NORWEGIAN UNIVERSITY OF SCIENCE AND TECHNOLOGY

TTK4550

CYBERNETICS AND ROBOTICS, PROJECT THESIS

Robotic rehabilitation of upper-limb after stroke

Author
Mads Johan LAASTAD

Main supervisor
Associate professor Øyvind STAVDAHL *MSc, PhD*

Co-supervisor
Erik Kyrkjebø *MSc, PhD*

December 21, 2016

NTNU

Abstract

Globally, stroke is one of the main causes of permanent neurological damage and death[18]. Partial or total paralysis in the extremities is the most common complication after a stroke, with paralysis in upper-limbs being the most prevalent. Efficient and readily available rehabilitation therapy is essential for the recovery process. General physical therapy is a common approach, but resource intensive. Research on robotic rehabilitation aims to provide a viable alternative.

The motivation for this thesis is to contribute in the design of a rehabilitation robot, based on a standard robot manipulator, aimed at rehabilitation of stroke patients with paralyzed upper-limbs.

The main objective is to design a robotic simulation environment for testing and analysis of robotic rehabilitation strategies. Secondary objectives include identification of implementation challenges, and suggesting feasible solutions.

A review of relevant simulation software is presented. The recommended candidate is used to create a robotic simulation environment for the Universal Robots UR5 manipulator. The resulting simulation environment proved to be insufficient for the requirements of the robotic rehabilitation project. The main challenge identified was limitations in available control inputs within a robotic software driver, because the driver does not facilitate force control. Suggested solutions, alternative software drivers and improvements to the simulation environment are also presented.

Several implementation challenges are identified and partially solved including; programming platform, firmware limitations, unwanted controller behavior, wrist mounting, patient criteria and minimal sampling rate.

Sammendrag

Hjerneslag er en verdensledende årsak til permanente nevrologiske skader og dødsfall. Delvis eller total lammelse i ekstremiteter er den vanligste komplikasjonen etter et hjerneslag, hvor lammelser i øvre lemmer er mest utbredt. Effektiv og tilgjengelig rehabiliteringsterapi er essensiell for å gjenopprette funksjonaliteten. Generell fysioterapi er utbredt, men ressurskrevende. Forskning innen robot rehabilitering har som formål å tilby et levedyktig alternativ.

Motivasjonen for denne prosjektoppgaven er å bidra til utviklingen av en rehabiliterings robot, basert på en standard industrirobot, som skal brukes til å rehabilitere hjerneslag pasienter med lammelser i øvre lemmer.

Hovedoppgaven er å konstruere et robot simuleringsmiljø for testing og analyse av strategier brukt av rehabiliterings roboten. Sekundære oppgaver er å identifisere implementeringsutfordringer, og komme med løsningsforslag.

En gjennomgang av relevant simuleringssoftware blir presentert. Den anbefalte kandidaten blir brukt til å konstruere et robot simuleringsmiljø for Universal Robots sin UR5 manipulator. Det resulterende simuleringsmiljøet er ikke tilstrekkelig for robot rehabiliteringsprosjektet. Hovedutfordringen er manglende pådragsmuligheter som er støttet av robot software driveren, den støtter ikke kraftstyring. Løsningsforslag, alternative drivere og forbedringsforslag blir også presentert.

Flere implementasjonsutfordringer ble identifisert og delvis løst, blant annet; programmerings plattform, firmware begrensninger, uønsket kontroller oppførsel, håndledds montering, pasient kriterier og minimal samplings rate.

PROJECT ASSIGNEMENT

Robotic rehabilitation of upper-limb after stroke

BACKGROUND

Approximately 15 000 people in Norway experience a stroke each year, and the incidence is believed to increase by up to 50 per cent in the next 20 years due to an ageing population. Stroke patients require extensive rehabilitation training to recover. Intense repetitions of coordinated motor activities in rehabilitation therapy and training constitute a significant burden for therapists, and due to economic reasons the duration of primary rehabilitation is getting shorter and shorter.

Recent systematic reviews of trials comparing conventional rehabilitation therapy by physiotherapists (CT) with robotassisted therapy (RT), suggests that RT gives similar upper-limb rehabilitation in terms of motor function recovery, activities of daily living (ADL) and motor control. Also, extra sessions of RT in addition to regular CT have been shown to be more beneficial than regular CT alone in motor recovery of stroke patients [1]. Robot rehabilitation of upper-limb after stroke have thus gained increasing interest in recent years, and new robotic devices have been designed to assist in physical rehabilitation of stroke patients [2].

These robot rehabilitation devices may significantly improve on existing practice through increased effectivity in rehabilitation training with less man-power, closer monitoring of patient progress, higher quality and more objective therapeutic sessions, and more personalized follow-up for each individual patient.

TASK

A number of different control strategies have been employed for robot rehabilitation ranging from high-level strategies – such as assistive, challenge-based, haptic or couching strategies designed to provoke motor plasticity and thus improve motor recovery – to low-level strategies to control force, position, impedance and admittance factors of the high-level strategies [2].

Recent advancements in light-weight and affordable robot manipulators that complies with safety regulations can operate without safety guards between robots and humans, and have a potential for decreasing the cost of custom built robotic rehabilitation devices significantly. The project shall propose a robot control strategy for rehabilitation of upper-limb after stroke using a standard industrial robot manipulator. The proposed robot manipulator is the UR5 robot from Universal Robots. The proposed strategy should be verified in simulation and experiments.

Task description

1. Present a short review of software used for robotic simulation environment and choose a suitable simulation software candidate for further use
2. Set up a robotic rehabilitation simulation model of the UR5 robot manipulator from Universal Robots
3. Choose and implement a control strategy for robotic rehabilitation in the simulation model, and analyze the simulation performance
4. If time permits: Suggest a functional design for experimental verification of the results, implement and analyze the robot rehabilitation control strategy in experiments

Useful resources may be the Matlab Robotics System Toolbox and the V-REP robot simulator.

Objective and purpose
The objective of the project is to design a simulation environment to analyze a chosen robot rehabilitation control system for a standard industrial robot manipulator for upper-limb stroke patients.

Subtasks and research questions

- Is simulation of robotic rehabilitation with with standard robot manipulators feasible?
- What are the main challenges in robotic rehabilitation using standard robot manipulators?

Main supervisor: Associate professor Øyvind Stavdahl, ITK

Co-supervisor: Erik Kyrkjebø, Sogn og Fjordane University College

Bibliography

- *Norouzi-Gheidari, N., P.S. Archambault, and J. Fung, Effects of robot-assisted therapy on stroke rehabilitation in upper limbs: Systematic review and meta-analysis of the literature. J. of Rehab. R&D, 2012. 49(4): p. 479-495.*
- *Maciejasz, P., et al., A survey on robotic devices for upper limb rehabilitation. J Neuroeng Rehab, 2014. 11(1).*

Preface

I would like to thank Erik Kyrkjebø MSc, PhD and Øyvind Stavdahl MSc, PhD for supervising this project thesis. Further, I would like to thank student colleagues and PhD candidates at the Depertment of Cybernetics for all their help and advice. Finally, a big thank you to Christina Laastad and Bjørn Johan Laastad for proofreading the project thesis.

This project thesis is part of a robotic rehabilitation project led by Erik Kyrkjebø, and is an extension of previous work[5][4] done by Kristine Blokkum during the fall of 2015 and spring of 2016. In her thesis, Blokkum presents background theory on different robotic rehabilitation strategies and suggested a control strategy for implementations. A key focus area for this thesis has been on maintaining a steady progress in the robotic rehabilitation project and to not recreate or retrace results presented in other works.

All of the software used in this thesis were new to both the supervisors and myself. The steep learning curve, and software complexity seized much of the time dedicated to the thesis. Even though the resulting simulation environment was insufficient for the needs of the robotic rehabilitation project, it provided useful information for future implementation efforts. My personal programming skills, and knowledge related to Linux, Matlab, ROS and Gazebo have improved significantly while working with this thesis.

Audience

To benefit from this thesis the reader is advised to have decent background knowledge in control engineering, and robotic modeling and control. Recommended literature on these subjects are respectively [8] and [32]. There are no requirements for background knowledge in medicine, physical therapy or programming. The relevant theory for these subjects will be presented in chapter 2 and chapter 3.

It is recommended to read the project thesis[5] and master thesis[4] written by Kristine Blokkum during the fall of 2015 and spring of 2016. It provides valuable insight into available robotic rehabilitation strategies and the basis for suggested controller design intended for implementation on the simulation environment.

Contents

List of Figures

Abbreviations

6DOF = Six degrees of freedom
OS = Operating system
VM = Virtual machine
ROS = Robot Operating System
ROS-I = ROS-Industrial
API = Application programming interface
GUI = Graphical user interface
UR3 = Universal robot model 3
UR5 = Universal robot model 5
UR10 = Universal robot model 10
OSRF = Open source robotics foundation
URDF = Unified robot description format
SDF = Simulation description format
XML = Extensible markup language
TCP/IP = Transmission control protocol/Internet protocol
TCP = Tool center point
LQR = Linear quadratic regulator
RTDE = Real-time data exchange
VPL = Visual programming language
CB = Control box
URControl = Universal robot controller

Chapter 1

Introduction

1.1 Motivation

Stroke is the leading cause of permanent neurological damage and one of the leading causes of death among adults [13] in Norway with over 15 000 confirmed incidents annually. On a global scale, about 15 million people suffer from stroke annually, with over 5 million casualties and another 5 million left with permanent neurological damage[18]. According to the Norwegian directorate of health $60 - 80\%$ of the stroke patients will experience complications in the immediate aftermath of a stroke. That number increases to above 90% after $3 - 6$ months[13]. Common complications include; paralysis, epileptic seizure, falling, swallowing difficulties and shoulder pain. These symptoms often result in reduced muscle strength, reduced motion control, reduced movement speed, disturbed coordination and increased rate of muscle fatigue. Paralysis is a serious symptom developed after a stroke with a crippling affect on the patient's daily life. Partial or total paralysis in the extremities occur in more than 80% of the stroke patients[13]. The severeness and prevalence of paralysis makes it one of the biggest challenges related to stroke rehabilitation, and is the focus of a robotic rehabilitation project was pioneered by Erik Kyrkjebø, vice-rector of R&D, Sogn og Fjordane University College, Norway.

Traditional rehabilitation of upper-limb after stroke provided by physical therapists have proven to be effective, but it is highly resource intensive and cannot keep up with the growing demands. Robotic rehabilitation can hopefully become a viable, affordable and safe rehabilitation alternative. Several studies[19][15] have already shown that robotic rehabilitation, utilizing customized and expensive robotic exoskeletons, can rival traditional rehabilitation in terms of quality. The robotic rehabilitation project hopes to show how an inexpensive industrial robot can be used safely in the rehabilitation of stroke patients as a valuable supplement to the traditional treatments. Achieving this will help society reduce the costs associated with stroke rehabilitation, and more importantly improve the stroke patient's quality of life.

Figure 1.1: Example of robotic rehabilitation with stroke patients using an exoskeleton device called ArmeoPower. Figure taken from [16].

1.2 Thesis goals

The main objective of this thesis is to design a robotic simulation environment to be used for testing and analysis in the robotic rehabilitation project led by Erik Kyrkjebø.

The thesis will present a short review of robotic simulation software. Then, with that insight, recommend a suitable candidate to be used as a platform for creating a robotic simulation environment using Universal Robots UR5 manipulator and a previously developed robotic rehabilitation control strategy.

Other, secondary objectives of the thesis is determining the simulation environments feasibility, and investigating some of the implementation challenges with robotic rehabilitation using standard robot manipulators. The thesis is also intended to be the basis for a future master thesis for the robotic rehabilitation project with a focus on physical implementation with a real UR5 manipulator.

1.3 Contributions

The thesis presents a functional review of four possible robotic simulation software platforms, and the best software candidate is used for creating a robotic simulation environment for the robotic rehabilitation project. With a defined software platform, several useful ROS repositories and packages for both a robotic simulation environment and a physical implementation are identified. The robotic simulation environment is built with the necessary sensory and patient force inputs for realizing the suggested controller design. Complications regarding the simulation environment required a deeper understanding on how the UR manipulators are programmed and controlled. The resulting information is presented, together with a review of some selected drivers for the UR manipulators based on one of

the programming and controlling options. A constant focus on identifying implementation challenges resulted in a number of interesting observations and reflections that will contribute to a physical implementation in a future master thesis. Finally, recommendations regarding programming basis, drivers, patient criteria and minimal sampling rate for future implementations are presented in chapter 6.

1.4 Report outline

This report structure is based on the introduction, methods, results, and discussion(IMRAD) structure[30]. However, the report structure does not follow the the work on the project thesis in chronological order. This leads to some smaller differences in the report structure. In chapter 2 it was sometimes necessary to draw natural conclusions in order to maintain a proper flow in the report. Also, the results presented in chapter 4 is somewhat limited due to complications with the resulting simulation environment.

Chapter 2 presents the theoretical background for this thesis. It includes an overview of previous work regarding the robotic rehabilitation project. Background knowledge on contained induced therapy and the Universal Robots UR5 manipulator. Information about the robot operating system(ROS) is presented together with some relevant ROS repositories and packages that are used in this thesis. Finally, details about programming and controlling the UR manipulator together with a review of relevant drivers.

Chapter 3 presents a review of simulation software and identifies the best suited candidate to be used in the thesis.

Chapter 4 presents the resulting simulation environment, together with some necessary sensor modifications to suit the needs of this thesis.

Chapter 5 analyses and discuss some of the challenges with the simulation environment, and provides some suggested solutions. Some interesting observations about unwanted robotic behavior is presented.

Chapter 6 provides the conclusion to the main objective, sub-tasks and research questions presented in the project assignment.

Finally, **chapter 7** presents a number of suggestions for future work within the robotic rehabilitation project.

The appendix contains some short code examples and information about the TÜV safety classification for Universal Robots UR5 manipulator.

Chapter 2

Theoretical background

This chapter examines important theory related to this thesis. It contains a quick review of previous work, robotic rehabilitation theory, challenges with force control and information about the robotic operating system(ROS).

2.1 Robotic rehabilitation strategies

The following section presents a short summary of previous work done by Kristine Blokkum in her project thesis[5] and master thesis[4]. Theory regarding constraint induced movement therapy is presented here, as it will influence the functional design of the physical interface between the rehabilitation robot and stroke patient. This is discussed further in chapter 5

2.1.1 Previous work

Blokkum examines several robotic therapy options, and recommends active robotic therapy as the basis for the rehabilitation strategies used by the robotic rehabilitation project. Several implementations of active robotic therapy are discussed in her work, the most relevant are mentioned in this report. Her suggestions for a functional design together with the suggested control strategy form a basis for the work presented in this thesis.

Active robotic therapy

Active robotic therapy requires the patients to take an active part in the movement by applying a minimum force threshold. The supervising physician or therapist should be able to change the force threshold value depending on the patient's needs and capabilities.

There are two main rehabilitation strategies that are based on the active robotic therapy, assistive and challenge-based rehabilitation. Both presume that there is a predefined task or movement which the patient is trying to perform. In assistive rehabilitation the robot

assist the patient in performing a task. This can be done by amplifying the movement the patient is trying to perform as long as the force threshold value is met.

Challenge-based rehabilitation is in many ways the opposite of assistive rehabilitation. The goal is to resist the patient's movements or in other ways challenge the patient while performing a task. One implementation involves making the robot act as a weight for the weak arm to move, another can be error-amplification of the movements that are not directly contributing in performing the desired movement.

Functional design

The functional design suggested by Blokkum in her master thesis[4] is based on results from a literature review presented in her project thesis[5]. The suggested approach provides challenge-based training for the strong patients, and assistive training for the weak patients. The design can also be expanded to include challenge-based error-amplification for the patients that have progressed further along in the rehabilitation.

Controller design

The rehabilitation strategies and their implementations can be realized by a joint space hybrid position/force controller. The suggested controller design[4] requires switching between a position controller and a force controller. For example, if the chosen rehabilitation strategy is assistive rehabilitation a dead-zone will be defined in an orthogonal plane around a predefined path in the desired direction. The controller will enable the force controller while the patient is inside the dead-zone, and the gradient vector is either zero or in the desired direction. Moving outside the dead-zone triggers the switching to a position controller that regulates the patient arm until it is located back inside the dead-zone. Movements with a gradient that is opposite the desired gradient will also trigger the position controller. Further details about the switching are not relevant for this thesis, but can be found in the original work[5][4].

In order to create a closed loop position controller it is necessary to gather measurements of the robot's position, in addition to a method of controlling its position. Similarly, a force controller needs force measurements of the force acted upon the robot by the environment and a way to control the force the robot acts upon the environment.

2.1.2 Constraint induced movement therapy

A common rehabilitation strategy in physical therapy consists of reducing the degree of freedom in the afflicted limb in order to retain some degree of functionality. The strategy is based on the human body's natural reaction to trauma or injuries. For example, a patient suffering from trauma or injuries in the lower limbs will naturally adopt a compensating gait pattern with fewer degrees of freedom in order to retain locomotive mobility. This is well documented in the gait pattern of elderly suffering from hip fractures[20][17].

The limited functionality enabled by the constraint induced movement therapy then becomes a platform for rebuilding muscle strength, a more complex repertoire of functional movements and other rehabilitation efforts.

2.2 Hardware and relevant challenge

The following section presents an overview of Universal Robots UR5 manipulator with a small selection of technical specifications,and a relevant challenge regarding force control in robotic manipulators.

2.2.1 Universal Robots UR5

The Universal Robots UR5 robot[26] is a relatively inexpensive 6-axis industrial manipulator with a reach of 85 cm and a maximal payload of 5 kg. The manipulator consists of six rotational joints shown in figure 2.1, all with a maximum joint ranges of $\pm 360^\circ$ and maximum joint speed of $\pm 180^\circ/s$. Several built-in safety mechanisms are constantly active with standard firmware running. This includes an emergency shutdown[25] triggered when a maximum force acting on the robot Tool Center Point(TCP) exceeds 100N, and a maximum momentum of the robot arm exceeds $10\frac{kgm}{s}$. There are some measurement uncertainty related with the robot's internal sensors. The producer specifies a force trueness of 25 N, and a momentum trueness of $3\frac{kgm}{s}$.

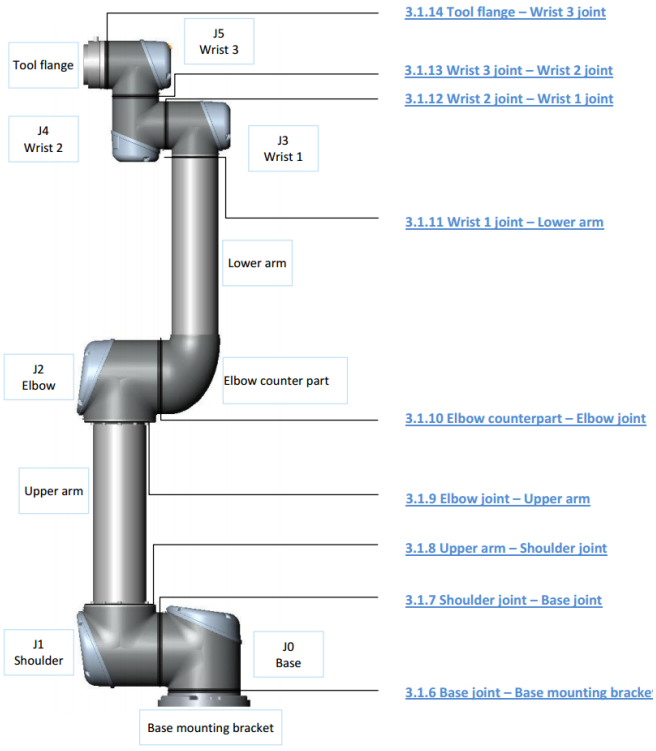

Figure 2.1: Configuration of the UR5 manipulator. Figure taken from [24].

2.2.2 Challenge with force control in robot manipulators

Universal Robot (UR) do not provide direct insight into the firmware controller design, probably because it is regarded as a trade secret. This thesis attempts to better understand why adequate force control is difficult without direct control of joint torque or joint velocity. This research question is a highly relevant challenge for this thesis.

At the mechanical level every joint in the UR manipulator is moved by a three-phase AC motor, the motors are in turn controlled by integrated motor drives with space vector control[24]. Manipulating the frequency and voltage input to the motor lead to an almost linear correlation[6] to joint speed and joint torque. The inner control loop handles joint torque control and joint velocity control in each joint directly. Outside this inner control loop is a joint position feedback control loop. Figure 2.2 illustrate the joint controller architecture from a similar 6DOF robot manipulator. Where τ_{ffw} = feed forward joint torque, $\dot{q}_{ref}$ = feed forward reference joint velocity and q_{ref} = reference joint position. In the UR manipulator each joint is individually controlled by a similar version of the joint controller, with a central computer that calculate references for each of the joints.

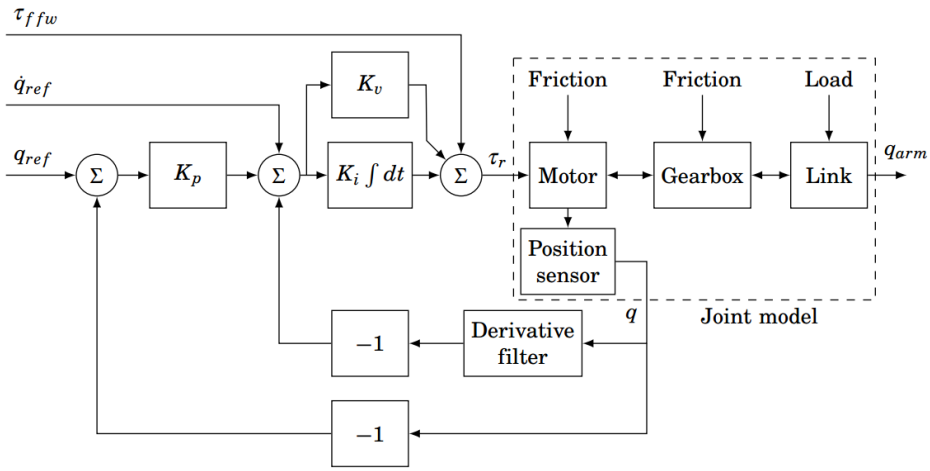

Figure 2.2: Low level joint controller with feed-forward velocity and acceleration control inputs. Figure taken from [31] .

Preferably it is possible to get access to all three control inputs; position, velocity and acceleration, but as illustrated in chapter 5 this is not always the case. Challenges rise from trying to design a force controller which is located outside the cascaded control structure in figure 2.2. It might work in a theoretical world with perfect mathematical models and perfect signals generation, but in an actual implementation it can lead to some unwanted behavior. The force control inputs need to be kinematically transformed, integrated twice and differentiated twice before they are applied as a joint torque, greatly amplifying any noise signals in the process. This will undoubtedly cause internal fighting between the control loops and give unpredictable results. In the author's interpretation of a best case scenario it will respond to force inputs and move in the desired direction, but with a twitch-

ing motion as each waypoint feed to the position controller leads to a constant acceleration and deceleration of the manipulator. Tuning the controller will also be a challenge. This implementation will most likely not meet the soft real-time requirements for the intended robotic rehabilitation strategy.

2.3 Supportive software

Most of the software used in this thesis were unknown to the author and the supervisors. Therefore, a detailed description of relevant software tools used in this project thesis will be presented in the following section. A compressed overview of high-level concepts of ROS[12] is presented, followed by a quick description of ROS repositories and packages used in this thesis.

2.3.1 Robot Operating System (ROS)

The Robot Operating System, hereby referred to as ROS, is an attempt to create a universal and flexible framework for writing robot software. Based on the provided functionality, it is more precise to call ROS a common robot programming interface, instead of a robot operating system. Having a standardized framework makes it easier to produce robotic software across different hardware platforms and programming languages. It is developed and maintained by the Open Source Robotics Foundation. A main goal for ROS is to facilitate collaborative robotic software development across different hardware platforms by utilizing a large library of repositories.

Figure 2.3: Artistic impression of different applications using ROS. Figure taken from [9].

2.3.1.1 ROS Master

The ROS Master provides naming and registration services to the entire ROS network. It enables individual ROS nodes to locate each other across different software platforms through standard TCP protocol. Once the connection have been established the nodes communicate with each other peer-to-peer.

Another important task the master node provides is a parameter service. Parameters are shared global variables within the ROS network. Although ROS is not intended to be used as a real-time system, it supports threading. The master node's parameter service avoid common problems related to shared variables in a threaded system.

2.3.1.2 Nodes and topics

ROS uses a graph based architecture for communication. In its most basic form, a graph consists of two elements; nodes and edges. A node is graphically depicted as a circle or a square and an edge is graphically depicted as a dotted or solid line connecting two nodes. All communication between nodes happen over the edges or as they are refereed to in ROS terminology; a topic. A node can publish information to a topic, or subscribe to a topic of interest. There are no limitations on how many topics a node can publish or subscribe to. The communication protocols are over TCP/IP with each node having it's own unique port number. Since all communication between nodes are standardized the nodes can run on different computers with different operating systems. In addition, the nodes can be programmed in different programming languages. This is the true strength of ROS; a robust cross-platform comparability.

2.3.1.3 ROS packages

Software in ROS is organized in packages. The packages provide useful functionality with easy application in order to encourage code reuse within the robotic community. Individual packages might contain ROS nodes, ROS-independent library's, configuration files, SDF model files, third-party software, or anything else that constitutes a useful module. Repositories contains a collection of useful ROS packages.

2.3.1.4 Catkin workspace

A software workspace introduces a standard for organizing all the files and directories in a large and complex project into one coherent structure. This allows the programmer to efficiently maintain a large project, organize various source code and resource into a single ROS package; and thereby helps to promote code sharing and cross platform code re-usability.

The Catkin workspace is popular within the ROS community, and it is therefore the preferred workspace in this project thesis. The workspace is divided into several subspaces each with a specific task. This thesis will focus on the source space located inside the src/ folder. It will contain all the ROS packages needed to build a robotic simulation environment, and custom scripts used for development and testing purposes. A generic package file structure can be seen in the example below.

```
workspace_folder/          -- WORKSPACE
  src/                     -- SOURCE SPACE
    CMakeLists.txt         -- The 'toplevel' CMake file
    package_1/         -- Generic package name
      CMakeLists.txt       -- CMake build file
      package.xml      -- Metadata about package
      include/       -- C++ include headers
      msg/       -- Folder containing Message(msg) types
      src/       -- Source files
      srv/       -- Folder containing Service (srv) types
      scripts/       -- Executable scripts
```

2.3.1.5 Messages and services

ROS uses a simplified message description language for describing data values that ROS nodes publish. Message descriptions are stored in $.msg$ files inside the *src/package_1/msg/* sub-directory. In a similar manner a simplified service description language is used for describing ROS service types. It builds directly upon the $.msg$ format to enable request/response communication between nodes. Service descriptions are stored in the $.srv$ files in the *src/package_1/srv/* sub-directory.

2.3.2 Applied ROS packages

The following subsections describes important and relevant ROS packages that are used for building the robotic simulation environment in chapter 4.

2.3.2.1 ROS-Industrial

ROS-Industrial (ROS-I) is an open-source repository containing libraries, tools, sensors, interfaces and drivers for industrial hardware. The repository overview seen in figure 2.4 provides all the necessary infrastructure for building robot models from the included vendor specific packages.

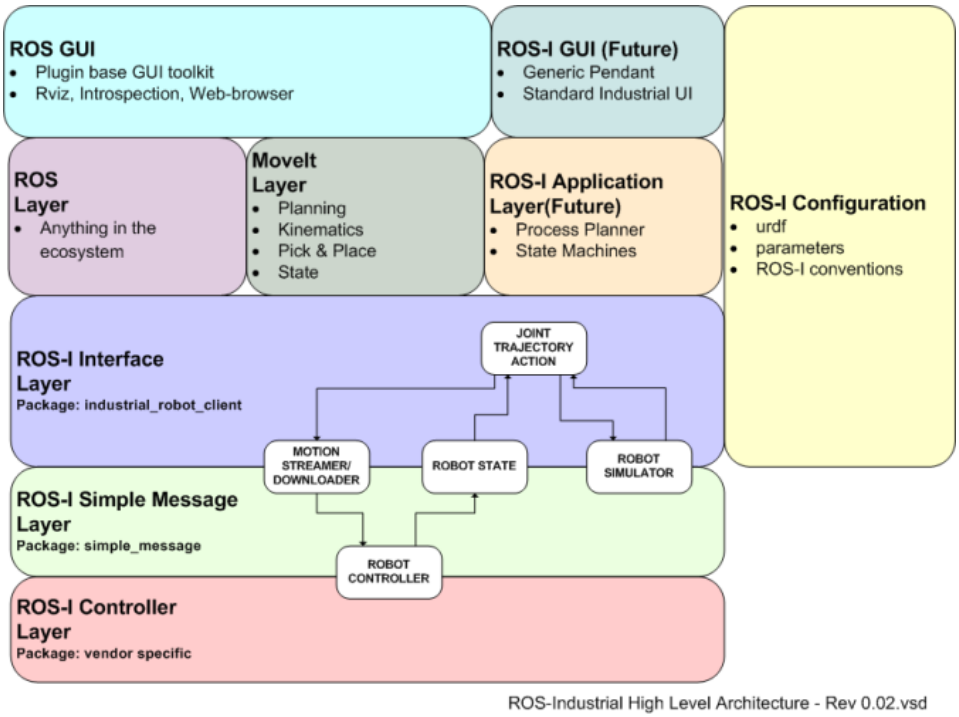

Figure 2.4: ROS Industrial Robot architecture. Figure taken from [11]

Importantly, one of the supported vendors are Universal Robots. The ROS-I repository forms the basis for the simulation environment used in this thesis.

2.3.2.2 Universal Robots

The *universal_robot* repository is part of the larger ROS-I repository. It contains vendor specific robotic models, parameters, interactions, communication interfaces, standard controllers and visual representation. Included in the supportive packages are URDF files for the UR3, UR5 and UR10 robot manipulators. These features makes it a highly useful repository for this project thesis.

2.3.2.3 URDF

The Unified Robot Description Format(URDF) is a ROS package used for standardized description of all the elements in a robot. The files are written in extensible markup language (XML) format, and contain kinematic and dynamic properties of a specific robot model. Chapter 4 will utilize the URDF architecture seen in figure 2.5 to build in a plugin for measuring the force and torque on the UR5 end effector.

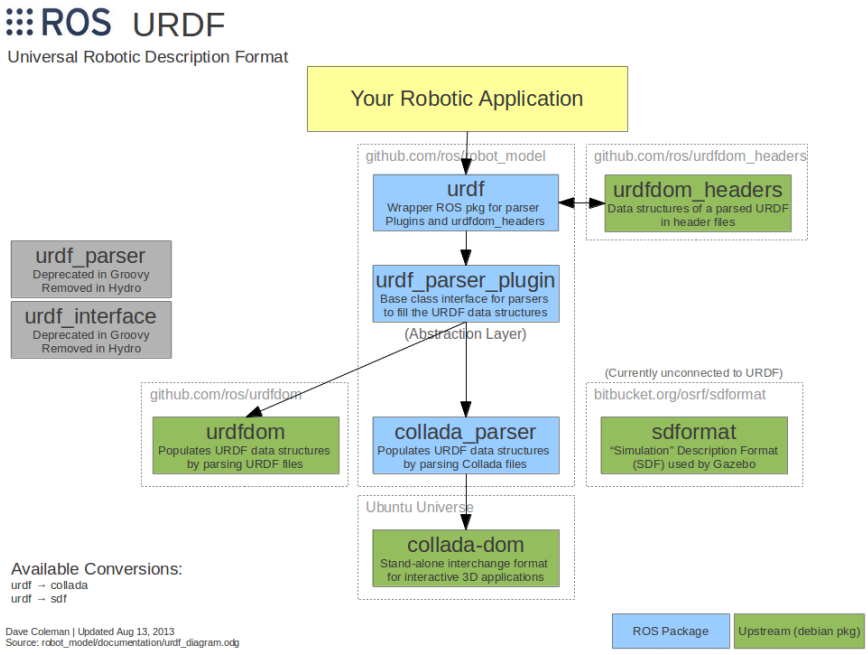

Figure 2.5: URDF overview. Figure taken from [27].

This modular approach to URDF files is in line with the method of reuse-ability that ROS was founded on. There are however, some drawbacks when model's objective is to interact with the surrounding simulation environment. For example, friction and inertia are two important environmental interactions that are not described in a URDF file. To overcome these shortcomings in a Gazebo simulation, a more complex SDF model file is built around the generalized URDF model.

2.3.2.4 RVIZ

RVIZ is a 3D visualization environment for ROS. It provides a graphic overview of the robot's sensory inputs, trajectory planning and motion execution. Visualization and logging of sensory information is an important part in developing and debugging. For this thesis RVIZ was used together with MoveIt, a motion planning library, for early testing and debugging.

2.4 Controlling the UR manipulators

During preliminary testing of the simulation environment presented in chapter 4 several challenges appeared which required a more detailed understanding of the way the UR5 is programmed and controlled. The information gathered is presented in this section. It describes the three main ways of programming the Universal Robot, gives details about the Ethernet connections of the standard UR firmware controller (URControl) and discuss a selection of drivers that can be used for programming the robot via these network connections.

2.4.1 Programming

There are three different ways of programming the UR manipulators; with a teach pendant, or using code generated in URScript or C. Both the teach pendant and URScript runs on the original firmware with a version of a low-level robot controller called URControl. The internal controller sends commands to the joint servos at 125Hz and is thus able to evaluate a new set of instructions every 8^{th} millisecond.

2.4.1.1 C programming

Universal Robot provides a C library for direct control of all joint servos in the robot. With direct control of the applied joint torques it possible to create a custom low level controller that runs on the robot with minimal delay. The fastest implementation will still have a updating speed limited by the joint servos updating rate of 125 Hz.

Using programs or drivers with the C API will be the best control options for this project. However, there are several disadvantages. It is necessary to install the program on the robot controller box in order to use the C API interface. This might have some legal ramifications that will be discussed further in chapter 5.

2.4.1.2 Teach pendant

The teach pendant is running a program called Polyscope which is a graphical user interface (GUI). It provides control and programming interactions with the UR manipulator. Polyscope can be a powerful tool for fast implementation, prototyping and debugging, but it was mainly developed for users without a programming background. Polyscope is mostly a visual programming language (VPL) and can not be used for offline programming.

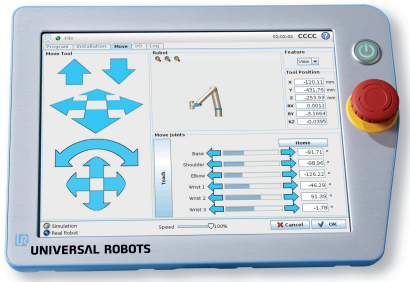

Figure 2.6: Universal Robot teach pendant. Figure taken from [21].

2.4.1.3 URScript

URScript is a robot programming language developed by Universal Robot. Like any other programming language, URScript provides basic functionality like variables, types, flow of control statements and basic threading. It also provides a lot of robot specific functionality such as built-in variables and functions that monitor and control the I/O and movements of the robot. However, it does not support classes, and the threading functionality is only basic. It supports nested threading and basic mutex operations to avoid race conditions, but it does not support parameters in thread function calls or semaphores. More limitations in thread handling is found in the documentation; *"A couple of things should be noted. First of all, a thread cannot take any parameters, and so the parentheses in the declaration must be empty. Secondly, although a return statement is allowed in the thread, the value returned is discarded, and cannot be accessed from outside the thread."*

Although it is not explicitly stated in the documentation, the impression is that URScript does not support shared variables. In the URScript language, there are four different methods of moving the robot; move, servo, speed and force.

Move

Move commands are simple point-to-point movements. Motions commands can be linear in Cartesian or joint space, or circular in Cartesian space. Speed and acceleration parameters in the function call controls the trapezoidal speed profile of the move command. A move trajectory with several waypoints will make the robot momentarily come to a hold at each point, unless a blend parameter is given. The blending functionality can be show in a simple trajectory example with three waypoints.

Figure 2.7 illustrates how URControl uses the blend radius parameter to merge two trajectories together with a smooth transition. The new trajectory prevents the robot from coming to a complete stop at the WP_2 waypoint at the cost of never accurately reaching it. This makes move commands with blend parameters unsuited for accurate position control.

Servo

The servo commands gives direct control of the position of individual joints in order to provide a high level of precision and control. It moves the joint to the desired position with a trapezoidal speed profile and then stops all motion. Firmware 3.1 or later versions provides the user with a lookahead_time and a gain parameter in the function call. This works in a similar manner as the blend parameter, it either soften or sharpen the desired trajectory. Section 2.5.3 will discuss an alternative method for trajectory smoothing called interpolation.

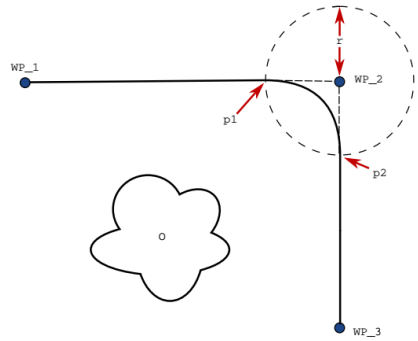

Figure 2.7: Blending over WP_2 with radius r. Figure taken from [26].

Speed

The speed command makes the robot accelerate linearly to the desired velocity and move with a constant velocity in either joint space or Cartesian tool space. Once a desired velocity is reached, the robot keeps moving at the desired velocity until a new command is issued, or a joint limit is reached.

Force

The force mode command enables simple built-in force control, making the robot compliant along one or more axes while still moving along other axes. Along/around compliant axes the robot arm will comply with the environment, which means it will automatically adjust its position in order to achieve the desired force. It is also possible to make the robot arm itself apply a force to its environment. However, this force controller design is fixed, and this will severely limited the desired applications for this project. Force commands is therefore believed to be unsuited as a basis for force control in the suggested controller design.

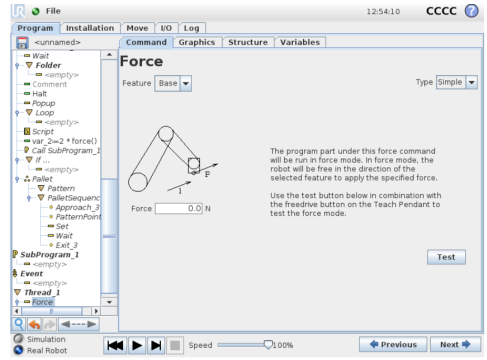

Figure 2.8: Polyscope view of force control. Figure taken from [26].

2.4.2 Programming summary

This thesis have outlined the three main methods of programming the UR manipulators. The information presented and the following discussion suggests that the teach pendant is unsuitable for this project thesis. The uncertainties raised in section 5.2.4 regarding C programming are not definitively confirmed, but the concerns does not grant further use in this project thesis. The remaining option would be to control the robot through URScript commands. However, the URScript language have some limitations that makes it unsuitable to be the preferred programming language for a large and complicated project. The best option is a driver written in a well known programming language that wraps custom programs into URScript functions for the URControl to read and execute. Using standard firmware, the driver communicates with URControl over Ethernet connections.

2.4.3 Ethernet connections

With the stock firmware running, data regarding the robot's state, temperature, joint positions, joint orientations, etc. are continuously streamed through four TCP/IP server sockets in the controller. It is possible to create custom programs or drivers to read and write to the sockets.

2.4.3.1 Dashboard server

The dashboard server is used by the robot GUI, called Polyscope, to load and execute programs and is found on port 29999. The main functions of the server are to load, play, pause and stop a URScript program. As of firmware version 3.0, it can also be used to power the arm on and off, release the brakes and, from version 3.1, unlock the protective stop.

2.4.3.2 Primary and secondary client interfaces

The controller exposes two interfaces which a client can connect to; primary port 30001 and secondary port 30002. The data streams provide robot state information like joint position and orientation, voltages, temperatures, Cartesian position, IO status, TCP force, etc. The primary interface transmit robot state data and additional messages, while the secondary interface transmit robot state data and as of firmware version 3.1, it streams firmware information. Both interfaces accept URScript commands with a 10 Hz update rate, however it can not be recommended because the commands are not read in a deterministic manner.

2.4.3.3 Real time interface

The real time interface is found on port 30003. It provides a similar set of robot state information at a much faster update rate of 125Hz. The data stream contains information about the actual joint position and speed, target joint position, joint moment, joint current, and joint temperature. The interface can also receive URScript commands with the same 125Hz update rate.

2.4.3.4 Real-time data exchange interface

For the newest firmware versions, CB3/CB3.1 with software 3.3 or later, UR has added a fourth client interface called the real-time data exchange (RTDE) on port 30004. It provides an interface to synchronize external applications with the UR controller over a standard TCP/IP connection, without breaking any real-time properties of the UR controller. The synchronization is configurable and can include outputs like robot-, joint-, tool- and safety status, analog and digital I/O's.

2.5 Review of existing drivers

This section will discuss a selection of existing drivers for the UR robot manipulators. Drivers like the ur_c_api written by Kelsey Hawkins from Georgia Tech and ur_sanntid developed[29] by Torstein Anderssen Myhre and Johannes Schrimpf form NTNU and Sintef Raufoss Manufacturing will not be considered since their work relies on the previously discussed C API.

2.5.1 ur_driver

The ur_driver[2] has been part of ROS since the development of ROS Electric in 2011 and was originally written by Stuart Glaser. Significant contributions have later been made by Felix Messmer, Alexander Bubeck and Shaun Edwards. The ur_driver is a package in the universal_robot repository that provides a ROS interface for controlling the UR robot manipulators. In chapter 4, it is used to create a simulation environment for the UR5 robot.

The driver is written in Python and exposes a joint trajectory interface. It is theoretically capable of supporting communication update rates of 125 Hz. The actual update rate is disputed, as several sources report significantly lower update rates. During initialization is spawns several threads for communication and control. The two threads connects to port 30002 and 30003 respectively, where they parse the received robot state information and publishes it onto ROS topics. The last thread opens a new socket on port 50001 where it sends a custom URScript program to the robot. Byte-encoded instructions are sent over the new socket to URControl where they are parsed by the new custom program which call the appropriate URScript functions. The custom program allows joint trajectory control based on the basic move and servo commands. The lack of control interfaces provided by the ur_driver will be discussed further in chapter 5. The joint trajectory interface uses interpolation to generate a smooth trajectory that goes through all the waypoints, instead of using the blend or lookahead_time parameter discussed in section 2.4.1.3. Figure 2.9 illustrate examples of a trajectory made from discrete waypoints with a linear point-

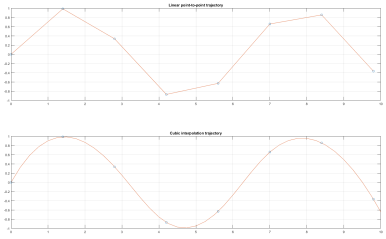

Figure 2.9: Linear and interpolated trajectory.

to-point solution, and with a cubic interpolation solution. The driver performs a quadruple oversampling of the interpolation to ensure a smooth trajectory execution of servo commands.

When URControl revives a new program it will immediately stop any program currently running and execute the newest program. This might lead to an abrupt transition in the robot's motion, and is not properly handled by the ur_driver. The custom URScript is long, slow and complicated. It introduces significant delays from a new command is received until the robot actually moves. Although the robot should be controllable at 125Hz, experiments done by [3] shows that the actual performance is 2-10 times slower. Additionally, comparability with the latest firmware is also questionable due to protocol changes in the robot's TCP/IP client interfaces.

2.5.2 python-urx

The python-urx[28] driver was developed in 2012 by Oliver Roulet-Dubonnet for Sintef Raufoss Manufacturing. It is originally intended for pick and place operations, but it has been used for welding and other sensor based applications that do not require high control frequency.

The driver is written in python and communicates mainly on the slower primary and secondary interfaces at a update rate of 10 Hz. It has built-in functionality to listen to the real time interface at 125 Hz, but it is turned off by default. The driver can only send programs to the robot through the slower 10 Hz interfaces. As previously mentioned, sending programs over the primary or secondary interfaces is problematic because the programs are not guaranteed to be executed in a deterministic manner. The developer comments in the source code and warns about a delay of several 10^{th} of a second from sending a new program to the robot starts moving. The driver handles both joint position changes and requests for I/O changes. It provides support for URScript all commands move and speed, but it does not support the servo and force commands.

2.5.3 ur_modern_driver

The ur_modern_driver[3][1] package was developed by Thomas Timm Andersen, a robotics engineer from the Technical University of Denmark, as part of his PhD thesis on sensor-based real time control of robots. The driver is written in C++ and is designed to be a direct replacement for the ur_driver with a lot of improved functionality.

The driver provides two interfaces; a joint velocity interface and a joint trajectory interface. The joint velocity interface is based on the speed command discussed in section 2.4.1.3, it takes the published velocities and directly forwards them to the robot controller for the fastest possible execution. The trajectory interface is based on the servoj command discussed in section 2.4.1.3 and closely resembles the ur_driver solution with a custom program calling URScript commands. This is done to ensure backwards compatibility with existing robotics projects. The trajectory interface receives new goals and verifies that the trajectory is valid. If it is approved, the driver does cubic interpolation between the trajectory goals and transmits the intermittent joint position to the custom script running on the UR controller.

During initialization the main thread spawns a total of four threads, two socket communication threads and two threads for parsing the information. The first thread briefly connects on the primary client interface to obtain the current firmware version. This is done in order to ensure backwards compatibility with older firmware versions. The first thread then disconnects from the primary and connects to the secondary client interface where it continuously monitor I/O and the robot's current state. A new thread is spawned to read data from the socket, parse the information and via a condition variable it notifies the main thread when new data is available. Then the main thread spawns a second communication thread that connects to the real time interface on port 30003 and a new thread is spawned that reads, parses and pass information back to the main thread. This means that the driver can operate at a 10Hz or 125Hz update rate.

The driver supports I/O interactions and all main URScript commands for motion, including commands needed for providing force control of the UR manipulator. It is also compatible with large ROS repositories that provides position and force control.

Chapter 3

Robotic simulation software

This chapter contains a short review of four potential simulation software candidates, arguments for choosing one for implementation, and a description of the software setup used in this project thesis.

The work presented here was done without some of the extensive background knowledge presented in chapter 2, especially the details about the ur_driver was at the time unknown to the author. This is important to note, as this information would have altered the selection outcome.

3.1 Short review of simulation software

Creating a accurate simulation environment is one the main goals of this project thesis. Previous attempts[4] at simulating the newly developed controller have failed due to a number of unknown reasons, but it was believed that communication problems between the Matlab script containing the controller and the simulation software V-REP was the main cause.

Some of the requirements for a potential simulation tool includes a preexisting UR5 model, accepting external controller inputs generated in Matlab, an adequate library and preferably a large community providing essential guidance and support.

Another preference by the author is easy compatibility with the robot operating system(ROS), preferably through a preexisting application programming interface(API).

Based on these requirements and preferences a total of four simulation software candidates where evaluated; V-REP, URSim, Gazebo and Matlab.

3.1.1 V-REP

V-REP is the simulation software used by Blokkum in her master thesis[4]. The software is developed by Coppelia Robotics and supports several programming languages and is available for Windows, Mac OS and Linux users. V-REP provides several external control inputs[23] options; child scripts within the V-REP platform, plugins, remote API,

ROS interfaces or through a tailored application script. The remote API contains built-in functions that allow communication with V-REP through the languages C/C++, Python, Java, MATLAB/Octave, Urbi and Lua. The ROS interfaces also allows V-REP to receive communication and control inputs from ROS nodes.

V-REP has several advantages; an intuitive GUI, accurate graphics engine, large community, extensive documentation and good online tutorials. However, it has some drawbacks, most noticeably the communication problems pointed out by Blokkum[Blokkum]. She chooses to communicate and pass control inputs from MATLAB to V-REP though a remote API. To avoid the problems discussed above it might be wise to choose a different way of communicating external control input, for example via the ROS interface or a different plugin.

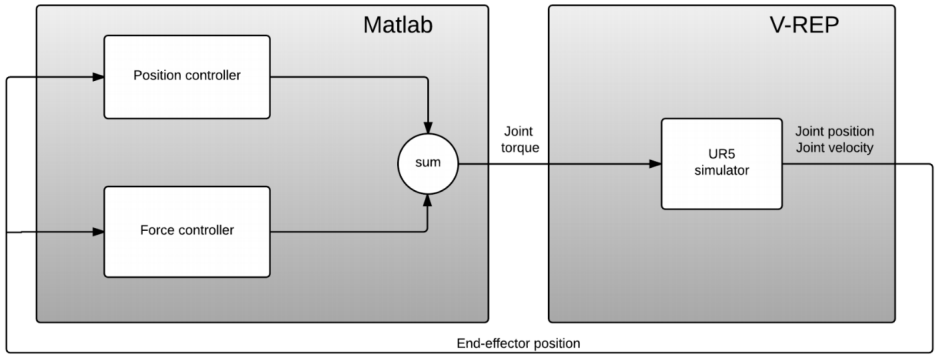

Figure 3.1: MATLAB to VREP communication used by Blokkum. Figure taken from [4].

Blokkum also speculated[4] that there might be an underlying built-in controller in the V-REP's UR5 model that interfered with the controller inputs. Some efforts have been made by the author to clarify this statement without yielding any conclusive results. It might be caused by the challenge with implementing force control described in section 2.2.2, but time restrictions prevented a deeper look into the code for the V-REP models she used.

Specific model code generated for the V-REP simulator can not be reused for the physical implementation, but using a ROS interface allows much of the code for controlling and analyzing to be written in Matlab and subsequently be reused at a later stage.

3.1.2 URSim

Universal Robots has created its own offline simulator[22]. It mainly runs on Linux(Mint 17.1 and Ubuntu 14.04) but it can also run on Windows inside a custom virtual machine(VM) that is based on Linux (Ubuntu).

URSim is a 3D robot manipulator simulator with the same GUI, called Polyscope, used with the physical UR robot manipulators. The simulation software comes with preexisting models of the UR3, UR5 and UR10. Using a simulator that is directly developed by the relevant robot manufacturer is tempting. One of the advantages is that it guarantees that

the kinematics and dynamics of the robot is correct. Detailed information of this kind is however actively shared by Universal Robots to all software developers and therefore provides no real advantage. There are some tutorials about URSim available, but they are limited and URSim lacks a proper support community.

URSim supports programming of the UR manipulator with Polyscope and URScript. The ability to reuse code used in the simulator for a physical implementation is considered an advantage, but as we discussed in chapter 2.4.1 there are some limitations using Polyscope and URScript. There is no available ROS interface for URSim, theoretically network communication between the simulator and Matlab would be possible. The required workload for establishing such an interface is extensive. Reading through the documentation revealed a potential problem with some of the software limitations to this simulator. *There are some limitations to the simulator since no real robot arm is connected. Specially the force control will be limited in use.* Although the exact source of the problems quoted in this statement is unknown to the author, they pose a serious threat to the desired outcome of the simulation environment. It raises some serious doubts about URSim's ability to handle controller inputs from an external source. In addition, the lack of proper documentation raises concerns about the accuracy and quality of the simulation.

3.1.3 Gazebo

Gazebo is a 3D dynamic simulator with the ability to accurately and efficiently simulate robots in complex environments. It was originally developed in 2002 by Dr. Andrew Howard and PhD Nate Koenig, and is now supported and developed by the Open Source Robotics Foundation(OSRF). Several high level concepts are involved in running a Gazebo simulation[10].

World files The world description file contains all the elements in a simulation, including robots, lights, sensors and static objects. The file is formatted using the Simulation Description Format(SDF), and typically has a *.world* extension. The Gazebo server reads this file to generate and populate a simulation environment.

Model files The model file uses the same SDF format as world files, but it only contains one specific model. The purpose of these files is to facilitate model reuse and simplify world files. Several models are provided by Gazebo's online model database, and even more can be sourced from the extensive community of Gazebo users.

Environment variable Environmental variables are used to locate files, and set up communication between the Gazebo server and clients. A proper software workspace will generate this set up automatically, but it can also be done manually by the user.

Gazebo server Through a command line the server parses a world description file and then simulates the world using a physics- and sensor engine. The server generates all the raw simulation data, but it does not include any graphical representation.

Graphical client The graphical client connects to the Gazebo server and visualizes all the elements in the simulation environment. It also provides basic tools for interacting with the running simulation. This interaction includes but is not limited to; spawning

and deleting objects, moving and scaling objects and modifying camera angle and lighting.

Plugins Plugins provide simple and convenient interfaces with Gazebo. They can be loaded though command lines, or by modifying a world/model file. Most plugins are loaded by the server, but some can be loaded by the client.

Gazebo is the preferred simulation environment used by the ROS community. Some key features include multiple physics engines, rich library of robot models and environments, wide variety of sensors and an intuitive GUI. It is supported by RVIZ, a powerful robot visualization tool, that provides visualization of path planning, current states, sensor inputs and logging information for debugging. Gazebo runs on Linux, supporting almost all main versions of Ubuntu depending on the chosen ROS version. It has an extensive library of preexisting robot models, a large community and several independent tutorial sources available. A large robotics repository called ROS-Industrial includes a model of the UR5 and UR10.

Since Gazebo and RVIZ are built on the ROS platform they use standard ROS interfaces by default. However it is possible to interact though other custom interfaces. The UR5 and UR10 models from ROS-Industrial is designed with a ROS interface that can communicate with a UR5 robot simulation in Gazebo and a physical UR5 robot. This ability to interconnect between a simulated and a physical robot is highly appealing for this project. It means that code generated for the simulation can be reused for implementation on a physical UR5 robot.

3.1.4 Matlab

Matlab has a free integrated package extension called Robotic System Toolbox with some existing models of robot manipulator. Of these, the *PUMA560* model is an older 6 DOF robot consisting of the standard layout of six prismatic joints. The model has an adequate level of real world accuracy and is fully integrated in the Matlab platform. The main idea is to retrofit the *PUMA560* robot model with the physical and kinematic parameters of the UR5. There are some clear kinematic differences between the UR5 and the PUMA560, but the PUMA560 model could be used as a template for a UR5 model, and it has the added benefit of avoiding cross platform communication problems between Matlab and a third party simulation software. Attempts have been made at contacting the author, or any of the code maintainers, of the Robotics Toolbox package through a support forum[14] with only one reply.

Unlike the other software discussed in this section Matlab is not a dedicated robotic simulation software. It lacks preexisting code support for robotic interactions with the environment like friction, collision, etc.. A member of the online community also warns about the lack of support for simulation of human machine interactions.

3.2 Selecting simulation software

URSim was for a long time a serious simulation software candidate for this thesis, but it had too many drawbacks with accepting external controller inputs and incomplete docu-

mentation. Blokkum's experiences and the inability to locate the source of the problems was a decisive factor for not choosing V-REP. Matlab falls short because it is not a dedicated simulation software, and it seem that the PUMA560 robot model is no longer be regularly maintained by the developers.

Ultimately, Gazebo was chosen as the simulation software for creating the simulation environment in this thesis. The ability to hot-swapping from simulator to hardware, large online community, extensive tutorials, preexisting UR5 model and recommendations from colleges at the cybernetics community at NTNU's department of engineering cybernetics were important factors in this decision.

In hindsight, as we will see in chapter 5, choosing Gazebo was not the best decision for the desired outcome of this project thesis. This is partly due to the challenges with implementing force control described in chapter 2.2.2, and will be discussed in greater detail in chapter 5.

3.2.1 Software setup

There are some software requirements that follow from using a simulation environment generated with Gazebo and the relevant ROS repositories and packages. ROS is mainly supported on Linux distributions including all stable versions of Ubuntu. There is no direct comparability with Windows and Mac OS, but a custom Linux based virtual machine(VM) with a pre-installed version of ROS is available for download. The latest three distributions of ROS are in chronological order from oldest to latest release; ROS Indigo Igloo, ROS Jade Turtle and ROS Kinetic Kame. All three distributions are still maintained and supported by the development team. The ROS-I repository requires the older distribution version ROS Indigo Igloo. In turn this ROS distribution version needs to run on a trusty version of Ubuntu 14.04.

Matlab is intended to be used for data analysis and direct interaction with the robot simulation. It is therefore important to have a ROS interface with the simulation environment. This can be provided through the Robotic System Toolbox that among other things includes a ROS interface for Matlab. It is uncertain how stable Matlab will be on a Linux platform, and as a preventive measure it was decided to run Matlab on a Windows OS. The architecture should not interfere with the cross-platform communication, since ROS is made to communicate with TCP/IP protocols. For example, the communication between the simulation environment in a Linux based VM and ROS nodes created in Matlab running in Windows.

The resulting software setup is a base OS running Windows 7 with and a VM running Ubuntu 14.04. The VM runs ROS Indigo Igloo with the ROS repositories and ROS packages discussed previously. From the base OS we will run Matlab version R2016b with the added extension Robotic System Toolbox.

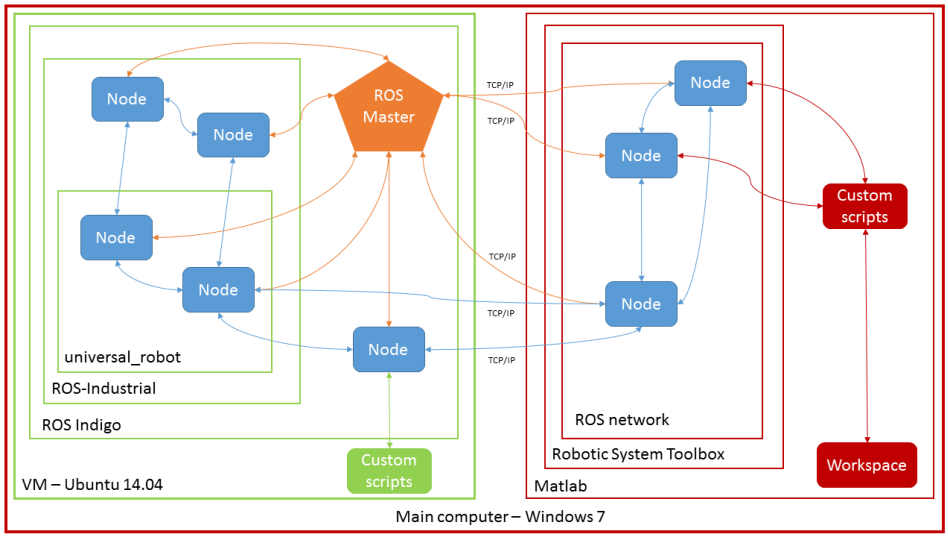

Figure 3.2: Final software setup. Base OS shown in red, VM in green, ROS nodes blue and ROS master in orange. All ROS communication are sent over TCP/IP, especially highlighted over the virtual bridge between the main computer and VM .

Chapter 4

Implementation and results

4.1 Building the simulation environment

The entire simulation environment is located in the /src folder inside the established catkin workspace. It consist of two packages;

- universal_robot
- project_thesis

The universal_robot package is part of the ROS-I repository, it contains the UR5 manipulator model, drivers, kinematics, configurations and a basic position controller. The project_thesis package contains python code for testing and analysis of the simulation environment.

4.1.1 Modifying the simulation environment

We need to make some modifications to the simulation environment to make it suit the needs of the robotic rehabilitation project.

4.1.1.1 Force Torque sensor

In order to implement a hybrid force and position controller we need to know the applied force by the patient at the end effector. In the physical UR5 robot this can be done by installing a force sensor at the contact point between the end effector and the patients wrist. In the Gazebo simulator this can be achieved through an existing plugin. Installing this plugin into the simulation environment is done by modifying a URDF file in the universal_robot package, specifically the *universal_robot/ur_description/urdf/ur.gazebo.xacro* file.

The following code is added to the URDF file;

```
<?xml version="1.0"?>
<robot xmlns:xacro="http://www.ros.org/wiki/xacro">
  <xacro:macro name="ur_arm_gazebo" params="prefix">
    <!-- Start of experimental code -->
      <gazebo reference="${prefix}wrist_3_joint">
        <provideFeedback>true</provideFeedback>
      </gazebo>

      <!-- Experimental code: The ft_sensor plugin -->
      <gazebo>
        <plugin name="ft_sensor" filename="libgazebo_ros_ft_sensor.so">
          <updateRate>80.0</updateRate>
          <topicName>ft_sensor_wrist_3_joint_topic</topicName>
          <jointName>${prefix}wrist_3_joint</jointName>
        </plugin>
      </gazebo>
      <!--End of experimental code -->
  </xacro:macro>
</robot>
```

As previously mentioned the URDF files are all written in XML. To the original URDF file we have added a ft_sensor plugin. This plugin adds a force/torque sensor to the wrist_3_joint. We can specify the desired sampling rate. According to Gertjan Ettema PhD, professor at Department of Neuroscience, Faculty of Medicine NTNU, the quickest human reaction time is approximately 25ms. The reaction time is measured from the initial human sense input is registered by the central nervous system, to movement begins in the extremities. This insight was gained during a course lecture[7] and subsequent discussions with the professor.

It is safe to assume that all other voluntary movements have a dynamic rate that is slower than 40Hz. In accordance with Nyquist sampling theorem we set the sampling rate to be twice as fast as the quickest dynamic in the system, in this case 80Hz.

The sensor output is published on a topic called ft_sensor_wrist_3_joint_topic. It is contained in a ROS message called geometry_msgs.msg.WrenchStamped. More information about the structure of the WrenchStamped ROS message can be found in the appendix.

From here it is relatively easy to create a ROS node that subscribes to the newly established topic and extracts the relevant force variables. Ultimately, these variables will be the sensor basis for force control of the UR5 robot simulation.

4.2 Communication between Matlab and ROS

In this thesis, Matlab is intended to be a tool for analysis and generation of patient force inputs. In order to establish a connection between Matlab and the ROS master running on the VM we need to initiate a ROS node inside Matlab, aptly named ”MatlabNode” for this thesis. It is then possible to utilize built-in functions provided by the Robotic System Toolbox to interact with the ROS network.

Figure 7.2.4 in the appendix shows a built-in function called applyForce. It takes in a torque and force vector and applies it to any Gazebo body in the simulation environment.

In this example we apply it to the robots end effector, simulating the contact point as between the robot and the patient.

4.3 Tools for preliminary testing

A ROS package called MoveIt was intended as a development tool for preliminary testing of the control interfaces. It is used together with RVIZ, which was discussed in section 2.3.2.4. MoveIt comes installed with the ROS-I repository and contains control theory, path planning algorithms and cost functions needed for trajectory planning and control. RVIZ provides an easy to use GUI with real-time visual representation of the robots state and animations of the planned path.

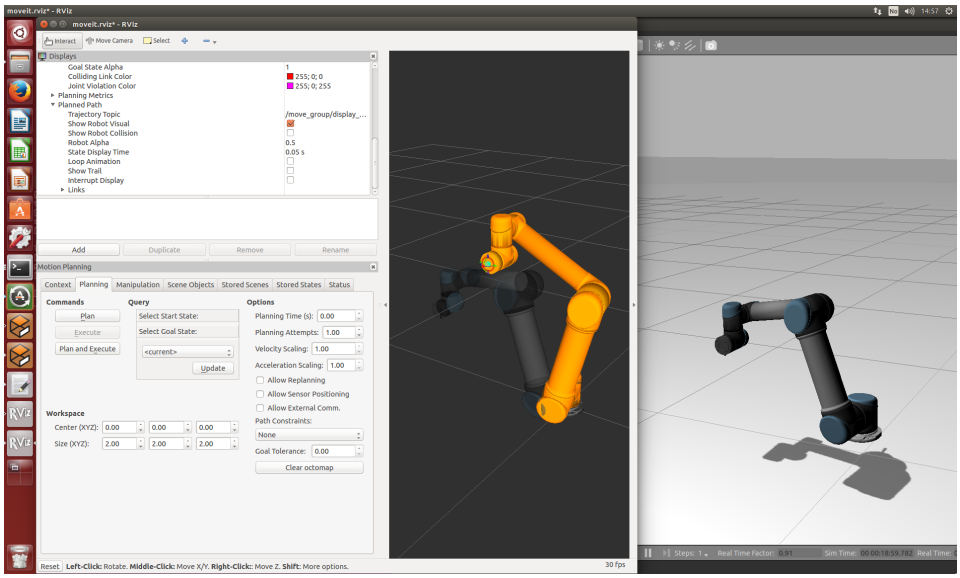

Figure 4.1: RVIZ with MoveIt used for preliminary testing of the control interfaces.

After preliminary testing are complete, it is intended to use Matlab for more detailed measurements. Including position and force control inputs, and calculation of the controller efficiency.

4.4 Resulting simulation environment

Combining everything we have learned about Gazebo, ROS, ROS-I and universal_robot we are now able to create a simulation environment for the Universal Robots UR5 manipulator inside the VM. The sensor modifications previously mentioned and the connection with Matlab are also running in the background.

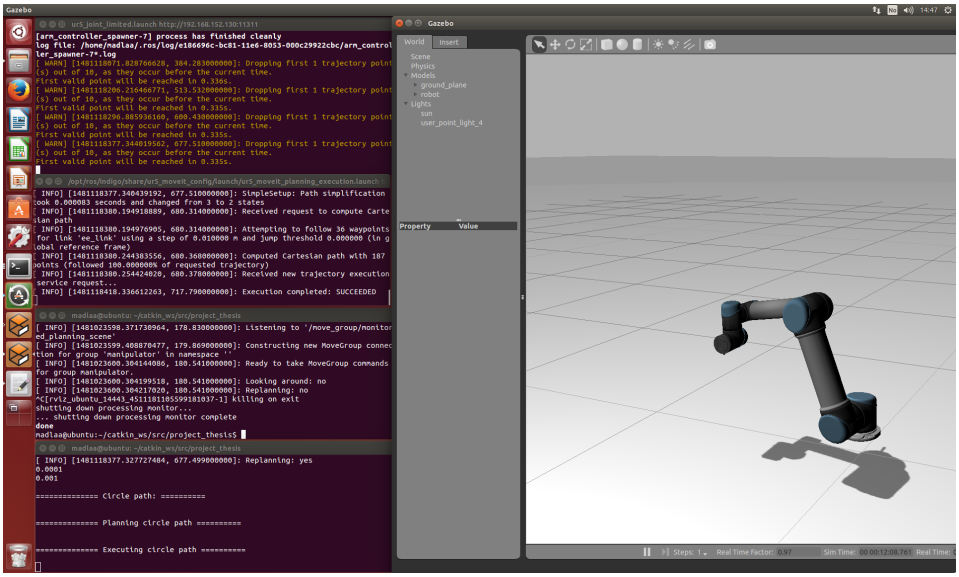

Figure 4.2: Simulation environment of Universal Robots UR5 manipulator.

Chapter 5

Analysis and discussion

5.1 Challenges with the simulation environment

Preliminary testing of the simulation environment revealed a number of challenges. This chapter gives a quick description of the main challenges and the suggested solutions.

5.1.1 Position control

Testing of position control in the simulation environment were done with MoveIt, an existing ROS motion library described in section 4.3. Interactions with MoveIt were done with RVIZ and a simple python test script. Testing reveled a challenge with using a highly developed and efficient motion library. Normally users feed a set of waypoints to MoveIt, which in turn generates a joint trajectory that is sent to the ur_driver for parsing to URControl or a simulation before being executed. Testing the standard trajectory builder revealed trajectories that rotates the end effector in a way that might harm the patient. This can happen even if the user specifies the exact position and orientation of the end effector in each waypoint. MoveIt is a sophisticated and advanced motion planning library, and as a consequence it will always try to find the optimal solution for a trajectory. All the joints in the UR5 robot has the potential to rotate $\pm 360^{\circ}$. An optimal solution according to MoveIt might include a large angular rotation in one of the robot joints in order to achieve a higher range of potential angular motion in the specific joint. This can result in a trajectory, with a fixed waypoint orientation in point A and point B, where the robot twists severely around one of the joints while transitioning from A to B. In the setting of this robotic rehabilitation project, with a delicate interaction between an industrial robot manipulator and a stroke patient, this is not a desired behavior.

Adopting a simpler Cartesian trajectory planner from the same library was the simple solution for testing purposes. It computes a simple and predictable trajectory where the end effector moves linearly in Cartesian space between the provided waypoints with a fixed orientation. A drawback with the new trajectory planner is an increased path planning time and a higher rate of rejected trajectories.

5.1.2 Force control

A big challenge was identified while testing force control in the simulation environment. The UR5 robot's standard firmware support both velocity and position control inputs, which is needed for realizing the suggested controller design described in section 2.1.1. During preliminary research of the universal_robot repository an incorrect assumption was made about the available control input options. It was believed that the ROS repository, designed with vendor specific details and real hardware interaction, would support both position and velocity control inputs. This assumption turned out to be incorrect. Inside the universal_robot repository the ur_driver package is responsible for providing the user with a communication and control interface to the robot. Research into the ur_driver, presented in section 2.5.1, showed that the driver only supports position control inputs through the joint trajectory interface. Without the ability to directly apply joint torque or joint velocity inputs it will not be possible to simulate the force control presented in the suggested controller design. Contacting the driver developers and other resource persons inside the ROS community concluded that it would be very difficult to alter the ur_driver to support velocity control.

Further research into alternative drivers for the UR manipulators are presented in section 2.5. The most promising candidate was the ur_modern_driver, which was created as a directly replacement for the ur_driver. The developer addressed and improved several problems with the original driver, and importantly added support for direct velocity control. Attempts to run the simulation environment with the new ur_modern_driver failed due to initialization problems. During the ur_modern_driver initialization process several threads are spawned to establish the firmware version and network connection protocol with the robot, described in more detail in section 2.5.3. However, this information is not provided by the simulated robot model. This makes it very difficult to connect the ur_modern_driver with the simulation environment. Attempts at altering the initialization of the driver by generating fake communications signals from the simulated robot have not been successful.

5.1.3 Simulation of patient force input

During testing patient force inputs were generated in Matlab by a function called applyForce and sent over a ROS interface to the Gazebo server. The function calls a ROS service provided by Gazebo called gazebo_msgs/ApplyBodyWrench.srv. This service applies linear force and torque to any Gazebo body with parameters that decide magnitude, starting time, duration and reference frame of the applied wrench. The specified Gazebo body was in this case the end effector of the UR5 robot, and the force was applied relative to the world frame. Generated patient force could then be registered by the force/torque sensor we added in section 4.1.1.1 and be used for direct force control with a feedback loop. This was an easy way of generating force on the end effector for early testing, but it is an inadequate representation of a human arm. Preliminary plans were made for adding the patient arm into the simulation environment, but time restrictions caused by the challenges described above did not allow for further research into this subject.

5.2 Implementation challenges

5.2.1 Controller design

Making changes to the suggested controller design is not a major focus in this thesis. However, there are some potential issues with the design that should be addressed.

The suggested controller design described in 2.1.1 switches between a position controller and a force controller. Regardless of the chosen rehabilitation strategy, instantly switching between a force- and position controller will be experienced by the patient as a sudden and abrupt change. To a patient that is inside the dead-zone while using assistive rehabilitation, moving outside the dead-zone and enabling the position controller will feel like hitting a brick wall. A dampening effect in the transition from force- to position controller would be preferred.

5.2.2 Wrist mounting

A suggested functional design of the physical interface between the rehabilitation robot and the stroke patient is a wrist-guard. The wrist-guard would also intentionally limit the stroke patients range of motions in the wrist. This could be a basis for constraint inducement movement therapy discussed in section 2.1.2.

Constraint induced movement therapy in the upper-limb after a stroke have been discussed[7] in October 2016 with Torunn Askim PhD, associate professor at Department of Neuroscience, Faculty of Medicine NTNU. She stated that; "If there are any benefits, they are not well documented within upper-limb stroke rehabilitation." She does not recommend this rehabilitation strategy on stroke patients. As a suggestion she recommends to only include patients that are able to move their wrist $\pm 10^{\circ}$ in all directions. This would ensure that patients have a minimal functionality in the wrist, and therefore be more responsive to the robotic rehabilitation efforts.

5.2.3 Maximum force emergency shutdown

It is unclear whether a built-in safety mechanism, where all forces acting on the robots TCP exceeding 100N will trigger an emergency shutdown, is going to be a challenge in the implementation process. The human arm constitutes approximately 6,5% of the total body weight[4], given an average human male weighing 80 kg this translates to roughly 5.2 kg. Since the UR5 robot has a maximum carrying capacity of 5 kg, we have to assume that stroke patients in the robotic rehabilitation program are unable to generate a enough muscular force to counteract the pull of gravity and stay within the UR5s carrying capacity. However, patients that meet the requirements for participating in the robotic rehabilitation program might suffer from spasticity, a medical condition leading to a sudden stiffness or tightness in muscles. It is likely that weak patient suffering from spasticity will pose a challenge for the robotic rehabilitation project. Triggering the emergency shutdown could occur when the robot is carrying much of the patient's arm weight and the patient experiences a spastic arm reflex in a vector pointed downwards.

Patients with a high frequency tremble might also pose a similar challenge to the robotic rehabilitation project. It is not possible to disable the safety mechanism while

using standard firmware.

5.2.4 Programming basis

In section 2.4.1.1 the possibilities of using the C API as a programming basis for control over the UR5 manipulator is discussed. Legal speculations are a bit outside the scope of this thesis, but the plausibility and potential ramifications makes it a relevant subject. As an example, the TÜV safety certificate is issued directly to URSafety, a safety subsystem of the original firmware. More details about the UR5 specific TÜV safety classification can be found in the appendix. Running a custom program parallel with, or as a replacements of, the original firmware might legally be seen as a significant change of the original product. Completely replacing it means that the built in safety mechanisms in the original firmware will be ignored. Such an intrusive change will most likely result in the UR5 product losing its TÜV safety and ISO classifications. Such a scenario would pose a big challenge for the robot rehabilitation project, where the end goal is close collaboration between the UR robot manipulator and stroke patients. Work towards approving the project for human clinical trials will most likely be set back significantly in time, and the cost of reinstating the safety classifications.

A decision to use the C API or URScript as a programming basis will significantly affect the controller design. Using the C API as a total replacement for the original firmware will undoubtedly be the best solution for a low level controller, as it provides direct control over joint torque and joint velocity in each joint. Using a driver based on the C API will result in a similar level of control without the added workload of designing a new driver. The URScript programming basis is safer in a legal sense but more constrained. The available motion commands have some clear limitations, and the lowest control level provided by drivers based on this method is a joint velocity interface provided by the ur_modern_driver.

Chapter 6

Conclusion

6.1 Feasibility of simulation environment

The resulting simulation environment handles position control inputs but currently fails at providing a simulation basis for force control inputs. With the current setup it is not possible to realize the suggested robotic rehabilitation strategy inside the simulation environment created in this thesis.

6.2 Physical implementation

6.2.1 Constraint induced movement therapy

Based on the recommendation made by Torunn Askim the decision is to not implement constraint induced movement therapy for this project- and master thesis. This implies that the wrist mounting between the robot and the patient must be developed to ensure minimal restrictions on the patients wrist.

6.2.2 Minimal sampling rate

The discussion in section 4.1.1.1 regarding the force/torque sensor sampling rate resulted in a recommended minimal sensor sampling rate. The same argumentation can be used for a physical implementation.

The recommended minimum sampling rate regarding for the physical force/torque sensor and the suggested controller design is 80 Hz.

6.2.3 Programming basis for UR manipulator

Based on the concerns discussed in 5.2.4, all drivers based on the C API were excluded from the reviews presented in 2.5. This approach is also believed to be unsuited for further physical implementations.

The recommendation is a driver based on URScript commands. Several potential drivers were reviewed in section 2.5. The ur_modern_driver is believed to be the best candidate for further development. The driver exposes a velocity interface that could be a used as a basis for force control, but the resulting controller needs to be verified in experiments.

6.2.4 Patient exclusion

The research and conclusions presented in this thesis set some requirements the patients must fulfill before they are able to participate in the robotic rehabilitation program.

The decision to use standard firmware implies that weak patients suffering from spasticity or high frequency trembles should be excluded from robotic rehabilitation until they are able to support the majority of their own arm weight. The exact criteria needs to be established based on further research.

Another requirement is that patients should be able to generate a minimal wrist motion of $\pm 10^{\circ}$ in all directions in the affected arm.

Together, these criteria ensures a minimal level of functionality in the stroke patients upper-limbs. This provides a platform for further rehabilitation effort provided by the robotic rehabilitation strategies.

Chapter 7

Future work

Work with the robotic rehabilitation project will be continued by the same author in a master thesis, with the guidance of Erik Kyrkjebø and Øyvind Stavdahl. This will ensure continuity and steady progress for the robotic rehabilitation project.

7.1 Improvements to the simulation environment

7.1.1 Implementing force control

With an experienced programmer and enough time it is possible to create support for force control input within the current simulation software setup by modifying the current ur_driver or the newly developed ur_modern_driver. The latter option should be preferred as it is still regularly maintained and supported by the developer. The challenges with force control described in section 2.2.2 should also be avoided.

Another solution is to try again with the V-REP simulator, based on the review presented in section 3.1 it should provide a good basis for the simulation environment. It is recommended to use a ROS interface for providing control inputs, instead of the plugin Blokkum used in [4] for communication with Matlab.

7.1.2 Simulation of patient force inputs

The robotic rehabilitation project require a simulation environment with a realistic representation of patient force inputs. The temporary solution applied in this thesis was always intended to be improved, but time restrictions prevented further development.

7.2 Physical implementation

The cybernetics department at the Norwegian university of science and technology will provide a physical UR5 robot for experimental implementation of the robotic rehabilitation

project in a future master thesis. The department will also contribute a 6-axis force/torque sensor that can be used for accurate measurements of patient force inputs.

7.2.1 Improvement to controller design

Improvements to the controller design need to be made in order to achieve the desired dampening effect discussed in section 5.2.1.

For example, a suggested controller design solution related to assistive rehabilitation can be realized by a linear quadratic regulator (LQR). It requires redefining the dead-zone around the desired trajectory from a hard constraint, to a soft constraint. The LQR would than penalize deviations from the desired path, and at the same time, penalize the controller for failing to amplify the patient force inputs. In a similar manner, rehabilitation based on challenge-based error-amplification can be realized by inverting the sign of the cost function parameters. This controller design will be provide the desired behavior related to the two main rehabilitation strategies, but with some drawbacks. The feasibility of the newly suggested controller design is unclear. It requires a mathematical model of the system, something that is currently not developed. Tuning the cost parameters will also be a challenge. Further development and research into the newly suggested controller design is necessary.

7.2.2 Patient exclusion criteria

The minimal muscle strength required by stroke patients suffering from spasticity and high frequency tremble needs to be defined. There should be a general limit, and a specific limit for Universal Robots UR5 manipulator. A standard medical test suited for determining the muscle strength in stroke patients should also be identified.

7.2.3 Wrist mounting

Before a physical implementation of the robotic rehabilitation project can be realized, the physical interface connecting the robot end effector and the wrist of the stroke patient needs to be developed. Based on the discussion in section 5.2.2, the wrist mounting should not interfere with any movement in the wrist.

7.2.4 Experimental verification

Implementing the robotic rehabilitation strategy with a driver based on the URScript and the provided Ethernet interfaces needs to be verified in experiments. The feasibility, efficiency and safety of the resulting controller is unknown. There is a small chance that a velocity interface will not provide a sufficient basis for the robotic rehabilitation requirements.

Bibliography

[1] Sorce code for ur_modern_driver. https://github.com/ThomasTimm/ur_modern_driver, Desember 2016.

[2] Source code for ur_driver. https://github.com/ros-industrial/universal_robot/tree/indigo-devel/ur_driver, Desember 2016.

[3] Thomas Timm Andersen. *Optimizing the Universal Robots ROS driver.* Technical University of Denmark, Department of Electrical Engineering, 2015.

[4] Kristine Blokkum. Design and analysis of a hybrid position/force controller for robotic rehabilitation of upper limb after stroke. Master's thesis, Norwegian University of Science and Technology, Department of Engineering Cybernetics, 2016.

[5] Kristine Blokkum. Functional review on robotic rehabilitation of upper limb after stroke. Master's thesis, Norwegian University of Science and Technology, Department of Engineering Cybernetics, 2016.

[6] Sylvester J. Campbell. *Solid-State AC Motor Controls.* Marcel Dekker, Inc., 1987.

[7] Faculty of Medicine NTNU Department of Neuroscience. Bev3103 - movement control and motion problems. http://www.ntnu.edu/studies/courses/BEV3103, October 2016.

[8] Jens G. Balchen & Trond Andresen & Bjarne A. Foss. *Reguleringsteknikk*. NTNU, Institutt for teknisk kybernetikk, 2003.

[9] Open Source Robotics Foundation. Artistic impression of ros. http://www.ros.org/history/, December 2016. See figure 3.1.2 on page 11.

[10] Open Source Robotics Foundation. Gazebo simulator tutorials. http://gazebosim.org/tutorials, December 2016.

[11] Open Source Robotics Foundation. Ros-i overview. http://wiki.ros.org/Industrial, December 2016.

[12] Open Source Robotics Foundation. Ros-i overview. http://www.ros.org/, December 2016.

[13] Helsedirektoratet. Nasjonal retningslinje for behandling og rehabilitering ved hjerneslag. *ISBN-978-82-8081-153-0*, 2010.

[14] Mads Johan Laastad. Retrofitting the puma560 into a ur5 model. https://groups.google.com/forum/#!topic/robotics-tool-box/9RNSE6021Fs, September 2016.

[15] P. Maciejasz. A survey on robotic devices for upper limb rehabilitation. *J Neuroeng Rehab. R&D*, 2014.

[16] Hocoma media center. Example image of robotic rehabilitation. https://www.hocoma.com/uploads/pics/1508_armeopower.jpg, December 2016.

[17] Lofthus & Meyer. Consequences of hip fracture on activities of daily life and residential needs. 2004.

[18] Christopher Murray and Alan Lopez. The world health report 2002 - reducing risks, promoting healthy life. pages 7–14, 2002.

[19] N. Norouzi-Gheidari, P.S. Archambault, and J. Fung. Effects of robot-assisted therapy on stroke rehabilitation in upper limbs: Systematic review and meta-analysis of the literature. *J. of Rehab. R&D*, pages p. 479–495, 2012.

[20] Prestmo. The trondheim hip fracture trial. 2015.

[21] Industrial robot. Teach pendant image. http://industrialrobotinfo.apps-1and1.net/wp-content/uploads/2015/04/ur_teach_pendant_moving-co.jpg, December 2016.

[22] Universal Robot. Offline simulation of universal robot products. http://www.universal-robots.com/download/?option=23952#section16597, December 2016.

[23] Coppelia Robotics. External controller tutorial. http://www.coppeliarobotics.com/helpFiles/, December 2016.

[24] Universal Robots. Service manual for ur5 with cb3.0/cb3.1-controller. https://www.universal-robots.com/download/?option=15833#section15832, December 2016. See figure 3.1.2 on page 11.

[25] Universal Robots. Service manual for ur5 with cb3.0/cb3.1-controller - limiting safety functions. https://www.universal-robots.com/download/?option=15833#section15832, December 2016. See table on page I-14.

[26] Universal Robots. Ur5 user manual. https://www.universal-robots.com/download/?option=22045#section22032, December 2016.

[27] ROS. Urdf overview. http://wiki.ros.org/urdf, December 2016.

[28] Oliver Roulet-Dubonnet. Source code for python-urx. `https://github.com/SintefRaufossManufacturing/python-urx/tree/master/examples`, December 2016.

[29] Johannes Schrimpf. *Sensor-based Real-time Control of Industrial Robots*. Norwegian University of Science and Technology, Department of Engineering Cybernetics, 2013.

[30] Luciana B. Sollaci and Mauricio G. Pereira. *The introduction, methods, results, and discussion (IMRAD) structure: a fifty-year survey.* Journal of the Medical Library Association, 2004.

[31] Andreas Stolt. *On Robotic Assembly using Contact Force Control and Estimation Department of Automatic Control.* PhD thesis, Lund Institute of Technology, Lund University, 2015. Figure 2.4, page 23.

[32] Mark W. Spong & Seth Hutchinson & M. Vidyasagar. *Robot modeling and control*. John Wiley & Sons, Inc., 2006.

Appendix

ROS message WrenchStamped

We can get more information about the structure of the WrenchStamped ROS message using the ROS command "rosmeg show <ROSmsgtype>":

```
  madlaa@ubuntu:~/catkin_ws$ rosmsg show WrenchStamped
  [geometry_msgs/WrenchStamped]:
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
geometry_msgs/Wrench wrench
  geometry_msgs/Vector3 force
    float64 x
    float64 y
    float64 z
  geometry_msgs/Vector3 torque
    float64 x
    float64 y
    float64 z
```

Matlab node

The following code snippet shows a skeleton framework for establishing a Matlab ROS node, and connecting it to the Gazebo simulation environment.

```
%% Initialize matlab node and connect to ROS master
ip_matlab_node = '129.241.154.109';
ip_ur5_robot = '192.168.152.130';
rosinit(ip_ur5_robot, 'NodeHost', ip_matlab_node, 'NodeName', 'MatlabNode'
    );

%% Get list of all models currently running in gazebo
models = getSpawnedModels(gazebo);

%Lanch file catkin_ws/src/universal_robot/ur_gazebo/launch/ur5.launch
    spawns UR5 model with name tag 'robot'
gazebo = ExampleHelperGazeboCommunicator();
ur5 = ExampleHelperGazeboSpawnedModel('robot', gazebo);
[ur5Links, ur5Joints] = getComponents(ur5);

%% Generate force on the UR5s end effector
stopTime = 5; % [Seconds], if negative time the force acts indefinitly
forceVector = [20 -20 -0];      % [Newtons], applyed in world frame
torqueVector = [0 0 0];     % [Newton-meters], applyed in world frame
applyForce(ur5, ur5Links{7}, stopTime, forceVector, torqueVector);

%rosshutdown;
```

The launch file used to spawns the UR5 model automatically generates a name tag to the Gazebo model, in this case 'robot'.

ZERTIFIKAT
CERTIFICATE

Hiermit wird bescheinigt, dass die Firma / *This certifies, that the company*

Universal Robots A/S
Energivej 25
DK-5260 Odense S
Denmark

berechtigt ist, das unten genannte Produkt mit dem abgebildeten Zeichen zu kennzeichnen.
is authorized to provide the product mentioned below with the mark as illustrated.

Fertigungsstätte:
Manufacturing plant:

Universal Robots A/S
Energivej 25
DK-5260 Odense S
Denmark

Beschreibung des Produktes:
(Details s. Anlage 1)
Description of product:
(Details see Annex 1)

Universal Robots Safety System URSafety 3.1
for **UR10, UR5** *and* **UR3 robots**

Geprüft nach:
Tested in accordance with:

EN ISO 13849-1:2008, PL d

Registrier-Nr. / *Registration No.* 44 207 14097602
Prüfbericht Nr. / *Test Report No.* 3515 4327
Aktenzeichen / *File reference* 8000443298

Gültigkeit / *Validity*
von / *from* 2015-06-02
bis / *until* 2020-06-01

Zertifizierungsstelle der TÜV NORD CERT GmbH

Essen, 2015-06-02

TÜV NORD CERT GmbH Langemarckstraße 20 45141 Essen www.tuev-nord-cert.de technology@tuev-nord.de

Bitte beachten Sie auch die umseitigen Hinweise
Please also pay attention to the information stated overleaf

ANLAGE
ANNEX

zum Zertifikat Registrier-Nr. / ***to Certificate Registration No.*** **44 207 14097602**

Allgemeine Angaben: *General information:*	**The system under test and certification is the URSafety 3.1, but not the entire UR10, UR5 and UR3 robots**
Typenbezeichnung: *Type designation:*	**Universal Robots Safety System URSafety 3.1 for UR5**
Gelenk: *Joint:*	**6 rotating joints (3 x joint size 3, 3 x joint size 1)**
Nennspannung: *Nominal voltage:*	**100-240 VAC, 50-60 Hz**
I/O Spannung: *I/O power supply:*	**24 V, 2 A in control box**
Umgebungstemperatur: *Ambient temperature:*	**0-50 °C**
Nutzlast des Roboters: *Payload of the robot:*	**5 kg**
Reichweite des Roboters: *Reach of the robot:*	**850 mm**
Auslenkung der Gelenke: *Joint range:*	**±360° on all joints**
Max. Geschwindigkeit im Betrieb: *Max.speed in normal operation:*	**Joint: 180°/s** **Tool: approx. 1 m/s**
Schutzart: *Protection class:*	**Robot arm: IP 54** **Control box and teach pendant: IP 20**

F. Hegenfeld

Zertifizierungsstelle der TÜV NORD CERT GmbH — Essen, 2015-06-02

ANLAGE
ANNEX

zum Zertifikat Registrier-Nr. / ***to Certificate Registration No.*** **44 207 14097602**

Allgemeine Angaben: *General information:*	**The system under test and certification is the URSafety 3.1, but not the entire UR10, UR5 and UR3 robots**	
Typenbezeichnung: *Type designation:*	**Universal Robots Safety System URSafety 3.1 for UR5**	
Hardwareversion: *Hardware version:*	**Safety Control Board:**	**Revision B**
	Joint size 1:	**Revision H**
	Joint size 3:	**Revision I**
	Screen board:	**Revision D**
	Current distributer	**Revision F**
Softwareversion: *Software version:*	**SCB µCA:**	**3.1.453**
	SCB µCB:	**3.1.189**
	Joint 0:	**3.1.17195**
	Joint 1:	**3.1.17195**
	Joint 2:	**3.1.17195**
	Joint 3:	**3.1.17195**
	Joint 4:	**3.1.17195**
	Joint 5:	**3.1.17195**
	Teach Pendant 1, 2:	**3.1.17195**
	Polyscope	**3.1.17195**
Checksumme: *Checksum:*	**SCB µCA:**	**0x5C6D7EE1**
	SCB µCB:	**0xC447B415**
	Joint 0:	**0xC0592E78**
	Joint 1:	**0xB3D67736**
	Joint 2:	**0x90602481**
	Joint 3:	**0x052D3334**
	Joint 4:	**0xC8F438BE**
	Joint 5:	**0xC5E3E9A1**
	Teach Pendant 1, 2:	**0xBF02C153**
	Polyscope	**0x20439EA3**

F. Hegenfeld

Zertifizierungsstelle der TÜV NORD CERT GmbH — Essen, 2015-06-02

ANLAGE
ANNEX

zum Zertifikat Registrier-Nr. / ***to Certificate Registration No.*** **44 207 14097602**

Allgemeine Angaben: *General information:*	**The system under test and certification is the URSafety 3.1, but not the entire UR10, UR5 and UR3 robots**
Typenbezeichnung: *Type designation:*	**Universal Robots Safety System URSafety 3.1 for UR10, UR5 and UR3 robots**

Geprüfte Sicherheitsfunktionen:
Safety functions tested:

- **Emergency Stop (Execution monitoring of the emergency stop)**
- **Safeguard Stop (Execution monitoring of the safeguard stop)**
- **Joint Position Limit (Monitoring of the joint position limit)**
- **Joint Speed Limit (Monitoring of the joint speed limit)**
- **Joint Torque Limit (Monitoring of the joint torque limit)**
- **TCP Pose Limit (Monitoring of the TCP pose limit)**
- **TCP Speed Limit (Monitoring of the TCP speed limit)**
- **TCP Force Limit (Monitoring of the TCP force limit)**
- **Momentum Limit (Monitoring of the momentum limit)**
- **Power Limit (Monitoring of the power limit)**
- **System Emergency Stop Output (Monitoring of the System Emergency Stop Output)**
- **Robot Moving Digital Output (Monitoring of the Robot Moving Digital Output)**
- **Robot Not Stopping Digital Output (Monitoring of the Robot Not Stopping Digital Output)**
- **Reduced Mode Digital Output (Monitoring of the Reduced Mode Digital Output)**
- **Not Reduced Mode Digital Output (Monitoring of the Not Reduced Mode Digital Output)**

F. Hegenfeld

Zertifizierungsstelle der TÜV NORD CERT GmbH — Essen, 2015-06-02

ANLAGE
ANNEX

zum Zertifikat Registrier-Nr. / *to Certificate Registration No.* 44 207 14097602

Allgemeine Angaben: *General information:*	**The system under test and certification is the URSafety 3.1, but not the entire UR10, UR5 and UR3 robots**
Typenbezeichnung: *Type designation:*	**Universal Robots Safety System URSafety 3.1 for UR10, UR5 and UR3 robots**
Kategory: *Category:*	**The system possesses a heterogeneous system architecture. Detailed information about the category acc. EN ISO 13849-1:2008 for each safety function can be found in the technical report (report No. 3515 4327, 2015-05-27).**
Wichtiger Sicherheitshinweis: *Important safety notice:*	**The robot is designed to be programmed/ integrated by users for specific collaborative applications. A risk assessment is therefore required for each installation of the robot. It is particularly important that all information in the user manual should be read with close attention and all the safety instructions in the user manual must be rigorously followed.**

F. Hegenfeld

Zertifizierungsstelle der TÜV NORD CERT GmbH — Essen, 2015-06-02

TÜV NORD CERT GmbH Langemarckstraße 20 45141 Essen www.tuev-nord-cert.de technology@tuev-nord.de