ensured that all data sent by the robot could also be read using this class by upgrading it.

To send data to the robot, URScripts commands are used. These are scripts that are used to program the robot using the teach pendant or the UR simulator in a virtual machine. When a string with the correct URScript syntax is sent to the robot, the robot will execute it. This only happens if the robot is not running a program or if the overwrite function is enabled. All the URScirpt commands that are integrated and the functions details can be found on the URScript manual [50].

Structure and use of the classes

The monitor is build-up with functions that can return all available parameters of the robot. Therefore first the full data must be received. The total package size is 1060 bytes, split up in multiple subpackages such as joint targets, actual joint positions, joint temperatures and so on. To use this class only a host IP adress must be given and the start function must be called. From that moment the class is making connection to the UR3 robot and the wait function must be used to ensure the connection has been made. As already shown in section 2.5.2 of the literature study, different ports can be used. For the monitoring class, port 30013 is used.
The send class has common properties. Again, only an IP address needs to be provided to run the real time command sender. Just as the monitor class, it uses a real-time port of the universal robot, port 30003. All major URScript functions are made in this class. To send the strings these functions return, an extra function has been made that converts the strings to a byte, which is understood by the robot.

4.2.2 Robot definition

Working method

Just as in the MATLAB environment, there was a need for a class that contains the properties of the robot. First of all, a library was sought that could be used. Because Peter Corke's MATLAB toolbox was clear and easy to work with, it was first searched whether it was never processed into a Python library. In fact this was the case, Peter Corke himself started writing a Python library. Unfortunately, it was not fully developed. For example, the kinematic functions and transformations are included in the library, but the dynamic functions have not yet been developed. The most recent version can be found at the GitHub repository of Aditya Dua and this library is called robopy [51].

Structure and use of the class

My co-supervisor Mr De Ryck first asked me how I would like to do this. Finally, when I came up with the idea to use the robopy library, he offered me his version. All non-essential classes were removed and the classes were left with the necessary functionality. This left us with only a class for the links, a class for the robot that describes all functions such as inverse and forward kinematics and a number of methods to calculate transformations.
Another function for the Jacobian was added based on the Peter Corke toolbox from MATLAB [32]. Using the MATLAB symbolic math toolbox and Euler-Lagrange equations, a function was then drawn up for the mass matrix, the Coriolis and centripetal matrix and the gravitation matrix. This specifically happened for the UR3 robot because the symbolic calculation with all parameters was an impossible task. It took forever for a symbolic function to be generated, so it was decided to manually enter the kinetic and dynamic parameters such as DH-parameters, inertias, center or gravities and the masses of the links. Because of this the robots parameters did not become an input of these functions and the calculations were performed much faster. Inspiration was taken from the GitHub repository of Nathalie Majcherczyk where the same was done but for a seven degrees of freedom link in two dimensions [52]. Because in this thesis work has to be done in three-dimensional space, it took much longer to determine all functions. Ultimately, these functions could be used in Python using the matlab.engine library. In section 2.1.6, the algorithm using the Euler-Lagrange equations, is shown for a two degree of freedom manipulator.

4.2.3 Trajectory generation

Working method

In a first stage, it was planned to take a fixed point as reference just as in MATLAB. Because of its simplicity, a more improved way to let the robot do a task was sought. Therefore, the robot was initially following a rectangular path using straight lines and with a maximum acceleration. This was not a smooth trajectory and a solution had to be found. Due to the meetings with my co-supervisor, the idea of using a trajectory generator came up. Because he had been working on this himself, his interpolator was used. Different methods to generate a trajectory were made: a cubic, quintic, septic and much more methods. Eventually, a function is made to create a path through multiple points using quintic partial trajectories.

Structure of these functions

First of all, multiple interpolators were made. These made a number of trajectory positions between two points in a given trajecttime. This was done by a cubic, quintic, septic and nonic method. These functions use other functions that make a n^{th}-order polynomial, corresponding to the order of the path, to be interpolated and devided it in a number of intermediate points. To get the coefficients of the polynomial, an extra function was made using an algebraic solver.

4.2.4 Main program

Working method

The main program was chosen to provide a main loop in which a cycle is performed with a trajectory between four points that remains the same. To break the loop, a GUI was made that eventually simulates external force on the robot arm due to unforeseen circumstances. The path the robot followed was created using the interpolator discussed earlier. Then, this path was saved so that with a second run with the same coordinates and the same cycle time, there is no need of waiting until the interpolator creates a path. This could simply be loaded from an existing file. After knowing the desired point from the trajectory, the force was measured. There were two options. First, a simulated force from the GUI could be used as input. Second, the force applied on the real robot could be used. The simulation of the force was done by sliders to create a force in Cartesian space. The real force does not only contain the external force on the end-effector but also the force due to the manipulators inertia, Coriolis and centripetal force. The external force must be extracted using the manipulators dynamics. By doing this, we had all the necessary parameters to perform the impedance control. This was written in an extra function. Here, the Cartesian target was used together with the force previously determined to obtain a new Cartesian speed. This velocity was converted to a position in Cartesian coordinates and then, using inverse kinematics, to a new joint configuration. This joint configuration was sent to the robot applying the servoj command, which moves the robot to this configuration while a new point is determined using the same calculation cycle again.

Structure of the program

The structure of the program is shown on the basis of the class schedule in appendix A.1. In appendix A.2 the complete cyclus diagram is illustrated. In the main program there are some extra functions created in order to keep the overview in the main while loop. In the following enumeration properties and functioning of these functions are discussed.

- initialisation(): This function recalibrates the internal force sensor, sets the payload to zero and sets the tool frame.
- gotostart(initialposition): This function is used to go to the initial start position. Until it reaches this position it will be in a continuous while loop.
- gettraj(): This function get the trajectory that is saved for the requested points. If this is not avaiable or the same points where not used or the sample rate is different, a new trajectory will be created using the trajectcalc function. This new trajectory will be saved for a subsequent use, using the savetext function.
- impedancecontrol(xtarget, vtarget, atarget, told, xold, vold, fext): The main part of the control loop. This function ensures that the impedance control is applied to the predefined axes. Then the desired joint positions are determined using inverse kinematics.
- updateplotarrays(actualpose, targetpose, force): This function is created to store the data that is created in arrays, so that all data can be reviewed afterwards.
- savealldata(): If the continuous while loop is interrupted, this function ensures that all data is stored in an Excel file.

General control system

Figure 4.3 shows the control scheme that is used in the program.

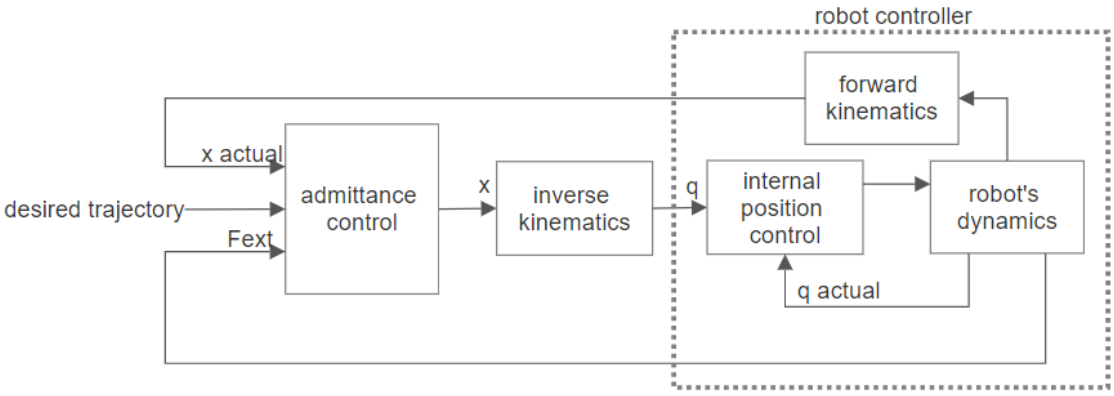


Figure 4.3: The applied control scheme

Since the admittance controller is built on top of the UR3 controller, there is no need for an extra PID controller to control the position.

4.3 Theoretical approach additional application

The following sections discuss possible applications that can be made with this admittance controller and by extension impedance control.

4.3.1 Bolt in hole application

Situation sketch

In this first section, an application is discussed that was meant to work out in practice. Due to a lack of time, this was only discussed theoretically. It concerns an application to screw a bolt into a not exactly drilled hole with a robot arm. Because a robot arm is usually very stiff, it is difficult to make small adjustments to the position when the robot is not exactly positioned to turn a bolt into a hole.

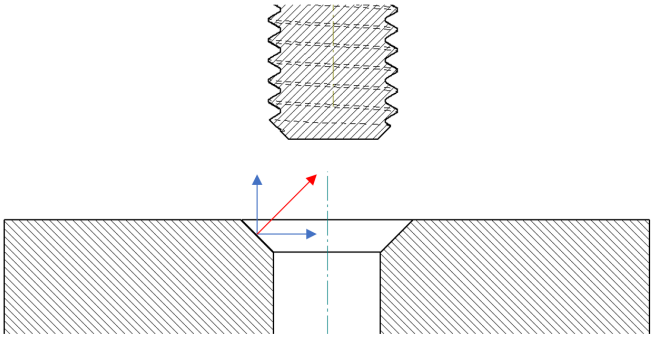


Figure 4.4: Application sketch

For this application it is assumed that the precipitation of the hole is correct up to a maximum deviation of the horizontal length of the chamfer of the hole (s is less than c in Figure 4.5). Because of this, there will always be contact with the chamfer of the bolt and the hole, so that the forces that occur can always be resumed in the forces shown in Figure 4.4. Since these forces are measured in base frame, they can always be related to the workpiece.

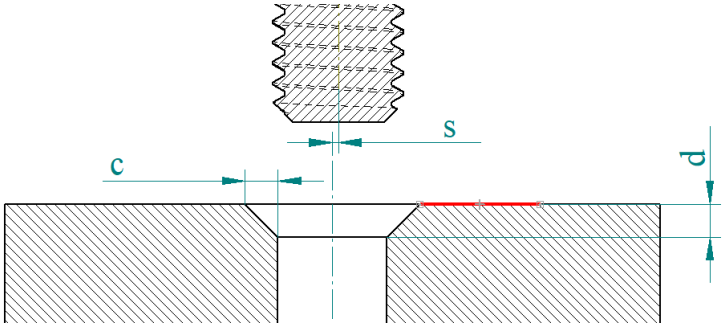


Figure 4.5: Situation sketch

Program sequence

The robot will first use the bolt to check whether there is a workpiece and if so at what distance. To do this, the robot will make a slow downward movement next to the hole (red strip in Figure 4.5) until a certain force is reached in the vertical direction. When completed, the robot will move to the preset position above the hole. It will then sink until the lower surface of the bolt is parallel to the surface of the workpiece. When this is achieved, the admittance control and turning of the bolt at a certain speed proportional to the vertical displacement speed is activated. Then the bolt slowly sinks with the set value, the end position of the bolt fully screwed in. When the spring damper mass principle works in the three translation axis, the bolt will move itself to the middle of the hole, thus obtaining a grip on the thread. When the bolt is at its end position, it can be chosen to provide a certain tightening torque, which can also be measured with the robot arm force measurements.

Reflections on this design:

- If the bolt has found the middle of the hole, there will be no force in the horizontal direction. Because of this, the target value in x and y direction must always be updated to the calculated values of the admittance controller. Taken the axis of the bolt and the hole seen as z axis.
- This method does not take into account deviations in the parallelism of the workpiece surface to the xy plane. This can be incorporated into the probing of the workpiece on the red surface of Figure 4.5. It is always known that the contact force is perpendicular to this surface. If the z axis positions itself according to this perpendicular axis, the same method can be obtained.
- In the admittance controller a difference has to be made in stiffness. The force in z direction gradually increases because the target also drops lower. If the same stiffness is taken for the z axis as for the x and y axis, a too large displacement will be observed. If l equals the hole depth and c is the horizontal length of the chamfer, we can state the following:

$$F_{zmax} = l * K_z \tag{4.3.1}$$

$$F_{xymax} = c * K_{xy} \tag{4.3.2}$$

  As the force in z direction equals the force in x and y direction due to the chamfer of 45 degree and l is normally larger than c. The spring coefficient in z axis (K_z) must be smaller than the spring coefficient in x and y direction (K_{xy}).

4.3.2 Other applications

With force control in general, many different applications can be made. It is especially interesting to use this when there is interaction between the robot and its environment, as in the previous applications. For general moving applications however, this can be used less well, due to the control error that occurs. For force control, for example, sanding a surface with a robot arm can be used, in this case impedance control would be applied. Also when gripping fragile products, this can certainly be used when, for example, a maximum force can be applied to an object, for example fruit. When pressed too hard, rotten spots can occur, which is not desirable. Admittance control can mainly be used when any environmental force acts on the system. For example, helping to move a heavy object, is a good application.

5 Results

This chapter presents and discusses the results of the previous MATLAB and Python experiments. Screenshots and graphs are used to show the end-effector coordinates and the joint configurations in function of time and the applied force. These graphs are also discussed and indicated for any deviations from expected results or matters that could be improved.

Due to the corona measures everything is kept in simulation environment. However, this also means that all results displayed are simulation data. It can be said from experience that these simulations cannot always replace practical tests.

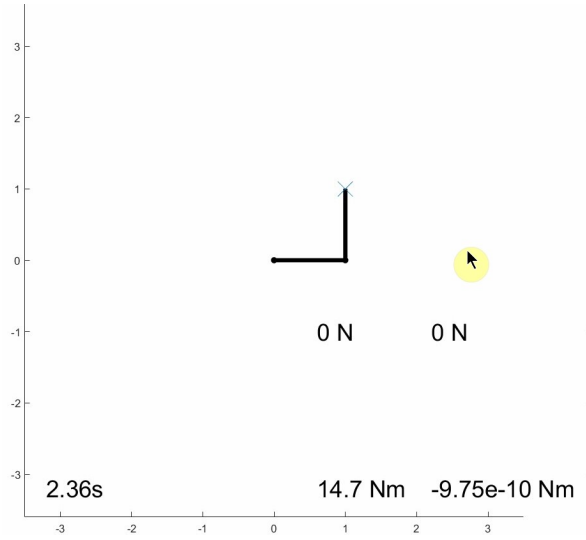
For each of the experiments in chapter 4 an overview is given of the obtained results. First the results of the MATLAB experiment of the two degrees of freedom manipulator are given. In this experiment the goal is to create an impedance controller where the link has a fixed point as target. In addition, the influence of the dynamics on the system was considered with different input parameters. A second experiment was to follow a trajectory with the same MATLAB controller. This section was split into two parts: a first part where admittance control is done in Cartesian coordinates and a second part where admittance control is done in joint space. Finally, a six degrees of freedom admittance control was made in MATLAB. The goal was demarcated to a fixed point due to the slow operation of the application. Eventually, the switch was made to the Python controller, using all previous techniques to design a force compliant controller that follows a path and where an external force can act as disturbance.

5.1 Two degrees of freedom MATLAB simulations

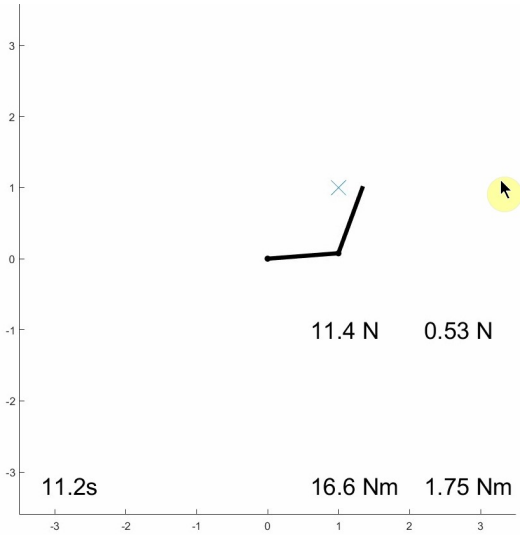
5.1.1 Simulation results fixed target point

Admittance control fixed point

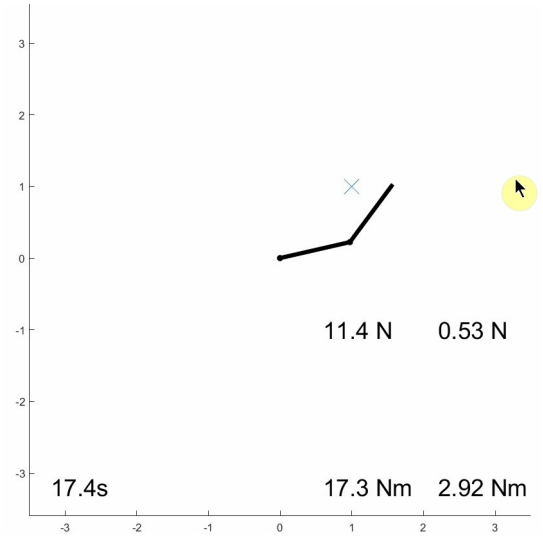
For these first simulations a general MATLAB figure is built. In this application, clicking with the mouse and dragging in the figure creates an external force in the desired direction, which also appears on the screen. In addition, the joint torques that must be generated due to the dynamic behaviour of the manipulator are also on the screen as shown in Figure 5.1. In this application there is no absolute need to calculate the joint positions. This is informative and can be used in subsequent applications. That's why it's not shown here.



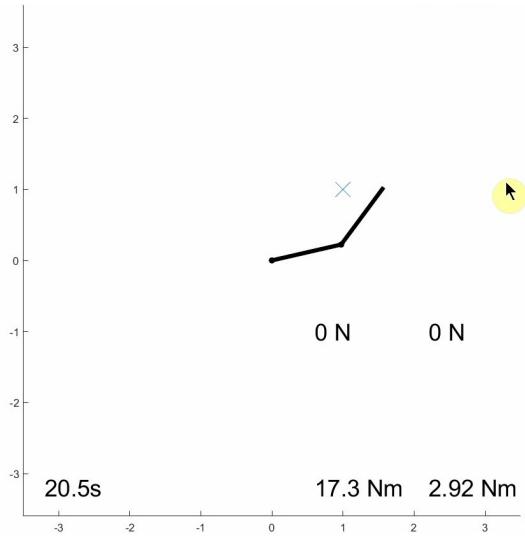
(a) Manipulator without any external force, rest position



(b) Manipulator moving due to an external force



(c) Manipulator just before the force is released



(d) Manipulator goes back to its rest position

Figure 5.1: Manipulator from rest position to steady state position with applied force

In Figure 5.1a the manipulator is in rest at the target position. Thereafter an external force is applied due to mouse dragging; the result is shown in Figure 5.1b. The manipulator then reaches its new steady state position in Figure 5.1c. After the external force is released, the manipulator returns to the target position as shown in Figure 5.1d. The following figures show the data plots from the trajectory followed by the manipulator.

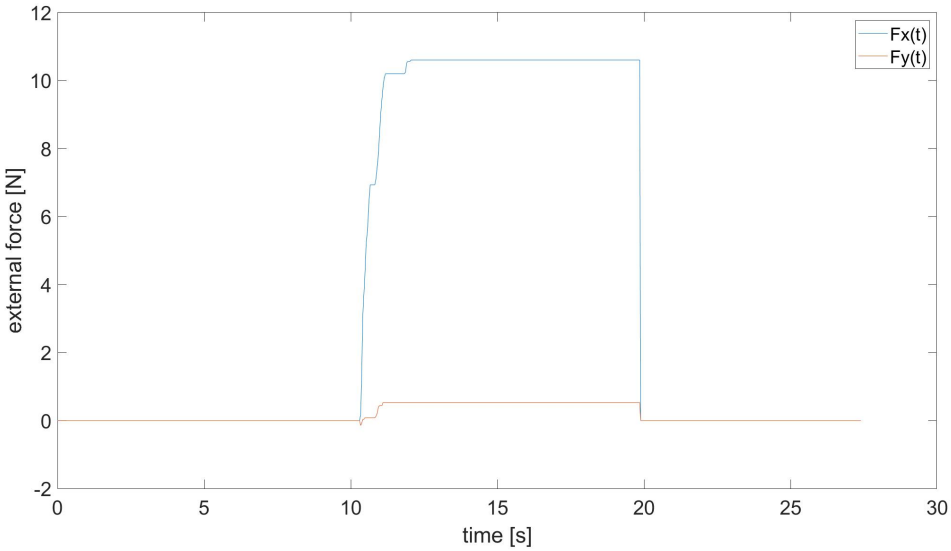


Figure 5.2: Forces in x and y

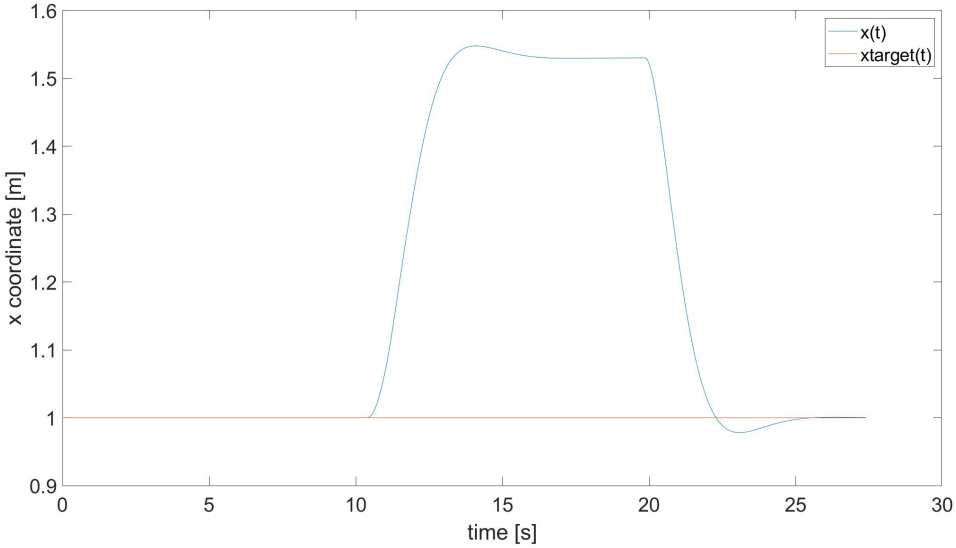


Figure 5.3: Coordinates in x

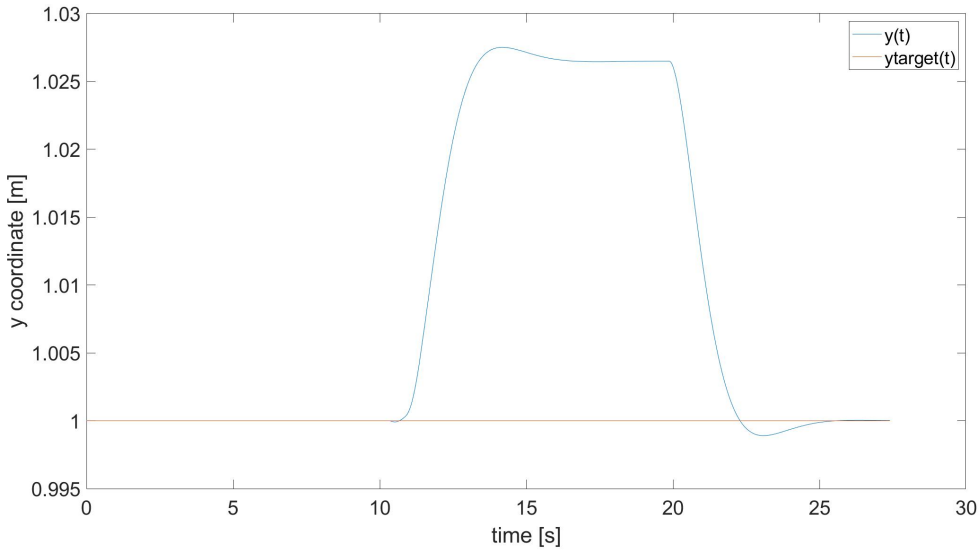


Figure 5.4: Coordinates in y

In Figure 5.3 it is visibly clear that there is a large deflection due to the external force in the x direction given in Figure 5.2. We also see a good, fast and well damped response. If desired, either the mass coefficient can be set lower or the spring constant higher increasing the natural frequency. However, the faster the system responds, the higher the torques will be. Finally Figure 5.4 shows us the deflection in y-coordinates, which is significantly small due to a small external force.
All previous figures are included in a test where the length of the links are one meter and the mass is one kilogram. The parameters for the spring damper system are respectively 20N/m and 10kg as mass. The damping factor was always calculated on the basis of a damping coefficient of 0.7. This also gives a damping factor of 19.80Ns/m for $Kd = 2 * \zeta * \omega_n * M$.

A small check can be carried out on the correctness of the results obtained. Since the speed and acceleration targets are zero, the steady state velocity and acceleration are also zero. The general equation of impedance control can be simplified to equation 5.1.1.

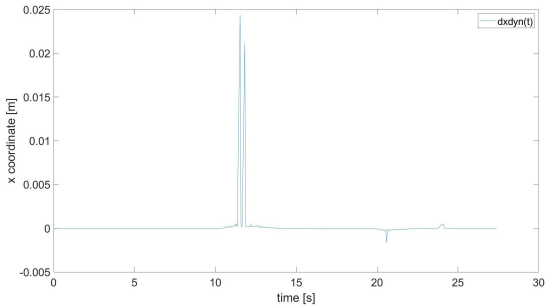
$$\begin{aligned} 0 &= F_{ext} + K_p(x_{tar} - x) \\ 0 &= 11.4 + 20(1 - x) \\ x &= \frac{20 + 11.4}{20} \\ x &= 1.57 \end{aligned} \tag{5.1.1}$$

This can be checked by means of graph 5.3. There, a displacement relative to the reference due to the external force of 11.4N in x direction can be observed equal to 0.57m.

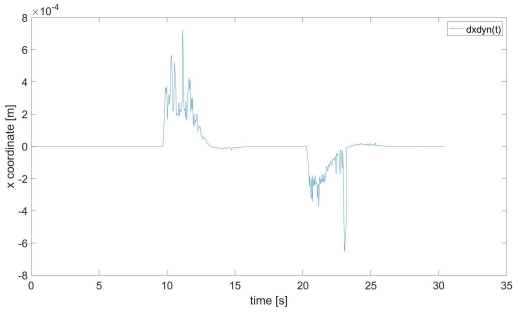
Note: When K_p is lower than 10N/m, the natural frequency must be higher than 4,5 rad/s in order to have good performance. When the end-effector velocity is too high, the inverse kinematic solver is not capable to find an accurate solution and the application will go crazy.

Influence of the manipulators dynamics

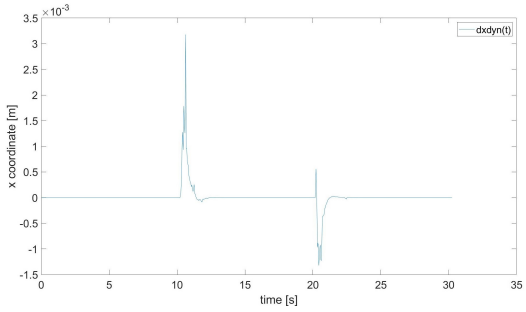
A second part of the results had to show if the dynamics of the manipulator had a major influence or not. For this, some tests were done in which the parameters of the links as well as the spring damper parameters were changed.
The first set of parameters are: link mass equals 10kg, link length equals one meter, the spring coefficient equals 20N/m, the virtual mass equals 10kg and the damper is calculated using a dampcoefficient $\zeta = 0.7$ and equals 19.8Ns/m. If we integrate the dynamics, we can plot the difference between the coordinate determined without and with dynamics, as in Figure 5.5a. From Figure 5.5a to 5.5d the same is done with varying parameters listed in the capture of the figures.
To generate this graphs the same routine is used: an external force in the x-axis is applied arround 15N just after the 10th second and the force is released just after the 20th second.



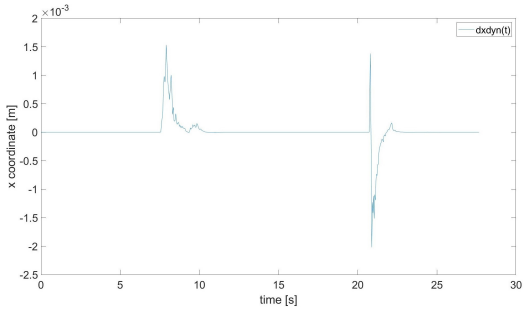
(a) $Kp = 20N/m, M = 10kg, mass = 10kg, linklength = 1m$



(b) $Kp = 20N/m, M = 10kg, mass = 1kg, linklength = 1m$



(c) $Kp = 20N/m, M = 1kg, mass = 1kg, linklength = 1m$



(d) $Kp = 20N/m, M = 1kg, mass = 10kg, linklength = 1m$

Figure 5.5: Error due to dynamic integration

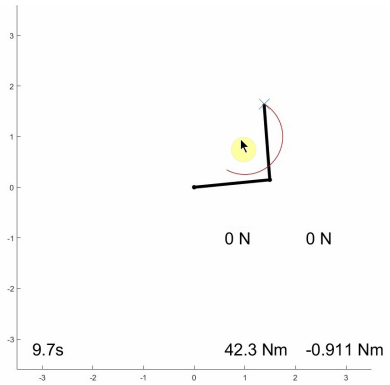
The graphs in Figure 5.5a to 5.5d show us the difference between the normal impedance controller error and the error caused by the impedance controller charged with the dynamics of the manipulator. Since the scale of the y axis is a factor of a thousand or ten thousand smaller, we can say that the influence of the dynamic parameters here is minimal. The maximum difference between both errors is approximately $24 * 10^{-3}$. Given that the spring constant of the impedance controller is 20N/m and the applied force is arround 15N, the diversion of the manipulator is about 0.75m. The error difference between the two methods is significantly small, so the dynamics of the manipulator can be omitted. However, if very large accelerations have to be performed, the error will become larger and larger. This has to be kept in mind.

5.1.2 Simulation results circle tracking

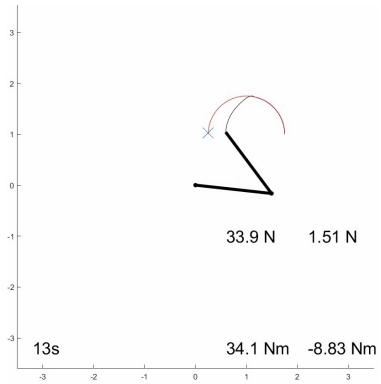
These simulations are an extended version of the previous test. It allows the manipulator to follow a trajectory and reacts on an external force. The second part of this test is to get an idea of how the manipulator behaves when impedance control is applied in Cartesian or joint space.

Cartesian space

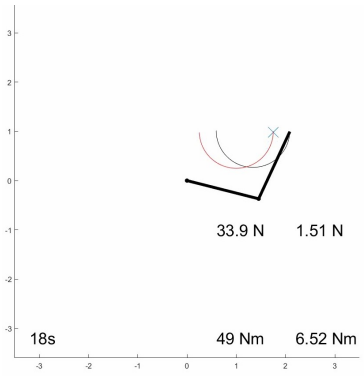
The same program is used as in the fixed point simulation for this application, only a little change has been made in the following trajectory. Here the choice is made to create a circular trajectory with a certain rotation frequency and amplitude. In Figure 5.6 a few screenshots are taken to illustrate the working principle.



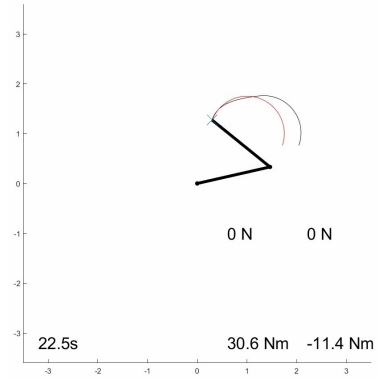
(a) Manipulator before a force is applied



(b) Manipulator going to steady state when force is applied



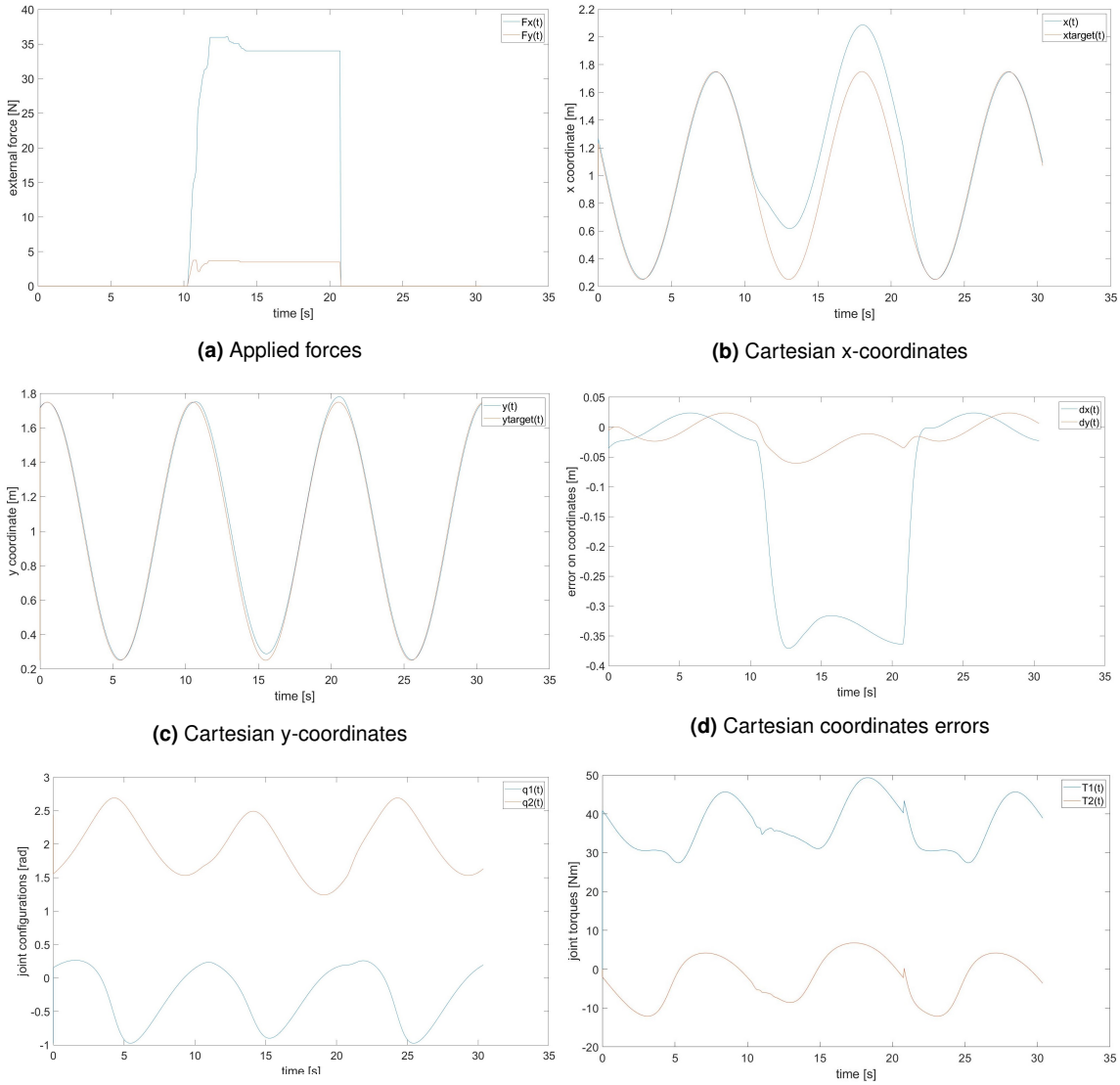
(c) Manipulator with force in steady state



(d) Manipulator without force in steady state

Figure 5.6: Manipulator from rest position to steady state position with applied force and back

In the following graphs in Figure 5.7a the same motion is recorded in several graphs from applied forces and Cartesian positions to joint positions and torques. Finally the graph in Figure 5.7d is made where the errors in Cartesian coordinates are made.



(a) Applied forces

(b) Cartesian x-coordinates

(c) Cartesian y-coordinates

(d) Cartesian coordinates errors

(e) Joint positions

(f) Torques

Figure 5.7: Graphs obtained from the execution in Figure 5.6

In Figure 5.7a the applied force is shown. As a result, a diversion mainly occurs in x direction, shown in Figure 5.7b. Figure 5.7d shows us that there is always an error in each direction. This was not expected as a result, but when the theory was reconsidered, this is one of the features of impedance and admittance control. The impedance/admittance control law needs to have an error on acceleration, velocity or position to move.

If the deviation from the reference must be zero, a PI controller can be switched on if no external force is acting on the system. In order to get a good control, a PI controller must be made for each joint separately. For this, the dynamic system needs to be described exactly. In this two degrees of freedom example, this can still be determined. When a six degrees of freedom application is made, it is already more complex due to the variable dynamic parameters.

In this example it is not clear that the dynamics play a role in the error that occurs. Should this prove to be the case, feedforward control can be applied. In this case, the influence of the dynamics is already eliminated by determining the inverse of the system and adding this to the result. If necessary, this can be integrated as given in an article about model-based feedforward control for motion systems [53]. This article describes the feedforward method that could be used for a two degree of freedom example. For a six degree of freedom example this is more difficult as an exact description of the system or a good approximation is needed.

Figure 5.8 shows an example of what a feedforward control can look like. In this application, the conventional controller is equal to the admittance controller and the neural controller equals the inverse dynamics of the robot arm.

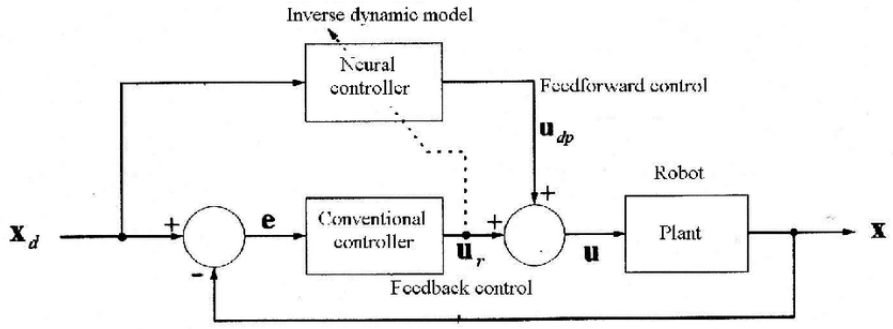
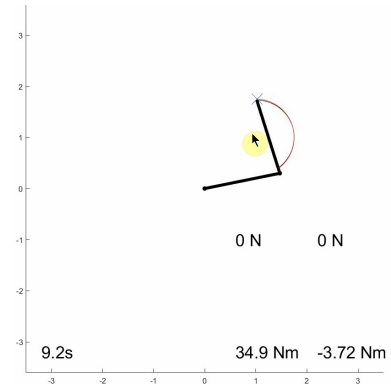


Figure 5.8: Feedforward control robot [27]

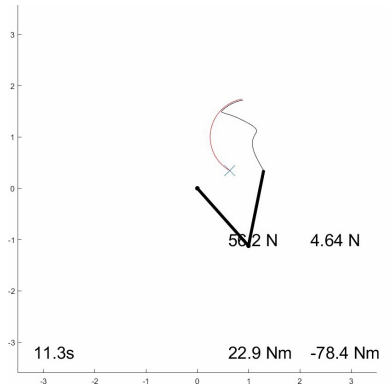
The same can be done to eliminate the error on the target. However, this will eliminate the dynamic system desired for admittance control and the response of a second order system will no longer be visible. Since in this thesis the goal is to obtain this behaviour, this is not done.

Joint space

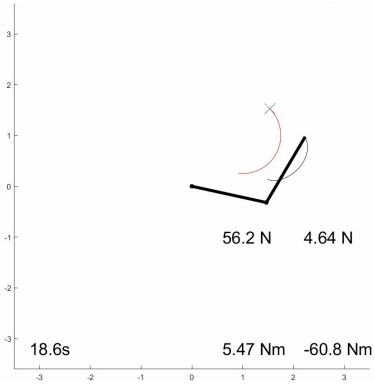
In this section the MATLAB graphs are represented of the two degree circle tracking approach in joint space. First of all, the visualisation of the response in Cartesian space is represented the same way as in the previous section. First no force is applied in Figure 5.9a. Next a force is applied in Figures 5.9b and 5.9c. Finally the force goes back to zero and the manipulator goes back to the desired path, Figure 5.9d.



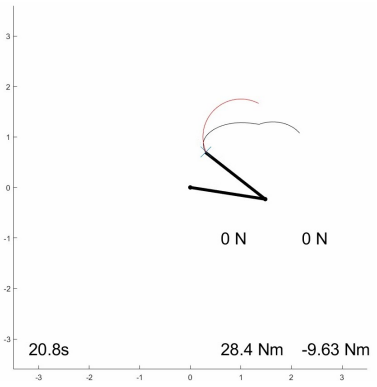
(a) Manipulator before external force is applied



(b) Manipulator just after external force is applied



(c) Manipulator in steady state when force is applied

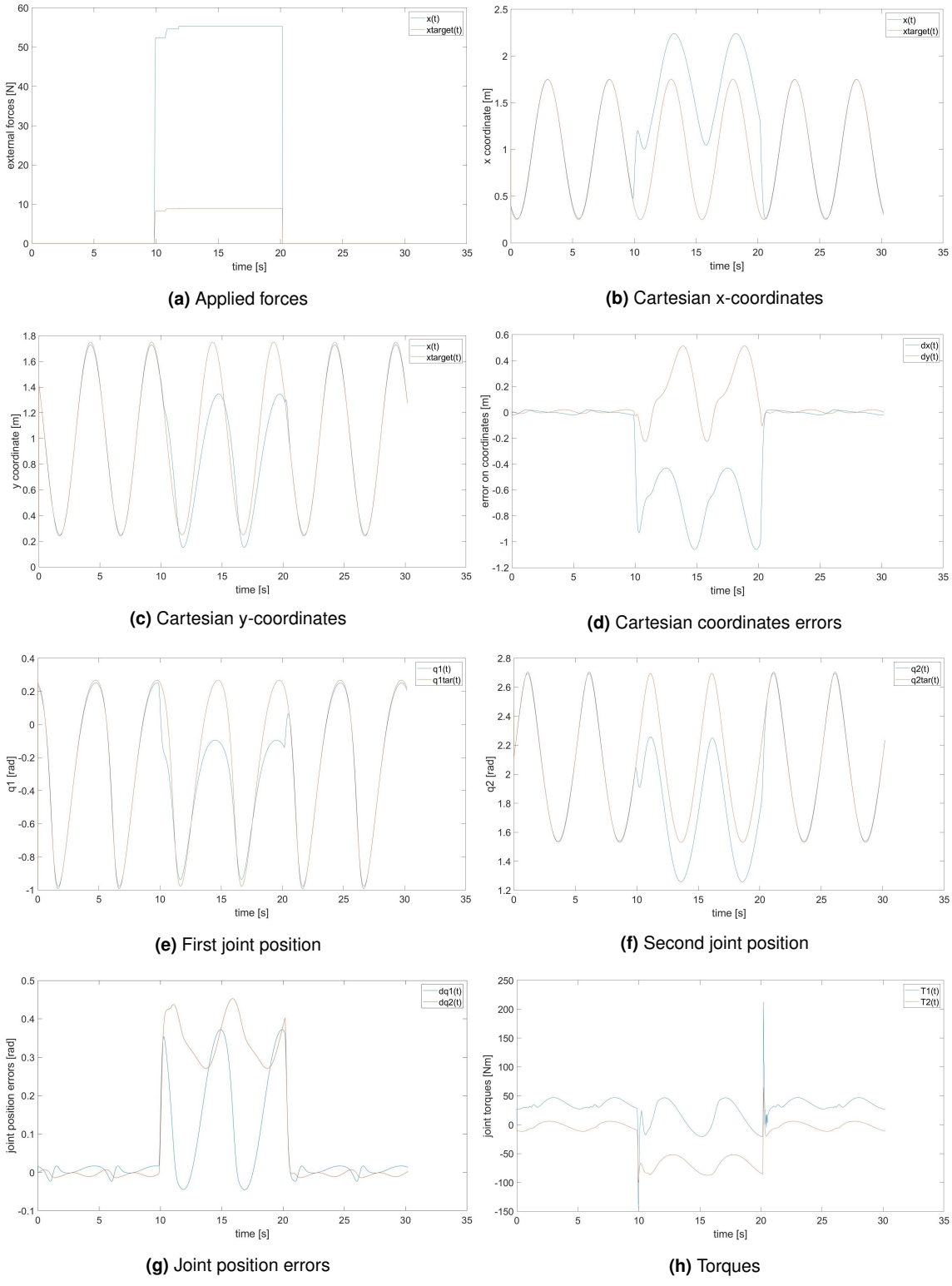


(d) Manipulator going back to desired position

Figure 5.9: Manipulator from rest position to steady state position with applied force and back

In this implementation we can see from Figure 5.9c that the path followed by the manipulator no longer corresponds to what it was in the Cartesian space control method. This makes sense since we control the joint positions and the transformation matrix provides this distortion in Cartesian coordinates.

As in the previous section, there are different graphs showing the configuration in Cartesian as well as in joint space. This gives us a better insight into what happens in reality. The displayed manipulator in Figure 5.9 shows us what happens in Cartesian space. In Figure 5.10a is shown when the external force did apply, from the 10th second until the 20th second. In Figure 5.10, these moments are extended to see the influence of the applied force.



(a) Applied forces

(b) Cartesian x-coordinates

(c) Cartesian y-coordinates

(d) Cartesian coordinates errors

(e) First joint position

(f) Second joint position

(g) Joint position errors

(h) Torques

Figure 5.10: Graphs obtained from the execution in Figure 5.9

Figures 5.10b and 5.10c give a picture of how the x and y coordinate progress. There is a smooth transition in the coordinates when a force occurs as seen in the overview of Figure 5.9. Unlike the experiment in Cartesian space, we cannot conclude that the error in both x and y directions in Figure 5.10d remains approximately constant at steady state. However, if we look at what happens in joint space, we see that in Figures 5.10e and 5.10f the spring, damper and mass control is more visible. When the external force is released, an adjustment to the target is visible with a small overshoot. Looking at the error on the joint positions, in Figure 5.10g is shown that this is also not constant. This can be explained by the fact that the external force creates a torque that is not constant because it depends on the Jacobian or the joint configuration and it changes over time. Finally, we can see in Figure 5.10h that when the external force is removed, a peak in torque occurs due to the acceleration. Thereafter it seems that the value before the peak in the first joint and after the peak is almost the same. While for the second joint it has a certain offset, due to the displacement of the target position and the resulting change in gravitational compensation torques.

Issues

A major problem arose during execution. The inverse kinematics of MATLAB's rigidbody toolbox were too slow for this application, about seven milliseconds each calculation. As a result, a closer look was taken at the function. The possible adjustable parameters were looked up in order to obtain an accurate result. Since the MATLAB rigidbody class was used to create the manipulator, the inverse kinematic solver of this class was also used.

An inverse kinematic solver needed to be defined using the function: inverseKinematics('RigidBodyTree', 'SolverAlgorithm', SolverParameters'). On the MathWorks site about the inverse kinematic solver, the two different solvers are explained in detail, according to which principle they work and what parameters can be given [54].

First, there is the BFGS gradient protection algorithm that could be used. This solver algorithm is based on the quasi-Newton method and uses the second derivative to determine the step that must be taken in the second iteration. A gradient projection method is used to deal with boundary limits on the cost function that the joint limits of the robot model creates. The parameters that can be obtained with this algorithm are: maximum number of iterations, maximum time, gradient tolerance, solution tolerance, joint limits, if random restarts are allowed and the minimum step that is allowed.

Secondly, there is the Levenberg-Marquardt algorithm. This algorithm uses an error-damped least-squares method. In addition to the BFGS gradient projection algorithm, this algorithm has some extra input parameters such as: threshold on the change in end-effector pose, a constant for damping and an indicator for the use or non-use of the damping.

Both algorithms could be studied in detail but this was not the purpose of this thesis. For more information, reference is made to the MathWorks site about the BFGS algorithm and the Levenberg-Marquardt algorithm [55][56]. In general the BFGS algorithm is set default because it is more robust at finding solutions and it's more effective for configurations near the joint limits or when the initial guess is not close to the solution. If the initial guess is close to the solution, the Levenberg-Marquardt algorithm is faster. In this application the previous joint configuration was known and therefore the Levenberg-Marquardt algorithm was prefered [54].

5.2 Six degrees of freedom MATLAB simulations

As in previous chapters, an overview is given of the manipulator's response to the external force in two degrees of freedom examples. This experiment is about the translation of those experiments to a six degree of freedom manipulator. At this point another library was used where the plotting was integrated. In Figure 5.11 the visualistation of the UR3 robot is given. At this point the manipulator is moving in the positive x direction due to an external applied force applied on the end-effector. This external force is applied by using a slider which has a value between 0N and 20N in the positive axis direction.

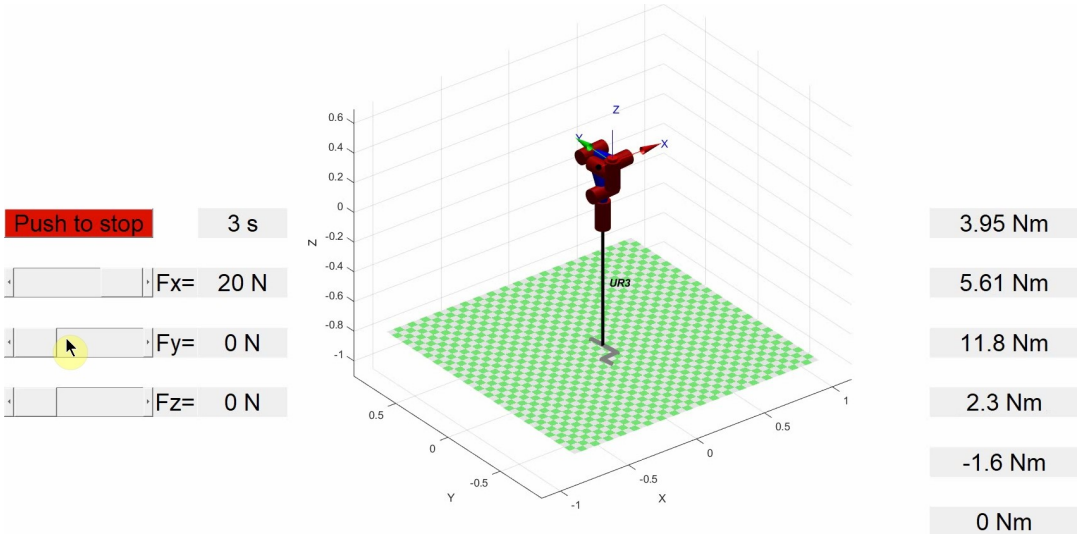
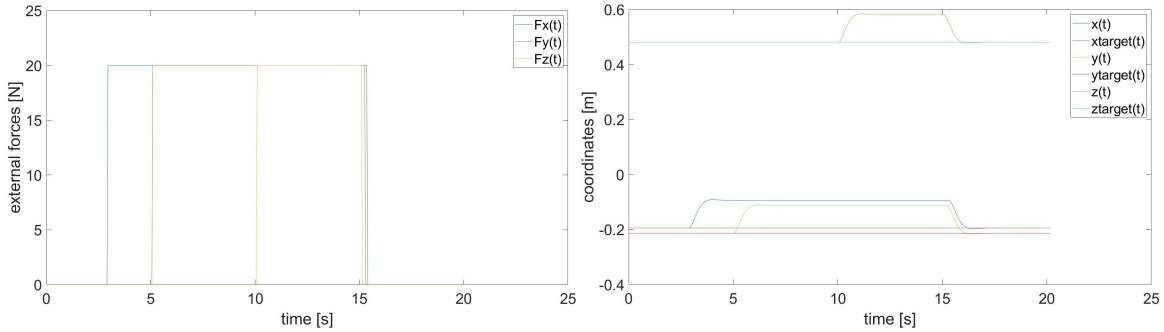


Figure 5.11: Manipulator with external force in x direction

Since this previous figure is not very clear to view the displacement due to the external force, the graphs in Figures 5.12a and 5.12b are a better reference for deciding that the application works fine. The extensive graphs for the joint configurations and others have been omitted, as this is of little importance since the admittance control is conducted in Cartesian space. In addition, Figure 5.13 has been added, showing the joint torques.



(a) Manipulator without any externale forces

(b) Change in Cartesian coordinates

Figure 5.12: Applied external force and the corresponding displacements

This response can be compared to the expected response shown in Figure 4.2. There a rising time of 477ms is measured. Since a sample time of 50ms is taken, this cannot be observed at the same precision. Table 5.1 shows the measuring points in which the rise time can be determined.

	y value [m]	**time [s]**
rest state	-0.2132	5.05
10%	-0.2032	5.25
90%	-0.1233	5.75
steady state	-0.1133	7.15

Table 5.1: Rise time sample points

From Table 5.1 it can be concluded that the system responds as previously desired, since the rise time between the two sampled points is approximately 500ms.

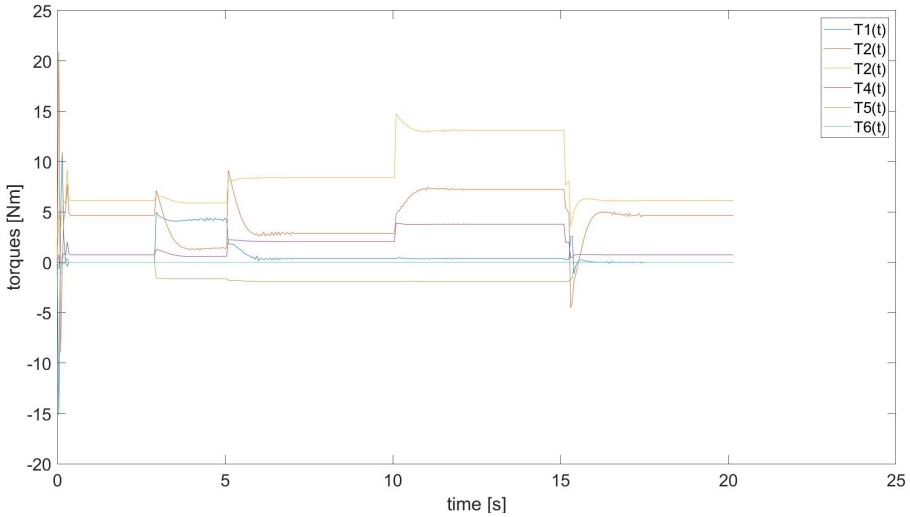


Figure 5.13: Joint torques

The maximum moments are never exceeded in Figure 5.13. If this does happen in the program, another joint acceleration with the maximum torque is calculated on the basis of the dynamics of the manipulator. However, this means that the path that has been followed no longer follows the desired trajectory, the coordinates are adjusted in function of the accelerations of the joints, so that an unexpected movement can occur.

The maximum torques are $[56,56,28,12,12,12]Nm$ for base, shoulder, elbow, wrist one, wrist two and wrist three. If for example the torque of the elbow must exceed the maximum torque of 28Nm for instance $[20,14,32,5,4,3]Nm$, the maximum value will be inserted in the actual torques $\tau_{adjusted} = [20,14,28,5,4,3]Nm$ and a new joint acceleration will be calculated using equation 5.2.1. The new final coordinates wil then be determined for this, as has been done previously.

$$\ddot{\theta} = M(\theta)^{-1}(\tau_{adjusted} - C(\theta,\dot{\theta})\dot{\theta} - G(\theta)) \tag{5.2.1}$$

When starting the application we see that the torques vary quickly. This is because the plotter automatically starts on its home configuration and then moves to the desired configuration as quickly as possible. As can also be seen here, the torques do not exceed the maximum torque of each joint. With the physical robot this transitional phenomenon will never happen because the initial configuration will always be known and will not be far from the target. This means that no oversized torques will occur in the joints. If the desired joint configuration that is sent to the robot must be reached way to quick, the internal PID controller will slow this down. Due to this the target will not be reached at the desired time but the maximum torques will not be exceeded as well.

Issues

As in previous experiments, a too slow inverse kinematics solver was also a problem here. Because the inverse kinematics solver of the Peter Corke toolbox has no input parameters that can be customized, we opted for a slow motion simulation with a sample time of 50ms. Because of this, the graphs are representative, but it takes longer to record them.

5.3 Six degrees of freedom Python simulations

Finally an admittance control application is developed using multiple classes. The Python code and MATLAB code are available on my GitHub repository and will not be updated anymore [49]. The application uses multiple classes: two communication classes, a robot discription class and some functions that include pad generation and admittance control. The repository with all the created and used code, gives the best result. In addition, the physical operation can also be used to look at the data that is stored in an Excel file. In order to preserve clarity in graphs, it was decided to process this data into a few graphs using MATLAB.
Since this experiment was only performed in simulations, a small GUI was created to simulate the external forces. In Figure 5.14 the layout of the GUI is given. It shows three simple sliders each going from -50N to 50N. The exit button is applied to end the continuous while loop that's created to simulate a continuous task. If it proves necessary to be able to control the rotation axes, only three sliders need to be added. Since the same method is used for the calculations for each axis and rotation axis, it is limited to three forces for simplicity and visualization.

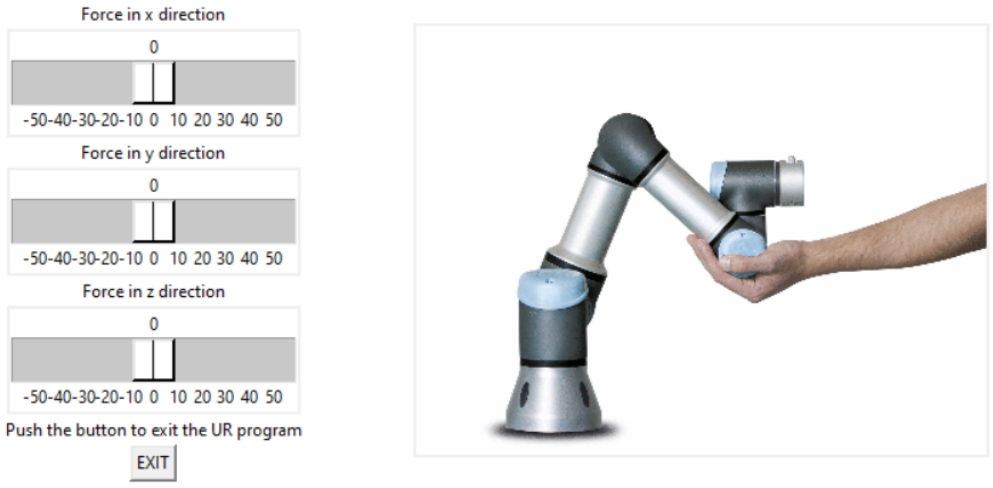


Figure 5.14: GUI layout

In Figure 5.15 the external force is plotted into a graph. In the title of this figure all used parameters are given such as the spring, damper and mass constants that are used and which axis are controlled by the admittance controller. In this case: $Kp = 500N/m$, $Kd = 22Ns/m$, $M = 0.5kg$ and $axis = [1,1,1,0,0,0]$.

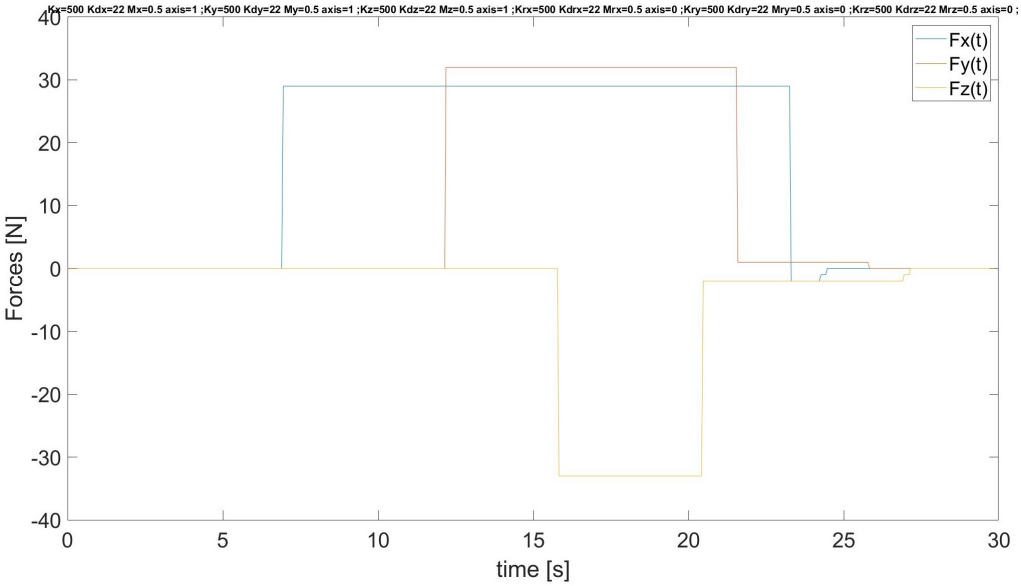


Figure 5.15: Simulated applied external force

Due to the external force in Figure 5.15 a response of the robot also appears. This is shown in Figure 5.16 and, as in other chapters, it has been viewed with respect to the target.

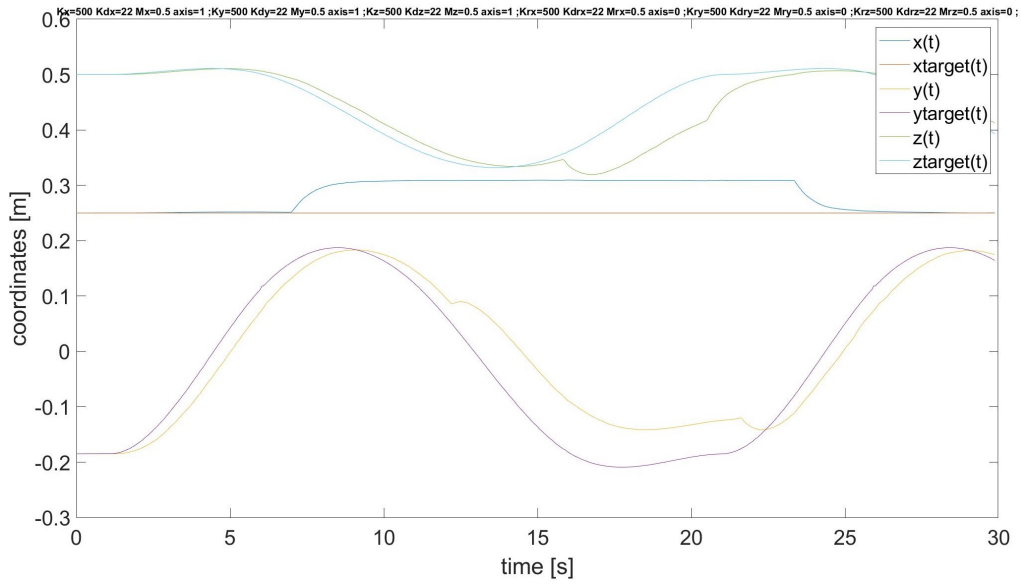


Figure 5.16: Response on the external forces small mass

The target in x direction stays constant in Figure 5.16 as the robot's path is made in a plane parallel to the yz plane of the robot's base. Since the simulated mass of the mass damper spring system was chosen equal to 0.5 kg, the response by the external force is sudden. This is not always the intention. If the mass is chosen larger, namely five kg as in Figure 5.17, then we see that the impedance control causes a large error on the trajectory to be followed.

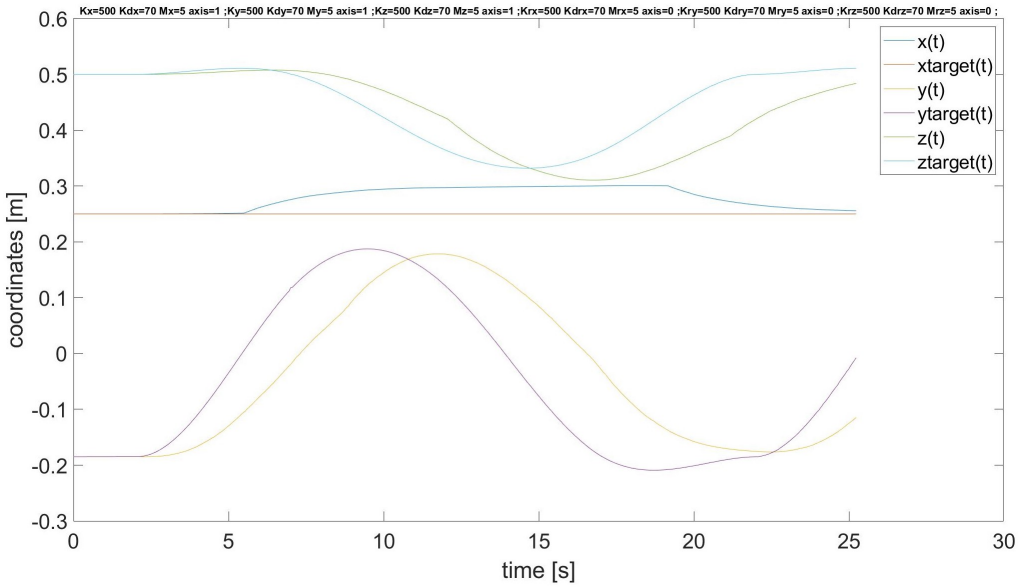


Figure 5.17: Response on the external forces large mass

If the virtual mass becomes zero, we can see in Figure 5.18 that an error still occurs due to the damper and the robot controller. This case zooms in on the robot controller and the error that could be obtained by the servoj URScript command.

A joint position is sent into the robot controller. The robot controller will move the end-effector of the robot to this position as soon as possible. The gain in the controller to reach its desired point, can be given by the servoj command. This gain occurs in the position control loop shown on page 23 of the literature study. This gain can vary from 300 to 2000 [50]. In Figure 5.18 it was placed at 500 in comparison to Figure 5.19 where the gain is placed at 2000. However, when a virtual mass or a virtual damper is added to the system, the error will increase due to the admittance control.

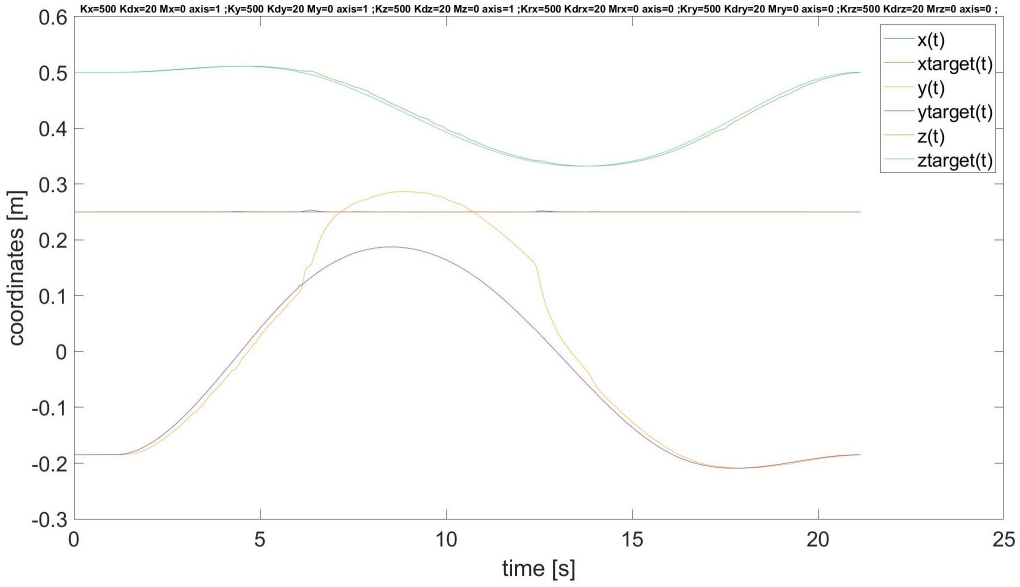


Figure 5.18: Response with servoj gain equal to 500

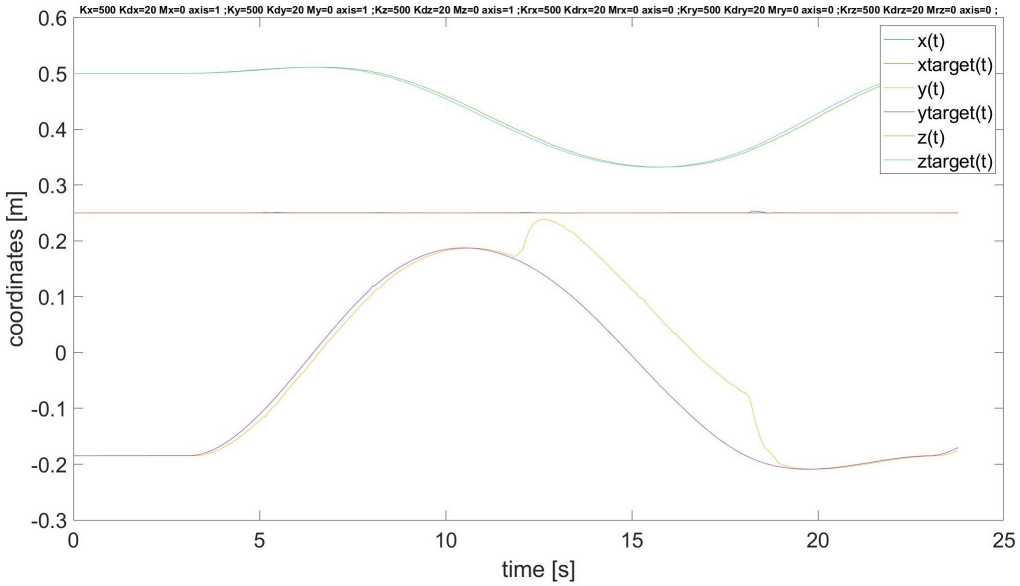


Figure 5.19: Response with servoj gain equal to 2000

Nothing meaningful can be derived from the graphs in Figure 5.18 and Figure 5.19. If we look at the data of the target position and the real position in Tables 5.2 and 5.3, we see that when the target is equal to 0.060... for example, both errors are approximately equal. As a result, it can be decided that the error that occurs can only be caused by the virtual mass and damper.

yforce	yposition	targetyposition	deltay
0	0,043986	0,056224	-0,01224
0	0,047319	0,06004	-0,01272
0	0,050706	0,063822	-0,01312
0	0,055024	0,067569	-0,01254
0	0,058828	0,071279	-0,01245
0	0,062537	0,07495	-0,01241
0	0,065018	0,07858	-0,01356
0	0,067771	0,082167	-0,0144
0	0,072929	0,085709	-0,01278
0	0,076167	0,089204	-0,01304

Table 5.2: Data record servoj gain=500, Figure 5.18

yforce	yposition	targetyposition	deltay
0	0,042441	0,056224	-0,01378
0	0,046845	0,06004	-0,01319
0	0,05056	0,063822	-0,01326
0	0,054652	0,067569	-0,01292
0	0,057902	0,071279	-0,01338
0	0,062412	0,07495	-0,01254
0	0,065637	0,07858	-0,01294
0	0,06894	0,082167	-0,01323
0	0,072338	0,085709	-0,01337
0	0,075893	0,089204	-0,01331

Table 5.3: Data record servoj gain=2000, Figure 5.19

Since the error increases at higher speed, there is no problem when a precision task has to be performed. The speed becomes so low when approaching the final point, that the error becomes almost zero. This is shown in Figure 5.18, where the x-coordinate is constant, the robots x-coordinate almost equals the target x-coordinate. When looking at the z-coordinate, this can also be decided. These coordinates change more slowly over time, which means that a smaller error occurs. If the error has to be eliminated, it is possible to use a PI controller or feedforward control. This can converge the error to zero. Because one of the characteristics of the admittance control is that it always causes an error, which is not the aim of this thesis. As mentioned in the literature study, this method of control should not be used when an exact position needs to be reached.

In chapter 4.1.1 it is noted that the three variables, position, velocity and acceleration, were converted to one variable. At that time, this seemed like a logical approach, as the new configuration changes all variables. This was followed up once to see whether this was the case. Figure 5.20 shows the result of this first approach. In Figure 5.21 only the acceleration needed in this point is determined. When we make the conversion to the speed and position, we obtain the new target corresponding to the acceleration that must occur in the current position. Both graphs are made with virtual mass equal to 5kg, damper equal to 20Ns/m and a spring equal to 500N/m.

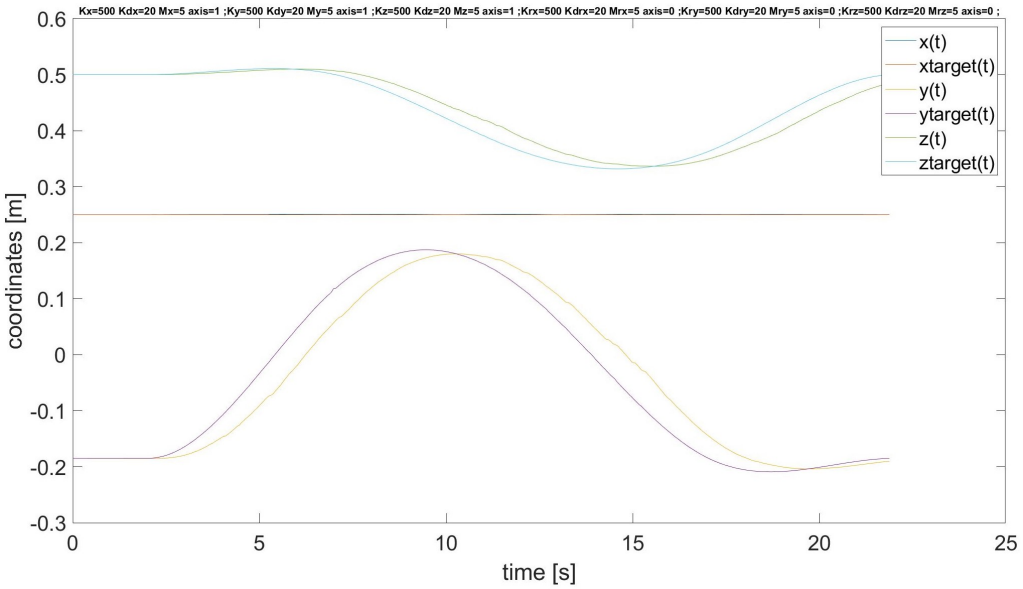


Figure 5.20: Response in first approach

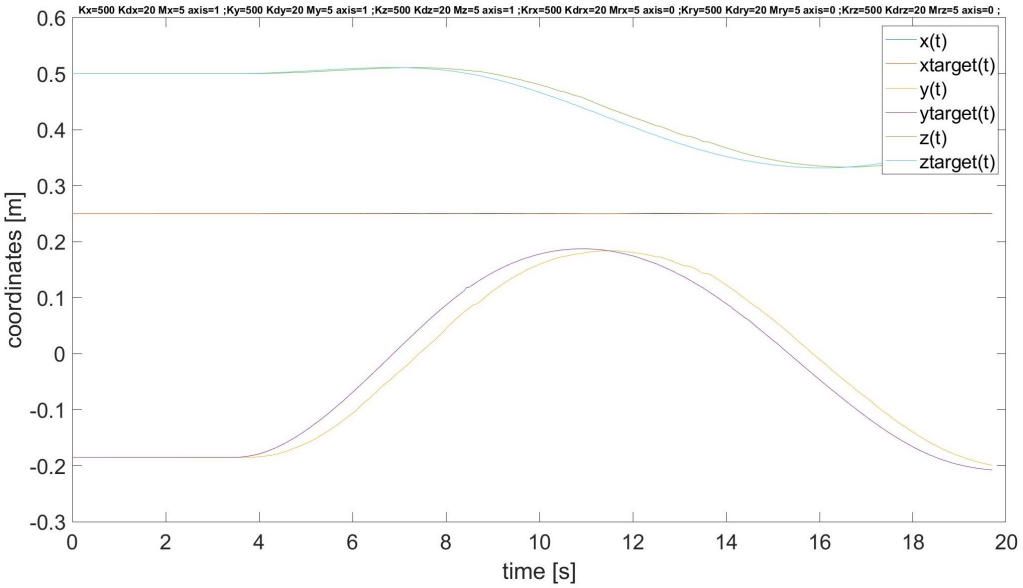


Figure 5.21: Response in second approach

The last column of Table 5.4 shows the difference in the y-coordinates between one method and in the other. It can therefore be stated that the second method leaves a smaller error in approaching the trajectory. Both data are recorded on the same target point, but the conclusion also applies to the rest of the measured values. Over the range defined for this test, the mean absolute error is 28% less in the second approach. This means that the admittance control is performed with the current values of the position and velocity. From this, the acceleration in that specific point is calculated and converted to a position by integration.

	time	yforce	yposition	targetyposition	delta y
first approach	4,612603	0	-0,1156	-0,06409	0,051513
second approach	6,061926	0	-0,10238	-0,06409	0,038296

Table 5.4: Difference in error between the two approaches on the same point in the trajectory

Issues

In this experiment several problems also occurred. This both in the communication to the robot, in the definition class of the robot as well as in the force output of the robot. In the following enumeration the different problems have been mentioned.

- **communication class:** Because the framework for the monitor class had already been made and only minor adjustments were made or extra functionalities were added, no problems were encountered. In contrast, the sending class encountered various problems. First and foremost, it was thought that a string should be sent and that the syntax was ultimately clear. A little later it turned out that a letter p always had to be placed before the array for the Cartesian coordinates. As a result, all functions that were controlled by Cartesian coordinates did not work at first. Later, during the testing, it also turned out that not all functions were made to send to the robot in real time. For example, the movel or movej functions do result in a very shaking robot. In the end, it was found on Universal Robots forums that the servoj function is the most used function for this. However, a relatively long time has to be given and the gain that is used is shouldn't be the maximum for a smooth course in the process. This caused the necessary delays in the configuration to be achieved. This is not a problem for this application, but for more precise applications it can cause problems.

- **robot definition class:** In previous experiments, the inertia of the robot links was calculated using an approach in which the links were presented as cylinders. Therefore this was not the exact inertia of every link. Because my aim was to work as accurately as possible, the distributors of Universal Robots were emailed. They replied they could not provide this information. When inquiring Universal Robots, this information apparently cannot be obtained. Because of this I went looking for another way to get it. On the wiki ROS page an urdf file was found containing all details of each link [57]. These inertias were then used. These seemed reliable, as many have used this in ROS projects according to GitHub.

  Later during testing it also became clear that the inverse kinematics were far too slow. The inverse kinematics calculations took an average of 0.2 seconds. Therefore, while real time is performed in this application, a higher operating frequency was desired. A closer look was taken at the cause of the slowness. It appeared to be the minimize function for which

the documentation can be found on the scipy docs site [58]. It was also encountered as a problem in MATLAB, where the Levenberg-Marquardt algorithm was chosen. Unfortunately, this is not reflected in this position. Many other algorithms were found such as: CG, BFGS, Newton-CG, L-BFGS-B and so on. Because every algorithm relies on different methods, it was investigated in an experimental manner which algorithm worked the fastest for this application. The previous joint position, a tolerance and the transformation matrix were provided as input. Because some algorithms need the Jacobian at the desired point, these could not be used. In general, the solution was always obtained between 30 and 500 milliseconds with a tolerance of 0.00001 on the joint angle solution. Only two algorithms stood out above the rest, namely the L-BFGS-B and SLSQP algorithm. For this, the time to find a solution was on average 30.0ms and 27.3ms. In table 5.5 an overview is given of the algorithms each with their calculation time with a certain tolerance. This table has been determined experimentally and is based on the average of ten randomly measured values in milliseconds.

Tolerance	**0.0001**	**0.00001**	**0.000001**
Nelder-Mead	171.6	116.0	225.7
Powell	371.3	453.7	469.6
CG	118.0	177.3	185.8
BFGS	38.7	44.4	46.0
L-BFGS-B	24.2	26.6	30.0
TNC	129.5	212.2	215.1
COBYLA	120.8	264.5	330.0
SLSQP	17.9	21.2	27.3
trust-constr	146.4	151.2	182.5

Table 5.5: Table with algorithm calculation times in milliseconds

A second problem was identified when calculating the dynamic parameters. These were determined on the basis of MATLAB functions. These functions have more than 2000 lines of code where the MATLAB syntax applies. Converting this to Python is an impossible task. As a result, a solution was sought so that the MATLAB functions could still be used in Python. Using the matlab.engine library was the easiest to work with [59]. Since Python 3.8 was installed, this was not self-evident since the library is only made for python 3.5, 3.6 and 2.7. Unfortunately this was not tested due to problems installing the necessary software in MATLAB and the corresponding library in Python.

- **robot force output:** Last but not least, the forces sent out by the robot are never zero. While at first it was thought that this turned out to be an external force, it is not. The dynamic forces are incorporated into this force. So it can be said that first the dynamic torques have to be calculated, then they have to be converted to forces on the end effector. These forces have to be subtracted from the obtained output of the robot, since no way was found to quickly calculate these parameters. This was not applied in this application either. However, the few measurements that were made showed that the dynamic forces were always relatively small so that they could be neglected. However, this will result in a dynamic behaviour that is not desired.

6
Conclusions

The goal of this dissertation was to create a force compliant controller and use it in an application. For this purpose, a fluent thesis has been provided with all the necessary information to bring this to a successful conclusion. Starting with the necessary knowledge and ending with some experiments and results.

This thesis mainly contains all the content that is needed to: (1) understand the kinematics of a robot, (2) describe the dynamics of a robot with multiple degrees of freedom, (3) choose a force control method and (4) apply all of this to a two or six degrees of freedom application. In addition, several things have been discussed such as the neglectable influence of the dynamics in the control loop and why for example it is not recommended to do force control in joint space.

In the first experiments, namely the one's in MATLAB, the admittance control loop was tested with a simple example such as a two degrees of freedom application with a fixed target. This application was gradually extended to an application with two degrees of freedom where a moving target was given. Finally a six degrees of freedom application was made in MATLAB where a fixed target had to be reached as well. After this, the switch was made to a Python controller for the UR3 robot.

In this Python admittance controller, different classes were created for this robot in order to establish communication between the program and the UR3 robot on the one hand, and to obtain a desired robot path and behaviour on the other hand. This program was tested and the obtained results were discussed. A correct operation is obtained in which the predetermined goal was achieved, namely to create a path tracking application in which an external force results in a defection. However, this was only tested in the URSim simulation environment due to the COVID-19 pandemic. A second application using admittance control is discussed in theory during the experiments, but was not made due to lack of time and the unavailability of labs due to the pandemic.

In the end, it is clear that this thesis created a profound base for future research and education purposes. People who are not yet familiar with robot arms and want to learn something about them quickly, can make use of the literature study. Those who start working with a robot from Universal Robots in combination with Python, will mainly find information in the written Python code that is available on GitHub [49]. Those who would like to know more about admittance control and how it works can consult this thesis and the MATLAB examples.

7
Bibliography

[1] F. C. P. Kevin M. Lynch, *MODERN ROBOTICS MECHANICS, PLANNING, AND CONTROL*, 2017.

[2] R. S. Andersen, “Kinematics of a ur5.”

[3] J. Dinesh, Robu.in.

[4] R. Y. A., “Geometric jacobians derivation and kinematic singularity analysis for smokie robot manipulator and the barrett wam,” *5th International Conference on Robotics and Mechatronics (ICROM)*, 2017.

[5] “Motor description,” ctrl-elec. [Online]. Available: http://www.ctrl-elec.fr/ctrl-clec/commande-de-moteur/commande-mcc/modelisation-et-commande-dun-moteur-cc/

[6] D. I. D. Vanoost, “Dc-drives,” 2018, course slides.

[7] S. L. Yoonho Kim, German A. Parada and X. Zhao, “Ferromagnetic soft continuum robots,” *Science Robotics*, vol. 4, no. 33, August 2019.

[8] G. Jeronimidis, “Materials strategies and systems for soft robotics,” 2014. [Online]. Available: http://www.robosoftca.eu/public_downloads/plenary1/RoboSoft_GeorgeJeronimidis.pdf

[9] C. P. Gabriel Felippe, “Desenvolvimento e implementaÇao do controle de uma mao robotica do tipo multifinger,” p. 9, December 2014.

[10] P. F. Andrea Calanca, Riccardo Muradore, “A review of algorithms for compliant control of stiff and fixed-compliance robots,” *IEEE/ASME TRANSACTIONS ON MECHATRONICS*, vol. 21, no. 2, pp. 613–624, April 2016.

[11] A. W. Tomasz Winiarski, “Indirect force control development procedure,” *Cambridge University Press: Robotica*, vol. 31, pp. 465–478, July 2012.

[12] D. M. Sergio A. Pertuz, Carlos H. Llanos, “Simulation and implementation of impedance control in robotic hand,” *24th ABCM International Congress of Mechanical Engineering*, 2017.

[13] A. H. Ganwen Zeng, “An overview of robot force control,” *Cambridge University Press*, vol. 15, no. 1, pp. 473–482, December 1996.

[14] Y. N. Christian Ott, “Admittance control using a base force or torque sensor,” pp. 1–6.

[15] M. S. C. DEDE, “Position/force control of robot manipulators,” Master’s thesis, THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES OF THE MIDDLE EAST TECHNICAL UNIVERSTY, 2003.

[16] R. N. J. Mohamed Elbanhawi, Milan Simic, “Continuous path smoothing for car-like robots using b-spline curves,” *Journal of Intelligent and Robotic Systems*, January 2015.

[17] L. V. G. O. Bruno Siciliano, Lorenzo Sciavicco, *Robotics Modelling, Planning and Control*, 2009.

[18] V. Hlaváč, “Robot trajectory generation,” 2009, p. 161-189.

[19] “Universal robots ur3,” Universal Robots. [Online]. Available: https://www.robots.com/robots/universal-robots-ur3

[20] *UR3 Service manual*, Universal Robots. [Online]. Available: https://www.pi4.de/fileadmin/material/Produktkatalog/Universal_Robots/pdf_components_partner/UR3__docs_en/ServiceManual_UR3_en_3.1.3.pdf

[21] “Max. joint torques - 17260,” Universal Robots. [Online]. Available: https://www.universal-robots.com/how-tos-and-faqs/faq/ur-faq/max-joint-torques-17260/

[22] “Real-time data exchange (rtde) guide,” Universal Robots. [Online]. Available: https://www.universal-robots.com/articles/ur-articles/real-time-data-exchange-rtde-guide/

[23] A. S. Tanenbaum and D. J. Wetherall, *Computer Networks*, 2012.

[24] D. S. Aditya Sundararajan, Aniket Chavan and A. I. Sarwat, “A survey of protocol-level challenges and solutions for distributed energy resource cyber-physical security,” *Energies*, September 2018.

[25] “Overview of client interfaces,” Universal Robots. [Online]. Available: https://www.universal-robots.com/how-tos-and-faqs/how-to/ur-how-tos/overview-of-client-interfaces-21744/

[26] “13 unique features of python programming language,” Data Flair. [Online]. Available: https://data-flair.training/blogs/features-of-python/

[27] V. S. Lavdim KurtaJ, Ilir Limani, “Dependence of cmac neural network properties at initial, during, and after learning phase from input mapping function,” *ResearchGate*, August 2012.

[28] “Parameters for calculations of kinematics and dynamics,” Universal Robots. [Online]. Available: https://www.universal-robots.com/articles/ur-articles/parameters-for-calculations-of-kinematics-and-dynamics/

[29] E. Y. G. Wei Wang, Robert N.K. Loh, “Passive compliance versus active compliance in robotbased automated assembly systems,” *Industrial Robot*, vol. 25, no. 1, pp. 48–57, 1998.

[30] P. Corke, *Robotics, vision and control*, 2017.

[31] Z. X. W. L. Q. Asghar khan, Cheng Xiangming, “Closed form inverse kinematics solution for 6-dof underwater manipulator,” *IEEE/International Conference on Fluid Power and Mechatronics*, pp. 1171–1176, August 2015.

[32] P. Corke, “Peter corke toolbox.” [Online]. Available: http://petercorke.com/wordpress/toolboxes

[33] D. S. Singh, “Key slides from lectures 1-5 robotics and automation,” 2018, homework from course ”Robotic Manipulation and Mobility” given by Dr. V. Krovi.

[34] G. Terörde, *Electrical Drives and Control Techniques*, 2009.

[35] A. Sabanovic and K. Ohnishi, *MOTION CONTROL SYSTEMS*, 2011.

[36] H. Neville, “Impedance control of industrial robots,” *Robotics and Computer-Integrated Manufacturing*, vol. 1, no. 1, pp. 97–113, 1984.

[37] M. W. S. Robert J. Anderson, “Hybrid impedance control of robotic manipulators,” *IEEE/JOURNAL OF ROBOTICS AND AUTOMATION*, vol. 4, no. 5, pp. 549–556, October 1988.

[38] G. H. Friedrich Lange, Mirko Frommberger, “Is impedance-based control suitable for trajectory smooting,” pp. 1–6, 2006.

[39] S. C. Hrishi Shah, “Kinematic and dynamic control of a two link manipulator,” homework from course ”Robotic Manipulation and Mobility” given by Dr. V. Krovi.

[40] C. X. Yimei Chen, Dapeng Wu, “Nonlinear direct robust adaptive control using control lyapunov method,” *TELKOMNIKA Indonesian Journal of Electrical Engineering*, vol. 12, no. 1, pp. 477–486, January 2014.

[41] O. B. Haifa Mehdi, “Stiffness and impedance control using lyapunov theory for robot- aided rehabilitation,” *International Journal of Social Robotics*, vol. 4, no. 1, pp. 107–119, November 2012.

[42] *User Manual*, Universal Robots. [Online]. Available: https://www.usna.edu/Users/weaprcon/kutzer/_files/documents/User%20Manual,%20UR3.pdf

[43] G. McMillan, “Socket programming howto,” Python. [Online]. Available: https://docs.python.org/3/howto/sockets.html

[44] R. J. T. L. YoonSeok Pyo, HanCheol Cho, *ROS Robot Programming from the basic concept to practical programming and robot application*, December 2017. [Online]. Available: http://www.pishrobot.com/wp-content/uploads/2018/02/ROS-robot-programming-book-by-turtlebo3-developers-EN.pdf

[45] L. Joseph, *Robot Operating System (ROS) for Absolute Beginners*, 2018.

[46] “Impedance control for a 2-link robot arm - user-interactive,” Mathworks. [Online]. Available: https://nl.mathworks.com/matlabcentral/fileexchange/57853-impedance-control-for-a-2-link-robot-arm-user-interactive

[47] “Python-urx,” Sintef Manufacturing. [Online]. Available: https://github.com/SintefManufacturing/python-urx

[48] “Universal robots support download,” Universal Robots. [Online]. Available: https://www.universal-robots.com/download/?option=18940

[49] M. Detailleur, Github.

[50] *The URScript Programming Language*, Universal Robots. [Online]. Available: http://www.me.umn.edu/courses/me5286/robotlab/Resources/scriptManual-3.5.4.pdf

[51] A. Dua, “robopy.” [Online]. Available: https://github.com/adityadua24/robopy

[52] N. Majcherczyk, “Seven link manipulator.” [Online]. Available: https://github.com/NathalieMajcherczyk/RBE501-Balancing-Posture/tree/ce87f3e0ccd2db2fd812658cafb6d46b749ba125/SevenLinkManipulator

[53] P. L. M. v. d. W. Matthijs Boerlage, Maarten Steinbuch, “Model-based feedforward for motion systems,” *IEEE International Conference on Control Applications (CCA)*, vol. 2, pp. 1158–1163, June 2003.

[54] “Inverse kinematics algorithms,” Mathworks. [Online]. Available: https://nl.mathworks.com/help/robotics/ug/inverse-kinematics-algorithms.html

[55] W. J. L. Mohd. Asrul Hery Bin Ibrahim, Mustafa Mamat, “Bfgs method: A new search direction,” *Sains Malaysiana*, vol. 43, pp. 1593–1599, October 2014.

[56] M. I. A. Lourakis, “A brief description of the levenberg-marquardt algorithm implemened,” February 2005.

[57] “Ur3 joint limited robot urdf xacro,” ROS. [Online]. Available: http://ros.org/wiki/xacro

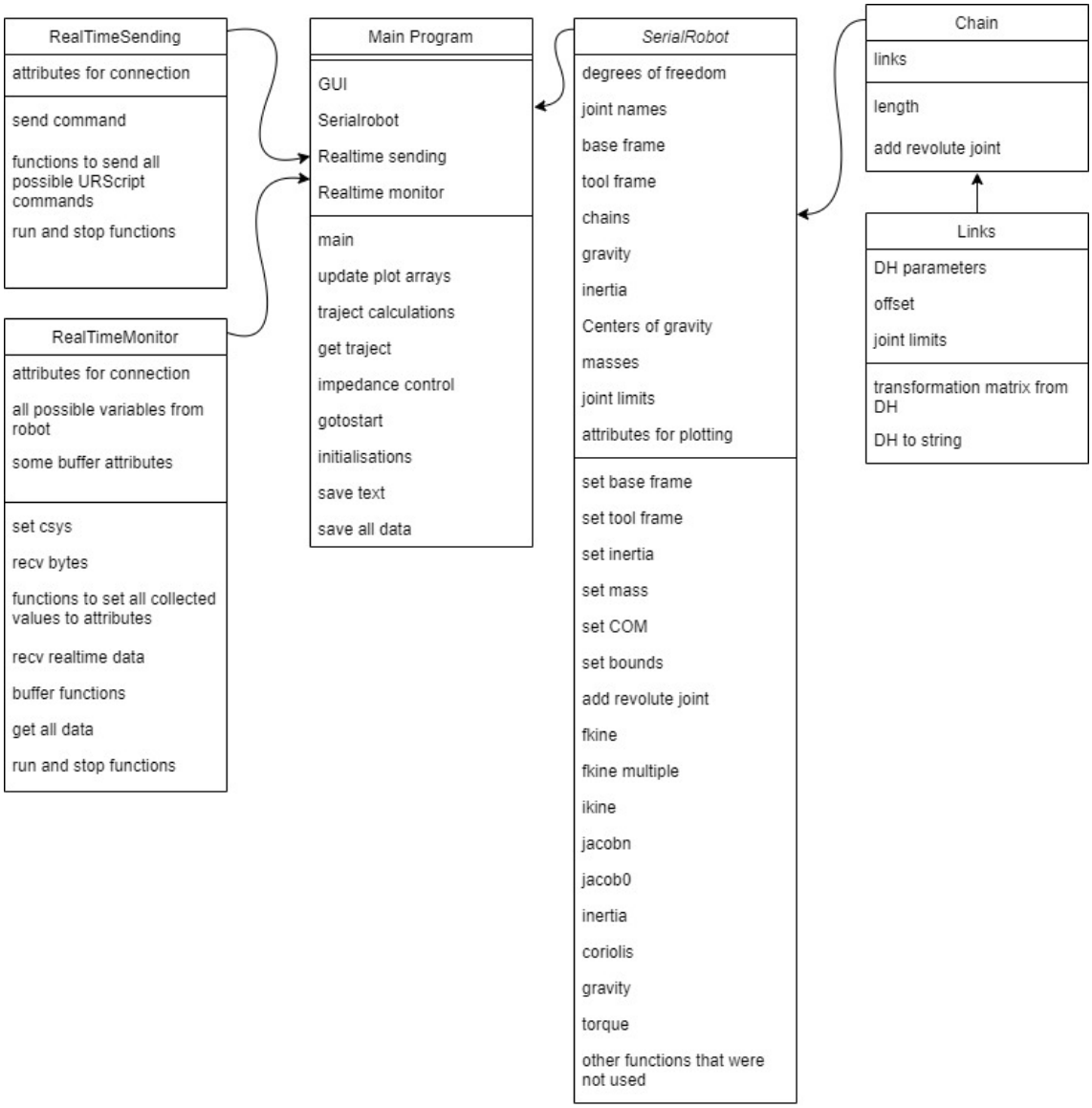
[58] “scipy.optimize.minimize,” SciPy. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html

[59] “Calling matlab from python,” MathWorks. [Online]. Available: https://nl.mathworks.com/help/matlab/matlab-engine-for-python.html

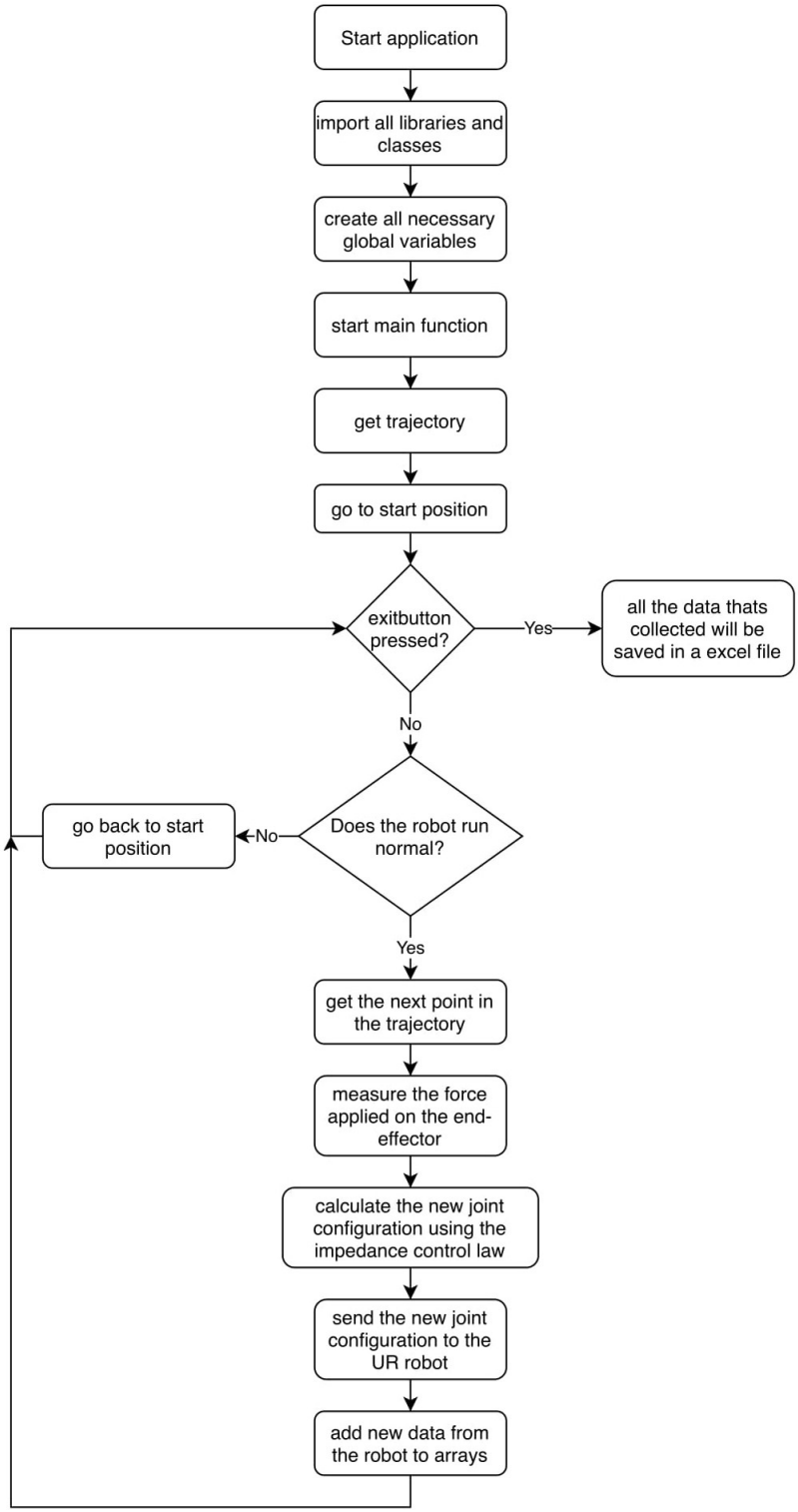
A

Attachments

A.1 Classes diagram Python controller



A.2 Cyclus diagram Python controller



FACULTEIT INDUSTRIËLE INGENIEURSWETENSCHAPPEN
CAMPUS BRUGGE
Spoorwegstraat 12
8200 BRUGGE, België
tel. + 32 50 66 48 00
iiw.brugge@kuleuven.be
www.iiw.kuleuven.be