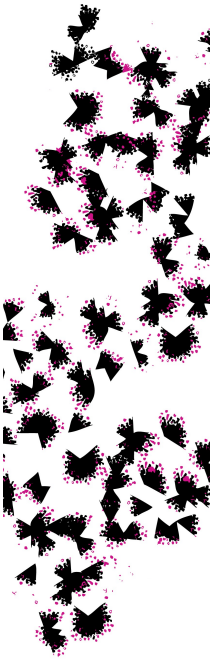


Teleoperation using passive control and LUNA network channels

S.F.J. (Sjoerd) Nijhof

MSc Report

Committee:
Dr.ir. P.C. Breedveld
Dr.ir. D. Dresscher
Dr.ir. D. Reidsma

June 2018

012RAM2018
Robotics and Mechatronics
EE-Math-CS
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

Summary

This is the report of a master thesis project executed at the Robotics and Mechatronics (RaM) group at the University of Twente. In the project, two research lines are combined. i-Botics is a joint innovation center with current research on teleoperation systems. Teleoperation is the control of a robot at a remote location by a human user. Usually, position of the robot is controlled by the user and the user receives force feedback of interaction between the robot and the remote environment. The other research line is focused on design of embedded control software for robotic applications. A certain way of working is developed at the RaM group. The TERRA tool and LUNA software framework are created to support this method. Recently, the LUNA framework is complemented with network channels, which enable the design of systems distributed over multiple PC's via a network. An example of such a system is a teleoperation system.

The goal of this project is to design, implement and test a teleoperation system to assess the suitability of LUNA network channels for teleoperation. Properties of the LUNA network channels that could limit the performance of a teleoperation system are introduced time delays and limited data transfer rate.

In this work a teleoperation system is realized by controlling a KUKA LWR robot (the slave device), based on commanded motion given by a user via an Omega.7 device (the master device). Control of the slave device and force feedback to the user are generated using an impedance controller, which can be seen as a virtual spring. The motion of the end effector of the slave device follows the motion of the master device and the force required to perform this is used as feedback to the user. The similarity of force and position at master and slave side (transparency) is used as technical performance measure. The slave device can only be actuated by joint torques, whereas the position of the end effector must be controlled. The geometric Jacobian is used as a power-continuous transformation between joint space and Cartesian space of the end effector. Virtual damping is added to the controller to prevent undamped motion. The damping is done with a velocity estimate based on position measurements via state variable filters.

Time delays are in general a problem for control systems. Passivity layers are used to deal with time delays introduced by network communication. The controller is created as a passive component, such that stability of the system can be guaranteed. Passivity layers monitor the interaction energy between the controllers and devices and limit control action such that energy in the controller is always nonnegative. By preventing output of 'virtual' energy, the controller is guaranteed stable. It was not possible to use LUNA network channels at the desired operation rate of the controller, therefore proper rate conversion was designed for signal and energy exchanged between master and slave controller via communication.

Performance of the teleoperation system with LUNA network channels was compared to teleoperation systems without network communication and ROS network communication to find relative differences. Functional tests were executed to test timely execution of digital control actions and to test system transparency. Timely execution was significantly different from the design for the system with LUNA network channels. Tests with other systems performed as designed. Nondeterministic timing in the implementation with LUNA network channels is thought to be caused by high computational load due to the current implementation of the LUNA network channels. Transparency was tested by means of an integration test. Results show that performance of systems with LUNA network channels and ROS communication is slightly worse than the system without network communication due to oscillations and delayed position and force signals.

Experiments with ten human users and a real robotic setup were conducted to see the relative influence of LUNA network channels on teleoperation task performance. A free space positioning task and peg-in-hole task were performed per implementation. Performance was measured by the mean absolute position error and mean absolute interaction force respectively. Perception of the users was measured by asking users to give a score on the difficulty of using each system. It was found that results for the positioning experiment are significantly equal. Significant equality could not be judged about the peg-in-hole experiment and user perception due to large variance in the results. Nevertheless, results suggest approximately equal performance.

Based on the measurement results, the suitability of the LUNA network channels for teleoperation was assessed. It was concluded that the transparency of systems with LUNA network communication and ROS network communication is slightly lower than for a system without network communication. Nevertheless, performance and perception measurements of task execution are suggested to be approximately equal with lack of significance, for teleoperation systems without network communication, with LUNA network communication and ROS network communication.

The main recommendation for the LUNA network channels is that the computational load could be reduced by a redesign of the current version. The main recommendation for enhancement of theory for teleoperation systems is to improve the interaction-energy-estimate-based limit for controller effort saturation for systems with multiple degrees of freedom.

The main limitations in this work are the user experiments. Experiments results could be improved by for example replacing the real robotic setup with a simulation, such that properties of the LUNA network channel could be assessed with more significance.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Context

Robots that are controlled from a remote location by humans to conduct a certain task are called teleoperation robots. Teleoperation robots offer a solution for scenarios in which a (complex) task must be executed and it is impossible or undesired to let it be performed by humans directly. Examples are tasks in extraterrestrial, submarine, toxic or radioactive areas. Complex tasks require decision making and dexterous manipulation of objects. Teleoperation utilizes cognitive skills and muscle motor skills of humans at a remote location to perform a task. The human control of teleoperation robots is often facilitated with haptic force feedback to let the human obtain an intuitive perception of the environment of the robot. Design challenges in the field of teleoperation are the perception of telepresence for the operator and the guarantee of a stable system despite time delays, when the robot moves in free space and when the robot interacts with an object or the environment (Hogan, 1989).

In nature, all physical systems are considered passive, i.e. do not produce energy. In digital control algorithms for physical systems, it is possible to create 'energy producing' controllers. With passive control, passivity is used to design control algorithms that guarantee stable behavior. This always holds, also in the presence of time-varying destabilizing factors such as variable communication delays found in teleoperation systems (Franken et al., 2011). Passivity is therefore a useful concept for the design of teleoperation systems.

i-Botics, a collaboration between TNO and the University of Twente, is a joint innovation center for interaction robotics. The two main research lines are telerobotics (control of possibly semi-autonomous robots from a distance) and exoskeletons. The activities with respect to telerobotics are in line with teleoperation and this project. i-Botics also focuses on the development of reusable software, such that robotic software development time can be reduced.

Next to research for teleoperation, research is done on software development for robotics at the Robotics and Mechatronics (RaM) group at the University of Twente. A certain 'first-time right' way of working (Broenink et al., 2016) is developed, which aims to have correctly functioning control software the first time a computing platform is connected to a mechanical setup. This is supported by the TERRA tool (Bezemer et al., 2012) and LUNA framework (Bezemer et al., 2011). LUNA is a hard real-time, multi-threaded, CSP-capable execution framework designed for embedded control software. In Wijnholt (2017) a DDS-based network channel is realized with the capability of connecting two real-time LUNA applications. With this network channel it is possible to design applications in LUNA that run on two computing devices.

The recent developments for robotic software design in LUNA have enabled the design of a system of computing devices connected via a network. The research of teleoperation can be combined with the embedded software design methodology by creating a teleoperation system implemented within the LUNA framework.

1.2 Problem Statement

The design methodology as described by Broenink et al. (2016) was first supported with a software framework for single computing platforms. With the realized LUNA network channel (Wijnholt, 2017), it is now possible to extend the design methodology to systems that consist of multiple platforms. An example system is a teleoperation robot. The LUNA network channel does not guarantee hard real-time performance, which will result in limitations for the implementation of a teleoperation robot. Limitations are mainly due to time delays and low data transfer rates, which can cause stability issues. As is recommended by Wijnholt (2017), the application of a passive controller as a layer above the LUNA network channel is a next step

to guarantee a stable closed loop system. However, application of a passive controller does not solve every aspect of a bad network, the teleoperation system could for example have a bad servo performance or barely present haptic force feedback to the user. The LUNA network channel limitations need to be investigated for effects on the operation of a higher-level teleoperation system.

1.3 Project Goal

The goal of this project is to design, implement and test a teleoperation system that includes the LUNA network channel. To evaluate task performance of a user with a teleoperation system, it is required that the system supports motion in multiple degrees of freedom. Therefore, a teleoperation system using a serial robotic manipulator is the subject in this project. The main research question in this project is:

"To which extend is the LUNA network channel suitable for application in teleoperation systems?"

An answer to this question will be generated by performing functional tests and a user-based study on the complete system. Functional tests will give result about differences in technical performance changes of the system by including LUNA network channels. The user-based study will give result on a higher level. The effects of the LUNA network channel, or a technical performance change in general, can be evaluated based on the effect on the performance of task execution. This relates technical performance of a part of the system to performance on the level of system operation. The results can subsequently be used to assess the suitability of the LUNA network channel for teleoperation systems. An additional outcome of the project could be a base teleoperation system that can be used in future research.

1.4 Outline of this Report

This work consist of several chapters. In chapter 2 background information is provided. Main topics are the basics of teleoperation, the 'first-time right' design approach and characteristics of different middleware. In chapter 3, the goal of this project is considered more in depth and design choices combining hardware, theoretical concepts, implementation methodologies and testing are motivated. In chapter 4, the design and implementation are discussed in detail. Materials and methods for verification of the design using functional and user-based tests are described in chapter 5. In chapter 6, the results of described tests are shown and interpreted. In chapter 7, the conclusion of this project is stated and recommendations for future work are given.

2 Background

In this chapter it is described which existing information is relevant for this project. It can be separated into interesting concepts to be implemented (2.1), background information (2.2, 2.3, 2.5, 2.6) and methods that can be used for implementation (2.4, 2.7). Sections can be skipped if the reader is familiar with the corresponding subjects.

2.1 Teleoperation

As already stated in the introduction, a teleoperation robot is a robot that is controlled from a remote location by a user. Teleoperation must not be confused with telerobotics, as telerobotics is also concerned with autonomous tasks that a robot can perform. A typical setup for teleoperation is schematically depicted in figure 2.1, which consist of a master and slave side that communicate. If the communication between master and slave is implemented as (part of) a network, the system could also be called a network-distributed control system. Both master and slave sides consist of a controller and a device. The devices can either be a device that interacts with the human user or a robot that interacts with its environment. The main function is to let the slave device be controlled by the user and give haptic force feedback to the user if the slave device has interaction forces with its environment. If the master device presents information about the interaction between the slave system and remote environment to the user, there is a bilateral signal flow between master and slave sides (Franken et al., 2011). Different physical properties can be communicated between master and slave controllers. To keep this section as a general overview, further details and different approaches for a detailed design of a teleoperation system will be discussed in chapter 3.

Teleoperation is beneficial in situations where a (complex) task must be executed without physical presence of humans. Complex tasks require decision making and dexterous manipulation of objects. The concept of teleoperation utilizes cognitive skills and muscle motor skills of human beings at a remote location. A teleoperation system can serve as extension of the human, but this extension will be limited by the system performance.

In literature, two criteria of bilateral teleoperation systems are separated: transparency and stability (Franken et al., 2009, 2011; Lawrence, 1992; Mersha et al., 2014). Transparency is used as a term to describe how well the complete system enables the user to interact with the slave device environment as if the system was not there. Transparency is technically achieved if slave position and force follow master position and force (Hashtrudi-Zaad and Salcudean, 2001). For effectiveness of the system and intuitiveness for the user, a high level of transparency is usually aimed for. A high level of transparency often comes at the cost of stability. Stability of a system must always be guaranteed for proper operation and safety of the environment.

Stability is a property that must be achieved by control systems in general. Inevitable factors that harm stability in bilateral controllers are for example: digital sampling of a continuous time system, relaxed user grasps, hard contacts in the remote environment, stiff position and force control settings, and time delays and package loss in the communication between master and slave controllers (Lawrence, 1992; Colgate et al., 1993; Franken et al., 2009, 2011; Mersha et al., 2014). A solution to prevent unstable teleoperation systems is to design a controller with the property of passivity.

2.1.1 Passivity

Passive systems (also passive components) have the property that they cannot generate energy. Energy input can only be stored or dissipated by such a system. Consequently, energy output cannot be more than the energy that was stored in the system (Colgate and Schenkel, 1994;

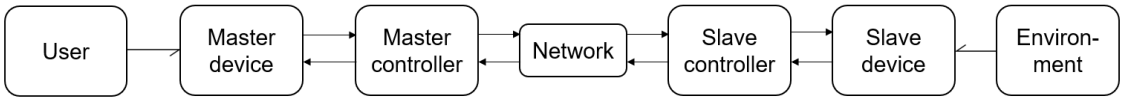


Figure 2.1: Schematic overview of a bilateral teleoperation chain.

Teeffelen, 2018). Additionally, the interaction between passive systems is always stable and any proper combination of passive systems will result in a passive system (Schaft, 1999). In the case of a teleoperation system, the total energy in all controller components ($H_T(t)$) can be considered at each time instance t:

$$H_T(t) = H_M(t) + H_C(t) + H_S(t) \tag{2.1}$$

Here, $H_M(t)$, $H_C(t)$ and $H_S(t)$ represent the energy at the master controller, communication channel and slave controller respectively. If it is assumed that there is no initial energy in the system, the following condition must hold for a passive controller:

$$H_T(t) \geq 0 \tag{2.2}$$

Energy is exchanged between the master device and user as well as between the slave device and the environment. Energy is also exchanged between device and controller at each side, which can be monitored by the controllers such that energy within the controller is known. Equation 2.2 could also be used in the following form:

$$\dot{H}_T(t) \leq P_M(t) + P_S(t) \tag{2.3}$$

Where $P_M(t)$ and $P_S(t)$ are the power flows from device to controller of the master and slave respectively. $\dot{H}_T(t)$ can be considered as the rate of change of the energy balance in the complete digital part of the system. By providing certain rules for the controller output, power flows and energy exchange with the devices can be limited such that the controllers and communication show passive behavior. The notion of passivity is used as a general concept in the design of passive controllers.

2.2 Description of the KUKA LWR

The KUKA LWR4+, schematically depicted in figure 2.2, will be used for practical experiments in this project. The LWR is a seven degrees of freedom (DOF) serial arm with only rotational joints and a workspace of approximately 1.84m^3 (KUKA Laboratories GmbH, 2012). The arm is connected with an internal computer, the KUKA Robot Controller (KRC). The KRC provides internal compensations for joint friction and gravity (KUKA Laboratories GmbH, 2012). The KRC can connect via UDP to an external commanding computer. Commands could for example be a setpoint to internal impedance or position controller, or joint torques provided by an external controller. The control commands by an external computer for actuation of the arm are embedded in a widely used C++ library (Stanford University, 2014). With this library it is possible to communicate with the internal computer and control the LWR arm.

The LWR has restrictions for operation, such that joint torques, joint positions and joint velocities are limited. Maximum values are denoted in table 2.1. If one or more of these limits is violated, the KRC will take appropriate safety measures. The rated maximum payload to be moved by the end effector is 7kg (KUKA Roboter GmbH, 2012).

The LWR arm is powerful and capable of causing material damage and injuries to humans. Safety measures must therefore be taken during operation. The LWR is placed in a closed lab

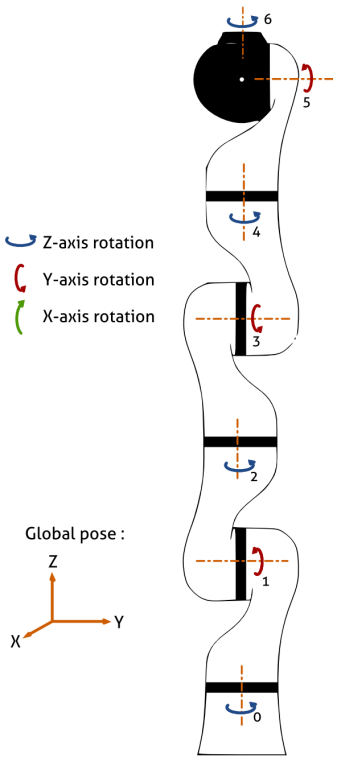


Figure 2.2: Schematic representation of the KUKA LWR. Joints are indicated with positive rotation directions. From LAAS-CNRS/ONERA (2010).

Table 2.1: KUKA LWR maximum ratings per joint. From (KUKA Roboter GmbH, 2012).

Joint axis	**Maximum torque** [N·m]	**Range of motion** [deg]	**Maximum velocity** [deg·s^{-1}]
0	176	±170	110
1	176	±120	110
2	100	±170	128
3	100	±120	128
4	100	±170	204
5	38	±120	184
6	38	±170	184

environment without access to unaware persons. Motor drivers must explicitly be activated by the operator when the LWR is operated. In case of high joint velocities, the KRC will automatically inhibit commanded motion and motors will be deactivated. In all cases there is also a safety switch present to let the operator stop arm motion immediately.

2.3 The Notion of Hard Real-Time

Real-time execution is an important property for computations that are involved with the interaction of physical systems, for example robotics. Several levels of real-time execution can be separated, according to Bezemer et al. (2011). If a catastrophe could happen if a deadline in time is missed, the task is classified as hard real-time. If a result has use after a deadline is passed, it is classified as soft real-time. If there is no deadline to be met, a task is non-real-time. In figure 2.3 several levels of importance are shown for embedded software applications that are connected to hardware. Control loops and safety checks must always be executed within time, to ensure safe operation of the physical system. These are therefore classified hard real-time. Other tasks that are less time demanding can be given a lower priority, such that dead-

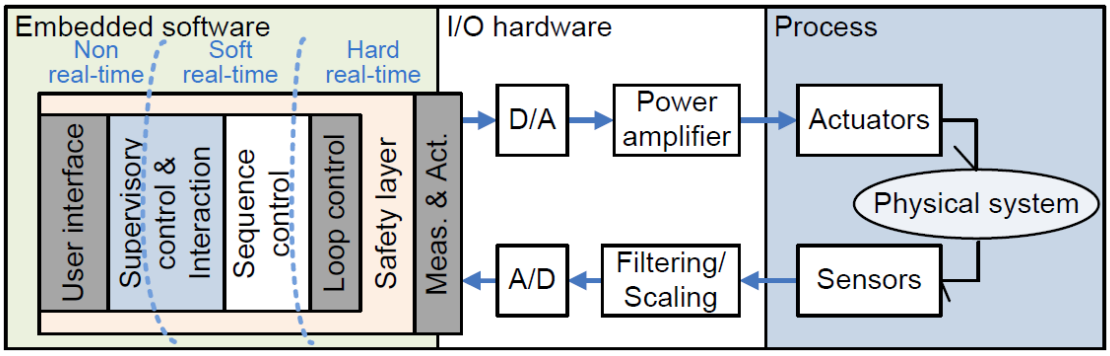


Figure 2.3: Software architecture for embedded systems. From Bezemer et al. (2011).

lines are attempted to be met, but guarantees are not required. A typical example for this situation would be a change in setpoint for the loop controller. Setpoint changes are relatively low-frequent with respect to the loop controller operation. The layering of tasks in figure 2.3 provides a separation that can be used to design embedded systems that interact with hardware.

The degree to which deadlines can be met says something about hard real-time performance. Hard real-time performance is the ability to execute a control loop deterministic at the designed frequency. Deviation of the controller frequency or irregular time durations between adjacent deadlines (jitter) are non-idealities that decrease hard real-time performance and could disturb control actions. If the degree of non-deterministic effects becomes too great, control actions are influenced and performance of the controller decreases. Deviation of one order of magnitude difference from the desired frequency is generally used as bound to ensure performance (van de Ridder, 2018).

2.4 'First-Time Right' Design Approach

At the University of Twente, an approach for designing control software for mechatronic and robotic machines is developed by Broenink et al. (2016). This is a multidisciplinary approach that considers the behavior of control algorithms, software infrastructure, I/O and machine that all influence each other. The total behavior is taken into account in a co-modeling approach to support co-design. This is a model-driven approach, such that all final software is generated from verified models, resulting in qualitative software. First, the approach and tooling are discussed, in section 2.4.1 the CSP software modeling language is discussed and in section 2.5 a middleware is discussed that supports platform-independent code generation from the used models.

An overview of the design approach is given in figure 2.4. The approach consists of four steps, in which different (physical) domains (tracks a.-e.) can be worked on concurrently. The focus is on embedded control software, therefore electronic and mechanic domains are out of scope. The design process starts with software architecture or plant dynamics modeling in step 1 (track b. or d.). Control law design (track c.) requires information from the plant dynamics, therefore this is not a logical starting point. Step 1 and 2 describe the functional system design and testing, in which the modeling of software, control laws and plant dynamics are stepwise refined and connected. Early testing is enabled because software and control law models are connected with the plant model. Step 3 and 4 are focused on implementation design. Design refinements are done to deploy software on the specific embedded system and I/O connections, which could involve code generation details, timing, signal or unit converters, and target specific drivers. Testing is refined from simulation (step 3a), to real-time simulation (step 3b) to the physical plant.

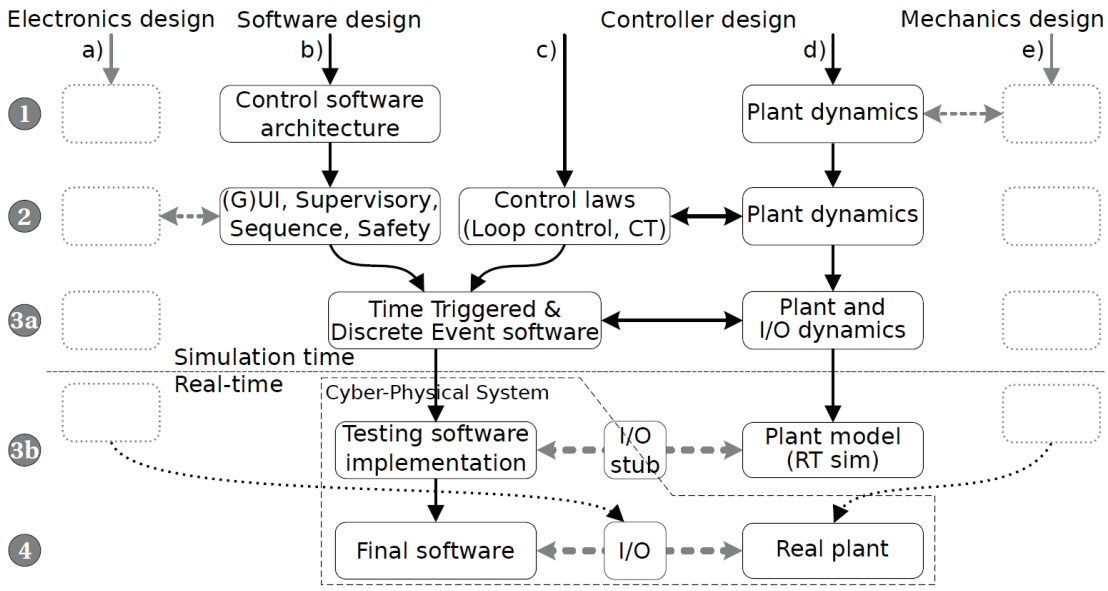


Figure 2.4: Overview of the co-design process. From Broenink et al. (2016).

The discussed design approach is model driven, therefore the use of modeling tools is inevitable. A combination of tools is required. Matlab/Simulink, LabVIEW and 20-sim are tools that have widespread support for physical modeling, but interaction with I/O hardware and software architecture modeling are barely supported. For that reason, the tool TERRA is developed at the University of Twente (Bezemer et al., 2012). TERRA is used to create (abstract) models of communicating sequential processes (CSP). CSP process algebra is a formal language that supports effective software design (Hoare, 1985). In short, CSP is used to describe communicating computational processes that run at the same time. The TERRA tool enables validation of models to conform certain abstract model definitions as well as additional rules. The tool can generate CSPm code that is readable by model checker FDR (Failure Divergence Tool) and can generate C++/LUNA code that is executable on embedded targets. The FDR can check the model for deadlocks, livelocks and other undesired behavior. Altogether, TERRA provides a solution for software architecture modeling. Additional features for hardware I/O, networking and timers are provided by the LUNA middleware and can be modelled with the CSP language in TERRA. In summary, the TERRA tool combined with LUNA middleware gives a solution for the modeling of software architecture and interaction with I/O hardware. This can be combined with code generation from tools that support physical modeling, so that the design process is completely supported.

2.4.1 CSP

Communicating sequential processes, developed by Hoare (1985), are used to formally describe software architectures. CSP models can be specified by textual language or by graphical CSP (gCSP) models (Jovanovic et al., 2004). The latter one is discussed here, as model specification in TERRA is also done in this way. CSP deals with the composition and communication of processes, in which processes are certain functional tasks of the software. Composition is done using certain constructs, such that a construct of processes can be seen as another process. Main constructs are discussed below and examples are visualized in figure 2.5.

- **Sequential:** a sequential construct of two processes results in a predefined execution order. First one process is executed. When the first process is finished, the second process is executed. The sequential construct is finished when the second process is finished.

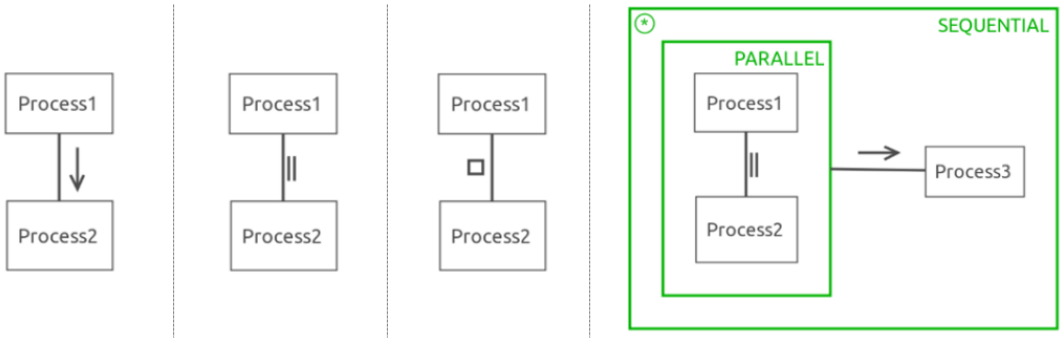


Figure 2.5: Examples of gCSP constructs, from left to right: sequential, parallel, alternative. At the right is a sequential construct in which first Process 1 and 2 are executed in parallel, after which process 3 is executed. Once all processes are executed, the recursion in the top left corner provides a restart of the construct.

- **Parallel:** a parallel construct allows two processes to run concurrently. Depending on the machine scheduler, the processes are mapped to specific cores or executed in sequence in one core. Formally spoken, the execution order of the processes in a parallel construct is fair. If all processes are finished, the parallel construct is finished.
- **Alternative:** an alternative construct of two processes choses one of the two processes for execution. This choice is made based on guards. Guards can be internal or external. The external guard will choose a process based on data that is available or not at one of the input channels (channels are discussed later). The internal guard will choose a process based on the value of a certain variable.
- **Recursion:** recursion is used to infinitely repeat a construct or a process.
- **Grouping:** constructs of processes can be grouped, such that a group can be part of another construct. This is used to explicitly define hierarchy between multiple constructs.
- **Hiding:** Hiding is a way to abstract processes. By making 'submodels', certain constructs of processes can be made unobservable at a higher level.
- **Priority:** the parallel and alternative constructs also have a variant that includes priority. If the process with priority is ready to execute at a certain time instance, the parallel construct will execute this process first. If there are multiple guards unblocked in an alternative construct, the process with priority will be chosen and executed.

Previous constructs only deal with execution order. Communication between processes is done using so-called channels. A channel is suited for a specific datatype. One process can provide data to the channel via a so-called writer and another process can obtain data via a reader. Writers and readers are predefined processes with the functionality to write to or read a variable to or from a channel. A writer and reader are used to communicate data from one functional process to another. This is done using so-called rendezvous communication, which is a specific way of communicating that always blocks the sending or receiving side. If the receiving side is ready to read from the channel, but the sending side has not sent anything, the receiving side is waiting. If the sending side is ready to communicate, but the receiving side is not, the sending side will wait. If communication is done, the constructs at both sides are allowed to continue. Figure 2.6 visualizes the specification of a channel in gCSP language and the rendezvous communication timing.

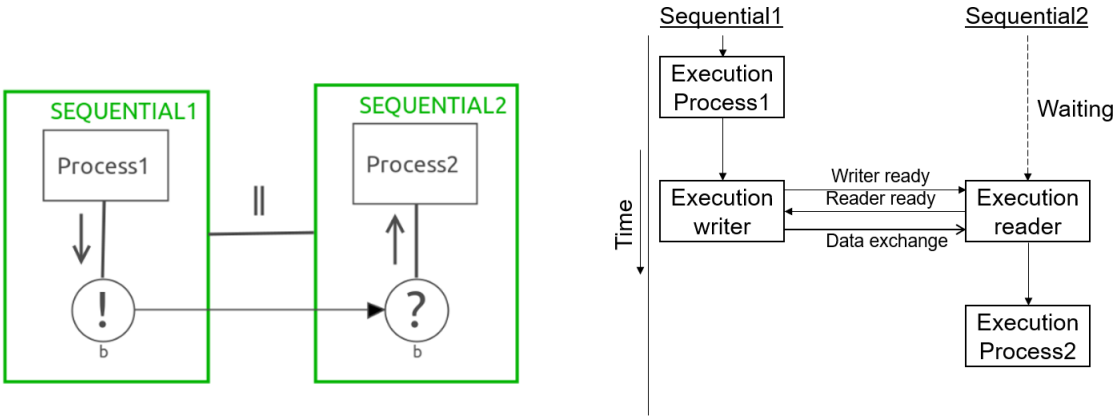


Figure 2.6: Rendezvous communication in gCSP architecture (left) and corresponding timing diagram (right).

With these few simple constructs and communication channels, complex structures can be designed. Designed structures can subsequently be formally checked for deadlock and other undesired behavior by tools that use mathematically proven methods, for example FDR.

2.5 LUNA as Middleware

According to Ellery (2017), middleware can be seen as software 'glue' that is required as interface between several software components. LUNA is a middleware that is designed to execute software components hard real-time, on multiple threads, with support for multiple hardware platforms and with a CSP execution engine (Bezemer et al., 2011; Wilterdink, 2011). LUNA is designed to support fast integration between software architecture design and software implementation (step 1b and 4 in figure 2.4).

LUNA is a C++ library that is built up from core components, high-level components and execution engine components, see figure 2.7. The core components level consists of platform supporting components to provide a generic functional interface for each supported device. This generic interface is used to map high-level components and execution engine components in a platform independent way. Support is provided for different operating systems, as well as different hardware components. The high-level components are platform-independent functionalities that build upon the core components (such as network connectivity or device drivers). The execution engine components determine the application flow. A CSP-based execution engine is provided, such that CSP software architecture models as described in section 2.4.1 can be mapped to embedded devices. The execution engine components are utilizing the core and high-level components to perform the desired task. To facilitate this process, LUNA code can be generated from models specified in TERRA.

Connectivity between basic hardware interfaces and CSP models specified in TERRA is provided. The use of basic hardware interfaces, such as timers, hardware I/O, PWM, etc. can be modeled within the TERRA tool. Hardware interfaces communicate with the CSP models using CSP channels. Timers will unblock the attached channel at specific intervals, by which they permit attached functional processes to run. Other I/O is implemented as a hardware port and will provide information to, or use information from a channel.

Opposed to the default channels in the CSP language, the TERRA and LUNA framework support buffered channels as well. Buffered channels could for example be used for communication between two processes that run on different frequencies. This would only be the case if buffered channels are nonblocking if the channel buffer is empty. Buffered channels in the current TERRA and LUNA version are blocking when the channel buffer is empty. This behavior is

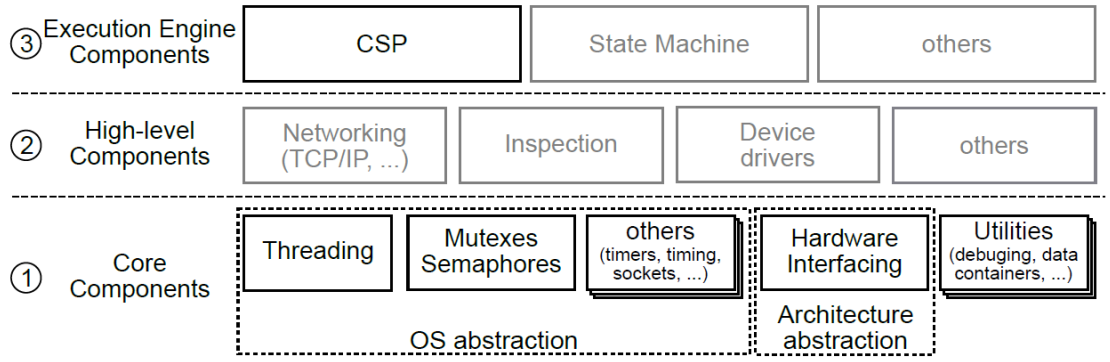


Figure 2.7: LUNA architecture. Grey boxes were not implemented in the initial design. From Bezemer et al. (2011).

not desired and therefore the current buffered channels in TERRA and LUNA cannot be used for rate conversion. Recent work has enabled nonblocking buffered channels (van de Ridder, 2018), but is not available for this project.

More advanced interfaces to LUNA have been created. A network 'bridge' between ROS and LUNA has been developed (Werff, 2016). The Robotic Operating System (ROS) is another middleware that is widely supported. ROS contains communication channels between software 'nodes' that are not real-time. The goal of the ROS-LUNA 'bridge' was to let a part of the system run a loop controller with hard real-time LUNA, while complex tasks are offloaded to a resource rich platform. A practical application for this is a drone that needs to be controlled in mid-air, while performing a certain image processing task.

The connection between robotic hardware and LUNA can also be made. Wijnholt (2017) created an interface between LUNA and the KUKA youBot. This interface was able to actuate and read out all individual robot joints in hard real-time. Different types of control could be used: joint control via PWM, torque, position or velocity were possible. The interface made by Wijnholt (2017) utilizes the youBot interface written in C, made by Spil (2016).

Wijnholt (2017) has designed and realized a DDS-based LUNA network channel that is capable of connecting two real-time LUNA applications running on different devices. The realization is such that the LUNA network channel can be seen as a hardware port in the CSP modeling process. CSP readers and writers can be used to communicate data over the LUNA network channel. Communication can be done via a publish-subscribe or rendezvous connection. OpenDDS (opensource Data Distribution System) has been chosen to provide a communication protocol for network communication. OpenDDS provides quality of service (QoS) settings and can perform soft real-time communication. QoS settings relate to for example: timely delivery, bandwidth, redundancy and persistence. Properties of this communication channel will be further discussed in chapter 3. Other network communication methodologies of LUNA applications have been compared in van de Ridder (2018). Communication based on ZeroMQ was characterized with the best timing properties. Network communication methods discussed by van de Ridder (2018) are used in a static network topology and do not provide QoS settings. As master and slave side in a teleoperation system could be separated by a large distance and network, QoS settings are required and therefore these methods are not further considered. The DDS-based LUNA network channels are used as a basis for network communication in this project.

2.6 ROS as Middleware

The Robotic Operating System (ROS) is a framework that is designed to simplify creation of complex and robust robot control algorithms in a platform-independent way. It is widely used, which means that existing algorithms and hardware interfaces can be used in new projects. ROS creates an environment by providing a set of tools, libraries and conventions. In the ROS environment, programs are run as nodes and different nodes with a specific functionality can be placed. Different nodes form a network on which the nodes communicate on specific topics via publish-subscribe communication. Due to this environment within ROS, timed execution of tasks is not guaranteed, making the framework not capable of providing hard real-time performance guarantees. In practice, ROS topologies work 'fast enough', such that the framework is capable of executing soft real-time tasks. The accessibility of ROS creates overhead which is a trade-off with timed execution performance.

Effort was put into combining ROS and DDS network communication (Woodall, 2018), such that the ROS environment can be extended to multiple platforms via the DDS protocol. A prototype has been designed, but is not mature. Additional effort is not put into this project, as a new version of ROS, ROS2, is being designed. ROS2 aims to support DDS, but is not ready yet. Nevertheless, ROS supports network communication via TCP between nodes running on different devices. The default network communication provided by the ROS framework could therefore form a basis for a teleoperation system.

2.7 Middleware Abstraction

Middleware such as LUNA and ROS are designed to streamline software development. Model-driven design is used to design a high level abstract model, which is generated as a middleware of empty components. Ellery (2017) reviewed this and observed that middleware provides the composition, coordination and communication in software. Computational and/or configurable components were not contained in this structure, and must be written by the programmer. Since these aspects of software design are separated, this allows for simple replacement of computational and configurable components, without the need to adjust the other middleware-dependent aspects. This allows for integration of pre-written computational software in middleware structures. By using a strict interface, the connection between any middleware and a general computational component is made. The interface also allows for different configurations of the general component, as computations are hidden for the middleware. The approach of Ellery (2017) allows for 'off-the-shelf' computational software components that can be used in different projects, which possibly use different middleware. The benefit of this approach is that computational software only has to be written once. In following projects, the existing generic computational software component can be reused, saving development time.

3 Analysis

The main considerations for design choices are described in this chapter. In section 3.1 the overall design is described, next the hardware that will be used is discussed in section 3.2, the controller that will be used for actuation of the slave device and force feedback to the master device is discussed in section 3.3, passivity methods for network communication are discussed in section 3.4, limitations that are imposed on the design are discussed in section 3.5, interesting properties and methods for testing are described in section 3.6, last in section 3.7 research questions and hypotheses required to start the design, implementation and testing are described.

3.1 System Overview

Part of the project goal is to design a teleoperation system. Such a system consists of different subsystems, see figure 3.1. The main goal of the system is to let the user operate a slave device in its environment and to present the user with force feedback about interaction between the slave device and the environment it is placed in.

In order to achieve a correctly functioning system, the master and slave controller must be designed such that they give meaningful outputs to the devices. The master device will be selected based on further analysis and the slave device will be the KUKA LWR. More choices regarding hardware are discussed in section 3.2. As depicted in figure 3.1, arrows with question marks are not yet determined, as they are not enforced by the chosen architecture. In section 3.3.1 the interface between controller and device at master and slave side will be discussed. In section 3.3.2 the network communication structure will be discussed. Additionally, the master and slave controllers must communicate via the LUNA network channel designed by Wijnholt (2017) and guarantee stable behavior. Stability is guaranteed through passivity. Various methods to obtain passivity are discussed in section 3.4.

As stated in the introduction, the goal of designing this teleoperation system is to assess the suitability of the LUNA network channel for this application. Transparency is an important aspect for teleoperation systems, but is a broadly-used term. The level of transparency that can technically be achieved is the degree in which slave position and force follow the master position and force (Hashtrudi-Zaad and Salcudean, 2001). Transparency is therefore measurable from controller signals. Seen from a higher level (the application level), transparency does not characterize the complete performance of a teleoperation system. A user will execute a task in the robot environment via the system. The degree in which this task can be executed, gives a measure for the performance of the designed system. Therefore, it is interesting to test the effect of the LUNA network channel on the performance of a task. Transparency will always be limited by the system as there are time delays and sampling of physical signals in a digital controller. Therefore, it is interesting to find a relative influence of the LUNA network channel on transparency and on a user task performance. For this relative comparison, it is required to have additional implementations without the influence of the LUNA network channel. There-

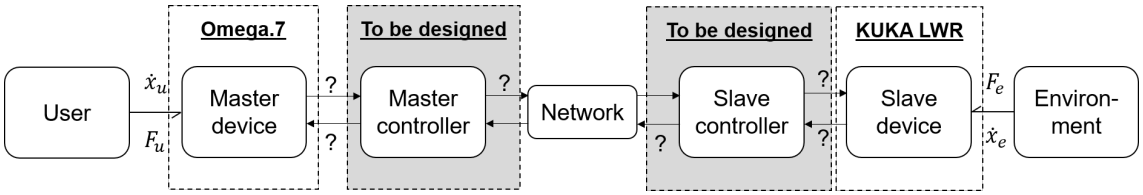


Figure 3.1: Schematic overview of the to be designed teleoperation system.

Figure 3.2: Omega.7 haptic device. From ForceDimension (2018).

fore, the design must be realized with at least two implementations to obtain relative measurement results. The influence of the LUNA network channel can subsequently be characterized by relative comparison of performance of different implementations.

3.2 Hardware Choice

3.2.1 Computing Platforms

Various computing platforms can be chosen to run control algorithms on. Important is that computing power is good enough, such that all controller calculations can be done within the corresponding real-time deadlines. Since a connection with the KUKA LWR and LUNA network channels must be established, it is required for the slave computing platform to have two network interfaces. The KUKA LWR is situated in a lab environment, therefore there are no additional requirements such as size or mobility for computing platforms. Two desktop PC's for master and slave computation platforms will be used.

3.2.2 Master-Side Device

Haptic devices that are available at the RaM group are the Omega.6 and Omega.7 devices (ForceDimension, 2018). The Omega.7 is depicted in figure 3.2. Both Omega.6 and Omega.7 variants cover the natural range of motion of the human hand (a sphere with a diameter 15cm), provide gravity compensation and register position and rotation (6-DOF). Force feedback is given in translational Cartesian directions. Additionally, the Omega.7 is capable of registering position and providing force feedback for 1-DOF grasping motion of a human finger. Force feedback in rotational directions cannot be performed. One of the Omega devices will be used in this project.

3.2.3 Slave-Side Device

Throughout this document, it was already stated that the KUKA LWR is used as slave device. This project initially set out with the KUKA youBot as a slave device, however a preliminary investigation shows that the KUKA youBot is not a transparent device, as joint friction requires relatively large controller forces for commanded motion. The high controller forces are too great to use for user feedback. More importantly, the dominant part in the feedback forces would be related to friction. This results in very inaccurate feedback from interaction between slave device and environment. The investigation regarding the KUKA youBot can be found in appendix A. As a comparison between KUKA youBot and KUKA LWR, the same measurements have been performed on the KUKA LWR, which can be found in appendix B. Compared to the KUKA youBot, the KUKA LWR shows more transparent properties and therefore the KUKA LWR is chosen for this project.

3.3 Transparent Control

The controllers at master and slave side must be able to control the slave device and provide the master device with feedback about interaction between slave device and the environment.

Design choices with respect to different device types and communication structures will be made. Impedance control will be used to perform control actions, based on the used device types and communication structure. All concepts are discussed in an one-dimensional case. In the actual system the KUKA LWR must be controlled, which is a serial robotic manipulator. Appropriate measures will be discussed.

3.3.1 Device Types

Devices can be classified as impedance or admittance type (Hashtrudi-Zaad and Salcudean, 2001; Ott et al., 2010; Lammers, 2017; Teeffelen, 2018). Impedance type devices have a force input and produce a velocity or position output as interface to a controller. As interface to the environment, they have force input and velocity output as well. Generally, there can be a stable interaction with stiff environments, but motion in free space can be inaccurate due to presence of friction and other (non-linear) dynamics such as play, dead zones and saturation. Typical properties of impedance type devices are low inertia, low friction and good backdrivability. Backdrivability is the ability to see a change in velocity or position due to an applied force. Admittance type devices have velocity or position input and force output as interface to the controller. Generally, admittance type of devices provide high position and velocity accuracy in free space but can result in an unstable or unsafe interaction with other objects. Admittance type devices are usually actuated with a high transmission ratio between actuator and device. These two different device types provide advantages and disadvantages, of which a compromise must be found.

The Omega devices measure the position of the device determined by a human user and provide force feedback to the user. The interface of these devices to the controller is thus impedance type. Also the mechanical properties such as low friction and low inertia are fulfilled by this device. The KUKA LWR interface gives the option to perform torque (effort-like) control and measure position, but also vice versa is possible. Therefore, the interface to the controller does not have a preferred type. External mechanical disturbance forces such as friction and gravity are internally compensated, making this device highly backdrivable. The mass of the device is relatively low compared to other serial robotic arms, but should not be underestimated. An additional mass compensation controller could be used to compensate for effects of mass, however this is not done in this project. As will be discussed in section 3.4.4, the Two-Layered approach requires impedance type devices. The KUKA LWR possesses preferred properties of an impedance type device and will therefore be actuated with joint torque commands. Position measurements can be used as input for the controller. This means both master and slave devices will be actuated as impedance-type devices and has as consequence that the controller must output effort-like control. For the Omega.7 Device, this means that only translational motion can be used.

3.3.2 Communication Structures

In bilateral controllers where master and slave devices must be controlled via a transparency layer, values related to power variables are transmitted between transparency layers at master and slave controller. Power variables are one of two physical quantities that can be multiplied to obtain power, e.g. force and velocity. One power variable is usually transmitted by each master and slave transparency layer. In this way, the communication between controllers can be seen as a power port. Communication of multiple power variables at each controller side would also be possible, but requires more advanced device controllers, based on environment modeling, by which the complexity of the controller is increased (Hashtrudi-Zaad and Salcudean, 2001). To keep the controller design intuitive and straightforward, communication methods with multiple power variables are not further considered.

Two different simple structures can be used in the communication between master and slave transparency controller. First, the position, related to the velocity of the master device, is send to the slave controller. The slave controller can subsequently send the slave position, resulting in position-position (PP) communication, or the slave controller can send the slave controller output force, resulting in position-force (PF) communication. The input of an impedance-type device is force. Both structures deal in a different way with the generation of these output forces by both master and slave controller.

If PP communication is chosen, both master and slave require a separate controller to convert a position difference to output force. If PF communication is chosen, the master transfers a position setpoint to the slave, the slave controller calculates the required force for the slave device and subsequently sends the same force output back to the master. The master controller will use this force directly to actuate the master device. With PF communication the energy in the communication channel is easily derived from force and position. This is caused by the choice of sending values related to two power conjugated variables between master and slave. The power flow is used by a single controller, such that energy can intuitively be shaped and monitored. Furthermore, in Franken et al. (2012) it was concluded that PP communication structures are ill-suited to deal with time delays from a transparency point of view. Based on the above arguments and the common use of the PF structure within the RaM group at the University of Twente, it is chosen to use PF communication.

3.3.3 Impedance Control

Both master and slave devices were determined to be used as impedance type, which correspond with position or velocity measurements and force outputs from the controller point of view. The slave device position must converge to the master device position by controller actions. An intuitive controller concept is a mechanical spring attached with one side to the master device and with the other side to the slave device. Such an impedance controller can perform stable operation during free space motion and during interaction with objects. This controller could be physically interpreted and is visualized in figure 3.3. The following relation holds:

$$F_s = K_v \cdot (x_m - x_s) \tag{3.1}$$

Here, F_s is the force output of the slave controller to the slave device, K_v the stiffness of the virtual spring, x_m the position of the master device and x_s the position of the slave device. Force feedback, corresponding to the other side of the spring can be given to the master device by: $F_m = -F_s$. In order to change position of the master device, the user is required to exert a force (F_u) on the master device such that the position changes.

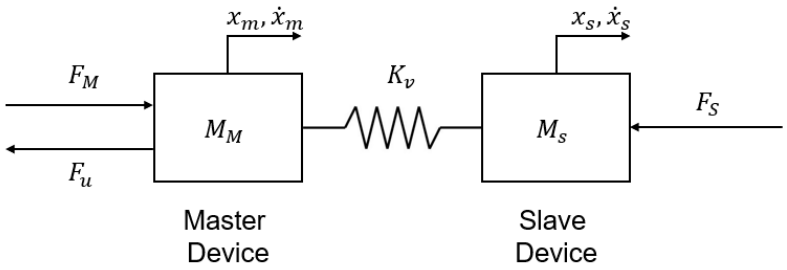


Figure 3.3: Schematic 1-DOF representation of two devices connected by a spring as impedance controller.

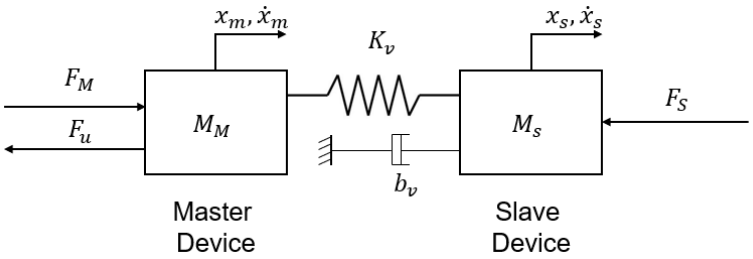


Figure 3.4: Schematic 1-DOF representation of two devices connected by a combined spring-damper controller.

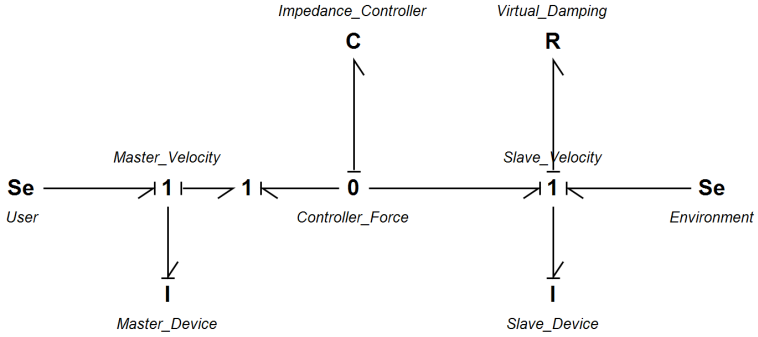


Figure 3.5: Bond graph notation of two devices connected by an impedance controller with damping of the slave device.

If a spring alone is used in the motion controller, undamped oscillatory behavior could be achieved in presence of inputs at the resonance frequencies corresponding to device masses and controller spring stiffness. Therefore, the control action must be augmented with a virtual damper. This is depicted in figures 3.4 and 3.5. For the damping the relation in equation 3.2 holds. From the bond graph notation, it becomes clear that the structure is almost symmetric: control action is used for the slave device via a virtual spring-damper system and the master device receives feedback forces. Additional damping could be placed on the master device, to reduce feedback force oscillations. It is chosen to perform damping on the slave device alone, because the master device interacts with a human, who will use muscles to damp the motion of the master device.

$$F_{D_v} = -D_v \cdot \dot{x}_s \tag{3.2}$$

In robotic systems, usually position measurements are available. This is also the case for the KUKA LWR arm. For the virtual damping, the slave velocity is required. Velocity can be derived by means of numerical differentiation, observers, state variable filters or damping injection. Numerical differentiation is very easy to implement but high-frequent noise is usually a problem. Observers are a suitable method, but require a model of the device which is simple in the depicted one-dimensional case, but difficulty increases for a serial robotic arm. State variable filters are only working for low frequencies, but are a viable option as the robot will also not reach high velocities. Damping injection works by interconnecting a digital mass and spring in series with the slave device, see figure 3.6. if $K_c >> K_v$ and $M_c << M_S$, the behavior of the slave device is not much affected by the virtual mass and spring. As the virtual mass will move with the slave device, the slave behavior is known digitally, which can be used obtain the slave device velocity for the damping controller. As additional elements are connected to the controller, the energy flows in the controller become more involved. Therefore, damping injection

seems a complicated solution to achieve the same functionality as state variable filters. This makes state variable filters the most feasible solution to use position measurements for velocity estimation.

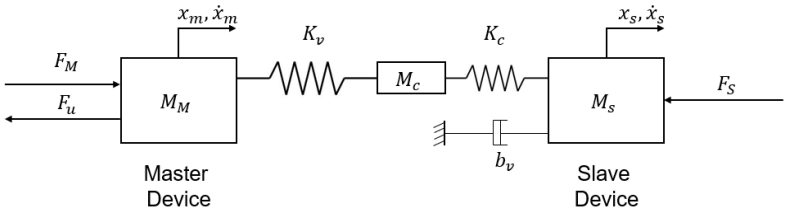


Figure 3.6: Schematic 1-DOF representation of two devices connected by a combined spring-damper controller. A virtual small mass and stiff spring are connected to the slave device for velocity measurements.

3.3.4 Control of a Serial Robotic Manipulator

In previous sections, a one-dimensional situation was considered. If a serial robotic arm is controlled, the situation becomes multidimensional. The end effector of the serial arm in three dimensional Cartesian space must be controlled, whereas it is only possible to actuate the joints of the arm. A part of Screw Theory (Stramigioli and Bruyninckx, 2001) offers a solution for the mapping between Cartesian space and joint space. This is described in appendix C.

It is requited to use an inertial reference frame to define and relate setpoint positions from the master device, the measured position of the slave device end effector and the wrench exerted by the control action. Mapping from measured joint position and velocities to end effector position and velocity can be done respectively using the exponential map (defined in equation C.8) and power-continuous geometric Jacobian (defined in equation C.4). Mapping from end effector wrench to joint torques can also be done using the geometric Jacobian (defined in equation C.10).

A multidimensional impedance controller can be designed in different ways. An single degree of freedom controller for each direction (3 translation, 3 rotation) can be designed, such that in total six impedance controllers generate the output wrench of the slave device end effector in reference frame. This was also performed in Teeffelen (2018). Another approach is to use a single impedance controller which compares the homogeneous matrices corresponding to slave device end effector and setpoint to output a wrench (Stramigioli, 1998). The latter is preferred, as it is a more general and complete way of performing multidimensional impedance control in three dimensional Cartesian space. After the calculation of a wrench by the impedance controller, the output wrench can be mapped to joint torques using the transpose of the geometric Jacobian (described by equation C.10).

If damping is added next to the impedance controller in Cartesian space, the behavior applied to the end effector will be damped. However to achieve this behavior, individual joints might contain redundant energy, such that a small difference in end effector behavior results in more motion of individual joints. This can also be described as motion in the null space of the robot. Damping in Cartesian space will be replaced by damping in joint space to overcome this behavior.

3.3.5 Additional Design Space Exploration

During the analysis of possible controller structures, a variable impedance controller and safety-aware impedance controller were found. These ideas do not contribute to the project goal, although they are interesting extensions to the current design choices. These extensions are mentioned here, but not used for the design.

A variation on impedance control can be made by using a variable spring stiffness ($K_v(\alpha)$ instead of K_v) in the controller (Walker et al., 2010; Chen et al., 2016; Teeffelen, 2018). A variable (α) can change the stiffness based on measurements. An estimate of co-contraction levels in the user's arm, based on electromyography measurements, is an example of a variable that can influence the controller (Teeffelen, 2018). An idea generated in this project to utilize this concept is to change the controller stiffness once communication delay or data loss is detected, which could result in less influence of the communication architecture on system transparency. Variable stiffness is an interesting concept, but will not be used in this work, as it does not directly contribute to the project goal.

Safety metrics can be incorporated within an impedance controller. Tadele et al. (2014) use passivity in a way to design an impedance controller suitable for human interactions. The controller spring stiffness and damping are modulated, based on the power flow from controller to device, potential energy in controller spring and kinetic energy in device mass. This structure provides safety based on quantitative safety norms, to safely allow humans in the workspace of the device. Safety is important, but in this work it is not specifically required to perform interaction tasks with humans. Therefore, safety metrics are not incorporated in the controller design.

3.4 Passive Control

Passivity will be used to guarantee stability of master and slave controllers and communication between them. Passivity is a property that must be explicitly designed. Motivation to use passivity for the controller and network communication can be found in subsections 3.4.1 and 3.4.2 respectively, as it is shown that passivity can be lost in controllers and communication if passivity measures are not taken. Different approaches from literature for passive controller design are subsequently discussed in subsection 3.4.3. Approaches are compared and a design choice is made in section 3.4.4.

3.4.1 Loss of Passivity in the Controllers

Loss of passive behavior must be prevented, but is implicitly present due to implementation on a digital system. Due to sampling of input signals and application of zero order hold to output signals, the signals in digital domain do not exactly correspond with the physical quantities. A continuous operation was described in section 3.3.3, but in reality the controller will have the behavior of a sampled spring over time. Compression and decompression of the spring will occur with steps, not corresponding to a continuous-time system and will therefore not have the correct energetic behavior. The digital spring will generate 'virtual' energy, violating the passivity of the system. The damping in the controller is also sampled. Damping usually dissipates energy in the system, however if the continuous time velocity changes sign, the applied damping will generate 'virtual' energy during the remainder of a single sample interval. The length of a sample interval and values of spring and damper constants also influence the active behavior, as higher values will result in a higher amount of generated 'virtual' energy. By application of high control frequencies and low frequent motions, these effects will become smaller. However, the effects cannot be neglected. Non-passive effects of the transparency layer must be negated by additional measures to guarantee passivity.

3.4.2 Loss of Passivity due to Communication

A simplified system, based on design choices made in 3.3.1 and 3.3.2 and without passivity layer, is described in this section to demonstrate loss of passivity in the communication caused by time delays introduced by a network. A single degree of freedom system is considered, see figure 3.7. Both master and slave devices output positions to the controllers and the controllers output efforts to the devices (impedance-type devices). The master controller sends the posi-

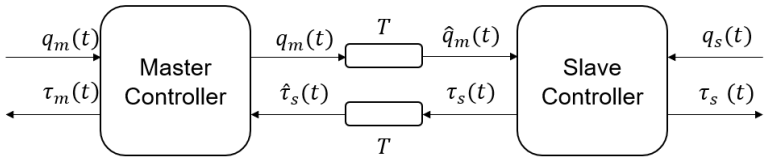


Figure 3.7: 1-DOF master and slave controller connected via a network that introduces time delays.

tion of the master device to the slave controller, and the slave controller sends the output effort from slave controller to the master controller.

Properties of passivity have been described in section 2.1. The energy contained in a passive component must always be nonnegative and the sum of power flows into the passive component must always be greater or equal to the rate of change of energy within the component.

If network delay is not present, then the power port of the master controller towards the network with variables $q_m(t)$ and $\hat{\tau}_s(t)$ is directly coupled to the power port of the slave controller towards the network with variables $\hat{q}_m(t)$ and $\tau_s(t)$. If a network delay of time T is introduced between master and slave, this situation changes as power variables are related to a power variable of the other controller from a previous moment in time. Instead of power ports with a direct connection, the energy flows do not match in time. The network delay changes equation 3.3 to equation 3.4.

$$\begin{aligned} \hat{q}_m(t) &= q_m(t) \\ \hat{\tau}_s(t) &= \tau_s(t) \end{aligned} \tag{3.3}$$

$$\begin{aligned} \hat{q}_m(t) &= q_m(t-T) \\ \hat{\tau}_s(t) &= \tau_s(t-T) \end{aligned} \tag{3.4}$$

If the interfaces of master and slave device towards the network are considered, the following power ports are appropriate:

$$\begin{aligned} P_{m,com}(t) &= \dot{q}_m(t)\hat{\tau}_s(t) \\ P_{s,com}(t) &= \dot{\hat{q}}_m(t)\tau_s(t) \end{aligned} \tag{3.5}$$

If equation 3.3 and a situation without delay are considered, both power flows are equal in magnitude. The flow going in the slave controller is exactly negative with respect to the flow going in the master controller. If equation 3.4 and a situation with delay are considered, a mismatch of power flows is created due to the introduction of delay. The network introduces time-variant behavior, by which an energetic mismatch occurs. At some time instances, the network provides dissipation of energy and at other time instances, the network provides generation of energy, depending on the values of the power variables over time. This means passivity is violated as a result of added time delay. If passivity is violated, stability of the system cannot be guaranteed. This makes time delays an important factor to take into account for the design of stable teleoperation systems.

3.4.3 Approaches for Passive System Design

In literature, several different approaches for design of passive systems exist. The methodology of several approaches, separated by Franken et al. (2011), will be briefly discussed here. If more details are desired to be read, the reader is referred to the corresponding literature. The most suitable method for this project is chosen according to the criteria: design intuitiveness, system transparency and freedom for choosing robot control techniques. These criteria are chosen

since the project will be done in limited time, the system must have transparency and follow-up projects will continue with this work if the project is successful. Methods that conserve passivity are:

- **Scattering/Wave-Variable-Based Approaches**
  The scattering and wave variable approach are developed by Anderson and Spong (1989) and Niemeyer and Slotine (2004). These approaches separate the communication channel, master and slave sides and aim to create each element in a passive way. The communication channel is made as a passive element by applying a coding scheme to force and velocity that are being communicated. Wave variables contain information that corresponds to the energy exchange between controllers and devices and the desired behavior at the other side of the channel. Although stability is guaranteed, problems for transparency arise due to time delays, delay variations in the communication channel and the decoding process. This results in delayed force feedback with respect to the commanded motion by the user.

- **Time-Domain Passivity Control**
  The Time Domain Passivity Control methodology utilizes a passivity observer (PO) and passivity controller (PC) (Hannaford and Ryu, 2002). The exchange of energy at master and slave side is monitored by a PO and must be simultaneously known. This means network delays are not taken into account. If the controller output tends to be active, the PC provides that passivity is maintained. Artigas et al. (2008) extended this idea with a PO and PC at each side of a communication channel. Based on a fixed communication delay and transmitted and received force and velocity by the PO's, the energy in the communication channel is estimated. Passivity at each side of the channel is maintained by the PC's. Several more extensions (Ryu and Preusche, 2007; Ryu et al., 2010) have been done to compensate for time-varying delays and optimize the controller algorithms. This methodology is characterized by limited transparency due to time delay.

- **Energy Bounding Algorithm**
  The energy bounding algorithm guarantees passivity based on the assumption that there is always friction present in the master and slave devices (Seo et al., 2008). The active behavior of the network distributed controller is limited by the amount of energy that can be dissipated by friction over a small time interval. This algorithm is based on a viscous friction model. A risk of this approach is that deviations between models and physical systems can result in an unstable system, however conservative friction models can be used to prevent this. A limitation of this approach is that a stationary system (with zero velocity) results in a constant force by the controller, which cannot adjust. The transparency of the system is severely limited, as this approach is based on slave device output force for the force feedback to the user instead of measured interaction force between slave device and environment.

- **Passive Set-Position Modulation**
  Passive Set-Position Modulation is based on a controller which consists of a spring and damper that are present between the master and slave plant position (Lee and Huang, 2010; Franken et al., 2011). The energy that will be dissipated by the damper in the controller is stored in an energy tank. Due to the discrete time controller, the setpoint position will 'jump' in time. The result is a controller spring potential that 'jumps' in time as well. The algorithm provides that this spring potential energy 'jump' is limited to the available energy in the tank. To guarantee passivity of the system, the parameters of the controller must be related to the physical parameters. For example, the controller damper must be at least twice as large as the viscous friction of the device. A disadvantage of this method is that the damper in the controller is always present and will limit

the servo performance and transparency. It is therefore hard to separate controller design and maintaining passivity.

- **Two-Layered Framework**
  Franken et al. (2009, 2011) describe a controller that separates two design goals of the controller. First there is a transparency layer that controls the slave device and provides the master device with force feedback of the interaction between slave and environment. Second, a passivity layer is used to maintain passivity of the complete system. The passivity layer monitors energy in- and outputs of the controller. Control actions will first be calculated by the transparency layer, subsequently the passivity layer will limit the control action according to the energy within the controller. The framework allows any bilateral controller type, assumed that effort variables (e.g. forces and torques) are output of the controller to the master and slave devices. The passivity layer and transparency layer are independent. The passivity layer controls an energy balance at the master and slave side and controls how much energy is in the communication channel. A drawback of this framework is that the system can become temporarily active. This happens in continuous time in a single sample interval and is inherent to a digital controller. Once the active behavior is measured, the passivity layer limits controller output until passivity is restored.

3.4.4 Passive Controller Choice

Several methods for implementation of passive controllers over a network have been discussed. The methods are compared with aid of table 3.1. It is found that all approaches except the Two-Layered framework score negative at some aspect. Therefore the most logical choice is to use the Two-Layered framework, designed by Franken et al. (2011). Another argument for this approach is that it is developed and widely used at the RaM group at the University of Twente. The discussion above might have been in vain, as there was a bias in the direction of the Two-Layered framework on forehand. Nevertheless, the proposed alternatives gave insight into different approaches for passive controller design.

The controller design freedom is not completely free with the Two-Layered framework, since it requires the control technique in the transparency layer to compute efforts to be applied to the devices. Measured data due to interaction with the environment (e.g. position, velocity or forces) can be used as input for the computation.

Table 3.1: Comparison of passive controller methods.

	Intuitiveness	System transparency	Controller design freedom
Scattering/Wave-Variable-Based	-	+	-
Time Domain Passivity Control	+	+	-
Energy Bounding Algorithm	-	-	-
Passive Set-Position Modulation	+	+	-
Two-Layered Framework	+	++	+

3.5 Limitations

Factors that will limit the overall performance of the system are described in this section. Effects that might occur due to these limitations are also discussed. First limitations corresponding to the available hardware will be discussed, where the combination of passivity layers and hardware is most important. Second, the non-idealities and corresponding consequences of the LUNA network channels and communication delay will be discussed.

3.5.1 Hardware Constraints

By combining passivity layers, impedance control, the Omega.7 and a serial robotic arm, several contradictory requirements arise. It is desired to control the serial robotic arm without constraints regarding maximum output power, degrees of freedom and workspace. This is desired because it allows a broader range of tasks to be executed by the teleoperation system in a remote environment. The first constraint is the limitation of the physical power that the serial arm can deliver to the environment. Therefore maximum output power is limited by the capabilities of the KUKA LWR and Omega.7.

Second, the energy flow from the connected devices is measured in the passivity layers. Physical quantities to derive the energy flow must be measurable at each sample interval. This is not a problem for the KUKA LWR interface, as output torques and input positions are known and energy flow can be derived (this will be discussed in chapter 4). The Omega.7 gives rise for a problem, as force feedback cannot be given in rotational directions. This means that the power port used for energy monitoring by the passivity layer is not present. Several solutions exist. As the Omega.7 device is impedance type, it gives a position to the controller and the controller provides a force. This feedback force is determined by the interaction of the KUKA LWR with its environment. All power variables for energy monitoring are known, so a value for energy exchange can be calculated. However, this energy exchange will never take place, because the Omega.7 cannot output the calculated rotational force feedback. This means that the controller is able to calculate a value for energy exchange, however transparency of the system in rotational directions is not present. If the rotational directions would be used as an input, they would be straightforward setpoint generators for the KUKA LWR. This causes additional energy consumption of the KUKA LWR which is not put into the controller by the rotations of the Omega.7 and thus an energy shortage would occur. Another option is to disregard rotational DOF's in the teleoperation system. In this project, the influence of the LUNA network channel must be found on system transparency and performance of a user doing a task via the teleoperation system. If rotational DOF's without force feedback would be used, a system with mixed properties would be created. The mixed properties being transparent translational DOF's and rotational setpoint generators. To increase repeatability of this work, there must be transparency in all DOF's. Therefore, it is chosen to only use translational DOF's. The orientation of the slave robot will be controlled to a fixed initial orientation. This choice limits the tasks that can be performed by the system, however with 3 translational DOF's it is possible to perform tasks so that experiments can be executed.

The third system limiting factor is the passivity layer, which works with a $1:1$ mapping between energy, force and position at master and slave side. The output position of the Omega.7 is used as a setpoint for the KUKA LWR. The reach of the Omega.7 is the natural reach of the human hand, whereas the reach of the KUKA LWR is approximately 1.84m^3 (section 3.2 and 2.2 respectively). The reach of both devices must be matched. A choice can be made to limit the reach of the KUKA LWR or to use the full workspace. If it is chosen to control the full reach of the KUKA LWR, scrolling methods can be applied to map the limited reach of the Omega.7 to a shifted position with respect to the previous position, such that the full range of the KUKA LWR is covered. If the reach of the KUKA LWR is allowed to be limited, the reach of the Omega.7 can be mapped to a certain limited reach of the KUKA LWR in which tasks can be executed.

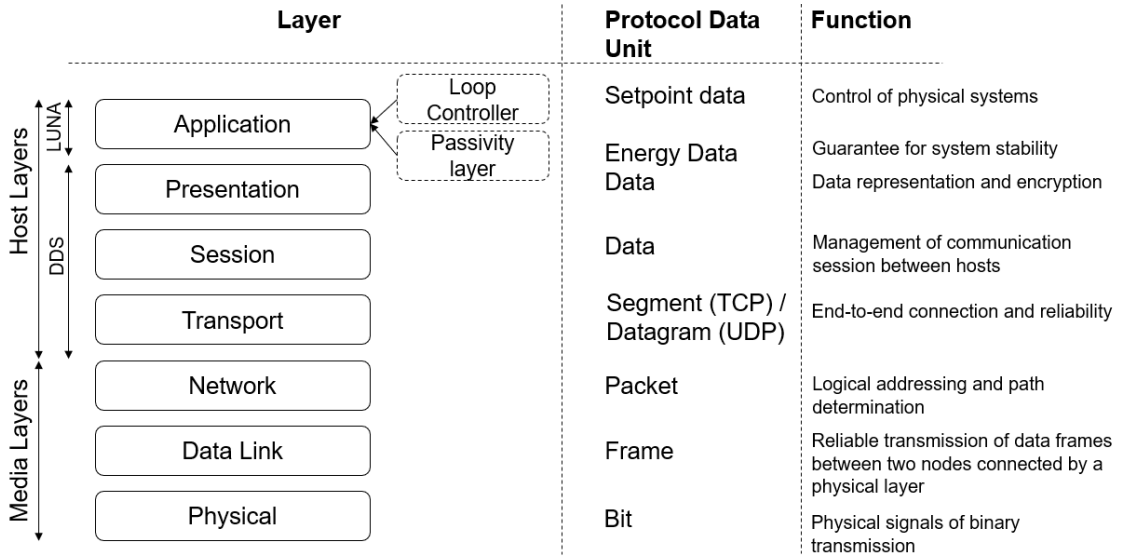


Figure 3.8: OSI model adapted for teleoperation systems. Based on Hewitt (2005).

It is also possible to apply position and energy scaling between the Omega.7 and the master controller. This enables a virtually enlarged reach of the master device to be matched with the KUKA LWR. Scrolling or scaling techniques make the system less generic and are not required for task execution, therefore it is chosen to do a straightforward design with a $1:1$ mapping between Omega.7 and KUKA LWR.

3.5.2 Communication Network Characteristics

Network communication via Internet between two devices can be decomposed using the OSI model, see figure 3.8. Layers can be separated in host layers and media layers. Host layers process higher-level data communication within a single device. The DDS-bases LUNA network channel described by Wijnholt (2017) provides this part of the communication. This part of the communication is seen as the interface between controller calculations and network hardware on the computation device, also network-accessing middleware. Media layers are inter-device connection topologies which provide the routing and actual transmission between devices. These layers are seen as the transport between network hardware of master and slave computation device. Each layer is not ideal, introducing delay into the communication. To test a teleoperation system based on LUNA network channels, the properties of host layers are of importance, as these are part of the LUNA network channel properties. Properties of media layers must be known, as these are unavoidable delays in network communication. Both network-accessing middleware and network transport (host and media layers) must be separated and are further analyzed.

Network Transport

Data transmission is considered to be via Internet and is characterized by varying time delays. This may have different causes, for example the distance between devices, differences in required routing, other network traffic and used transmission protocol. Transmission is performed using UDP or TCP protocols. The UDP protocol does not provide any reliability, such that packet loss, packet duplication and differences in order of packets may occur. On the other hand, the TCP protocol provides reliability with respect to the previously mentioned aspects, at the cost of more overhead. Therefore, more time jitter is introduced. The chosen method to guarantee passivity will fail in case of packet duplication or changes in the received order of packages, as energy monitoring would be disturbed. This could be prevented by sending

Table 3.2: Single-way communication delays of transport (in media layers) for several network topologies.

Connection	Result from	Mean Delay	Std. dev. Delay	Package Loss
The Netherlands - Italy	Franken (2011)	40ms	1ms	0.04%
The Netherlands - Australia	Franken (2011)	351ms	5ms	27%
Single switch	Wijnholt (2017)	0.3ms	0.003ms	0%
Single switch	This project	0.5ms	0.09ms	0%

timestamps with each packet as an additional measure. For convenience, communication via TCP protocol is used in this work.

Data from previous work is gathered to get an indication of the introduced transport delays. Franken (2011) measured the mean and standard deviation of delay, and packet loss of one-way connections between The Netherlands and Italy, and The Netherlands and Australia. Wijnholt (2017) measured similar delays properties of a single network switch. Measurements are done using ping requests, which measure the transmission time in the three media layers. Results are shown in Table 3.2. The delay introduced by transport is related to delays introduced by the LUNA network channels in section 3.5.2.

LUNA Network Channel

LUNA applications on a single device are characterized by hard real-time execution. The LUNA network channel can be seen as network-accessing middleware to connect LUNA applications to a network. The LUNA network channel cannot guarantee hard real-time execution, opposed to other parts of a LUNA application. Several properties were observed by Wijnholt (2017) and are analyzed here for consequences on teleoperation. The tests performed in Wijnholt (2017) are performed using two RaMstixes (RaM, 2017).

- The communication channel only supports the transport of scalars with the datatype 'double'.
- Communication is possible using publish-subscribe and rendezvous protocols. The maximum transmission frequency between two devices is 250Hz for a publish subscribe connection and 230Hz for rendezvous connection with the used hardware. Rendezvous communication would require master and slave side to wait with execution of processes until messages are received at the other side, which is not necessary and could even disturb timing of loop controllers. Network communication frequency is an interesting property to investigate, as it will also affect minimum network transport delays. In previous work, 1kHz is used as frequency for the master controller, slave controller and network communication (Franken et al., 2010, 2011; Teeffelen, 2018). This frequency is not achievable with the LUNA network channel, therefore it is interesting to investigate effects of lower transmission frequencies. Controller frequencies could be lowered to the network communication frequency, but this could harm performance of the passivity layer as energy exchanged between device and controller in one sample period is increased. The controller frequencies are therefore maintained at 1kHz. Since controller and network communication frequencies are different, a proper information exchange facility must be made between controller and communication software.
- For network transport, TCP and UDP protocols can be used. Tests by Wijnholt (2017) show that there is a slight difference in timing at the sending side between using TCP or UDP. Received channel information via TCP is bounded in time, whereas UDP transmission introduced packet loss and therefore varying receive time. Therefore, the use of TCP was recommended over UDP.

- Timing for publish-subscribe communication is analyzed by Wijnholt (2017), to chart latencies and the causes for latency. Latencies at the publisher were mainly caused by queuing and writing to DDS. Messages are asynchronously added to a queue and dequeued at 400Hz, generating a variable delay between 0 and 2.5ms. DDS is responsible for writing to the network. It is found that the writing operation contains a system call, which is the cause for additional time delay. The total average time delay at the publisher side is 2.5ms. At the receiving side, latency is introduced between the arrival of a message and the unblocking of a reader. Latency of communication via the TCP protocol is found to be 9.2ms. All mentioned latencies are dependent on the used computational devices.
- The performance of the network channel was tested on a network with additional traffic (0Mbps versus 50Mbps, where 67Mbps was the maximum achievable bandwidth). It was observed that publish-subscribe communication suffered from more jitter due to more network traffic. On average the timing was correct. The standard deviation of execution time for writing to the network at the publisher was increased from 0.17ms to 0.32ms. The standard deviation of execution time for the subscriber was increased from 0.99ms to 1.63ms. No high outliers were observed, which together with the jitter indicate that the network channel was able to communicate in a reliable way despite presence of other traffic. Comparable results were obtained for rendezvous communication.

Effects on Teleoperation

The properties of Internet communication and LUNA network channels limit the performance of a teleoperation system. The limitations can result in data (energy) loss in rate conversion between controller and communication frequencies, and time delays between master and slave controllers. Time delays are considered to be the most significant limitation to the operation of a teleoperation system. Based on the previous two subsections, an indication of average total delay between master and slave controller can be estimated. The total delay is the sum of delays introduced by publishing of data, network transport, and receiving of data. The result is based on the previous subsections and denoted in table 3.3. The result is that most delay is caused by operations of the LUNA network channel, thus in the network-accessing middleware. Based on this result, it is assumed that delay introduced by network transport via a single switch can be neglected with respect to the delay introduced by LUNA network channels.

Table 3.3: Communication delays separated per communication action.

Action	Publising data (LUNA network channel)	Transport (single switch)	Receiving data (LUNA network channel)	Complete transfer
Introduced delay (ms)	2.5	0.5	9.2	12.2

Time delays affect teleoperation systems in a number of ways. Due to time delays, controlled systems can become less-damped or unstable. Instability is prevented by the passivity layer, but the system may still show less-damped behavior. Oscillations will be mainly present in the controller output force to master and slave device, and position measurements of the slave device (Franken, 2011). Due to time delays, slave device commanded motion and force feedback to the user are performed later in time than intended. This results in less system transparency, and in higher contact forces during interaction between slave device and the environment. Due to time delays, the passivity layers will have to intervene more than without time delays. A direct cause is due to the receiving of delayed energy packages. If energy is not present in time at master or slave side, control actions cannot be performed. An indirect cause is due to oscillations resulting from less-damped behavior, which cause energy output at the slave side to be more than energy input at the master side. This can be explained by the transfer function of the

controller. If there is less-damped behavior, the slave device position will oscillate around the setpoint, which costs energy. If the setpoint is constant, energy is not put into the controller at the master side, whereas energy is consumed at the output to the slave side. All these effects are indications of time delays in a teleoperation system and can be used to design experiments and interpret measurements.

Quantitative performance has been observed from 10ms to 1s additional time delays in teleoperation systems which used the Two-Layered framework (Franken, 2011; Franken et al., 2011; Teeffelen, 2018). Observations are that oscillations increase for higher time delays and transparency decreases as signals are more delayed. Oscillations were mainly observed in master controller output force. Performance of correctly executing tasks by systems with delay was less than systems without delay (Teeffelen, 2018).

3.6 System Testing

Design and implementation of the teleoperation system must be tested to determine the performance. The main goal of testing is to see the effect of the LUNA network channels on teleoperation. This effect is hard to measure in an absolute way, therefore it is chosen to compare different implementations of the same design. Characteristics of teleoperation must be refined, such that performance can be quantified.

A system using LUNA network channels will be compared with the 'ideal' system without network communication and a system with network communication provided by the ROS middleware. For implementations that use a network, effects of network communication and middleware that provides access for the controller to the network, must be separated. Therefore, the LUNA network channels are compared with ROS network communication. In this comparison it is assumed that network transport properties are equal for both middleware. To keep effects of network transport as low as possible, it is chosen to introduce minimal network delay by connecting master and slave devices via a single network switch. In this way, the influence of network transport is of minor influence compared to the influence of network-accessing middleware (section 3.5.2).

Teleoperation is the execution of a task in a remote environment by a user via a teleoperation system. Transparency is a term which is inseparable connected with teleoperation and characterizes the technical performance of the teleoperation system. Transparency is technically achieved if slave device position and force follow master device position and force (Hashtrudi-Zaad and Salcudean, 2001). Master and slave position and force are measurable quantities that can be used to assess the transparency of a system. Transparency can subsequently be used to give a technical performance measure of the teleoperation system. The technical performance of a teleoperation system is an interesting property, as force and position over time are directly influenced by network delays.

However, considering technical performance (transparency) of a teleoperation system alone would give a limited view in the context of teleoperation. The degree in which a task can be successfully completed via the system would be a higher-level performance measure for teleoperation. The task to be executed must be chosen carefully, such that they are representative for a typical teleoperation application. Skills such as decision making and dexterity are typical human characteristics required for execution of a task via a teleoperation system. Another factor that offers a different aspect to teleoperation is the qualitative perception of the user. Perception could be measured by the assessment of task difficulty by users. Perception and performance can be combined to obtain both a qualitative and quantitative result corresponding to a task performed via a specific system implementation.

In Boessenkool et al. (2018), different types of tasks executed via a teleoperation system are evaluated. The main conclusion is that the performance measurements of task execution via

teleoperation systems inherently are characterized by large between- and within-person variance, making results less suited for decisive conclusions. Different characteristic tasks for teleoperation are among others: assembly, bolting and peg-in-hole (Boessenkool et al., 2018). All these tasks are based on interaction with the environment, which require the human decision making and dexterity skills. Besides interacting with the environment, free space motion will always be a part of any task executed via the teleoperation system. In other work (Teeffelen, 2018), teleoperation is successfully characterized by two user experiments: free space motion and random interaction force disturbances.

Based on results of previous work, it is chosen to perform a free space motion experiment and a peg-in-hole experiment in this project. The free space motion task is chosen as a dexterous task to characterize teleoperation. The peg-in-hole task is chosen to characterize interaction between slave device and environment, where dexterous movements and decision making by the user are required. The performance of this task can subsequently be used to characterize teleoperation. The limitations imposed by design decisions and the use of a system with real hardware instead of simulation are a basis for these choices. By combining the results of an interaction task and a free space motion task, it is expected that the performance of teleoperation and perception of users can be characterized.

Experiments are set up to characterize the influence of time delays caused by network-accessing middleware on the operation of a teleoperation system. Time delays are not the only limiting factor on teleoperation. It is therefore important to exclude other influential factors as far as possible (in the broadest aspect of teleoperation) and to keep remaining influential factors equal for all system implementations. Influences from design choices of the teleoperation system such as controller parameters must be kept equal for all implementations. Influential factors due to users or the environment are inevitable but must be kept as low as possible. If all limiting factors are suppressed enough, differences between implementations can only be caused by differences in network-accessing middleware.

3.7 Hypotheses

In this analysis chapter several design choices have been made, combining control theory, software constraints and available hardware. The design based on these choices will form a basis for several different implementations, such that properties of the LUNA network channel can be tested in a teleoperation system. Tests must be chosen to give an answer to the main research question as stated in section 1.3:

"To which extend is the LUNA network channel suitable for application in teleoperation systems?"

Based on the main research question and the described testing aspects and methods, several hypotheses are formulated to form a basis for experiment design. Methods for experiments are described in chapter 5. Experiments can be done to give an answer to each hypothesis, which can subsequently be combined to answer the main research question. The following hypotheses are formulated:

1. **"Computational platforms will not be a significant influence on real-time performance of the designed controller."**

   This hypothesis might sound basic, but is key for a correctly functioning system. If timing of a controller in the teleoperation system would be incorrect, the teleoperation system might not be functional and consequently not suitable as a basis for testing the next hypotheses. Significant influence on timing is chosen to be more than one order of magnitude deviation from design (section 2.3).

This hypothesis can be tested by measuring timely execution of controller actions. By measuring the length of consecutive sample intervals, the determinism and accuracy of timing can be determined and assessed.

2. **"Additional delay introduced by network communication increases oscillations and decreases transparency in a teleoperation system."**

   The reason for this hypothesis is that the effects of delay introduced by an implementation with network communication with respect to an implementation without network communication are expected to be clearly present in measurements of controller signals. This hypothesis is verified to be true by previous work (Franken et al., 2011; Teeffelen, 2018) and is used in this project to verify functionality of each teleoperation system implementation.

   This hypothesis can be tested by measuring controller signals. Especially the position and force of the slave side can be compared with the position and force of the master side to find the level of transparency.

3. **"Time delay introduced by network-accessing middleware does not decrease performance and perception of tasks executed via teleoperation systems."**

   Despite the effects of delay caused by the network-accessing middleware, human dexterity will be adequate such that these effects are insignificant for the performance and perception of task execution via a teleoperation system. This hypothesis is based on the assumption that delay caused by network transport is insignificantly small compared to delay caused by network-accessing middleware. As discussed in section 3.5.2, network transport via a single network switch corresponds to a situation in which this assumption is valid.

   This hypothesis can be tested by comparing the performance and perception of users that perform a task using the teleoperation system implemented with different network-accessing middleware.

4. **"Performance and perception of task execution on a teleoperation system with LUNA network channels do not decrease compared to a system with ROS network communication."**

   The reason for this hypothesis is that it is expected that time delays caused by LUNA network channels and ROS network communication are similar and similar time delays should cause the same performance and perception of task execution.

   This hypothesis can be tested by comparing the performance and perception of users that perform a task using the teleoperation system implemented with different network-accessing middleware.

4 Design and Implementation

In this chapter the detailed design and implementations of the teleoperation system in this project are described. First, the controller architecture is described and detailed. Subsequently, implementation details are addressed. Parameters or used constants that are described in the design are quantified in section 4.9.

4.1 Controller Architecture

This section is used to give an overview of the master and slave controller architectures. The master and slave controller architectures are given in figures 4.1 and 4.2 respectively using signal flow diagrams. A legend with signal symbols and corresponding explanation can be found in table 4.1. Boldface symbols denote a vector. Device interfaces, network interfaces, velocity estimation, passivity layers and transparency layers will be discussed in more detail. The transparency layer in the master controller is relatively simple. The transparency layer in the slave controller is built up from an impedance controller, forward kinematics and joint damping. Velocity estimation is performed within the Omega.7 device for the master side. For the slave side, velocities are estimated via state variable filters.

Different parts described in the controller architecture could run at different frequencies. Especially device and network interfaces are not necessarily capable of running at the controller frequency because they are designed and implemented outside the scope of this project. Rate conversions between controller frequencies and interface frequencies must be handled with care. This is described in section 4.4 for the LUNA network channel and in section 4.6 for device interfaces.

4.2 Transparency Layer

The transparency layers provide control actions for the slave device and give force feedback to the master device. The transparency layers in master and slave controllers are discussed separately. Main functionality of the transparency layers is implemented in the slave controller, therefore this is discussed first.

4.2.1 Slave Transparency Layer

The slave controller transparency layer generates a force for the master transparency layer and the slave device, based on position measurements. The position measurements of master and slave devices must be placed correctly with respect to each other, before they can be used for control actions. An inertial reference frame in Cartesian space is used to relate the position of the Omega.7 with the end effector of the KUKA LWR. The orientation of KUKA LWR is fixed, as discussed in section 3.5. Therefore, rotations of the Omega.7 device are not used and only the position is sent to the slave controller. To let the KUKA LWR end effector remain at a fixed orientation, a constant rotational setpoint is defined as R_o. The position of the KUKA LWR end

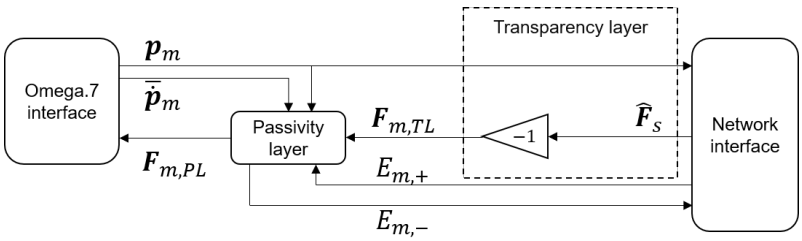


Figure 4.1: Master controller architecture.

Table 4.1: Definitions of discrete time signal names in master and slave controller.

Symbol	Definition
$\mathbf{p}_m$	Measured translational position of the master device.
$\hat{\mathbf{p}}_m$	Latest known $\mathbf{p}_m$ at the slave controller.
$\bar{\dot{\mathbf{p}}}_m$	Estimated translational velocity of the master device.
$\mathbf{F}_{m,PL}$	Output force master device limited by the passivity layer.
$\mathbf{F}_{m,TL}$	Output force master device calculated by the transparency layer.
$\mathbf{F}_s$	output force of the slave device end effector in translational directions.
$\hat{\mathbf{F}}_s$	Latest $\mathbf{F}_s$ after network communication.
$E_{m,+}$	Energy arrived at master side from slave side.
$E_{m,-}$	Energy to be sent from master side to slave side.
$E_{s,+}$	Energy arrived at slave side from master side.
$E_{s,-}$	Energy to be sent from slave side to master side.
$\mathbf{q}_s$	Measured joint positions of the slave device.
$\bar{\dot{\mathbf{q}}}_s$	Estimated joint velocities of the slave device.
$\mathbf{H}_{ee}^{0,0}$	Homogeneous matrix representing the position and orientation of the slave device end effector with respect to reference frame, defined in the reference frame.
$\mathbf{H}_{sp}^{0,0}$	Homogeneous matrix representing the position and orientation of the setpoint of the impedance controller with respect to reference frame, defined in reference frame.
$\mathbf{H}_o$	Homogeneous matrix representing constant rotation and position offset.
$\mathbf{W}_{ee}^{0,0}$	Wrench that the controller will apply on the slave device end effector, defined in the reference frame.
$\boldsymbol{\tau}_{TL}$	Joint torque outputs as calculated by the transparency layer.
$\boldsymbol{\tau}_{PL}$	Joint torque outputs limited by the passivity layer.

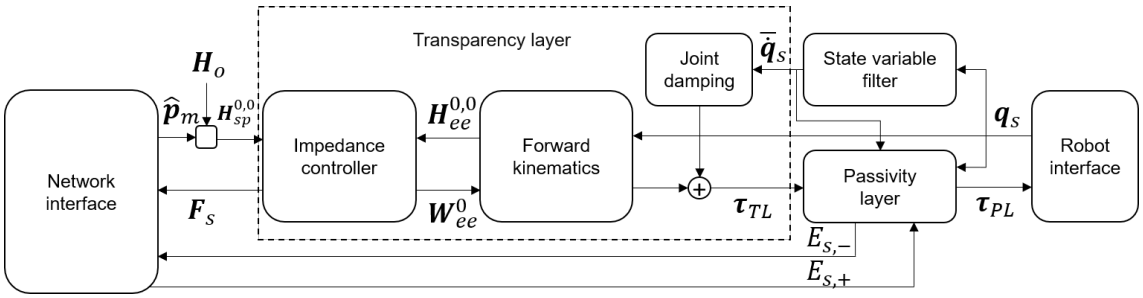


Figure 4.2: Slave controller architecture.

effector is based on a constant offset position (p_o) and the latest known position of the Omega.7 device ($\hat{p}_m(k)$). The offset position is used to place the KUKA LWR in the correct position in the environment, since the position range of the Omega.7 is limited. Calculation of the setpoint H-matrix in reference frame, with respect to reference frame is denoted in equation 4.1.

$$H^0_{sp}(k) = \begin{bmatrix} R_o & p_o + \hat{p}_m(k) \\ \mathbf{0}_3 & 1 \end{bmatrix} \tag{4.1}$$

Measurements of the joint positions of the KUKA LWR are input for the transparency layer. A mapping must relate joint positions to a H-matrix of the end effector in the inertial reference frame. This mapping is described by forward kinematics.

Based on difference in H-matrices that describe the master-position-based setpoint and KUKA LWR end effector in the same reference frame, it is possible to perform control actions. An impedance controller is used for the main control action to let the KUKA LWR end effector converge to the setpoint. The impedance controller acts as a mechanical spring between setpoint and end effector in all translational and orientational directions. The output of the impedance controller is a wrench (W^0_{ee}). The force components of the wrench are used for feedback to the master controller. The wrench must be mapped to joint torques for actuation of the KUKA LWR, which is described by forward kinematics (geometric Jacobian).

Controller output from an impedance controller alone could result in poorly damped (oscillatory) behavior. Therefore, additional viscous damping is added in the controller structure. Damping is applied in the joint space to avoid undamped motion in the null space of the device. The torques as a result of damping are added to the torques resulting from the impedance controller.

Detailed functionality of the impedance controller, forward kinematics and joint damping are discussed below.

Impedance Controller

The impedance controller from Stramigioli (1998) is used in this project. The main behavior of this block is to calculate a generalized force (a wrench, $W^{0,ee}(k)$) in reference frame based on the relative difference between the current setpoint and current KUKA LWR end effector location. This mimics the behavior of a mechanical spring. H-matrices are used to describe position and orientation of both setpoint and KUKA LWR end effector. The output of the impedance controller is a wrench to be applied to the KUKA LWR end effector, defined in reference frame. The force components from the wrench in reference frame calculated here, are also used as force feedback to the master device. The calculation steps for this process are denoted below. Calculations are based on spring constant matrices K_o, K_t and K_c, which are 3x3 matrices that describe orientational, translational and coupling spring constants respectively.

Calculation steps:

1. Calculation of a relative position and orientation:

$$H^{sp}_{ee}(k) = \left(H^0_{sp}(k)\right)^{-1} H^0_{ee}(k) \tag{4.2}$$

2. Calculation of momenta and forces in tilde form in end effector frame:

$$\begin{aligned} \tilde{M}^{ee,ee} &= -2\,\mathrm{as}\left(G_o\, R^{sp}_{ee}\right) - \mathrm{as}\left(G_t\, R^{ee}_{sp}\, \tilde{p}^{sp}_{ee}\, \tilde{p}^{sp}_{ee}\, R^{sp}_{ee}\right) - 2\,\mathrm{as}\left(G_c\, \tilde{p}^{sp}_{ee}\right) \\ \tilde{F}^{ee,ee} &= -R^{ee}_{sp}\,\mathrm{as}\left(G_t\, \tilde{p}^{sp}_{ee}\right) R^{sp}_{ee} - \mathrm{as}\left(G_t\, R^{ee}_{sp}\, \tilde{p}^{sp}_{ee}\right) R^{sp}_{ee}\right) - 2\,\mathrm{as}\left(G_c\, R^{sp}_{ee}\right) \end{aligned} \tag{4.3}$$

In these equations the time dependencies on sample instance k have been omitted for clarity. G_o, G_t and G_c are 3x3 orientational, translational and coupling co-stiffness matri-

ces respectively. These can be derived from 3x3 stiffness matrices K_o, K_t and K_c respectively, using equation 4.4. The anti-symmetric part of a square matrix can be calculated using equation 4.5.

$$G_* = \frac{1}{2}\text{trace}(K_*)I_3 - K_* \tag{4.4}$$

$$\text{as}(A) = \frac{A - A^T}{2} \tag{4.5}$$

3. Calculation of the wrench vector:

$$W^{ee,ee}(k) = \begin{bmatrix} M^{ee,ee}(k) & F^{ee,ee}(k) \end{bmatrix}^T \tag{4.6}$$

4. Calculation of the wrench in reference frame:

$$W^{0,ee}(k) = Ad^T_{H_0^{ee}(k)} W^{ee,ee}(k) \tag{4.7}$$

Forward Kinematics

Forward kinematics is implemented by H-matrix multiplication and the geometric Jacobian, as discussed in section 3.3.4. Detailed calculation steps and construction of the geometric Jacobian can be found in appendix C. The forward kinematics block has three main functionalities. First functionality is to transform joint positions to end effector position and orientation represented by a H-matrix in reference frame with respect to reference frame. This function is implemented by H-matrix multiplication. Second functionality is to transform a wrench to be applied to the end effector in reference frame to joint torques to be applied on individual joints of the KUKA LWR. This functionality is implemented by multiplication of the transpose of geometric Jacobian with end effector wrench. Third functionality is not implemented in the current design. The geometric Jacobian could additionally be used to transform joint velocities to end effector twist (generalized velocity) in reference frame, with respect to reference frame.

Joint Damping

Viscous joint damping is part of the control actions for the KUKA LWR. The damping is performed on joint velocity estimates ($\bar{\dot{\mathbf{q}}}_s(k)$), based on position measurements. In section 4.5.2 it is explained how the velocity estimates are calculated. The calculation for joint damping is denoted in equation 4.8.

$$\boldsymbol{\tau}_D(k) = -D_j \bar{\dot{\mathbf{q}}}_s(k) \tag{4.8}$$

4.2.2 Master Transparency Layer

The master transparency layer is relatively straightforward: the Omega.7 position measurement is sent to the slave side unchanged. The impedance controller at the slave side acts as a damped spring on the slave device end effector. The user of the teleoperation system will feel the other end of this spring. This also follows from the bond graph of the controller structure given in figure 3.5. The master controller must therefore output the negative of the slave controller output in Cartesian space. Communication between master and slave side transforms the slave controller output forces $\mathbf{F}_s$ to a delayed version $\hat{\mathbf{F}}_s$ upon arriving at the master controller. Previous reasoning results in:

$$\mathbf{F}_{m,TL}(k) = -\hat{\mathbf{F}}_s(k) \tag{4.9}$$

4.3 Passivity Layer

The Two-Layered framework has been designed for single degree of freedom systems in Franken et al. (2011). As the passivity layer functionality is a key concept used in this work, the functional concepts are repeated in section 4.3.1. In section 4.3.2, additional measures which extend the passivity layer to multiple degrees of freedom are discussed.

4.3.1 Core Functionality

The passivity layer created for this system is based on the single degree of freedom passivity layer described by Franken et al. (2011), which is described in this subsection. The function of the passivity layer is to create passive controller behavior. Passivity of the controller is obtained by monitoring energy flows between controllers and devices, and controllers and the network. Energy exchange between the controllers and the network is performed via the Simple Energy Transfer Protocol (SETP) as described in Franken et al. (2011). The SETP calculation steps are elaborated in appendix C.

Energy exchange with a device can be calculated using a bounded integral in continuous time (t) during one sample interval (T_s) at sample instant k ($k \in \mathbb{Z}$): $(k-1)\Delta T_s \leq t \leq k\Delta T_s$. The continuous time power conjugated variables $\dot{q}(t)$ and $\tau(t)$ are considered. The devices that interact with the controllers at both sides are impedance type, meaning that the controller exerts $\tau(t)$ on the device and the device exerts $\dot{q}(t)$ on the controller. The controller utilizes a zero order hold function to output digital signals, which implies that $\tau(t) = \tau(k-1)$ is constant during one sample interval. As a result, the energy exchange between device and controller (referred to as interaction energy) can be calculated as function of output $\tau(k-1)$ and measured positions $q(k)$ and $q(k-1)$:

$$\begin{aligned} \Delta H_I(k) &= \int_{(k-1)\Delta T_s}^{k\Delta T_s} P_I(t)\,dt = \int_{(k-1)\Delta T_s}^{k\Delta T_s} \tau(k-1)\dot{q}(t)\,dt \\ &= \tau(k-1)\int_{(k-1)\Delta T_s}^{k\Delta T_s} \dot{q}(t)\,dt = \tau(k-1)\big(q(k)-q(k-1)\big) \end{aligned} \tag{4.10}$$

Part the transparency layer at each side is a lossless energy tank, in which the total energy within the controller is stored. The energy level in the tank at sample instant k is denoted with $H_*(k)$, where $*$ denotes either master or slave side. Once energy within the tank is non-positive, passivity is violated. Violation of passivity can be avoided by reducing the energy output of the controller to zero. Energy packages to the other controller side will not be sent and interaction energy is reduced to zero. The interaction energy can be reduced to zero by limiting the transparency layer output effort to zero.

If transparency layer output is only limited to zero once the tank level is non-positive, the passivity layer actions will always be too late. To anticipate on loss of passivity, the passivity layer is extended with different saturation methods to limit the actions of the transparency layer. A schematic overview is given in figure 4.3. The transparency layer output is limited if the energy tank is empty, fundamental for passivity of the system. Furthermore, the limits $\tau_{lim,2}$ and $\tau_{lim,3}$ are introduced.

The second limit is setup to anticipate loss of passivity. Any function could be used to shape interaction energy between controllers and devices, based on available energy within the controller. This controller effort limitation is based on an estimate of the interaction energy during the next sample interval, which is described by:

$$\overline{\Delta H}_I(k+1) = \tau_{TL}(k)\bar{\dot{q}}(k)\Delta T_s \tag{4.11}$$

This estimate is based on the assumption that the device velocity is not influenced by the current effort $\tau_{TL}(k)$, such that $\dot{q}(k+1) \approx \dot{q}(k)$. The current effort will of course influence the

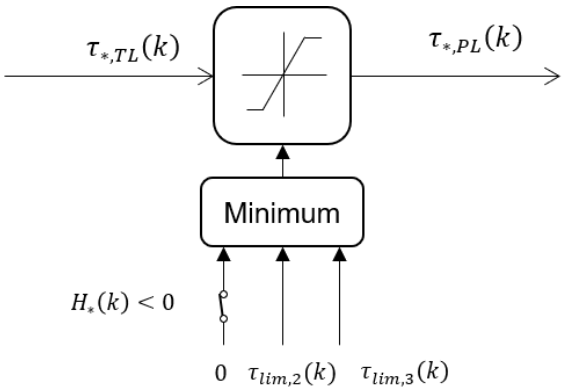


Figure 4.3: Functional block diagram of effort saturation in the passivity layer.

device velocity, which means the accuracy of this estimation is limited. The velocity estimate as described in section 4.5 is based on low pass filtering. This is beneficial since the velocity estimate is based on a relatively longer period, meaning less influence of the output effort during one sample interval.

Once the estimated amount of interaction energy is known, the controller output effort limit can be set up such that not more than the energy available in the tank is used in the next controller iteration. The limited interaction energy is a scaled down version of the estimated interaction energy. The interaction energy can be lowered by the controller by lowering the output effort. The limited interaction energy ($\Delta H_{I,L}(k+1)$) can therefore be described as in equation 4.12. This is not equal to the actual interaction energy $\Delta H_I(k+1)$ since it is based on a velocity estimate ($\bar{\dot{q}}(k) \neq \dot{q}(t)$) and the assumption made above. The corresponding controller output effort limit can subsequently be derived, see equation 4.13.

$$\Delta H_{I,L}(k+1) = H_*(k) = \tau_{max,2}(k)\ \bar{\dot{q}}(k)\ \Delta T_s \tag{4.12}$$

$$\tau_{max,2}(k) = \begin{cases} \dfrac{H_*(k)}{\bar{\dot{q}}(k)\Delta T_s} = \dfrac{H_*(k)}{\overline{\Delta H}_I(k+1)}\tau_{TL}(k) & \text{if } \overline{\Delta H}_I(k+1) \neq 0 \\ |\tau_{TL}(k)| & \text{otherwise} \end{cases} \tag{4.13}$$

The output of the passivity layer is a bounded version of the transparency layer output. The limits on controller effort could be appended with additional constraints, for example without dependence on the energy within the controller. One of these limits is the maximum absolute output effort of a device. This is a constant value, therefore $\tau_{max,3}$ can be set to this maximum directly.

Subsequently after all limits are determined, they must be combined. This is performed as a saturation depicted in figure 4.3 and described in equation 4.14:

$$\tau_{PL}(k) = \text{sign}\big(\tau_{TL}(k)\big)\ \min\big(|\tau_{TL}(k)|, |\tau_{max,1}(k)|, |\tau_{max,2}(k)|, |\tau_{max,3}(k)|\big) \tag{4.14}$$

The aforementioned energy flows and saturations will be dependent on the control actions of the transparency layer and movement of the connected devices over time. A deadlock situation could arise, in which both controller sides and the communication channel do not contain energy. This means controller output will also be limited to zero. If the controller output at both sides it set to zero, interaction energy (ΔH_I) will also be zero and energy cannot enter the controller anymore. As it is required to strictly separate transparency and passivity layer, this situation must be prevented. The passivity layer at the master side is complemented with a

Tank Level Controller (TLC), which is used to maintain a desired energy level (H_D) in the tank at the master side. The function of the TLC is to extract energy from the user once the energy tank level is below the desired tank level, to prevent and always recover from a deadlock situation. The TLC is implemented as a modulated viscous damper, to extract energy from the user to fill the master energy tank:

$$\tau_{TLC}(k) = \begin{cases} -\alpha\big(H_D - H_m(k)\big)\dot{q}_m(k) & \text{if } H_m(k) < H_D \\ 0 & \text{otherwise} \end{cases} \tag{4.15}$$

$\alpha > 0$ is used as a constant to parametrize the amount of energy received from the user. The effort generated by the TLC will be added to the transparency layer effort. The total controller output efforts are denoted as:

$$\begin{aligned} \tau_{out,m}(k) &= \tau_{PL,m}(k) + \tau_{TLC}(k) \\ \tau_{out,s}(k) &= \tau_{PL,s}(k) \end{aligned} \tag{4.16}$$

As a result, the effort that is fed back to the user at the master side is corresponding less with the efforts between slave controller and device. This is a known limitation of the transparency of the system.

4.3.2 Extension to Multiple Degrees of Freedom

In Teeffelen (2018), the passivity layer concept was extended to multiple degrees of freedom. The basic idea is repeated here: energy for motion of the master device is equal to energy for motion of the slave device. Therefore, a single energy tank is used for all degrees of freedom in each master and slave controller.

The extension to multiple degrees of freedom in Teeffelen (2018) has a flaw, as passivity layers were only used for translational motion of the KUKA LWR. By only making translational motions passive and not the rotations, the complete system is not made passive. Complete passivity of the system is required for stability.

The passivity layers are based on energy monitoring. In this work, all energy output to the slave device will be monitored, such that the complete system is passive. Due to the assumption that energy within the KUKA LWR is only exchanged between end effector and environment or between joints and controller, and the fact that the geometric Jacobian is power-continuous (equation C.9), the control of motion of the end effector can be made passive by monitoring the energy exchange between joints and controller. The assumption only holds if other non-ideal energy exchanges with the KUKA LWR, such as friction and play are relatively small (i.e. a transparent device must be used). By choosing the controller structure proposed in figures 4.1 and 4.2, energy exchanged between the slave controller and KUKA LWR joints are monitored, and energy exchanged between the master controller and translations of Omega.7 are monitored. Due to power continuity of the geometric Jacobian, energy exchanged between slave controller and KUKA LWR joints is equal to energy exchanged between end effector and environment. This means the complete teleoperation system is made passive.

It is allowed that the master passivity layer has three degrees of freedom and the slave passivity layer has seven degrees of freedom. As long as energy input at the master side is enough for energy output at the slave side, both motions will be similar.

The main change of a multiple degrees of freedom passivity layer instead of single degree of freedom is that in- and output flows and efforts are vectors instead of scalars, with the vector length corresponding to the degrees of freedom at each controller side. Monitoring of interaction energy must be performed separately per degree of freedom, after which all interaction

energies are summed to a resulting net interaction energy. Equation 4.10 is extended to vector form in equation 4.17, via the inner product.

$$\Delta H_I(k) = \boldsymbol{\tau}^T(k-1)\left(\mathbf{q}(k) - \mathbf{q}(k-1)\right) \tag{4.17}$$

The individual components of the transparency layer effort vector ($\boldsymbol{\tau}_{TL}(k)$) must be limited by the passivity layer. The first limit is rather straightforward, since the controller output must be zero if there is no energy present in the energy tank. This principle must hold for all individual vector components. The third limit is also straightforward for multiple degrees of freedom, as maximums can be set for individual components. For the second limit, equation 4.13 is considered. Franken et al. (2011) originally stated this limit as the left part of equation 4.13, which does not provide clear insight for extension to multiple degrees of freedom. By rewriting the expression to the right side of equation 4.13, a dependency on $\tau_{TL}(k)$ appears, which makes the equation also suitable for multiple degrees of freedom. The expression on the right could also be understood as a downscaling of controller efforts based on the ratio of the estimated interaction energy and available energy in the tank.

The second limit must be reconsidered as it was previously treated as a limit to the transparency layer, but this is not the case for all values of $\overline{\Delta H}_I(k+1)$. The value of $\overline{\Delta H}_I(k+1)$ can result in different cases, in which the design is not always intended to limit the transparency layer:

1. $\overline{\Delta H}_I(k+1) > H(k)$
   The estimated interaction energy is greater than the energy in the tank. Equation 4.13 will limit the output effort.

2. $0 < \overline{\Delta H}_I(k+1) \leq H(k)$
   The estimated energy is smaller than the energy in the tank. Equation 4.13 will result in a higher value for $\tau_{max,2}$ than the transparency layer effort τ_{TL}. This means the system is not tending to become non-passive. In this case, the minimum in equation 4.14 is determined by $\tau_{TL}(k)$.

3. $\overline{\Delta H}_I(k+1) = 0$
   The estimated energy is expected to be zero. Division by zero is not possible, therefore limit 2 is set to the amplitude of $\tau_{TL}(k)$. It is estimated that the interaction energy during the next sample interval is zero, therefore non-passive behavior is not expected.

4. $\overline{\Delta H}_I(k+1) < 0$
   The interaction energy during the next sample interval is estimated to bring energy into the system. This is only the case if $\tau_{TL}(k)$ and $\bar{\dot{q}}(k)$ in equation 4.11 have a different sign. This situation is likely to occur if $\bar{\dot{q}}$ is near zero (especially due to noise in the velocity estimate).

A conservative approach is chosen by treating negative energy estimates the same way as positive energy estimates, the same way as Teeffelen (2018). This means negative energy estimates will be limited in the system in the same way as in situation 1 if $\overline{\Delta H}_I(k+1) < -H(k)$, since the minimum in equation 4.14 uses absolute values of inputs. This approach is not performing as the intended limitation. The intended limitation only limits $\tau_{TL}(k)$ in situation 1. In situation 2, 3 and 4 it is not required to limit the transparency layer control action. The algorithm is not improved in this work due to time limitations. It is suggested to change the condition that enables the limitation from $\overline{\Delta H}_I(k+1) \neq 0$ to $\overline{\Delta H}_I(k+1) > H(k)$ in equation 4.13.

4.4 Network Interface

The analysis in section 3.5.2 shows that the LUNA network channel has a maximum operating frequency that is lower than controller frequencies used in teleoperation systems designed in

Franken et al. (2011) and Teeffelen (2018), which are 250Hz and 1kHz respectively. The controller structure must be adapted to deal with this difference. Communication between master and slave via the network is done between the transparency layers and between the passivity layers. Communication in both layers requires different adaptations.

For convenience T_{cntrl} is used to denote the sample interval of the controller and T_{com} is used to denote the sample interval of the communication. The situation in which $T_{com} > T_{cntrl}$ is considered in this section, since the LUNA network channel will operate with a bigger time interval than the controller.

4.4.1 Network Communication between Transparency Layers

The master transparency layer sends a setpoint position and the slave transparency layer sends an output force. Ideally, communication of both values is performed without additional delays. Realistically this is not possible. If communication is performed at a slower rate than calculation of controller values, an additional delay is introduced or not all information can be sent. It chosen to minimize delays by sending the latest calculated controller output and discard other controller outputs that have been calculated in the time between the previous and current communication. This is chosen because controller calculations performed before the latest calculation, are based on old measurement data. By communicating the latest calculated value, the maximum delay is $T_D < T_{com} + T_{cntrl}$, which is bounded. The communication could be seen as a "Last In, First Out" (LIFO) protocol.

4.4.2 Network Communication between Passivity Layers

The passivity layers at master and slave side communicate energy packets. It is desirable that a communication principle without loss of energy is used. If the LIFO method as for the transparency layer communication would be used, it is guaranteed that energy will be lost in the communication as data (or energy in this case) is discarded. To conserve energy, the sending interfaces will "integrate and dump" over one communication time interval. This action is denoted in equation 4.18. Here, E_{com} is the energy sent by communication, E_{sum} a buffer that sums energy output of the passivity layer and resets to zero once it is transferred to E_{com}, $E_{*,-}$ is energy output of the passivity layer at either master or slave side, t represents continuous time, δ is an infinitesimal time delay and $k, l \in \mathbb{Z}$ denote sample instances of controller and communication respectively.

$$
\begin{aligned}
E_{com}(l) &= E_{sum}(k) \\
E_{sum}(k) &= \begin{cases} 0 & \text{if } t = lT_{com} + \delta \\ \displaystyle\sum_{(l-1)T_{com} < kT_{cntrl} \leq lT_{com}} E_{*,-}(k) & \text{otherwise} \end{cases}
\end{aligned} \tag{4.18}
$$

At the receiver interfaces, the passivity layer expects an energy input every T_{cntrl} and the communication delivers energy every T_{com}. The method for rate conversion is denoted in equation 4.19. All energy is transferred from communication to the passivity layer once it is arrived. The input for the transparency layer is set to zero for sample instances where no energy is received from communication. Here, $E_{*,+}$ is the energy that is input for the passivity layer and E_{com} is the energy received from communication.

$$
E_{*,+}(k) = \begin{cases} E_{com}(l) & \text{if } lT_{com} < kT_{cntrl} \leq lT_{com} + T_{cntrl} \\ 0 & \text{otherwise} \end{cases} \tag{4.19}
$$

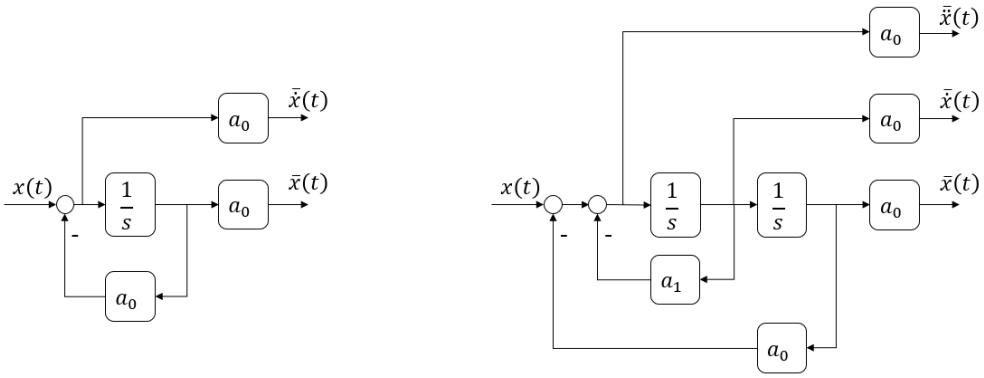


Figure 4.4: State variable filter structure. Left: first order filter, right: second order filter.

4.5 Velocity Estimation

Velocity of Omega.7 and KUKA LWR is estimated based on position signals. Velocity estimation is used in the master and slave controllers by the passivity layers. Additionally, velocity estimation is required in the slave controller to provide additional viscous joint damping. Velocity estimation of the Omega.7 is implemented within the device by the manufacturer. Here, it is tried to understand their design. Velocity measurements or estimation are not present within the KUKA LWR, therefore velocity estimation must be performed within the slave controller.

4.5.1 Velocity Estimation in the Master Controller

Velocity estimation of the Omega.7 device is done in the device internally. The velocity estimation is done using a low pass filter, implemented by a digital Finite Input Response (FIR) filter. The calculation is denoted in equation 4.20, where $k, n \in \mathbb{Z}$ and n is constant. The velocity estimate is based on a previous position sample subtracted from the current position sample. This method looks like numerical differentiation, but is different since a longer time interval is used. By taking a greater time interval between two position samples, a low pass filter is implemented, high-frequent noise is removed and the velocity estimate is smoothened.

$$\bar{\dot{x}}(k) = \frac{x(k) - x(k-n)}{nT_s} \tag{4.20}$$

4.5.2 Velocity Estimation in the Slave Controller

As discussed in chapter 3, state variable filters are used to estimate velocity in the slave controller. State variable filters implement a derivative operation based on integrators in a low pass filter. The structure of such filters in continuous time is depicted in figure 4.4. If the cut-off frequency is chosen as discussed above, then the filtered position output is approximately the position input ($\bar{x}(t) \approx x(t)$) and the derivative of the position output is approximately the derivative of the position input , which is equal to velocity ($\bar{\dot{x}}(t) \approx \dot{x}(t) = v(t)$). Higher order filters are capable of providing higher order derivatives, for example a second order filter could be used to obtain an estimated second derivative, corresponding to acceleration.

If the input signal contains a noise component, it will propagate directly to the highest derivative in the filter, such that the output signal of the highest estimated derivative will also contain noise. This situation can be circumvented by choosing the state variable filter order at least one order higher than the highest derivative output. In this way, the input signal passes at least one integrator (and low pass filter) which provides reduction of the high-frequent noise. In this project, a velocity estimate must be derived from a position signal, therefore a second order filter is chosen.

A continuous time filter cannot be implemented in a digital system. Therefore, the filter must be discretized. Discretization is done using the bilinear transform, denoted in equation 4.21. This is essentially a first order approximation of complex s in continuous time by complex z in discrete time and sample time T_s. The continuous time transfer function of the filter can subsequently be rewritten to a discrete time transfer function. The result for a second order filter is obtained by substitution of equation 4.21 in equation 4.22 and is denoted in equation 4.23.

$$s = \frac{2}{T_s}\frac{1-z^{-1}}{1+z^{-1}} \tag{4.21}$$

$$G_2(s) = \frac{\bar{\dot{x}}(t)}{x(t)} = \frac{a_0 s}{s^2 + a_1 s + a_0} \tag{4.22}$$

$$G_2(z) = \frac{\bar{\dot{x}}(k)}{x(k)} = \frac{a_0 \frac{T_s}{2}(1-z^{-2})}{\left(1 - a_1\frac{T_s}{2} + a_0\frac{T_s^2}{4}\right)z^{-2} + \left(a_0\frac{T_s^2}{2} - 2\right)z^{-1} + 1 + a_1\frac{T_s}{2} + a_0\frac{T_s^2}{4}} \tag{4.23}$$

Parameters a_0 and a_1 can be chosen such that the dynamics of the continuous time state variable filter are corresponding with the designed cut-off frequency. This can be done by equating the filter poles to a polynomial with designed pole locations, which is denoted in equation 4.24. A constant f_c is used to specify the cut-off frequency of the filter in [Hz]. Dynamics present in this project correspond to movements of the human lower arm and hand. Preliminary measurements showed that a cut-off frequency of approximately 30Hz is suitable as trade-off between capturing physical system dynamics and noise suppression.

$$\begin{aligned} s^2 + a_1 s + a_0 &= (s + 2\pi f_c)^2 \\ s^2 + a_1 s + a_0 &= s^2 + 4\pi f_c s + (2\pi f_c)^2 \\ a_1 &= 4\pi f_c \\ a_0 &= (2\pi f_c)^2 \end{aligned} \tag{4.24}$$

4.6 Device Interfaces

Both master and slave controllers must interface with attached physical devices. The interface to communicate measurements and controller output between controller and device is elaborated here.

4.6.1 Omega.7

The master controller requires an interface to communicate with the Omega.7 device. The interface utilizes functions defined in the "Haptic SDK" library given by the supplier. Both controller and the device software library use SI-units, so unit conversion is not required. Communication with the device is done at the same frequency as the controller. Functions to readout the current device position and velocity, and functions to set an output force are provided. All signals have three dimensions, corresponding with the three translational directions.

4.6.2 KUKA LWR

An interface between slave controller and the KUKA LWR is made using the Stanford FRI library (Stanford University, 2014). The reading of joint position measurements and the writing of controller outputs to joint torques is required. Both controller and the FRI library use SI-units, so unit conversion is not required. A velocity signal is not part of this interface and therefore the controller must generate it internally based on position measurements, which is described in

section 4.5.2. The KUKA KRC stops operation once the KUKA LWR arm moves with too high velocity (table 2.1), caused by too high torque commands. A saturation function is implemented in the interface between KUKA LWR and controller to make the interface more robust. The saturation is implemented by limiting the maximum joint output torques. The saturation in this project is set equal to $\tau_{max,3}$ from the passivity layer. If saturation takes place when a value lower than $\tau_{max,3}$ is chosen here, the actual interaction energy of the system is lower than the interaction energy calculated by the passivity layer. The result is that virtual energy is lost, which does not violate passivity but limits transparency. During initial testing it was found that the KUKA LWR interface runs at a predefined frequency of 500Hz. Due to time restrictions it was not possible to change this in the KUKA KRC to other frequencies.

A device interface working on a lower frequency than the controller is not necessarily a problem for control. For the transparency layer, the interface operating frequency must still be sufficiently higher than the device dynamics such that they are not influenced. The controller reads the same position measurements multiple times and output force commands are present for longer time periods to the device, due to a mismatch in operating frequencies. If position values are repeated, they result in zero difference, which results in zero energy flow between device and controller. Therefore, the value of the output force during that sample interval has no influence on interaction energy. Thus, the result is that passivity is not violated.

4.7 Component-Based Software Design

Computational components used within master and slave controller can be separated. Using figures 4.1 and 4.2, the following components are separated: passivity layer, impedance controller, forward kinematics and state variable filter. These components are implemented in a general way, such that they can be used in different applications with different signal dimensions. The general component structure (Ellery, 2017) in combination with the software library Eigen (Benoît and Guennebaud, 2018) for vector and matrix calculation are used to facilitate and structure the software. Facilities for testing and documentation provided by the general component structure are used to increase reusability of each general component.

4.8 Implementation in LUNA and ROS

Three different implementations of the designed teleoperation system are proposed in section 3.6. The implementations are based on: LUNA without network communication, LUNA with LUNA network channels and ROS with ROS-based communication. Implementation in LUNA and ROS is done in different ways. More practical information on how the LUNA and ROS applications and toolchains are set up, can be found in appendix E. Additional to the above mentioned implementations, an implementation in a single ROS program could be made as well. This implementation is described but not used for testing.

The first implementation corresponds with the 'ideal' teleoperation system, without delay between master and slave controllers. This implementation is done in LUNA without LUNA network channels in a single program. The architecture model and gCSP model of this program can be found in figures 4.5 and 4.6 respectively. The architecture model connects the main model containing the gCSP model with two timers. The timers generate the operating frequencies for the LWR interface and controller respectively. The gCSP model contains two sequential processes connected with a pri-parallel construction. One sequential process is interfacing with the Omega device and calculating the controller output. The other process is interfacing with the LWR. The priority is given to the LWR interface, as incorrect timing will trigger an error in the communication with the KUKA KRC. Priority is required to pass control actions to the LWR without time jitter. If priority would not have been given, both sequential constructs could be executed in different orders. This results in an undefined execution order per iteration. Data is communicated between parallel processes via global variables. This is not a nice

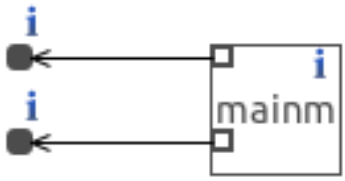


Figure 4.5: Architecture model for implementation in LUNA on a single PC.

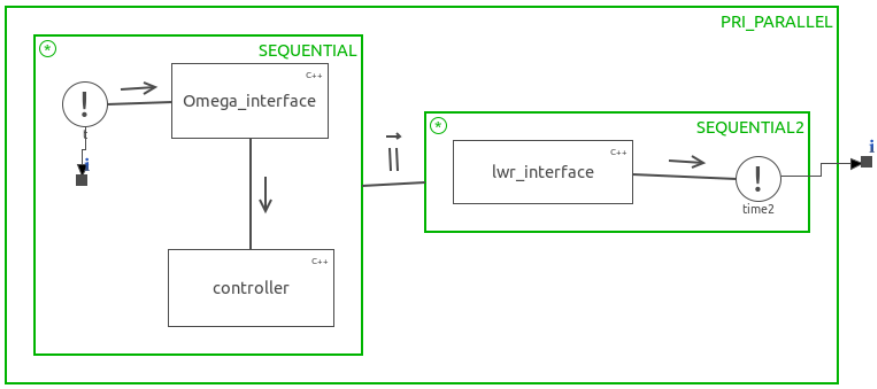


Figure 4.6: Main gCSP model for implementation in LUNA on a single PC.

implementation, as global variables could be altered by other processes. For now, the program size is limited an global variables are strictly maintained manually. The alternative would be to use nonblocking buffered channels, however these are not supported in the LUNA version with the DDS network communication.

The same implementation could be done in ROS. Here, a single node for the calculations of both controllers could be made. Device interfaces could be implemented as separate nodes communicating with the controller. The architecture of this implementation is depicted in figure 4.11. This implementation is not used for measurements.

The second implementation to be validated is done with LUNA network channels. Two LUNA programs with LUNA network channels have been made. One program is the master side controller and the other is the slave side controller, both including corresponding device interfaces. Master and slave programs are executed on different PC's. The LUNA network channels provide the communication between programs. The master controller architecture and main gCSP model can be found in figures 4.7 and 4.8 respectively. The master architecture connects the main gCSP model with two timers, four input and four output LUNA network channels. The timers generate the operating frequencies for the controller and communication. The four input LUNA network channels receive values corresponding to the slave controller output force (3 directions) and slave controller energy output. The four output network channels send the master position (3 directions) and master controller energy output.

The gCSP model of the master side consists of three different actions in parallel. The first action is to interface with the Omega device and to calculate controller actions. The energy communication with the slave controller requires conversion between operating frequencies of the controller and communication. For these conversions, processes are added before and after controller calculation, as well as before sending and after receiving. As sending of data must occur synchronously, the sending of all data is done after the rate conversion of energy com-

munication. Data is received asynchronously, since the communication is done over a network with variable delay. Dummy processes without functional code are used to explicitly denote independent asynchronous receiving of multiple variables (to show that there is a parallel construct of multiple recursive readers).

The slave architecture and main gCSP model are similar to the corresponding master controller models and can be found in figures 4.9 and 4.10 respectively. Additional to the master-side implementation, a timer is used for the LWR interface. The LWR interface is connected in priparallel with the control actions and communication.

The third implementation to be validated is done with ROS network communication. Two ROS nodes have been created, corresponding with master and slave controller. Device interfaces are implemented as nodes interfacing between the corresponding master or slave device and controller. Master and slave controller nodes and device interface nodes are executed on two PC's, a master and a slave. The ROS middleware provides communication between master and slave controller nodes. As ROS communication does not have a specified maximum speed, communication frequency is set to the same frequency as the controller. Since controller and communication frequency are the same, no rate conversion is required for this implementation. Implementation is done by creating two ROS nodes, corresponding to master and slave controllers. The master controller sends the master device position and energy to the slave controller and the slave controller sends slave controller output force and energy to the master controller. Both nodes interface with their corresponding device interfaces. The overall architecture is denoted in figure 4.12.

To keep delays as low as possible, messages communicated via topics should be received as quickly as possible after being sent. This suggested to use minimal buffer size for each topic. However, during preliminary tests, it was found that a significant amount of data was lost. Therefore buffered topics must be used, but the size is rather arbitrary, as buffers must be chosen 'large enough', but too large buffers introduce more delay. Due to poor documentation on topic buffers and lack of time, this is not further investigated. A buffer size of 1000 is used for communication between master and slave controller.

4.9 Parameters

The controller design as described in this chapter is implemented in different ways. All implementations will use the same controller parameters, such that difference in behavior is not caused by a change of parameters. Controller parameters are denoted in table 4.2.

Parameters corresponding with the passivity layer are based on Teeffelen (2018), as they resulted to be successful in a similar teleoperation system including the KUKA LWR and Two-Layered framework. The desired tank level (H_D) must be chosen high enough such that the system is operating smoothly under normal conditions, but must also not be chosen too high, to limit the amount of allowed active behavior. The TLC damping coefficient (α) must generate enough energy to let the master tank energy converge to H_D in time and must conversely not limit transparency and not disturb the user too much. The part of the local energy tank to be

Table 4.2: Controller parameters.

Parameter	Value	Unit	Parameter	Value	Unit	Parameter	Value	Unit
H_D	0.1	J	f_{cntrl}	1	kHz	k_t	500	$N{\cdot}m^{-1}$
α	200	$N{\cdot}s{\cdot}m^{-1}{\cdot}J^{-1}$	$f_{com,LUNA}$	250	Hz	k_o	50	N·m
β	0.01	-	$f_{com,ROS}$	1	kHz	k_c	0	N
$\tau_{max,3,m}$	12	N	f_c	30	Hz	D_j	0.1	$N{\cdot}m{\cdot}s{\cdot}rad^{-1}$
$\tau_{max,3,s}$	4	N·m	n	20	samples			

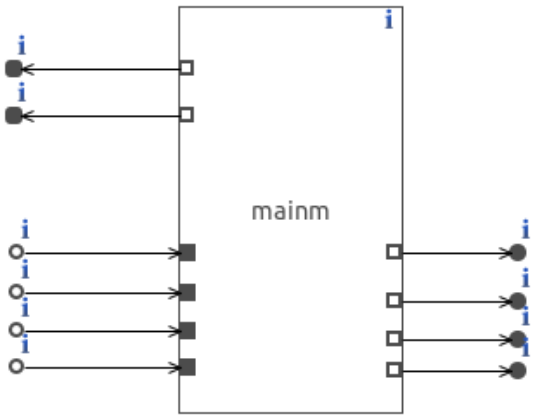


Figure 4.7: Architecture model for the master side of LUNA on two PC's.

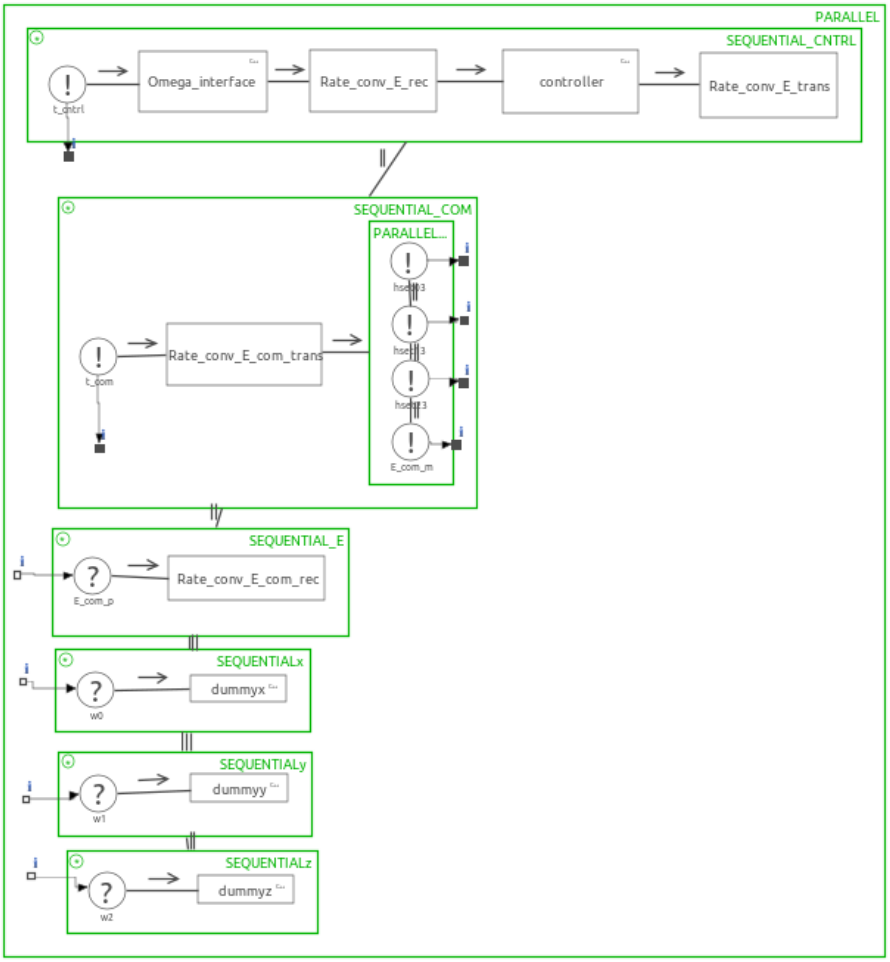


Figure 4.8: Main gCSP model for the master side of LUNA on two PC's.

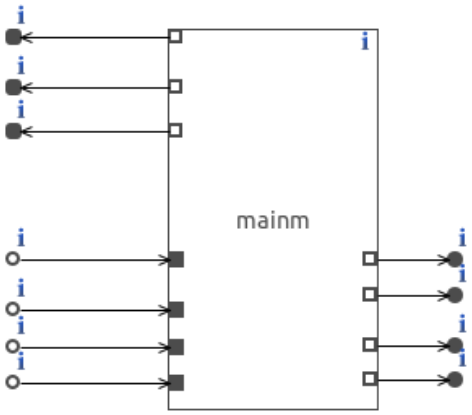


Figure 4.9: Architecture model for the slave side of LUNA on two PC's.

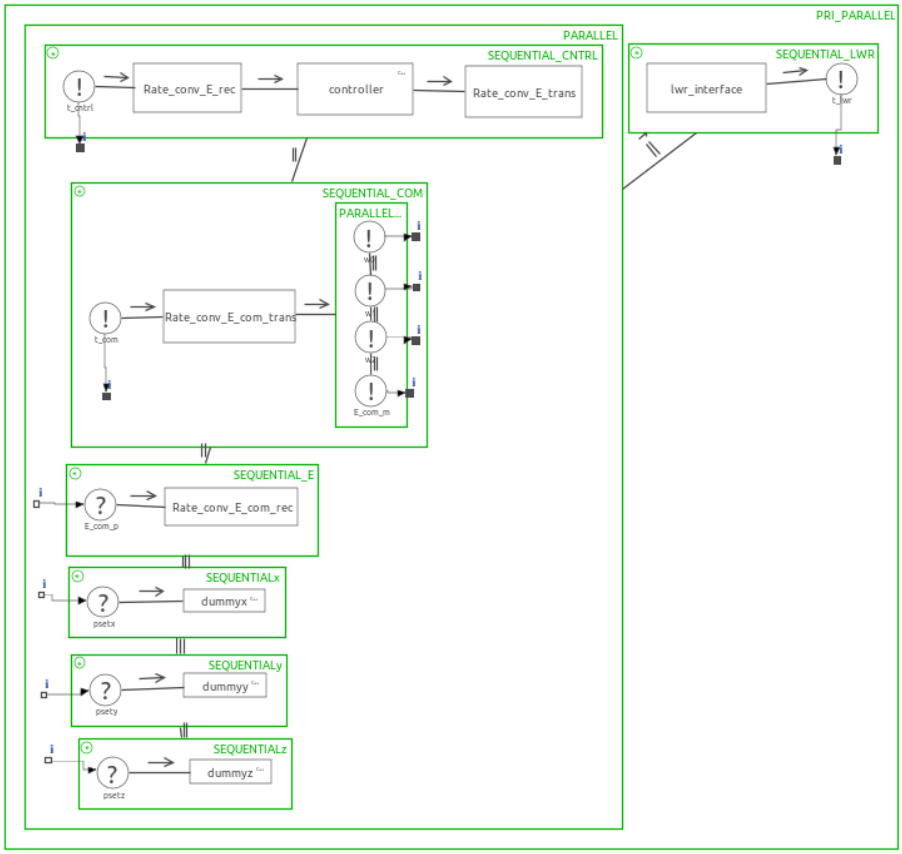


Figure 4.10: Main gCSP model for the slave side of LUNA on two PC's.

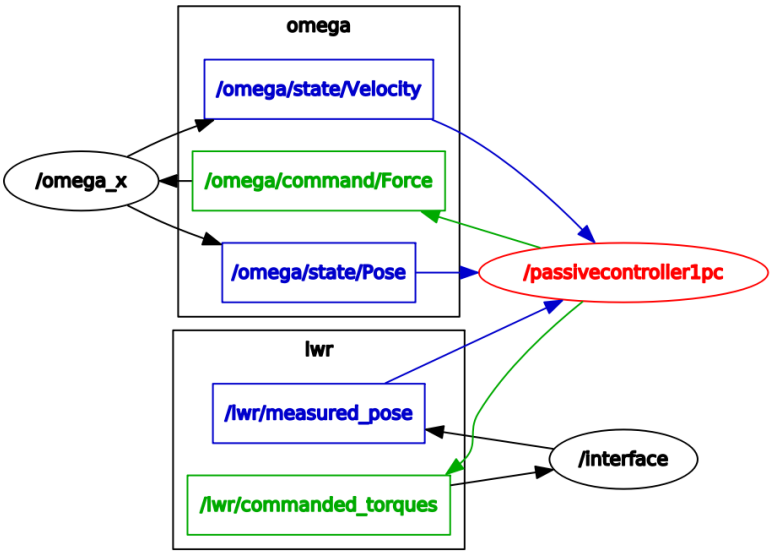


Figure 4.11: ROS nodes (ellipses) and topics (rectangles) for implementation on a single device.

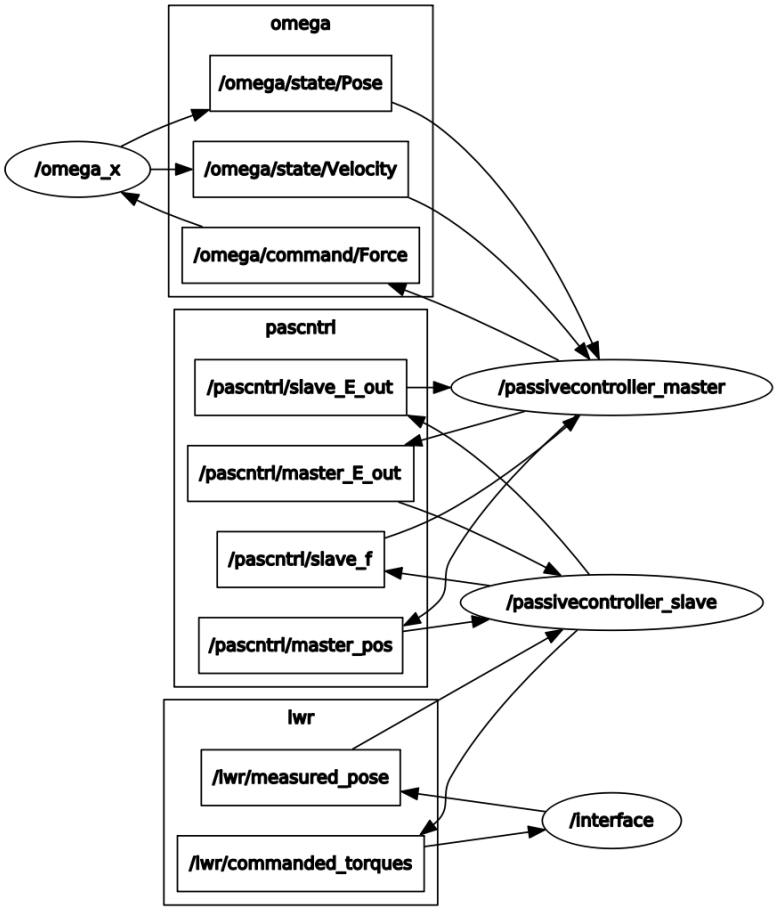


Figure 4.12: ROS nodes (ellipses) and topics (rectangles) for implementation with master and slave controller separated in two nodes.

sent to the other side (β) is chosen such that energy levels in master and slave controller converge quick enough and the system operates smooth.

The maximum output force for the master device ($\tau_{max,3,m}$) is set to the maximum output force of the Omega devices. The maximum output torque for slave device joints ($\tau_{max,3,s}$) is based on empirical testing. Typical motion and the given impedance controller settings showed that joint torques hardly exceeded the given value. This mainly serves as a safety limit, which also prevents the KUKA KRC from stopping operation. In this way, high torque commands are saturated on before they would have reached the KUKA KRC.

The controller frequency (f_{cntrl}) is based on systems from previous work that use the Two-Layered framework. Communication frequency (f_{com}) is set to the maximum for LUNA network channels and for ROS communication it is set to match the controller frequency. The cut -off frequency of the state variable filter (f_c) is based on empirical testing, such that velocity signals appeared smooth. The parameter n is used for the velocity estimation of the Omega device. This value is fixed in the current implementation of the Haptic SDK library.

All stiffness matrices are set to identity matrices multiplied with corresponding spring constants. The translational spring constant (k_t) is based on transparency measurements, described in appendix B. A compromise between device transparency and controller output force is chosen. The orientation spring constant (k_o) is found using empirical testing of adequate rotational movement of the end effector. Too high values result in oscillatory behavior, too low values result in errors between setpoint and measured orientation. The coupling stiffness (k_c) is set to zero, as coupling between commanded motion in rotation and translation domain is not used. Joint damping (D_j) is empirically determined with the goal of smooth operation given the impedance controller settings and typical commanded motion.

Additional to parameters mentioned in table 4.2, two more parameters are used in this design. First, the specifications of the master and slave PC's: Intel i7 @ 2.8GHz × 8, 12GB RAM, ubuntu 16.04, ROS Kinetic (master) and Intel core 2 @ 3.16GHz × 2, 4GB RAM, ubuntu 16.04, ROS Kinetic (slave). Second, offset H-matrices must be chosen around which the slave device will follow motion commanded by the master device. The offset matrices for functional and user experiments are denoted in equations 4.25 and 4.26 respectively. Offsets are chosen such that motion in the reachable range does not suffer from (near-)singularities and joint endstop limitations of the KUKA LWR arm.

$$H_o = \begin{bmatrix} 0 & 0 & -1 & -0.48 \\ -1 & 0 & 0 & 0.25 \\ 0 & 1 & 0 & 0.7 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.25) \qquad H_o = \begin{bmatrix} 0 & -1 & 0 & 0.42 \\ -1 & 0 & 0 & -0.23 \\ 0 & 0 & 1 & 0.06 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.26)$$

5 Materials and Methods for Testing

The materials and methods used for validation of design and implementations are described in this chapter. All validation tests will be done using a real slave device. Simulation tests are not executed, since the project set out to work on a real robotic setup and there is a time constraint. The drawbacks of using real systems instead of simulations are that the freedom of experiment design is limited and (unknown) disturbing factors, for example friction, are not controlled. The teleoperation system will be tested using functional tests described in 5.1 and human-dependent tests with human subjects described in 5.2.

The teleoperation system is implemented in three ways. The first implementation is on one device without network and is considered to be the 'ideal' implementation of this teleoperation system design, since network delays are not present. The second and third implementations are with network communication using LUNA network channels and using the ROS middleware respectively. The implementations will be compared by means of different tests with the goal to find out how much deviation between implementations is introduced as a result of delays caused by network-accessing middleware.

5.1 Functional Tests

Functional tests will be separated in two parts. The first part consists of unit tests to characterize timely execution of the discrete time controller and the second part consists of tests considering the overall performance of the teleoperation system.

5.1.1 Timed Execution Test

Goal: to verify that each implementation provides timely execution of the designed discrete time controller.

Execution of controller actions must be done timely, as controllers are designed for specific operating frequencies. The time between two consecutive sample instances of controller calculations is defined as the loop time of a certain process. The loop time will be measured for all timed controller actions in the system. These actions are: the computation of controller output values, interfacing with the KUKA LWR and communication. These correspond with all parallel processes in the controller architecture (section 4.8). The non-ideal loop times introduced by the slave PC on which the control law is executed will be measured. This is done to verify timely execution of control laws and to possibly relate with properties of the LUNA network channel as described by the original work (Wijnholt, 2017). If deviation from the designed teleoperation system in certain implementations is significant, the implementation is not suited for this teleoperation system. Significant deviation was defined in section 2.3 as one order of magnitude difference.

Absolute timestamps will be extracted from all implementations during 30 seconds of operation. Measurements will be evaluated using post processing to obtain the time difference between two consecutive sample instances (the loop time). The loop times can be used to assess non-ideal properties. Interesting properties of the time difference between two consecutive sample instances are: mean, standard deviation, minimum and maximum.

5.1.2 Teleoperation System Integration Test

Goal: to find the degree of transparency for each implementation.

Overall technical performance of the teleoperation systems will be tested based on functional tests described in other work in which teleoperation systems with the two-layered framework are designed (Franken et al., 2011; Teeffelen, 2018). The level of transparency is an indication

of the technical performance, as transparency describes the degree in which master position and force are similar to slave position and force. The teleoperation system is tested by moving the slave robot in x-, y- and z-direction, based on a predefined motion. The motion cannot be generated by a setpoint generator, as effects of force feedback would not be taken into account and the system would start in a deadlock situation without energy in the passivity layers of the controllers. The motion will first be a back and forth translation in free space, such that the system can start up. Subsequently, the motion of the slave device will be constrained by a wall. A vertical wall will be placed along the y-direction, such that motion in the x-direction provokes interaction with a static object. The wall is located at $x = 0$m, relative to the workspace of user defined motion. Relevant controller signals will be measured over time. These signals are: position and force in Cartesian directions, passivity layer tank energy levels of both master and slave, and TLC force of the master passivity layer.

Measurements of controllers implemented on two devices result in two measurements files (corresponding to measurements of master and slave controllers) that need to be synchronized for visualization of the functional test results. Synchronization is not straightforward to perform and will therefore result in limited accuracy of the measurement results. A symmetric network is assumed, which means communication delay from master to slave is equal to communication delay from slave to master. By monitoring a signal that is sent from master to slave side at both sides, and monitoring a signal that is sent from slave to master side at both sides, it is possible to do a synchronization. Synchronization is done manually by looking at signal characteristics of both signals. Notable signal characteristics such as extreme values or zero-crossings can be used to relate measurement files. Once both measurement signals are related in time, the network delay and measurement file time offsets can be calculated. Notable signal characteristics in two different signals will be at different time instances. This in combination with the assumption of a symmetric network might result in invalidly synchronized data, as network communication is characterized by variable delay. This means synchronization will limit the accuracy in time if master and slave signal properties are related. This will also make it difficult to quantitatively characterize delay. Synchronization methods from literature are not considered for this work due to time constraints, these could provide a better methodology and result.

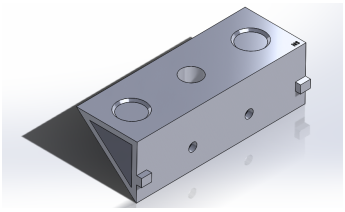
5.2 Tests with Human Subjects

Goal: to find the performance and perception of tasks relevant for teleoperation performed with each implementation.

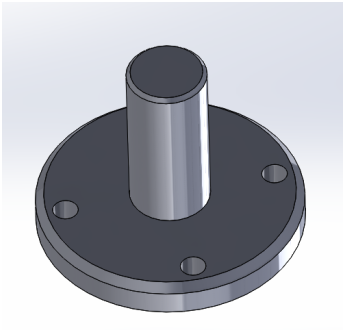
Tests with human subjects will be performed to characterize the influence of LUNA network channels on performance and perception of tasks execution. All users execute two tasks: positioning in free space and a peg-in-hole task. The choice for these tasks is based on reasoning in section 3.6.

The task of positioning will be done by using two static position setpoints separated by 120mm in the slave device environment. The user is required to move the slave device end effector as quick as possible from one setpoint to the other, and vice versa. Once a setpoint is reached, the user is required to remain at that position. The user will be given repeating audible command signals separated by 3 seconds. Once a new command signal is given, the user must execute the task of moving to the other setpoint as quick as possible.

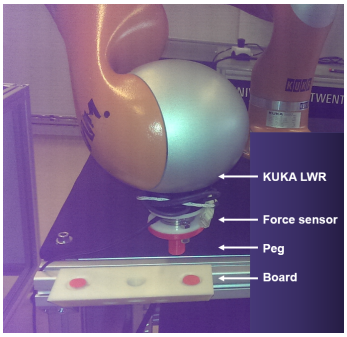
The peg-in-hole task requires the user to insert a peg attached to the slave device end effector into a hole present in the slave environment. The insertion task starts when the tip of the peg is moved far away enough from the hole. If the tip of the peg is outside a sphere of 50mm centered at the hole, it is considered to be far away enough. Once the peg is successfully inserted in the hole, the peg is removed again.



(a) Board with two position setpoints (lateral) and a hole (middle) for the positioning and peg-in-hole tasks.



(b) Peg to be attached to the slave device end effector.



(c) Complete setup with peg, board and KUKA LWR.

Figure 5.1: Hardware setup overview.

5.2.1 Procedure

Both tests must be repeated for each user and implementation. For the testing of one implementation with the positioning task, the cycle of: moving to the first setpoint, waiting, moving to the second setpoint and waiting, is repeated 10 times per user. For the testing of one implementation with the peg-in-hole task, the cycle of: inserting the peg in the hole, removing the peg from the hole and moving far away enough, is repeated 10 times per user.

Different implementations must be tested sequentially in time. The effect of learning by the users must not be underestimated, as it is assumed that users do not have any experience with teleoperation systems. To decrease the effect of learning on the results, several measures will be taken. First, users are given the chance to familiarize with the teleoperation system for as long as desired. Second, the positioning task is executed twice per different implementation per user. The first time will be considered as a practicing trial and the second time will be used for experiment results. Third, the positioning task will always be done before the peg-in-hole task and both tasks are performed for a single implementation before switching to a different implementation. Last, tests will be executed with different orders of implementations for different users. Users will be divided in groups, which perform the tasks with implementations switched in different orders. The tasks are performed using the ideal system on one PC without network communication (I), a system with LUNA network channels (L) and a system with ROS network communication (R). Sequences are: ILR, IRL, LIR, LRI, RIL and RLI.

5.2.2 Hardware Setup

The hardware used in this experiment is depicted in figure 5.1. A board with two position setpoints and a hole will be mounted in the slave device environment. A peg will be attached to the slave device end effector. Both parts are 3D-printed. The slave device orientation setpoint will be such that the peg is aligned with the setpoint locations and the hole. The diameter of the peg is 1mm less than the diameter of the hole. This value is chosen for a tight fit. Practical limitations influence this value: the printed parts have limited resolution, the peg and hole could be misaligned due to imperfect control actions and limited transparency of the teleoperation system increases task difficulty.

5.2.3 Participants

Ten healthy volunteers (all males, aged 20 to 27 years, 9 right handed) take part in the study. A person is considered healthy for this study if there are no problems with sight, hearing and functioning of the dominant hand. Participants without experience with teleoperation systems are selected. Participants are not informed about characteristics of different controller implementations.

5.2.4 Measurement Methods

Data analysis is done separately for the positioning and peg-in-hole tasks. During the positioning tasks, the setpoints are located along the y-axis. The position of the master device and slave device end effector along this axis are measured. The mean absolute error between setpoint and slave device position over time will be used give a measure for performance.

During the peg-in-hole task, slave controller output force could be used for measurements. The slave device has uncompensated inertia, therefore the controller output forces will be a combination of forces for device inertia and interaction with the environment. Using the slave controller output force is therefore a limited choice for characterization of interaction forces with the environment. A force-torque sensor (ATI MINI40-E) will be used. The sensor is mounted between the peg and the robot arm such that purely the interaction forces between peg and environment are measured. This is based on the assumption that the peg is lightweight and stiff enough to be a minimal influence on the measurement. A performance measure is related to the measured interaction force. The mean of the sum of the absolute interaction forces in all directions will be used as a performance measure. The sum of absolute interaction forces over time is first thresholded with a threshold value of 0.5N. This is done to calculate the mean only during time when interaction takes place. This threshold will cancel the effect of different experiment durations on calculation of a mean value.

Perception of the user is measured by asking the subjects to give a score on difficulty of using the system on a scale from 1 to 10, where 10 is extremely easy 1 is extremely difficult. Only integer numbers are used as a score. A score is given for each implementation.

5.2.5 Analysis Methods

Results of different system implementations must be statistically evaluated to assess the significance of results. Results of the positioning task and results of the peg-in-hole task are compared separately. Experiments with humans performing a task are characterized by high between- and within-person variance (Boessenkool et al., 2018), which could result in an outcome with low precision. Therefore it is chosen to use the noninferiority test described in Streiner (2003) to assess statistical significance of the results. This is a modification on a single tailed t-test. The modification describes an equivalence interval δ, which is based on technical judgment. The equivalence interval is the acceptable difference between results of different system implementations, such that performance can still be considered equal. Via the described method, a t-test is performed on group 1 with mean μ_1 as first entry and the sum of group 2 with mean μ_2 and equivalence interval δ as second entry.

A null hypothesis is described in equation 5.1: μ_1 is more than δ greater than μ_2. The alternative hypothesis is described in equation 5.2: the difference between both means is less than δ, which could also mean that μ_2 is greater than μ_1. The probability that the null hypothesis is true is indicated with the p-value. Low p-values are therefore an indication that the null hypothesis can be rejected. A p-value lower than 0.05 (5%) is considered to be statistically significant result.

$$\mu_1 > \mu_2 + \delta \tag{5.1}$$

$$\mu_1 - \mu_2 \leq \delta \tag{5.2}$$

If the null hypothesis can be rejected, the alternative hypothesis must hold. If for example the mean of results of a system implementation with network communication is chosen as μ_1 and the mean of results of the system implementation without network communication is chosen as μ_2 and the null hypothesis is rejected, it can be said that μ_1 is significantly not inferior to

μ_2. Or in the context of this project: a system implementation with network communication is significantly not inferior to the system implementation without communication.

The choice of δ is critical for this judgment. If δ is chosen to be zero, a normal t-test is performed and if δ is chosen too big, the null hypothesis will always be rejected. The value of δ must be chosen based on technical judgment. This choice is not trivial, since there is not a clear acceptance specification. Due to the user-based experiments, the variance in results is expected to be big. Since there is not a clear acceptance specification, it is interesting to find the equivalence interval δ which describes the boundary of results being significant or not. In this way, it can be judged if the equivalence interval is acceptable for equal performance. Future work can place the relevance of this project in perspective of their own performance acceptance specification.

6 Results

In this chapter, the results of tests described in chapter 5 are presented. The result of three different implementation of the same teleoperation system design are compared. First, results of the timing tests are shown in section 6.1.1. Second results of the functional tests considering the operation of the teleoperation system are shown in section 6.1.2. Last, results of tests with human subjects are shown in section 6.2. Additional measurement results not directly relevant for the project goal are shown in appendix D.

6.1 Functional Tests

6.1.1 Timed Execution Test

Results of timing tests can be found in table 6.1 and figure 6.1. For a more detailed segment of the measurement data, see appendix D. Considering the loop time of the controller, it is found that the controller implemented on a single PC using LUNA provides the behavior that is closest to the design: the mean value is exactly 1ms, corresponding with the chosen controller frequency, standard deviation is lowest and extreme values are closest to the mean value. Similar holds for the controller calculation time: mean, standard deviation and maximum are lowest of all implementations.

It is found that the mean controller loop time of the controller implemented with LUNA on two PC's is significantly different than the designed 1ms (1.42ms~704Hz). Standard deviation and the maximum value are both relatively high. These are all indications of poor hard real-time performance.

The LWR interface loop time is more deterministic than the loop time from other controller functions. This holds for all implementations. It is found that the LWR interface loop time in the implementation with ROS on two PC's is the best and the LWR interface in the implementation with LUNA on two PC's is the worst. For the LUNA implementations deterministic loop times are expected, as the LWR interface software was given highest priority. Deterministic interfacing with the LWR is beneficial from a practical view, as the KUKA KRC will not stop execution based on a bad connection ("FRI communication error") with the slave controller.

The loop times corresponding to transmission and receiving with LUNA on two PC's are slightly higher than the designed 4ms (corresponding with 250Hz). Standard deviation and maximum values are relatively high compared to ROS network communication. The standard deviation of receiving is slightly higher compared to transmission, which could be an indication of a network with variable delay.

The data transmission loop time via ROS network communication is characterized by approximately the same properties as the controller loop rate for that implementation. This can be explained, as transmission is done in each controller iteration. The standard deviation of transmission is lower than the standard deviation of the controller loop rate, which is also explainable since the accessing of the transmission ROS-topic is only a part of the performed actions within the controller. The receiving of data via ROS network communication is characterized by relatively low standard deviation and maximum values compared to LUNA network communication. The standard deviation in the receiving loop time is slightly higher compared to the transmission loop time, which could be an indication of a network with variable delay.

All mean timing properties of the implementation with LUNA on one PC and ROS on two PC's are significantly close to the designed system and are therefore suited implementations for the designed controller. The implementation with LUNA on two PC's shows overall worse performance. The mean loop time of the controller in this implementation is significantly different

from the design, meaning that actual controller behavior could be significantly different from designed controller behavior. In the original design of the LUNA network channels (Wijnholt, 2017), it was concluded that CPU load is significantly high if multiple variables are communicated. The cause for the nondeterministic timing properties in the controller with LUNA network channels is therefore thought to be that the LUNA network channels use relatively many computational resources, which interrupt the timed controller calculations. From the timing perspective, this means that the implementation with LUNA network channels is significantly worse than the other implementations.

Table 6.1: Timing test results. Values are in milliseconds (ms).

	Controller loop time		Controller calculation time		KUKA LWR interface loop time		Transmit loop time		Receive loop time	
	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
Ideal system	1.00	0.00	0.00	0.00	2.00	0.00	*	0.00	*	0.00
LUNA 1 PC	1.00	0.24	0.41	0.11	2.00	0.20	-	-	-	-
LUNA 2 PC's	1.42	1.53	0.93	1.57	2.01	0.70	4.11	1.38	4.27	1.83
ROS 2 PC's	1.03	0.29	0.57	0.26	2.00	0.04	1.02	0.23	1.00	0.60
	Min.	**Max.**	**Min.**	**Max.**	**Min.**	**Max.**	**Min.**	**Max.**	**Min.**	**Max.**
Ideal system	1.00	1.00	0.00	0.00	2.00	2.00	*	*	*	*
LUNA 1 PC	0.49	7.22	0.00	6.52	0.02	6.00	-	-	-	-
LUNA 2 PC's	0.36	18.26	0.00	16.65	0.01	8.28	0.04	12.06	0.03	12.23
ROS 2 PC's	0.40	11.49	0.00	11.24	1.55	2.46	0.40	6.54	0.02	9.71

* The designed communication frequency: none, 4.00ms or 1.00ms.

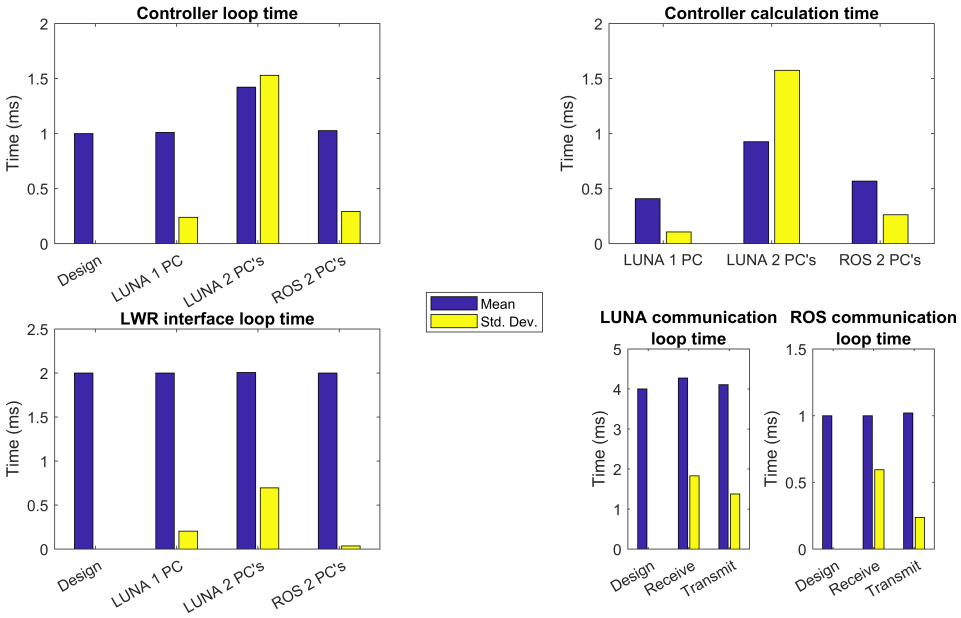


Figure 6.1: Timing test results. Each column shows timing measurement mean and standard deviation for the different implementations.

6.1.2 Teleoperation System Integration Test

Functional tests considering the operation of the complete teleoperation systems are depicted in figures 6.2, 6.3 and 6.4 for a LUNA implementation on a single PC without network communication, master and slave on two PC's communicating via LUNA network channels and master

and slave on two PC's implemented and communicating via ROS respectively. Results are analyzed in general first, to get a grasp of all measurement results. Second, the differences between results are compared and discussed.

General Analysis

In each test it can be seen that the free space motion of the master device is roughly followed by the slave device. This can for example be seen around $t = 15$s in plots corresponding with the y-direction in figure 6.2. There is a peak in the position plot, meaning that velocities of master and slave devices are first positive and subsequently negative. During the time the velocity is positive, the force feedback to the user is negative, meaning that the user must pull the mass of the slave device via the virtual spring in the controller. Once the velocity switches from positive to negative direction, the force feedback goes back to zero and becomes positive with approximately equal magnitude. The feedback forces during free space motion are mainly caused by the mass of the KUKA LWR. Feedback forces to the master device are a significant magnitude as they are in the same order of magnitude as force due to interaction with the wall.

If the user decides to give a relatively high change in velocity to the master device, the slave device will eventually follow. If the master device has already quickly moved and the slave device not, it can be seen that there is a peak of energy present in the energy tank of the master passivity layer. This energy is subsequently used for motion in of the slave device. Peaks observed in figures 6.3 and 6.4 are of greater magnitude than in figure 6.2. This can be justified as these figures correspond with systems that use network communication, and thus have a delay in the communication between master and slave controller. Due to the delay, there is more time between the motion of the master and slave devices.

During startup of the system, the energy within the tanks of the passivity layers of both controllers must be increased. This managed by the TLC which shows relatively high forces during the startup time correspondingly. Due to these high forces, transparency at startup is very poor.

If the master device moves beyond the wall position in the slave device environment in x-direction, it can be seen that the slave device cannot move further than the wall. As the master device position becomes more negative than the slave device position, the spring in the controller is stretched more. The spring acts in the opposed direction of the master device movement on the master side, therefore a relatively large positive feedback force occurs. It is also observed that tank energy levels are relatively high during interaction with the wall. This can be clarified because the master device position is changing whereas the slave device position is not. Since there is a feedback force present, the master side will bring energy into the system, whereas the energy is not going out of the system, since the slave device is placed against the wall and not moving. Clipping of the output force occurs, due to the combination of impedance parameters and position difference. This is not considered as a problem, since the force feedback indicates very clearly that there is interaction with the environment instead of just moving the weight of the KUKA LWR.

If lower impedance settings would have been chosen, feedback forces due to motion in free space would have been lower and feedback forces due to interaction with the wall would not clip. This result could be used to tune system parameters in later work.

Analysis per Implementation

In the implementation without network communication (figure 6.2) it is observed that TLC action is only required at startup. Afterwards, the energy level in the master controller remains above H_D. Since there is no delay in communication between master and slave controllers, there is also no mismatch in power variables communicated by the transparency layers of master and slave controllers. Therefore, energy flow between master device and controller is approximately the energy flow between slave device and controller, where the difference is caused

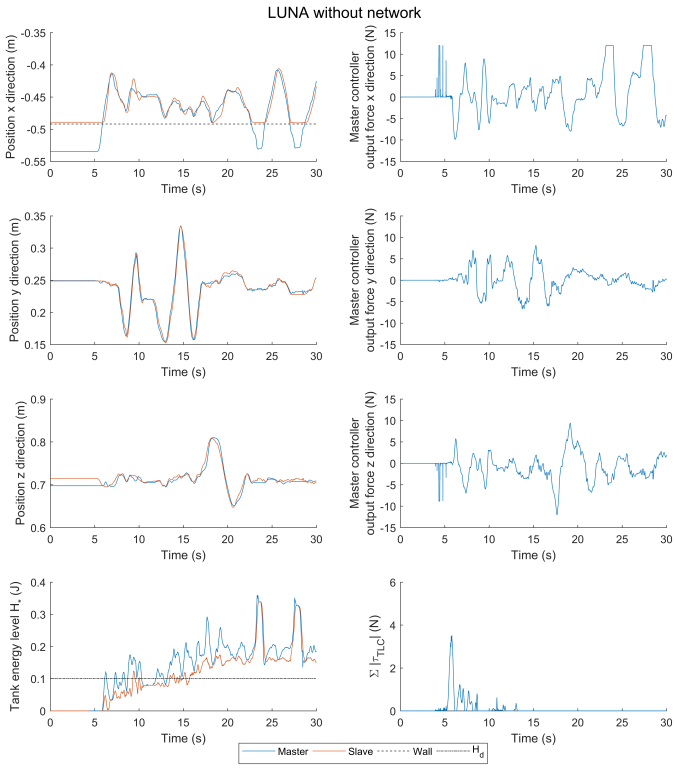


Figure 6.2: Results of the functional test with one LUNA application without network communication.

by the control algorithm. This is therefore considered to be ideal behavior for the given design. The energy level in the transparency layer energy tanks could go down because of non-idealities caused by implementation on a digital system (as discussed in section 3.4.1), however this cannot be found in the timespan of the measurement.

Systems with network communication included (figures 6.3 and 6.4) are limited by the time delay introduced by communication. Therefore, transparency layer signals (position and force) are slightly delayed by communication between master and slave. Results show that this is not significant as slave device position is still following the master device adequately and force feedback is provided in the same way as in the situation without communication delay. Exact quantitative results for transparency corresponding to the error between position and force signals at master and slave controller cannot be given, based on a number of reasons: synchronization of master and slave controller signals introduces errors in time, the traveled position is different for each implementation and interaction with the wall is different for each implementations. This is a limitation of the results, and therefore only a qualitative assessment can be given.

Despite above mentioned performance, subtle differences exist. More oscillations are observed in the master controller output force for implementations with network communication. It is assumed that this is also the cause for oscillations in master device position, as oscillations were felt by the user. Oscillations observed in the master controller output force from the implementation with ROS network communication are greater than with the implementation with LUNA network channels. More oscillations are a decrease of transparency of the teleoperation system.

Due to the delay introduced by network communication, the passivity layer must be activated more to keep the master tank energy level at the desired energy level H_D. This can be seen by nonzero TLC forces during the whole timespan of the measurements. This effect is found more

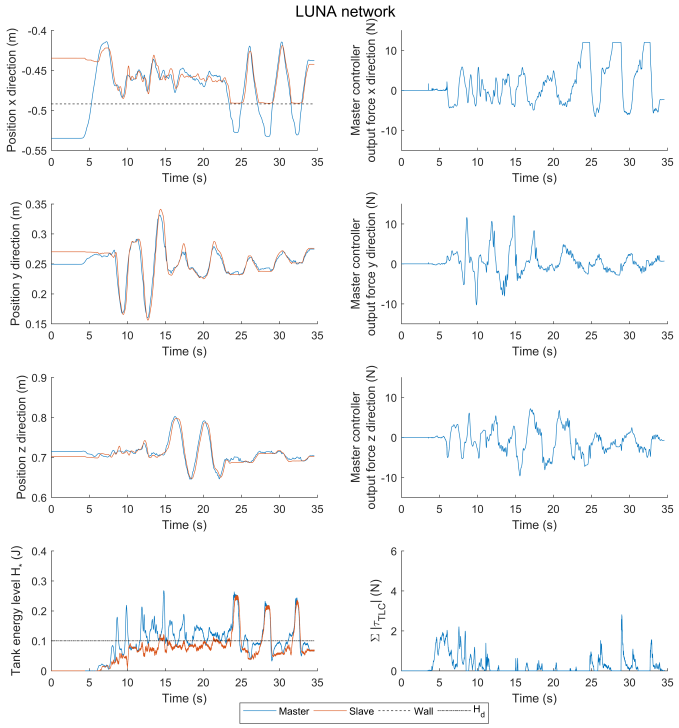


Figure 6.3: Results of the functional test with master and slave communicating via LUNA network channels.

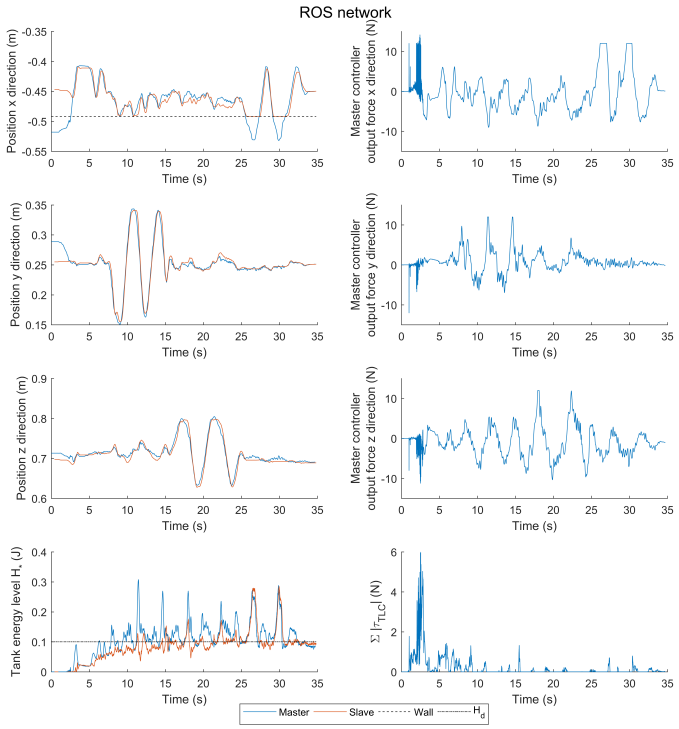


Figure 6.4: Results of the functional test with master and slave as ROS nodes communicating via ROS middleware.

for LUNA network channels than for ROS network communication. This effect is caused by a mismatch between communicated power variables at the master and slave side. A greater mismatch could be an indication of greater delays. However, since the motion of the master device is not exactly equal for different implementations, it could also be caused by different values of the communicated power variables. Despite more activation of the TLC, the teleoperation systems with network communication are stable, never show active behavior since the energy in both master and slave controllers is positive for all time and show proper functionality as position of the slave device follows position of the master device and the master device receives force feedback from interaction between the slave device and environment.

Looking more closely at the implementation with LUNA network communication, small oscillations are observed in the energy tank level of the slave device. This can be explained by the different communication and controller frequencies. If a closer look is taken at the measurement data, it is found that energy increases during one sample interval and energy decreases during the subsequent three sample intervals. This corresponds with a communication frequency that is four times lower than the controller frequency as the communication provides new energy every four controller sample intervals. The small oscillations do not form a problem, as the amplitude is insignificantly small compared to effects of system dynamics or communication delay.

6.2 Tests with Human Subjects

Results of performance in both user experiments are combined in figure 6.5. The results of the peg-in-hole experiment are on the horizontal axis and results of the positioning experiment are on the vertical axis. Additional to the results, the mean values and uncertainty region corresponding to the first standard deviation are depicted. The mean and standard deviation of both experiments, and results corresponding to the perception of users for each implementation are given in table 6.2. The mean and standard deviation of user perception per implementation can be found in figure 6.6. A detailed view on one recording of the positioning and peg-in-hole experiment can be found in appendix D.

Results corresponding to the performance measures for each experiment are spread which means it is not straightforward to separate different implementations. Looking at the results of each implementation separately, the results are spread, indicating a high between-person variance. Results of single users for each different implementation cannot be distinguished, indicating high within-person variance. By looking at the mean and standard deviation of each implementation, it is found that the system without network communication scores slightly better than other implementations in the positioning experiment. In the peg-in-hole experiment, the system without network communication scores slightly worse than the other implementations. The uncertainty of the implementation with ROS on two PC's in the positioning experiment is greater than the uncertainty of other implementations for that experiment. The uncertainty for the peg-in-hole experiment is similar for each implementation. More generally spoken, the results of all implementations are mixed, which indicates either that there is no difference between implementations or the measurements are not precise.

The results corresponding to perception of the users are similar for all implementations. Values for both mean and standard deviation are similar for all different implementations. Individual scores are given as integer values. 20% of the scoring range falls within the first standard deviation of the human perception results. The results are thus relatively coarse.

The results are statistically evaluated based on the method discussed in section 5.2.5. Two different implementations can be compared for significant noninferiority given that a small difference (the equivalence interval) in measurement results is acceptable. If one implementation is not inferior to another implementation, it is considered that performance is at least equal. A maximum equivalence interval can be determined, if the statistical significance level and the

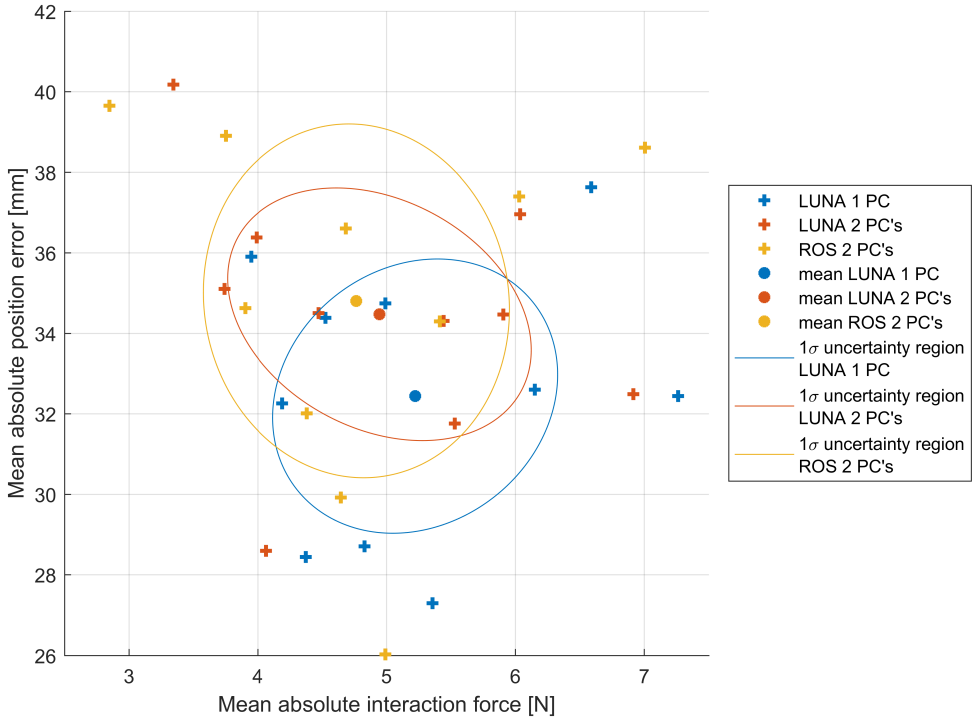


Figure 6.5: Combined result of both user experiment tasks.

Table 6.2: Mean and standard deviation per implementation of the positioning task, peg-in-hole task and perception of task difficulty. For perception, 1 means extremely difficult and 10 means extremely easy.

	Positioning task [mm]		Peg-in-hole task [N]		Perception [-]	
	μ	σ	μ	σ	μ	σ
LUNA 1 PC	32.44	3.41	5.22	1.11	7.40	1.08
LUNA 2 PC's	34.47	3.14	4.95	1.18	7.50	1.18
ROS 2PC's	34.80	4.39	4.78	1.19	7.10	0.99

measurement results are given. In this way, the maximum allowed difference between results is given as function of the significance level and the measurement data. Maximum allowable equivalence intervals for relevant statements are denoted in table 6.3. The interpretation of these statements is explained by considering the first entry in more detail: if a maximum difference of 3.99mm in performance measure between mean experiment outcomes is acceptable, the statement "A system with LUNA network channels is significantly not worse than the ideal system" is true.

It is now tried to place the equivalence intervals in the context of the experiments and teleoperation. The equivalence intervals for all statements corresponding to the position experiment results are equal to or below 4.11mm. This means all system implementations are significantly equal if a maximum error of 4.11mm between means of experiment outcomes corresponding to different implementations is accepted. In the positioning experiment, the users were required to move between two locations separated by 120mm. The equivalence interval is relatively small compared to the complete motion (3.4%). The equivalence interval is also relatively small compared to the mean absolute position errors that are used as performance measures ($\leq$ 12.7%). It must also be taken into account that the methods for this experiment are limited due to usage of a real robotic setup. In the context of teleoperation, a position difference of

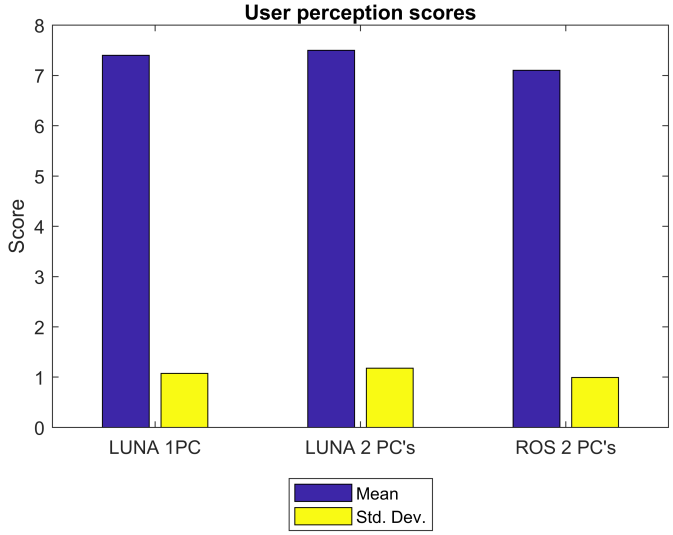


Figure 6.6: Mean and standard deviation of user perception per system implementation.

4.11mm would also not be problematic, as users can use their cognitive ability to assess the robot location in the environment and decide to move slightly further to complete the task (at the cost of using more time for a task). Combining these arguments, the maximum equivalence interval sizes for the position experiment are accepted as being sufficiently low for significant equal performance of each system implementation in the positioning experiment.

The peg-in-hole experiment equivalence intervals can be placed in perspective similar to the reasoning for the positioning experiment. The maximum equivalence interval is 0.68N, which is relatively small compared to the mean absolute interaction forces used as performance measures ($\leq$ 14.2%). It is complicated to relate this equivalence interval with the peg-in-hole task performance or teleoperation in an absolute way. Therefore the value of all equivalence intervals cannot be placed in perspective and statements about significance cannot be made. As statistical significance cannot be shown, it is assumed the results are not significant. Nevertheless, the results do not indicate a clear difference between different implementations and therefore the results suggest approximately equal performance. The equivalence interval for noninferiority of ROS with respect to the ideal system is very small. Therefore, a system with ROS network communication is significantly not inferior to the ideal system in the peg-in-hole experiment.

Based on the scores of perception for each implementation, equivalence intervals for noninferiority are generated. All equivalence intervals are greater than or equal to 0.51 on a scale from 1 to 10. Grades by single users are given as integer. If a grade would decrease with the amount of an equivalence interval, rounding would result in a lower grade which is a significant difference. Therefore it is chosen to not accept the results of perception as being statistically significant. Nevertheless, the results are close to each other and suggest approximately equal performance.

Table 6.3: Statistical evaluation of results: equivalence intervals for results to be significant.

Statement	**Equivalence interval** (acceptable difference)
Position results	
A system with LUNA network channels is significantly not worse than a system without network communication.	3.99 [mm]
A system with ROS network communication is significantly not worse than a system without network communication.	4.11 [mm]
A system with LUNA network channels is significantly not worse than a system with ROS network communication.	1.53 [mm]
A system with ROS network communication is significantly not worse than a system with LUNA network channels.	2.19 [mm]
Peg-in-hole results	
A system with LUNA network channels is significantly not worse than a system without network communication.	0.22 [N]
A system with ROS network communication is significantly not worse than a system without network communication.	< 0.01 [N]
A system with LUNA network channels is significantly not worse than a system with ROS network communication.	0.68 [N]
A system with ROS network communication is significantly not worse than a system with LUNA network channels.	0.32 [N]
Perception results	
A system with LUNA network channels is significantly not worse than a system without network communication.	0.61 [-]
A system with ROS network communication is significantly not worse than a system without network communication.	0.69 [-]
A system with LUNA network channels is significantly not worse than a system with ROS network communication.	1.31 [-]
A system with ROS network communication is significantly not worse than a system with LUNA network channels.	0.51 [-]

7 Conclusion

7.1 Conclusion

This project set out to design, implement and test a teleoperation system that includes network communication via LUNA network channels. The purpose of these actions is to assess the suitability of LUNA network channels for teleoperation, of which the main research question is derived:

"To which extend is the LUNA network channel suitable for application in teleoperation systems?"

To answer the main research question, a teleoperation system design was made. This design is based on an impedance controller that connects position and force of an Omega.7 master device with position and force of a KUKA LWR slave device. It was found that a KUKA youBot is insufficiently transparent to be used as slave device, due to joint friction. To test the properties of a teleoperation system with LUNA network channels, two other implementations where successfully made for relative comparison. Properties of an implementation without network, an implementation with LUNA network channels and an implementation with ROS network communication were compared.

To aid answering the main research question, hypotheses were setup, according to which experiments were performed. The hypotheses formulated in 3.7 are repeated here and answered with the measurement results:

"Computational platforms will not be a significant influence on real-time performance of the designed controller."

System functionality was tested on timely and deterministic execution of discrete controller actions. It was found that the implementations without network and with ROS network communication performed corresponding to the intended design. This shows that the computational platforms were suited for the designed controller and that the hypothesis is true. The implementation with LUNA network channels had significantly different timing properties than the intended design, which was mainly jitter of controller actions of which the cause is thought to be high CPU load from LUNA network channels. From a timely execution perspective, the LUNA network channels are therefore not suited for controlled systems and therefore also not suited for teleoperation systems.

"Additional delay introduced by network communication increases oscillations and decreases transparency in a teleoperation system."

The degree of transparency in a teleoperation system was technically evaluated based on comparison of master and slave position and force signals of the different system implementations. The teleoperation system integration tests showed that implementations with network communication resulted in more oscillations and a slight delay between master and slave position and force signals compared to the implementation without network communication. For the implementations with network communication it was also required that the passivity layer TLC was active during the complete experiment. Observations are indications of lower transparency due to network communication delay. The degree in which transparency is decreased is similar for LUNA network channels and ROS network communication. Therefore, it is concluded that the hypothesis is true. Additionally, it must be concluded that communication via LUNA network channels at a rate of 250Hz instead of 1kHz introduces a insignificantly small ripple on the energy within the controller over time.

The third and fourth hypothesis describe effects of different implementations on the execution of tasks relevant for teleoperation. The hypotheses were tested by letting ten different users

perform free space positioning and peg-in-hole tasks. The third and fourth hypothesis are respectively formulated as:

"Time delay introduced by network-accessing middleware does not decrease performance and perception of tasks executed via teleoperation systems"

"Performance and perception of task execution on a teleoperation system with LUNA network channels do not decrease compared to a system with ROS network communication."

Answers to the second and third hypothesis are based on the outcome of the user experiments. The user experiments were characterized by relatively large variance, mixing the results of different implementations. Results show that there is either no difference between implementations or the measurements are not precise. With a noninferiority test, statistical significance was assessed. Results of the positioning experiment for the different implementations were accepted to be significantly equal. Significance of results corresponding to the peg-in-hole experiment and overall perception of users could not be guaranteed. Nevertheless, corresponding results suggest approximately equal performance and perception for each implementation.

The results of user experiments do not give a strong answer to the second and third hypothesis, since statistical significance cannot be proven for the peg-in-hole task and perception of users. However, the results suggest equal performance for all implementations, which means that both hypotheses are true.

The integration tests show small effects of delay for implementations with network communication. These effects are not found back in the user experiments. User experiments suggest equal behavior of a system with LUNA network channels with respect to a system with no network communication and a system with ROS network communication. If the answer to the first hypothesis is compared with the answers to the second and third hypotheses, it can be concluded that differences between implementations is small. This is based on the small difference in results in the integration tests and the lack of difference in results of the user experiments.

To answer the main research question, a teleoperation system using LUNA network channels is similar to a teleoperation using ROS network communication on functional level and task performance level. Compared to a teleoperation system without network communication, slight delays and oscillations in position and force signals are introduced on the functional level. Performance and perception of task execution by all implementations of the teleoperation system in this project is suggested to be equal. A note from a timing point of view is that the use of LUNA network channels comes at the cost of significant computational resources. The use of LUNA network channels therefore limits other computational processes, which is not desired for integration of additional features in future projects.

7.2 Recommendations

Recommendations are generated during the course of this project. Recommendations can be made about several aspects: improvements on the work done in this project, improvements for the TERRA and LUNA frameworks, and enhancements of theory for passive teleoperation systems. Recommendations are ordered from high to low priority per category.

Improvements on the work done in this project:

- The user experiments conducted in this work could be improved. First, to make a better characterization of the system for teleoperation, more tasks relevant for teleoperation could be performed. Second, the awareness of the user of slave device environment could be improved by providing a view from cameras at multiple angles. Third, the interaction force was used as performance measure in the peg-in-hole experiment. It could be more interesting to use the interaction energy as a performance measure, based on

force and position measurements. Measuring of energy combines information from position and force signal, which could be a better characterization of interaction. Fourth, experiments with a real robotic setup could be compared with simulation experiments. By reducing the KUKA LWR inertia and friction a more transparent system would be created, such that the characteristics of network communication could be assessed more significantly.

- The inertia of the KUKA LWR was a cause for user feedback forces during free space motion of the slave device. These feedback forces were relatively high and could be reduced by using an inertia compensation controller, resulting in more system transparency. The inertia compensation controller must not compromise passivity of the complete controller, therefore measures must be taken to compensate inertia in a passive way. With a more transparent system, it would be possible to better characterize properties of the LUNA network channel.

- In this design it was chosen to use a TCP communication protocol. TCP provides reliability, such that data is not lost. It would be interesting to investigate the difference with communication via the UDP protocol, where loss of data is possible. Additional bookkeeping would be required to account for energy package duplication and changes in received order (Franken, 2011). With using the UDP protocol and loss of data, the behavior of the system might change. A similar investigation could be done to compare a Wi-Fi connection with the currently used Ethernet connection.

- Once ROS2 is released, it would be an interesting alternative to the implementations in this work. Performance of ROS2 can be compared with the performance of current middleware to check for possible improvements on current implementations, or assess the suitability of ROS2. ROS2 is an interesting alternative since it will use DDS-based communication, which is the same basis as is used for implementation of the LUNA network channels.

- Velocity estimation based on position information was done using state variable filters in this project. The Omega.7 device utilized built-in velocity estimation via a FIR low pass filter. Both velocity estimation methods were suitable for this system. The FIR filter could be an improvement on state variable filters, as performance is similar and computational complexity is lower.

- A communication frequency of 250Hz instead of 1kHz was not a great impact on the system. This could be because the communication frequency was sufficiently higher than the dynamics of the physical system. To reduce computational load, the complete controller could be operating at 250Hz instead of 1kHz. This requires further investigation, as the amount of virtual energy leakage and interaction energy per sample would also increase.

- In this project, the controllers were executed on a PC without real-time operating system. It would be an improvement if a real-time operating system would be used, such that timed controller actions are performed on more deterministic intervals.

Improvements on the TERRA and LUNA frameworks:

- High computational load was introduced by using LUNA network channels. It is recommended to revisit the design and implementation of the network channel. The recommendations made in Wijnholt (2017) should be followed in this process.

- Working with the current LUNA network channels was not user-friendly and should be improved. It is recommended to fully integrate the DDS-based LUNA network channels

in a stable build of TERRA and LUNA. Additionally, the process of setting the correct linking directories for DDS-libraries should be done automatically when makefiles are generated.

- TERRA and LUNA support model-to-model conversion, such that for example models of physical systems can be transformed to code models. Models of physical systems describe computations. In Ellery (2017), a method to structure computational software was introduced. The method of Ellery (2017) could be combined with model-to-model conversion, such that computational models are converted to structured code. An example would be that 20-sim models are converted to general computational components, such that they can be (re-)used by different middleware.

- Currently, only scalar variables are supported by communication channels in TERRA and LUNA. It would be more practical if array and matrix support is provided as well.

Enhancements for the theory of teleoperation systems:

- The passivity layer of the Two-Layered framework contains several limitations for transparency layer output. limitation 2 should be further investigated for systems with multiple degrees of freedom, as current functionality is not performing the intended actions. The sign of interaction energy should be accounted for.

- Due to interaction with the environment, it could be that the energy within the controller is relatively high. This is a risk for temporary active behavior and oscillations. To reduce the energy within the controller, a maximum energy level could be used. If energy within the controller exceeds the maximum allowed energy, a part of the energy could be discarded, such that there is a lower risk of active behavior and oscillations, which prevents possible destruction of the environment or slave device. This principle would lead to more action of the TLC, as the discarded energy must be re-obtained. Additional to the previous idea, the discarded energy could be stored in a separate energy tank at the master side. Later in time when the energy level of the default energy tank is below H_D, energy could be extracted from this tank instead of energy being extracted from the user by the TLC. The rate of energy extraction from this second tank must be limited, because if this is not done the active behavior is not limited.

- One goal of i-Botics is to create intuitive teleoperation systems. In Teeffelen (2018), impedance modulation based on electromyography signals was introduced. Impedance modulation could be based on communication channel properties as well (e.g. delays or loss of data), possibly resulting in a more intuitive or transparent teleoperation system.

- Oscillations were observed in master device position measurements for systems with delay during the functional tests. To obtain a more transparent system, the oscillations could be reduced by adding a damper to the master controller.

A Investigation of the KUKA youBot

This project was originally started with the goal to use the KUKA youBot arm as robotic platform. During preliminary tests, it was found that control torques required to actuate the arm were too high. This resulted in either a controller with high impedance and too large feedback forces or a controller with relatively normal impedance values and almost no movement of the youBot arm. Before this result was obtained, different properties of the youBot were already researched. This appendix is used for all information found in this project regarding the youBot.

A.1 Description of the youBot

The youBot is depicted in figure A.1. The youBot consists of a base and an arm that function separately. The base is actuated by four mecanum wheels, which enable independent linear motion in the x and y axis and rotation around the z axis of the base coordinate frame. The arm consists of six links which are connected with five single degree of freedom (DOF) joints. Each joint is actuated by an electrical motor and has specific limits to its range (limits indicated in figure A.1). A gearbox connects each motor to a joint or wheel. A two-fingered gripper is attached to the end of the arm, it can open and close.

All motors are driven by TMCM motordriver modules (Jasinska, 2015). These modules accept PWM, current, position or velocity setpoint. The module is able to measure current to the motors, position of the motor and velocity of the motor. Output current, motor position or motor velocity are regulated internally to the given setpoint using PID control. Via current control, it is also possible to control the torque in the joints.

Communication with the motor driver modules is done using the EtherCAT protocol, which uses standard Ethernet hardware and can achieve hard real-time requirements (Jansen and Buttner, 2004; Spil, 2016; Wijnholt, 2017). Setpoints can be given to the motordrivers, parameters can be adjusted, measurement data, status or error information can be accessed. The network topology that connects all robot joints consists of one (external) master device and for each robot joint a slave device. The operating principle is that all slave devices can insert or extract relevant information in a chain. The last slave will sent the fully processed data back to the master. Different topologies can be achieved, but must be known by the master on forehand.

A.2 Hardware Choice

A choice must be made regarding the hardware that is used to combine the youBot and an teleoperation system. This mainly considers the computing platform.

Several different computing platforms can be chosen for implementation of control algorithms and connection with the youBot hardware. Options that are discussed here, are a desktop PC, internal youBot PC, RaMstix and Raspberry pi. These are options that are feasible within the RaM laboratories.

- A desktop PC is characterized by relative large amount of processing power. Different operating systems can be installed, resulting in a lot of flexibility and resources. Low level connectivity is often not available, and must be added with expansion boards if desired. A new generation of Mini PC's, offers comparable capabilities as old fashioned PC's, but has the advantage of a small physical size.
- Part of the youBot robot is an internal PC (Jasinska, 2015). This PC is a mini-ITX with a dual core CPU which runs at 1.66GHz, the device has USB and 2x LAN connection.

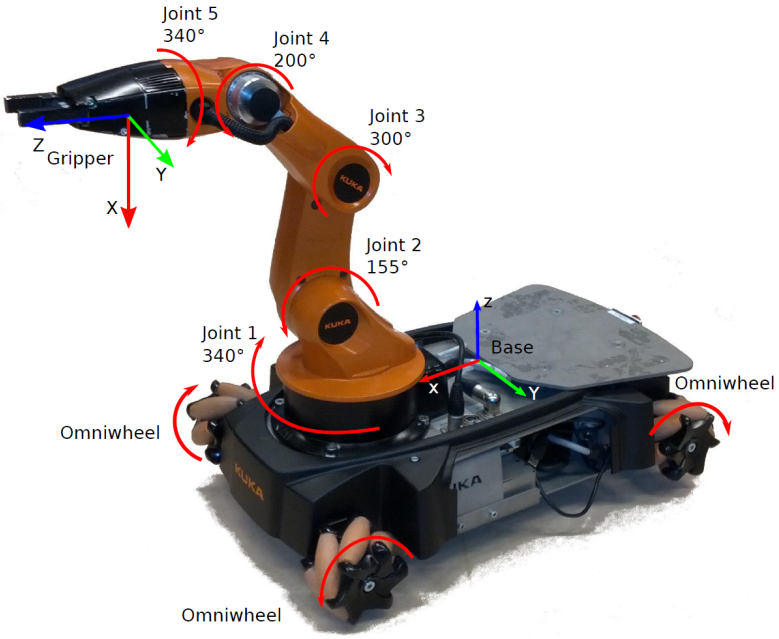


Figure A.1: The youBot robot with arrows that indicate positive directions of individual joints. From Frijnts (2014).

- The RaMstix is an embedded computing platform used by the RaM group at the University of Twente (RaM, 2017). Because this platform is used by the RaM group, a lot of new research is focused on this platform, providing support for many software features. The RaMstix is an expansion board for the Gumstix Overo module (Gumstix, 2018). The Gumstix Overo module is a single core computer which runs at 800MHz and supports WiFi and Bluetooth connections. The Gumstix Overo module runs on Linux with a normal kernel and Xenomai kernel to support hard real-time applications. The RaMstix expansion board provides interfaces for 100Mbit Ethernet, incremental encoder inputs, PWM outputs, digital input/output (I/O), USB, serial, CAN and provides onboard ADC's and DAC's.

- The Raspberry pi is a small single board computer that can run Linux. The latest version has a quad core processor that runs at 1.2GHz (Raspberry Pi Foundation, 2018) and supports digital I/O, PWM output, USB, Ethernet, WiFi and Bluetooth.

Limitations exist on all computing platforms. Most important is that the connections in figure 3.1 can be made. It is assumed that all user interface devices communicate via USB. Communication to the youBot is done using EtherCAT that communicates via Ethernet connection. The LUNA network channel works on Ethernet or WiFi network. It is assumed that WiFi will be used. This means that the master controller platform must require USB and WiFi connection and the slave controller platform must require WiFi and Ethernet.

The platforms that were discussed are compared here, see table A.1. Most important requirements are the Ethernet and WiFi connections, as the LUNA network channel must be tested in this project. Computation speed is relevant, as this must not be a limiting factor for the teleoperation system. Support for LUNA and OpenDDS is required for the implementation. Extending LUNA and OpenDDS with the support for a certain platform will not be done in this project. Size and mobility are important for the computing platform that is placed on the youBot. Low level interfacing could be convenient for testing, as digital I/O ports can be set or cleared after certain events, such that events at different platforms can be compared and timed in measure-

Table A.1: Computing platform comparison.

	PC	Mini-PC	RaMstix	youBot ITX PC	Raspberry Pi 3B	**Weight Master**	**Weight Slave**
Ethenet connection	1	1	1	2	1	1	3
WiFi connection	1	1	1	0	1	3	3
Computation speed	++	++	-	+	+	1	1
LUNA support	+	+	+	NT	NT	3	3
OpenDDS support	+	+	+	+	+	3	3
Size/mobility	–	+	++	+	++	0	1
Low-level interfacing	-	-	+	-	+	1	1
USB	*	*	*	*	*	3	1
Score Master	**15**	**15**	13	9	12		
Score Slave	13	15	**16**	9	14		

* = USB is supported.
NT = Not tested.

ments. USB connectivity enables support for user interface devices and for additional network interfaces, such that platforms that lack the required network adapter, can be complemented.

The scores in table A.1 are subjective and relative with respect to other devices. The youBot ITX PC and Raspberry Pi 3B are disregarded, as they do not score highest on both master and slave devices but offer interesting alternatives. The RaMstix has relative low computational power, but fulfills all other requirements. This is characterized by the highest score for slave device. It is therefore most suitable for usage as slave platform in this project. For the master platform, size and mobility is not important, which means that a PC or Mini-PC score highest in table A.1 and are most suitable. A PC is already installed and configured with the TERRA/LUNA environment and DDS, therefore this option is chosen.

A.3 Hardware Constraints

The youBot software interface gives the option to perform torque (effort-like) control and measure position, but also vice versa is possible. Therefore, the interface to the controller does not have a preferred causality type. Mechanical properties are a relatively high gear ratio and presence of gearbox and joint friction, corresponding to admittance type devices. For control methodologies as discussed in section 3.4.4, impedance type devices are required. External forces other than from the environment (e.g. gravity) and friction are a problem for the backdrivability of the device. A consequence due to these non idealities is less transparency of the overall system.

A.4 Limitations of the youBot

Previous work with the youBot (Corberan Ruiz, 2012; Keiser, 2013; Brodskiy, 2014; Weijers, 2015) has shown the presence of friction in the robot joints. Friction, backlash, saturations and dead zones are not desired, as they are hard to model, reduce the backdrivability and reduce system transparency. Friction is considered to be most influential on the system and could therefore be compensated. Several solutions for friction compensation exist. All of these solutions make a model for friction per joint, on which the compensation is based. Usually friction forces are characterized for several velocities. Friction compensation is performed by adding the modeled friction force to the controller output. This compensation violates the passivity of the controller, as energy is generated by the controller. In the ideal case, controller compensation energy matches the energy dissipated by friction perfectly. In the realistic case, the

modeled friction will deviate from real friction. If the friction model is too conservative, net friction force is still present and the backdrivability and transparency are still limited. If the friction model is overestimating the real friction, the controller generates energy, such that stability through passivity of the controller cannot be guaranteed. Franken et al. (2010) show a method for passive friction compensation with presence of noise for teleoperation systems. The result is a conservative but passive friction compensation algorithm.

Friction compensation is useful for improvement of the overall teleoperation system performance. The goal of this project is not to obtain a system with best possible transparency. The effect of friction would influence the system independent of the implemented network communication. Therefore, friction can be disregarded such that a relatively less transparent system is used for testing of the effects of the network channel on the teleoperation setup.

The setup will be tested on Earth, therefore gravity is acting on the devices. The Omega.7 has built-in gravity compensation, such that the user does not have to move the weight of the device. Gravity compensation could also be added to the youBot. Screw Theory (Stramigioli and Bruyninckx, 2001) provides a solution. Following similar reasoning as with friction compensation, this could be implemented for improvement of the total system transparency and backdrivability. If preliminary design tests show that gravity has a big impact on the system usability, compensation will be implemented.

The angle that can be reached by each joint in the youBot arm is limited. Relative high joint velocities towards a mechanical end-stops are harmful for the device. In previous work (Frijnts, 2014) it was found that gearboxes attached between joint and the motors are the weakest link and break. This must be prevented, therefore Frijnts (2014) designed a safety layer which generates opposing torques with respect to controller output torque when close to an endstop or when velocity becomes too high. This boils down to a position and velocity dependent saturation function for the controller torque output to each joint. Similar safety aspects must be implemented in this project as well to prevent self-destruction of the youBot.

A.5 Software Interface with the youBot

The KUKA youBot is controlled using the youBot interface from Wijnholt (2017). Joint position and velocity are measured using incremental encoders and Hall effect sensors respectively. Position signals in encoder counts are converted to radians. The position offset from measurement values should be matched with the angles known in the controller. Velocity measurements are given in rounds per minute and are converted to radians per second. Both position and velocity signals correspond to the motion of the motor axis and are converted to motion of the joint axis by the gearbox reduction ratio. The youBot interface sends motor current values to the youBot. Joint torques in Newton meter calculated in the controller are mapped to motor current in milliampere using the motor constant and gearbox reduction ratio. Interfacing with the youBot was performed at the same frequency as the controller. A saturation was implemented to limit the maximum joint output torque, to prevent damage and high-velocity behavior.

The software interface was tested by Wijnholt (2017) and Spil (2016), which resulted in observations of latency peaks of $50-400\mu s$ corresponding to a nominal execution time of $150-600\mu s$. The execution time was different for different implementations, which were using LUNA on a RaMstix ($350\mu s$), a C++ application ($150\mu s$ on a laptop, $600\mu s$ on a RaMstix) or 20-sim 4C on a RaMstix ($150\mu s$). Depending on the implementation, periodicity in latency peaks was observed. The cause for periodicity was assumed to be partially in the network stack of the RaMstix, but Wijnholt (2017) discussed that there must also be another cause.

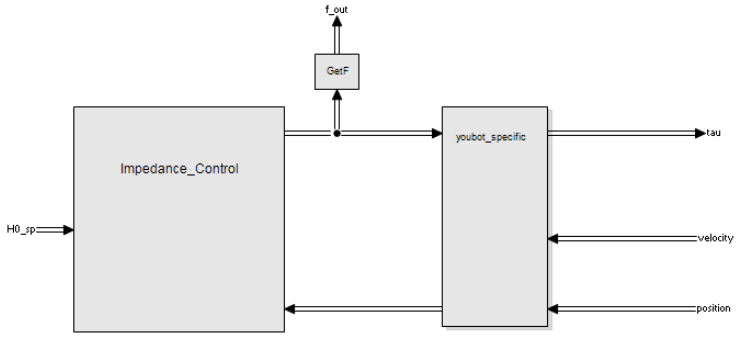


Figure A.2: Controller model.

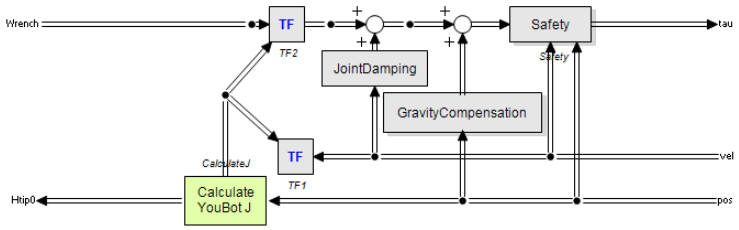


Figure A.3: Submodel of the youBot specific part of the controller.

A.6 Controller Model

The youBot model from Dresscher (2010) and controller from Frijnts (2014) have been implemented in 20-sim. The controller structure is depicted in figures A.2 and A.3. The impedance controller is implemented in a code model, based on section 4.2.1. The youBot specific part includes the calculation of: the H-matrix corresponding to the youBot arm end effector, geometric Jacobian, joint damping, gravity compensation and output limitations for safety.

A.7 Software Architecture

The controller software is made in the LUNA framework and consists of an Omega interface to communicate with the Omega.7 device, a youBot interface to communicate with the youBot and a controller. The structure is depicted in figure A.4. All functionalities are executed sequentially, where a writer is used to communicate with a hardware timer. The hardware timer channel runs with a frequency of 1kHz. The controller is executed in parallel with readers and writers to write and read variables that are accessible for the omega and youBot interfaces. This choice has been made since all processes are strictly placed in sequential order. Practically, this method is less cumbersome, since LUNA readers and writers only support scalar variables. The variables to be sent are the H-matrix corresponding with the setpoint, joint torques, joint positions and joint velocities. The combination of a sequential structure and a controller with both in- and outputs introduces delays into the control system, since not all inputs can be processed to the corresponding outputs in one iteration. This causes the use of values calculated in the previous iteration. The delay is limited to one iteration, introducing only a delay of a single sample time. This is assumed to be negligible.

A.8 Validation - Materials and Methods

An experiment is set up to validate the functioning of an impedance controller working on the youBot arm. The validation is done using a repeatable setpoint path in each direction.

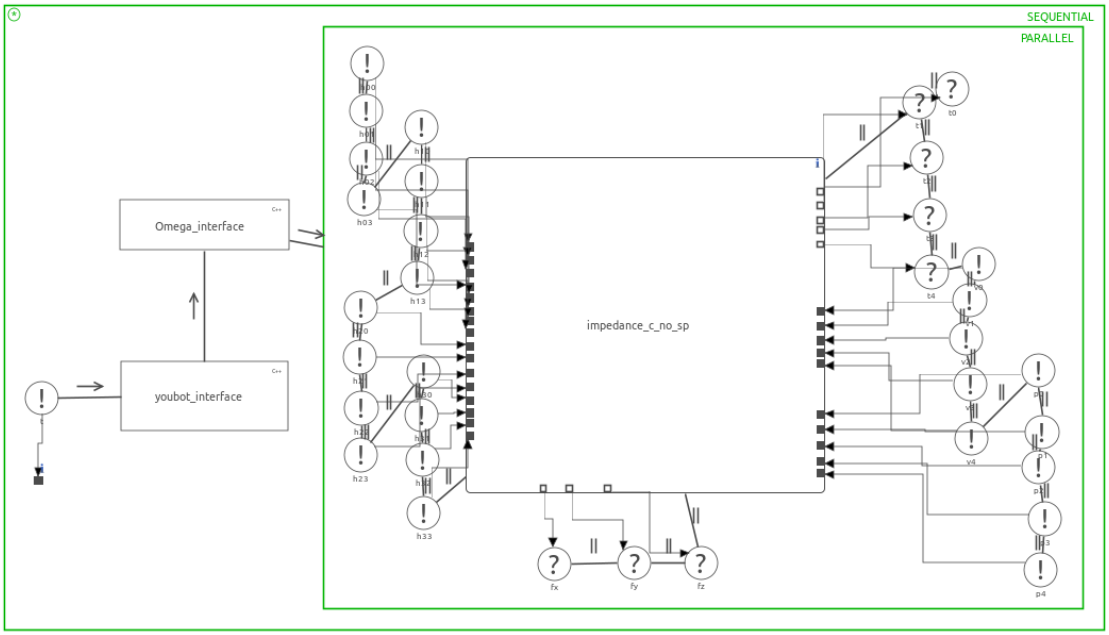


Figure A.4: gCSP software architecture of the impedance controller.

Table A.2: Used spring constants for different measurements.

	Unit	Relative high	Relative medium	Relative low
K_t	[N·m^{-1}]	5000	500	100
K_o	[N·m]	500	50	50
K_c	[N]	0	0	0

The setpoint is one period of a sine wave with an amplitude of 50mm and period of 5s, given in sequence in x-,y- and z-direction. The given setpoint setpoint, calculated position of the youBot arm end effector based on joint position measurement, and translational controller output forces in reference frame are measured. Measurements are done with different controller parameters, depicted in table A.2. Joint damping is set to $D_j = 0.5[\frac{Ns}{m}]$ for all joints and all controllers. The offset H-matrix to which the position setpoint is added, is denoted in equation A.1.

$$H_o = \begin{bmatrix} -1 & 0 & 0 & 0.05 \\ 0 & 1 & 0 & 0.2 \\ 0 & 0 & -1 & 0.1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.1}$$

A.9 Validation - Results and Interpretation

Results of validation are depicted in figures A.5, A.6 and A.7. In the measurements it is shown that the youBot positions and controller output forces are significantly different for different chosen spring constants in the impedance controllers. For the relatively high impedance settings, it is found that the youBot can reasonably follow the position setpoint. However, this comes at a cost of high controller output forces. For the medium and low impedance settings, the error between setpoint and measured position is great. The system shows lack of transparency and backdrivability, as controller output forces do not result in consequent motion of the youBot end effector. Controller output forces for low impedance settings are considered to be in reasonable range for force feedback. The reasonable range for force feedback is consid-

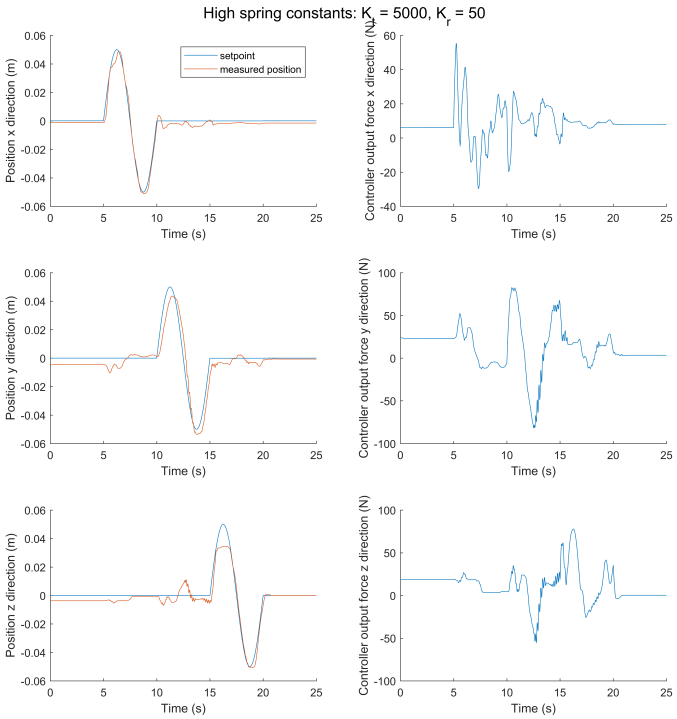


Figure A.5: Commanded motion with a relatively high impedance controller.

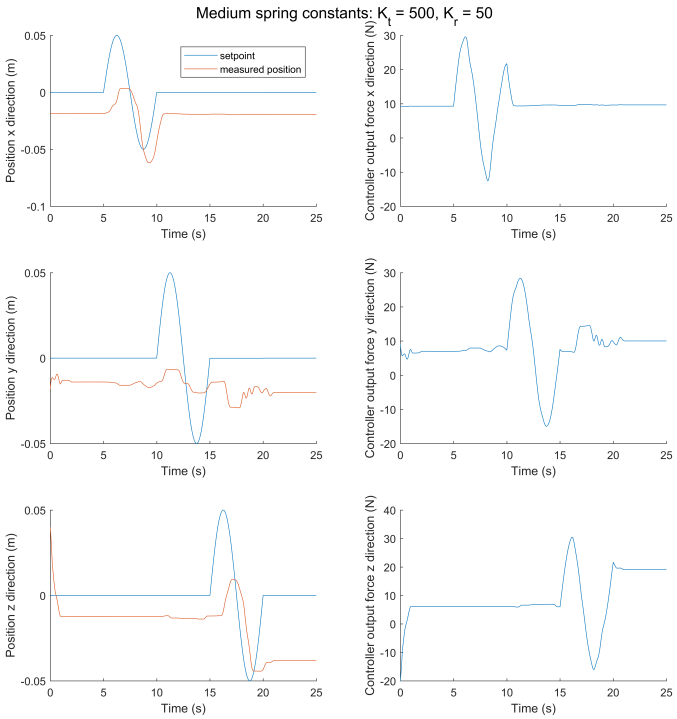


Figure A.6: Commanded motion with a relatively medium impedance controller.

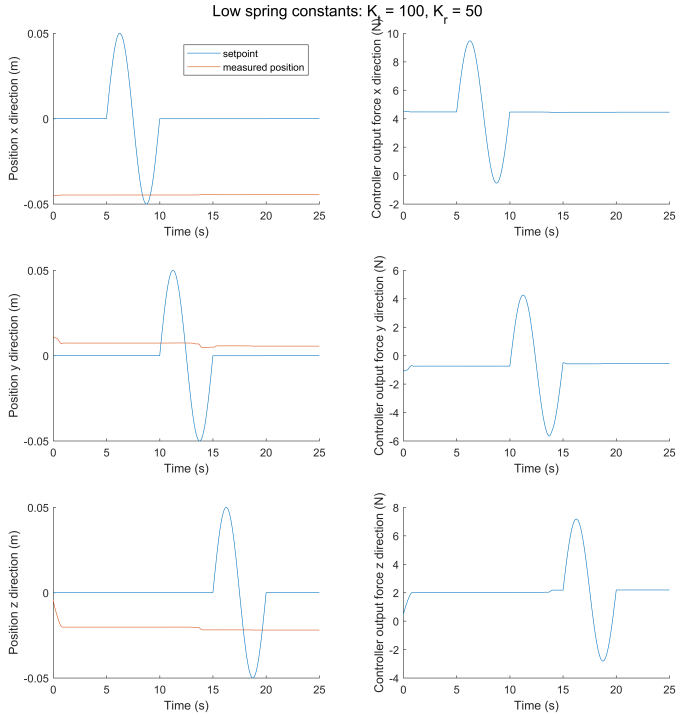


Figure A.7: Commanded motion with a relatively low impedance controller.

ered to be ±12N, as this is also achievable by the Omega.7 device. Reasonable controller output forces do not result in sufficient change in position of the youBot end effector. The cause for this effect is thought to be friction of the joint gears. Additionally, the controller output force in z-direction is constant positive for a constant setpoint (during $0-5$s), as the measured youBot end effector is lower than the setpoint. This could be the result of a slight offset in the gravity compensation algorithm.

A.10 Conclusion

By evaluating the analysis and measurements of the youBot arm, it is found that this arm is not a transparent or backdrivable device. Controller output forces for correct motion of the arm are too high, resulting in enormous force feedback to the user. The youBot arm was only tested for motion in free space. If the youBot arm would be interacting with the environment, controller forces could become even higher. With high controller output forces for motion in free space, it is impossible to distinguish between motion in free space and interaction with the environment. For a device with lack of transparency such as the youBot arm, it is impossible to test the network channel of a teleoperation system without great influence of the device. This result is a basis to choose for other devices when building a teleoperation system.

The main reason for lack of transparency or backdrivability is thought to be friction in gearboxes in the joints. It is therefore recommended to design a friction compensation controller for the youBot arm. Work of Weijers (2015) could be used as a basis for a friction compensation algorithm. If friction can be correctly compensated, the compensated arm could be more transparent or backdrivable.

B KUKA LWR Transparency Tests

It is interesting to review the transparency of the KUKA LWR for comparison with the transparency of the youBot. Therefore the experiment of Appendix A is repeated here for the KUKA LWR.

The controller that is used for the LWR is similar to the controller described in chapter 4, however the passivity layer is omitted and inputs are obtained from a digital setpoint generator instead of from the Omega device. A slight difference with respect to the controller used for the youBot is present, as all controller signals connect within one iteration.

B.1 Materials and Methods

The experiment described in section A.8 is repeated for the LWR. Controller parameters are denoted in table B.1. Rotational impedance settings are not changed, as higher values resulted in oscillatory behavior. Joint damping is set to $D_j = 0.1[\frac{Ns}{m}]$ for all joints and all controllers. The offset H-matrix to which the position setpoint is added, is denoted in equation B.1.

$$H_o = \begin{bmatrix} 0 & 0 & -1 & 0 \\ -1 & 0 & 0 & -0.25 \\ 0 & 1 & 0 & 0.85 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{B.1}$$

B.2 Results and Interpretation

Results are shown in figures B.1, B.2 and B.3. It is observed that command motion using a controller with high impedance results in very good tracking of the setpoint by the LWR end effector. Controller output forces are relatively high compared to lower impedance settings. When this controller is compared to the same controller connected to the youBot, it is observed that output forces are relatively low.

For a controller with medium impedance, the commanded motion is reasonably followed by the LWR end effector, however errors are clearly greater compared to the controller with high impedance. Additionally, controller output forces are lower than output forces generated by the controller with high impedance.

From figure B.2, it can be observed that the same setpoint in different directions causes different magnitudes of controller output. This can be explained by the configuration of the robot. Due to the configuration of a serial robotic arm, motion in a certain direction requires a different way of actuation by the joints in the serial arm.

The controller with low impedance shows poor performance, as the controller output forces are too low to generate the desired motion. This was also the case for the medium and low impedance controller connected to the youBot.

Table B.1: Used spring constants for different measurements.

	Unit	Relative high	Relative medium	Relative low
K_t	[N·m^{-1}]	5000	500	100
K_r	[N·m]	50	50	50
K_c	[N]	0	0	0

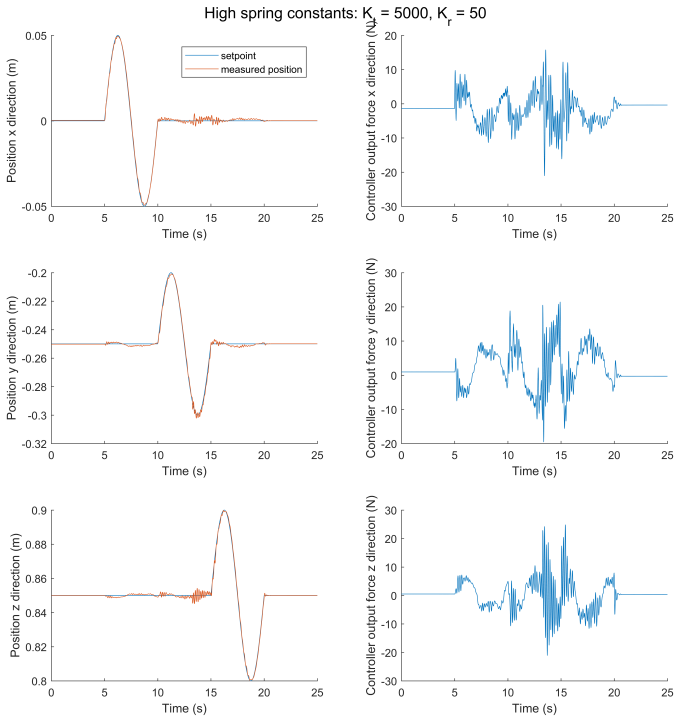


Figure B.1: Commanded motion with a relatively high impedance controller.

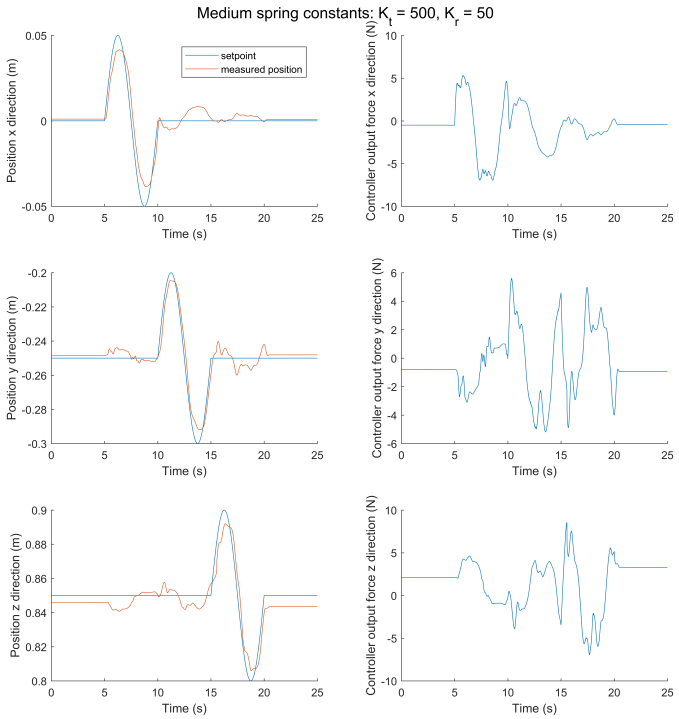


Figure B.2: Commanded motion with a relatively medium impedance controller.

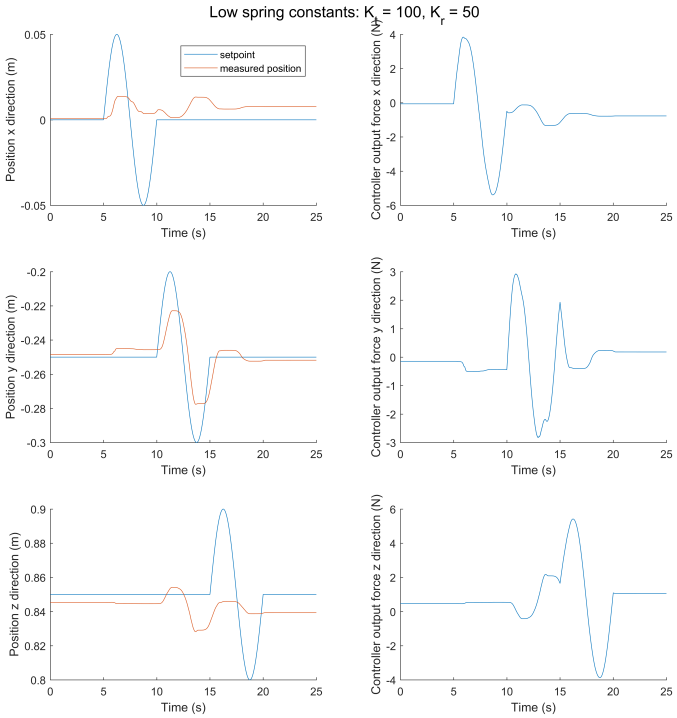


Figure B.3: Commanded motion with a relatively low impedance controller.

C Detailed Controller Calculation Steps

C.1 Summary of Screw Theory

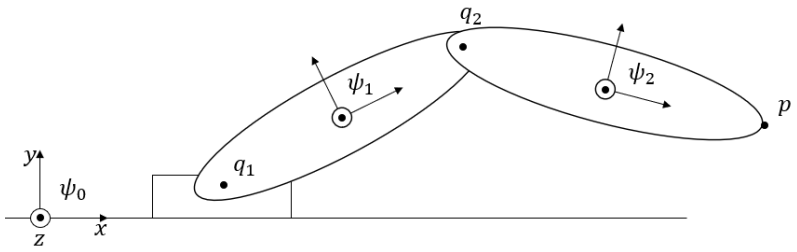


Figure C.1: Schematic representation of a serial robotic chain. Link 1 is connected with joint q_1 to an inertial frame, link 2 is connected with joint 2 to link 1 and point P is the outer end of link 2. Coordinate frame ψ_0 is connected to an inertial frame, frame ψ_1 is connected to link 1 and frame ψ_2 is connected to link 2, all have the positive z-axis towards the reader.

Results from Stramigioli and Bruyninckx (2001) are used for the mapping between joint space and end effector in Cartesian space of a serial robotic arm.

1. Consider a point p, visualized in figure C.1. A point can be quantified in a coordinate frame, for example point p in frame ψ_1 gives p^1. A homogeneous matrix can be used to transform the location of point p expressed in frame ψ_1 to the same location expressed in frame ψ_2: $\hat{p}^2 = H_1^2 \hat{p}^1$, with $\hat{p} = [p^T \quad 1]^T$ in homogeneous coordinates. The homogeneous matrix is composed of a 3×3 rotational matrix R_1^2 and translational vector p_1^2 which describe the relative rotation and translation between the coordinate frames:

$$H_1^2 = \begin{pmatrix} R_1^2 & p_1^2 \\ \mathbf{0}_3 & 1 \end{pmatrix} \tag{C.1}$$

   If a coordinate frame would be connected to each robot link, the points in link n could be expressed in frame 0 by multiplying homogeneous matrices:

$$H_n^0 = H_1^0 H_2^1 ... H_n^{n-1} \tag{C.2}$$

2. A general velocity can be written as a twist in vector form: $T_a^{c,b} = [\omega_x, \omega_y, \omega_z, v_x, v_y, v_z]^T$, which denotes the twist of frame a with respect to frame b expressed in frame c. A twist in vector form $T_a^{a,b}$ could be rewritten to a twist in matrix form, which is defined as:

$$\tilde{T}_a^{a,b} = H_b^a \dot{H}_a^b \tag{C.3}$$

3. In (Stramigioli and Bruyninckx, 2001) it is proven that a mapping from joint velocities $\dot{\mathbf{q}} = [\dot{q}_1, \dot{q}_2, ... \dot{q}_n]^T$ to end effector velocity with respect to inertial frame expressed in inertial frame exists. This is the geometric Jacobian:

$$J(\mathbf{q}) = \begin{bmatrix} T_1^{0,0}(q_1) & T_2^{0,1}(q_2) & ... & T_n^{0,n-1}(q_n) \end{bmatrix} \tag{C.4}$$

   with

$$T_n^{0,n-1} = \mathrm{Ad}_{H_{n-1}^0(q_n)} \hat{T}_n^{n-1,n-1} \tag{C.5}$$

and

$$\mathrm{Ad}_{H_{n-1}^0} = \begin{bmatrix} R_{n-1}^0 & 0 \\ \tilde{p}_{n-1}^0 R_{n-1}^0 & R_{n-1}^0 \end{bmatrix} \tag{C.6}$$

such that

$$T_n^{0,0} = J(\mathbf{q})\dot{\mathbf{q}} \tag{C.7}$$

Here, $T_n^{0,n-1}(q_n)$ is the twist of body n with respect to body $n-1$ expressed in the inertial frame as function of the n-th joint position. The geometric Jacobian in equation C.4 can be built up using equation C.5 and C.6. The $\hat{T}$ in equation C.5 are constant unit twist vectors that describe the direction of individual joint motions. They can have a rotational part with unit magnitude for rotational joint connections or a rotation vector with zero magnitude and a translational part as a unit vector for translational joint connections. With unit twists in matrix form it is also possible to give the current end effector position as function of joint positions:

$$H_n^0(\mathbf{q}) = e^{\tilde{\hat{T}}_1^{0,0} q_1} e^{\tilde{\hat{T}}_2^{0,1} q_2} ... e^{\tilde{\hat{T}}_n^{0,n-1} q_n} H_n^0(0) \tag{C.8}$$

4. If it is assumed that the robotic arm only exchanges energy via the end effector and the joint actuators, power continuity can be used to relate information about general velocities to information about general forces, called wrenches ($W^{ee} = [\tau_x, \tau_y, \tau_z, F_x, F_y, F_z]$):

$$P_{ee} = W^{0,n} T_n^{0,0} = W^{0,n} J(\mathbf{q})\dot{\mathbf{q}} = \left(J^T(\mathbf{q})(W^{0,n})^T\right)^T \dot{\mathbf{q}} = P_{joints} = \boldsymbol{\tau}\dot{\mathbf{q}} \tag{C.9}$$

$\boldsymbol{\tau}$ is a vector that contains the generalized force applied on each joint. Previous equation must hold for any joint velocity, therefore:

$$\boldsymbol{\tau} = J^T(\mathbf{q})(W^{0,n}) \tag{C.10}$$

C.2 Forward Kinematics Calculation Steps

The three functional computations within forward kinematics are denoted below:

1. Calculation of the slave device end effector position and orientation in reference frame: Using screw theory, the homogeneous matrix of the slave device end effector with respect to reference frame can be calculated. In equation C.11, numbers are used to denote the joints of the slave device. k is used to indicate that this calculation must be performed in each controller iteration.

$$H_{ee}^0(k) = H_1^0\big(q_1(k)\big) H_2^1\big(q_2(k)\big) ... H_7^6\big(q_7(k)\big) H_{ee}^7 \tag{C.11}$$

This calculation can also be performed iteratively, to obtain the position and orientation of each joint in reference frame. This is denoted in equation C.12. Here n can be substituted by any joint number of the slave device. This is beneficial for the calculation of the geometric Jacobian for the current device configuration. The geometric Jacobian can be constructed by calculation of C.12 for each joint and substitution in equations C.4, C.5 and C.6.

$$H_n^0(k) = H_{n-1}^0\big(q_1(k), ..., q_{n-1}(k)\big) H_n^{n-1}\big(q_n(k)\big) \tag{C.12}$$

2. Calculation of slave device joint torques corresponding to the calculated end effector wrench. Here $J(k)$ denotes the geometric Jacobian at sample instance k.

$$\boldsymbol{\tau}_J(k) = J(k)^T W^{0,ee}(k) \tag{C.13}$$

3. Calculation of slave device end effector twist in reference frame, with respect to reference frame, based on joint velocities:

$$\boldsymbol{T}_{ee}^{0,0}(k) = J(k)\dot{\mathbf{q}}(k) \tag{C.14}$$

C.3 Construction of the Geometric Jacobian

The construction of the geometric Jacobian is performed with figure 2.2 as reference. In total 9 coordinate frames are used. All coordinate frames are placed along the z-axis, centered in the x-y center of the KUKA LWR arm. All coordinate frames have the same orientation, equal to the orientation of the coordinate frame in figure 2.2. The coordinate frame in figure 2.2 is not used in the calculations. One coordinate frame is used as reference frame and is placed at the base of the arm. Seven coordinate frames are placed in the center of the joints of the KUKA LWR. The last coordinate frame is attached to the end effector. The relative positions of coordinate frames are denoted in equation C.15 and are obtained from the work of Luper (2017). Since all coordinate frames are oriented in the same way, only the relative position difference between coordinate frames are given. Relative H-matrices in reference position could be built up by using a 3-dimensional identity matrix as rotational part and the given positional differences.

$$\begin{gathered} p_1^0 = \begin{bmatrix} 0 \\ 0 \\ 0.102 \end{bmatrix}, p_2^1 = \begin{bmatrix} 0 \\ 0 \\ 0.2084 \end{bmatrix}, p_3^2 = \begin{bmatrix} 0 \\ 0 \\ 0.1914 \end{bmatrix}, p_4^3 = \begin{bmatrix} 0 \\ 0 \\ 0.2084 \end{bmatrix}, p_5^4 = \begin{bmatrix} 0 \\ 0 \\ 0.1914 \end{bmatrix}, \\ p_6^5 = \begin{bmatrix} 0 \\ 0 \\ 0.196 \end{bmatrix}, p_7^6 = \begin{bmatrix} 0 \\ 0 \\ 0.07 \end{bmatrix}, p_{ee}^7 = \begin{bmatrix} 0 \\ 0 \\ 0.03 \end{bmatrix} \end{gathered} \tag{C.15}$$

The KUKA LWR only consists of rotational joints, of which unit rotations are assigned. These are denoted in equation C.16. The direction of rotation for each joint is purely along an axis of the attached coordinate frame. Therefore H-matrices that describe the relative position of robot joints after a joint rotation, have a rotational part corresponding to pure rotation along an axis of the reference frame. The position part of these H-matrices remain as in equation C.15.

$$\omega_1^{0,0} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \omega_2^{1,1} = \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix}, \omega_3^{2,2} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \omega_4^{3,3} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \omega_5^{4,4} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \omega_6^{5,5} = \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix}, \omega_7^{6,6} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{C.16}$$

The unit twist for each robot body is constant and can be generated from the unit rotations and relative positions in tilde from, as is denoted in equation C.17.

$$\hat{T}_n^{n-1,n-1} = \begin{bmatrix} \omega_n^{n-1,n-1} \\ \tilde{p}_n^{n-1}\omega_n^{n-1,n-1} \end{bmatrix} \tag{C.17}$$

The geometric Jacobian is subsequently constructed from all unit twists as denoted in equations C.4, C.5 and C.6.

C.4 SETP calculations

- Receiving of energy packets
  Communication protocols between master and slave can be implemented in several ways (Franken et al., 2011), which could possibly be asynchronous. The received energy from communication with the other side during one sample interval is denoted with $H_{*,+}(k)$ (where $(*)$ can be both master or slave). It corresponds with the sum of all received energy packets $\bar{H}(i)$ during one sample interval:

$$H_{*,+}(k) = \sum_{i \in Q(k)} \bar{H}(i) \tag{C.18}$$

  $Q(k)$ represents the set of all energy packets received by a controller side at sample instant k. At each sample instant k the received set $Q(k)$ is emptied, as the energy is added to the tank. This mechanism supports asynchronous receiving of data, which is important as the communication channel is characterized by variable delays, resulting in a variable amount of energy packets received per sample instant.

- Sending of energy packets
  The Simple Energy Transfer Protocol (SETP) described by Franken et al. (2011) will be considered as an energy communication method. SETP requires both master and slave sides to send a fixed portion ($0 < \beta < 1$) of the tank energy level to the other side. This can only be done if energy is available in the energy tank. It is proven by Franken et al. (2011) that this algorithm will let the tank levels at both sides converge for any communication delay if interaction energy is not present. The sending of energy is executed after the received energy and energy from interaction have changed the tank energy level $H_*(k-1)$. Energy to be sent ($H_{*,-}$) is calculated according to:

$$H_{*,-}(k) = \begin{cases} \beta\big(H_*(k-1) - H_{*,I} + H_{*,+}\big) & \text{if } H_*(k-1) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{C.19}$$

- The tank energy level after all energy exchanges have taken place, can be calculated using:

$$H_*(k) = H_*(k-1) - H_{*,I}(k) + H_{*,+}(k) - H_{*,-}(k) \tag{C.20}$$

D Additional Measurement Results

D.1 Functional Tests

As discussed in chapter 4, an implementation on one PC using ROS was also made in this project (figure 4.11). Results with this implementation do not directly contribute to the project goal, nevertheless results of functional tests with this implementation are considered here. The functional test as described in section 5.1 for this implementation are depicted in figure D.1.

In figure D.1, it is shown that the implementation with ROS middleware on a single PC shows similar properties as the implementation with LUNA on a single PC. The TLC only supplies energy to the controller during startup, subsequently enough energy (tank energy level greater than H_D) remains in both master and slave controllers.

Functional timing tests could also be performed for the implementation on one PC using ROS. Table 6.1 is appended with the implementation of ROS on one PC in table D.1. The controller period and calculation time deviate slightly more than ROS implemented on two PC's. The reason for this is thought to be an increased amount of computational complexity introduced by running both master and slave controller at one PC. Interfacing with the KUKA LWR is found to be stable, which is the case for all implementations.

To give a better view to the timing test, two complete sets of measurement data are shown in figure D.2. In the top plot, the controller period and calculation time of a single LUNA application running both controllers is shown. In the bottom plot, the controller period and calculation time of the slave controller implemented in LUNA on two PC's are shown. Table 6.1 and D.1 consist of data extracted from sequences similar to these. It can be seen that the controller period of LUNA on one PC is much more stable around 1ms than the controller period of LUNA on two PC's. Similarly, calculation time is much lower for one PC than two PC's.

D.2 Tests with Human Subjects

Single runs of both user experiments are depicted in figures D.3 and D.4 to give more insight in the signals of which a performance measure is taken. The setpoint in the position experiment is switched between the two locations on the board (figure 5.1a) in one direction, the measured position of the slave device is measured in the same direction and the difference is taken as error between setpoint and actual position. The sum of absolute force in each direction is used as a performance measure in the peg-in-hole experiment. A threshold value is used to separate time of interaction from time without interaction. The mean of interaction forces during time of interaction is used as a performance measure.

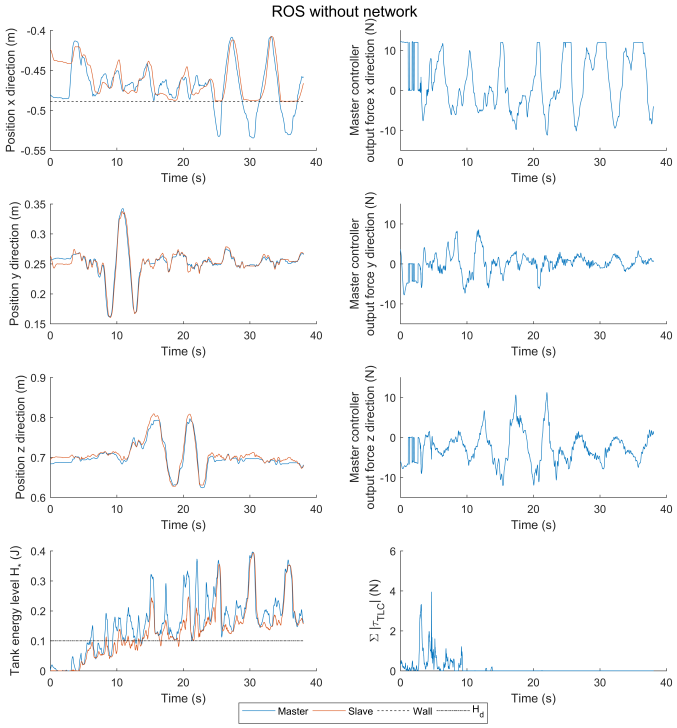


Figure D.1: Results of the functional test with one ROS node without network communication.

Table D.1: Timing test results with controller implementation in a single ROS node included. Values are in milliseconds (ms).

	Controller loop time		Controller calculation time		KUKA LWR interface loop time		Transmit loop time		Receive loop time	
Ideal system	1.00	0.00	0.00	0.00	2.00	0.00	*	0.00	*	0.00
LUNA 1 PC	1.00	0.24	0.41	0.11	2.00	0.20	-	-	-	-
LUNA 2 PC's	1.42	1.53	0.93	1.57	2.01	0.70	4.11	1.38	4.27	1.83
ROS 2 PC's	1.03	0.29	0.57	0.26	2.00	0.04	1.02	0.23	1.00	0.60
ROS 1 PC	1.04	0.33	0.68	0.31	2.00	0.04	-	-	-	-
	Min.	**Max.**	**Min.**	**Max.**	**Min.**	**Max.**	**Min.**	**Max.**	**Min.**	**Max.**
Ideal system	1.00	1.00	0.00	0.00	2.00	2.00	*	*	*	*
LUNA 1 PC	0.49	7.22	0.00	6.52	0.02	6.00	-	-	-	-
LUNA 2 PC's	0.36	18.26	0.00	16.65	0.01	8.28	0.04	12.06	0.03	12.23
ROS 2 PC's	0.40	11.49	0.00	11.24	1.55	2.46	0.40	6.54	0.02	9.71
ROS 1 PC	0.42	9.66	0.00	8.84	1.25	2.75	-	-	-	-

* the designed communication frequency: none, 4.00ms or 1.00ms.

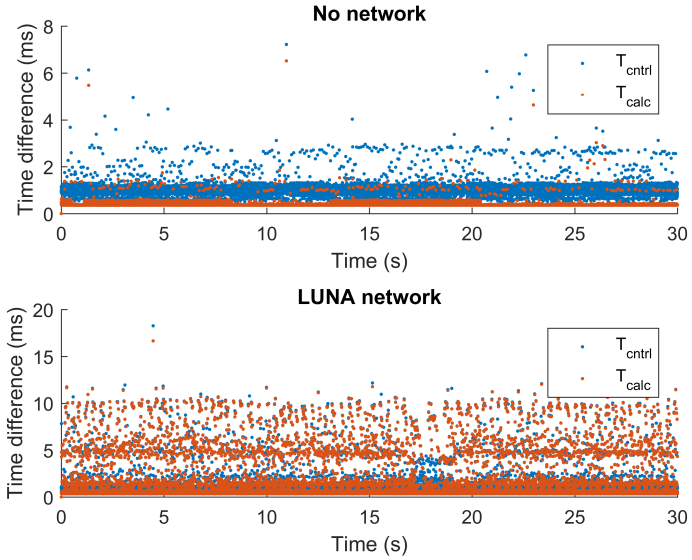


Figure D.2: Controller period and calculation time for 30s for the implementation in LUNA on one PC (top) and the slave controller of the implementation in LUNA on two PC's (bottom).

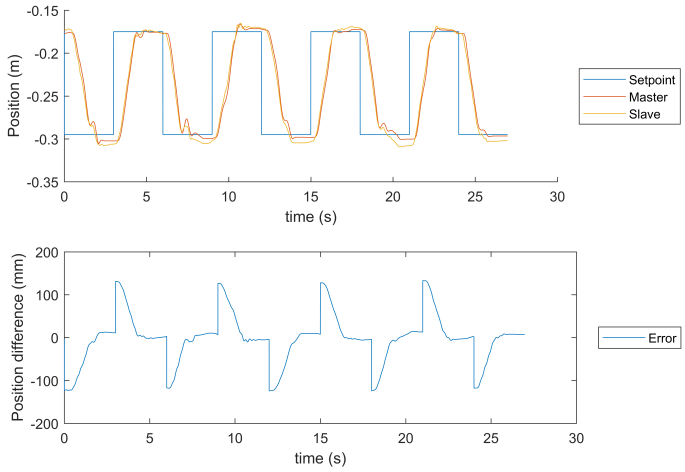


Figure D.3: Results of a single run of the position experiment. The error is derived from setpoint and slave device position in y-direction.

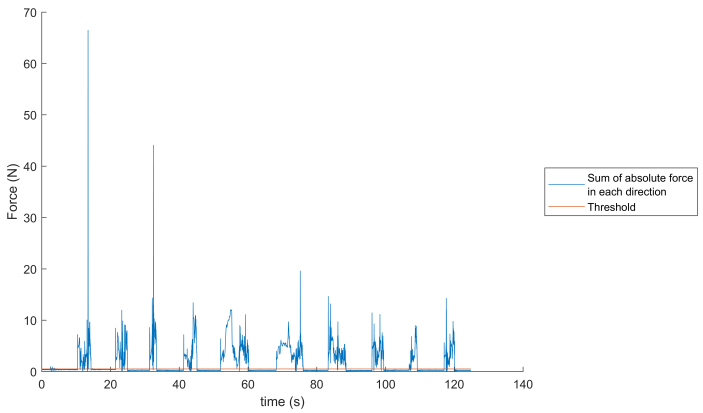


Figure D.4: Results of a single run of the peg-in-hole experiment.

E Instructions and Practical Notes for Software Interfaces and Toolchains

Software in this project is developed for Linux-based systems. This appendix is written to reproduce the results of this project, or to use the results for other projects with the same methodology. In this appendix it is covered how to setup and use TERRA/LUNA with DDS network channels created by Wijnholt (2017), how to embed general components in a software project and how to connect with the youBot, LWR and Omega devices.

```
cd [directory name]          # go to specified directory
ls  [directory name]         # show directory contents
mkdir [directory name]       # create a directory with the specified name
ps aux                       # show all active processes
ps aux | grep [name]         # active processes with the specified name
kill -9 PID                  # kill running process with PID
# kill all running programs with the given name
sudo killall -9 ./[program name]
[tab]                        # use to autocomplete names
./[program name]             # run C/C++ program
# run C/C++ program, store output in textfile.txt
./[program name] > textfile.txt
```

Listing E.1: Convenient Linux command line commands.

E.0.1 Generating Programs from Code

To generate a program from code, code must be compiled. This is done using a compiler (for example GCC), which builds and links relevant code files. Understanding of building and linking code (especially precompiled libraries) is a key concept in the software design of this project. A nice explanation can be found here: https://www3.ntu.edu.sg/home/ehchua/programming/cpp/gcc_make.html.

For small projects, the compiler could be called by the user by a single command line command, for example:

```
gcc code.c -o program_name
```

For larger projects, usually makefiles are used. If LUNA code is generated from the TERRA environment, makefiles are provided for the program structure. Makefiles can be executed by using the following command:

```
make          # compile a program
make clean    # remove results of compilation
```

ROS and general components can be converted from code to programs using CMake. CMake commands are specified in a CMakeLists.txt file and are usually used as:

```
# from code to program
mkdir build && cd build # create and goto build directory
cmake ..          # setup compile environment
make              # compile
# remove results of compilation
make clean
```

E.1 LUNA-DDS Development Environment

The description by Wijnholt (2017) on how to use the LUNA-DDS development environment was found to be practically too concise during this project, therefore it is elaborated here.

E.1.1 Step-by-Step Installation

1. Open a Linux terminal.
2. Select (or make) and goto the desired installation directory.
3. Obtain TERRA, Eclipse LUNA and the LUNA library files:

```
git clone https://git.ram.ewi.utwente.nl/wijnholtr/TERRA
git clone https://git.ram.ewi.utwente.nl/wijnholtr/Eclipse_LUNA
git clone https://git.ram.ewi.utwente.nl/wijnholtr/LUNA
```

4. The LUNA library must be compiled for the device on which it is used. LUNA can be compiled by the commands listed below. In the configuration menu, optional LUNA components can be (de)selected. Make sure that DDS channels are selected in *main menu* > *high-level components* > *link-drivers*. Optionally, facilities for 20-sim generated code or debug information could be enabled as well. Make sure to place the build files in a recognizable directory, for example a "lunabuilds" directory in the previously mentioned installation directory.

```
sudo apt-get install libncurses5-dev
sudo apt-get install svn2cl
cd LUNA/LUNA
make menuconfig      # Configure LUNA
make                 # Build LUNA
```

5. Open a file explorer, go to the Eclipse_LUNA directory and open Eclipse. It could be that JRE is not installed, in that case run the following command:

```
sudo apt-get install default-jre
```

6. In an empty java workspace in Eclipse: File > Import > Existing Projects into Workspace. Select the cloned TERRA project.
7. Select nl.utwente.ce.terra and click green play button (top menu). A new eclipse window should appear. The new eclipse window includes a DDS port in the architecture palette for architecture models.
8. In eclipse go to *window* > *preferences* > *TERRA* > *LUNA*. The binary and header files location should be set here. Also the correct target platform must be selected.
   For Linux select:
   /lunabuilds/luna-linux-x64/lib64
   /lunabuilds/luna-linux-x64/include

   For RaMstix select:
   /lunabuilds/luna-xenomai-arm-v7-Posix/lib
   /lunabuilds/luna-xenomai-arm-v7-Posix/include
9. Install OpenDDS in the main installation directory. OpenDDS 3.11 was used in this project. Follow the instructions from http://opendds.org/quickstart/GettingStartedLinux.html. Make sure that the DDS directory is placed at a convenient location. A precompiled version of OpenDDS for RaMstix can be found at https://git.ram.ewi.utwente.nl/wijnholtr/Demo_source.
10. Go to the directory where the LUNA library is located. Open the tools.mk file and change the absolute path to the appropriate install directory of OpenDDS. Do this for all LUNA library versions.

11. To compile for RaMstix, the appropriate operating system libraries are required. Download the following:

```
git clone https://git.ram.ewi.utwente.nl/wijnholtr/Demo_source
```

    Run the following script:

```
Demo_source/poky-glibc-x86_64-ramstix-20sim-image-cortexa8hf-vfp-
neon-toolchain-1.8.2.sh
```

12. Open the environment-setup-cortexa8hf-vfp-neon-poky-linux-gnueabi and change the absolute paths to the correct directory.

E.1.2 Compiling LUNA-DDS programs

General notes:

- Options can be set on a DDS channel hardware port in the architecture editor in TERRA. The commontopic and owntopic settings must be set to a unique topic name to communicate data between two applications. The config setting must be set to ../dds_config.ini, where dds_config.ini is a file with options for communication. The file must be copied into the main directory of the current TERRA project.

- The TERRA implementation of DDS channels is poor. Therefore an error will be generated when it is tried to compile. This error will be something like: "writer is specified with 7 arguments, but 8 are required" . This is the case for both sender and receiver. The last setting "waitForCompletePacket" has to be set manually to true or false in the C++ code of the architecture model "[architecture].cpp" (set to false in this project). This corresponds with the last argument in the following line in the "[architecture].cpp" file:

```
[port name]_DDSwrtrChannel = new LUNA::CSP::DDS::DDSChannel<double>("commontopic","
    owntopic", "../dds_config.ini", "BESTEFFORT", 1, 1, false,false);
```

- In Wijnholt (2017), the interface between OpenDDS and LUNA has been made. A directory "LUNA_sequence" was delivered, in which libraries are made that do the interfacing. In this directory, the library files can be generated for RaMstix and x64 devices. By going to the directory in a terminal and executing "make" the libraries can be rebuild.

  A copy of each precompiled library (in lib directory) can be placed in each project, or the absolute path of the "LUNA_sequence/lib" directory can be used for linking. The header files are included in the LUNA build, so that the LUNA library knows about the existence.

Notes for compiling for Linux:

- The location of files libidl.a, libpublisher.a and libsubscriber.a must be known in the makefile of the LUNA architecture for linking.

- The makefile of the LUNA architecture should be appended if DDS channels are used in the project:

```
## protected region to add additional statements on begin
DDS_APP = ../lib_pubsub
LDFLAGS+=-L$(DDS_APPS)/lib -L$(ACE_ROOT)/lib -l$(DDS_ROOT)/lib
LIBS+=-lpublisher -lsubscriber -lidl -lTAO_Svc_Utils -lOpenDDS_Tcp -
    lOpenDDS_InfoRepoDiscovery -lOpenDDS_Dcps -lTAO_PI -lTAO_CodecFactory -
    lTAO_PortableServer -lTAO_AnyTypeCode -lTAO -lACE
```

- Some of these libraries are located within the OpenDDS folder. Therefore it is required to do the following command:

```
source .../OpenDDS-3.11/setenv.sh
```

- The project root folder requires a dds_config.ini file.
- If above requirements are met, the project can be compiled by the command:

```
make
```

- If your project uses a timer, then the real-time system library should be added manually in the makefile of the LUNA architecture. This can be done by placing the following line in the protected region:

```
LIBS+=-lrt
```

- If GCC-ar is not found, check /usr/bin for available versions. Command:

```
ls /usr/bin | grep gcc
```

  In this project only gcc-ar-5 was available, which resulted in a compilation error. Changing the build.mk file in /lunabuilds/luna-linux-x64 from gcc-ar to gcc-ar-5 solved this issue.

Notes for compiling for RaMstix:

- Make sure RaMstix is selected as target platform, as well as selecting the appropriate LUNA version within TERRA.
- It is convenient to copy the file envAndXenomaiDir defined in https://git.ram.ewi.utwente.nl/wijnholtr/Demo_source/demo_ramstix into the project directory. The file should be modified, such that the paths in the file direct to the installation directory of the RaMstix libraries.
- The makefile of the LUNA architecture should be appended to make a program suited for execution on the RaMstix:

```
##Under the protected region document description on begin:
SYSROOT=$(SDKTARGETSYSROOT)

##Under the protected region to add additional statements on begin:
LIBS:=$(subst -lm ,,$(LIBS))
LIBS:=$(subst -lpthread,,$(LIBS))
CPPFLAGS +=-I$(SYSROOT)/usr/xenomai/include

#own implimentation libs
DDSLIBS:=-lpublisher -lsubscriber -lidl

#DDS dependency libs
DDSDEPLIBS:=-lTAO_Svc_Utils -lOpenDDS_Tcp -lOpenDDS_InfoRepoDiscovery -
    lOpenDDS_Dcps -lTAO_BiDirGIOP -lTAO_PI -lTAO_CodecFactory -lTAO_PortableServer -
    lTAO_AnyTypeCode -lTAO -lACE

#libary files are located here of DDS.
DDSLDFLAGS:=-L$(DDS_APP)/lib_ram_new -L$(ACE_ROOT)/lib -L$(DDS_ROOT)/lib

#add the header files of DDS
CPPFLAGS+=-I$(ACE_ROOT) -I$(ACE_ROOT)/TAO -I$(DDS_ROOT)
```

```
##Under the protected region additional targets on begin: (contents of lines starting with $ must
    be indended with a single tab)
[name of your ARCHITECTURE]: $(OBJECT FILES)
    $(CXX) -Wl,-Ur -nostdlib $(LDFLAGS) -Wl,@$(SYSROOT)/usr/xenomai/lib/cobalt.wrappers -
      Wl,@$(SYSROOT)/usr/xenomai/lib/modechk.wrappers $(SYSROOT)/usr/xenomai/lib/
      xenomai/bootstrap.o -L$(SYSROOT)/usr/xenomai/lib $^ $(LIBS) -o top_arch_controlloop.
      tmp
    $(CXX) top_arch_controlloop.tmp -Wl,--wrap=main -Wl,--dynamic-list=$(SYSROOT)/usr/
      xenomai/lib/dynlist.ld -L$(SYSROOT)/usr/xenomai/lib -lcobalt -lmodechk -lpthread -lrt -o
      $@
```

- To make a RaMstix program, the following commands must be executed in the project directory:

```
source envAndXenomaiDir
make
```

- If the error below occurs when trying to execute the program, then the LUNA program is set to the wrong priority. Error message:

```
<5> start - couldn't set thread priority (error: 22)
<5> EmergencyManager::emergencyShutdown - There was an EMERGENCY, the system will now
    shutdown.
```

  The solution for this error is to modify the main.cpp file that is automatically generated by the LUNA framework. Add the following line under the protected region manual tree modifications on begin:

```
LUNA::Threading::OSScheduler::Instance()->setSchedulerPolicy(FIFO);
```

E.1.3 Running LUNA-DDS programs

On each device on which it is desired to run a LUNA-DDS program, an OpenDDS folder must be present and installed. Multiple LUNA-DDS programs can be run at a single device if desired. Before running a LUNA-DDS program, the DCPS server must be running on one device in the network. This can be started by the following command:

```
source [path to OpenDDS installation]/setenv.sh
[path to OpenDDS installation]/bin/DCPSInfoRepo -ORBListenEndpoints iiop://:12345
```

Programs can be run by the following commands:

```
source [path to OpenDDS installation]/setenv.sh
./[program_name]
```

E.2 Connecting with General Components

This section is a stepwise description to use general components from i-Botics, in the structure designed in Ellery (2017). Most of these general components perform matrix and vector calculations using the Eigen3 library (Benoît and Guennebaud, 2018). It is assumed that this library is installed on the device on which general components are compiled.

E.2.1 Steps to build General Components

1. Copy the general component from git to a desired directory.

```
git clone git.ram.ewi.utwente.nl/[general component link]
```

2. Go to the general component directory and make a build directory.

```
cd [general component] && mkdir build && cd build
```

3. Make the component.

```
cmake .. && make
```

4. To verify that the component is made, the file [general component].a should be present in the build directory. To verify correct functionality, the component can be tested using the following command:

```
./build/test/tests/testing
```

E.2.2 Including Generalized Components in LUNA Projects

To add a general component to a LUNA project, makefiles must be appended. Makefiles of all higher-level structures must be appended. For this project, there was a LUNA architecture and main CSPm model of which the makefiles must be appended. If multiple levels of CSPm models would be used, more makefiles must be edited. The following lines should be added to the makefile:

```
## protected region to add additional statements on begin
CPPFLAGS+=-I/usr/local/include/eigen3/

LIBS+=-l[generalcomponent]
CPPFLAGS+=-I/.../[general component]/include
LDFLAGS+=-L/.../[general component]/build
```

E.2.3 Including The Generalized Components in ROS Packages

It is assumed that the reader is familiar with creating ROS projects. The general components can be added to a ROS package by adding or appending the following lines in the CMakeLists.txt of that package:

```
# note: ... can be replaced with other entries
include_directories(
 ...
${catkin_INCLUDE_DIRS}
/usr/local/include/eigen3/
 /.../[ general component]/include
)

link_directories(
 ...
 /.../[ general component]/build
)

target_link_libraries(node_name ${catkin_LIBRARIES} ... [general component])
```

E.3 Connecting with Omega Devices

For more information about the Omega and Omega SDK, look at:
http://wiki.i-botics.com/index.php?title=Omega.7
http://wiki.i-botics.com/images/ibotics/e/ea/User_manual_-_omega.x.pdf
http://wiki.i-botics.com/images/ibotics/6/65/User_manual_-_haptic_SDK.pdf

http://wiki.i-botics.com/images/ibotics/4/43/User_manual_-_robotic_SDK.pdf

E.3.1 Installation

1. The SDK can be downloaded from: http://forcedimension.com/download/sdk. Version 3.6.0 is used in this project.

2. Extract the archive and move the directory to a desired location. Run "make" in the SDK folder.

3. The SDK might require additional installation of:

```
sudo apt-get install glut
sudo apt-get install libusb
```

4. To allow the device to be used by any Linux user, the following udev rules are added:

```
sudo nano /etc/udev/rules.d/10-local.rules
```

   Copy the following lines in the file:

```
ATTRS{idProduct}=="0402", ATTRS{idVendor}=="1451", GROUP="users", MODE="666"
ATTRS{idProduct}=="0301", ATTRS{idVendor}=="1451", GROUP="users", MODE="666"
```

   The rules are not always loaded correctly. Therefore the udev rules can be reloaded:

```
Sudo udevadm control --reload-rules && sudo udevadm trigger
```

   Permission to use the Omega device can also be given manually with the command:

```
lsusb # list usb devices and properties
# Now, find the Omega device and the corresponding bus and device numbers
sudo chmod 666 /dev/bus/usb/[bus nr]/[device nr] # give permission
```

E.3.2 Using Omega Devices in LUNA

To access the device, functions specified in the Omega SDK can simply be called. Functions can be found in the manual or in example code. Once a LUNA program is ready to be compiled, the following lines should be added to the makefiles in the folders of the architecture and main CSPm model:

```
## protected region to add additional statements on begin
LIBS += -ldhd -lusb-1.0 -ldl
CPPFLAGS+=-I/.../[SDK directory]/include
LDFLAGS+= -L/.../[SDK directory]/lib
```

Of course, the Omega library should be included in the header file of the class in which SDK functions are called:

```
// protected region additional headers on begin
#include "dhdc.h"
```

If the program is unexpectedly terminated (unexpected for the Omega interface), the output force on the device remains. A clean exit function for the Omega device can be performed by using the following lines:

```
dhdClose (); // terminate Omega device connection
std::exit(1); // terminate program
```

E.4 Connecting with KUKA LWR

For connection with the LWR, the Stanford FRILibrary is used (Stanford University, 2014). The library can be downloaded at: http://cs.stanford.edu/people/tkr/fri/download/fril.zip. The downloaded file can be unzipped and moved to a desired location. Subsequently, the following commands must be used to compile the library:

```
Sudo apt-get install gcc-multilib g++-multilib # if not yet installed
cd ../ FRILibrary/Linux
make clean all
make
```

The program from which FRI library functions are called, is communicating via ethernet with the KRC. The "Alfa" arm at RAM is used, with the KRC configured to IP address: 192.168.42.1. The KRC is configured such that it listens to the IP adress: 192.168.42.40. This must be manually specified on the PC on which the program is executed. A manual connection can be setup using the graphical user interface for network settings (found in the drop down menu of the network icon on the top right of the screen) A manual connection can be setup, which requires the IP address 192.168.42.40, netmask: 255.255.255.0 and gateway: 0.0.0.0.

The program that is interfacing with the KRC must be executed as superuser.

The correct version of the precompiled FRI library must be known by the makefile of the LUNA or ROS project. It can be added in the same way as general component libraries:

```
## protected region to add additional statements on begin
LIBS+=-lFastResearchInterfaceLibrary -ldl -lm
CPPFLAGS+=-I/.../FRILibrary/include
LDFLAGS+=-L/.../FRILibrary/Linux/x64/release/lib
```

E.5 Connecting with KUKA youBot

Connection with the youbot can be setup using the youbot API (found at https://github.com/youbot/) or using the C library designed in Spil (2016). The work of Spil (2016) has been expanded to work from LUNA applications in Wijnholt (2017).

The youBot library is distributed in the folder "youbot_interface", which can be compiled for new devices using the method noted above. Makefiles for the calling program should be appended with the youbot library as well. The makefile for the architecture and the makefile for the main CSPs model should be appended with the following for linux devices (with the correct path set for your machine):

```
## protected region to add additional statements on begin
LIBS+=-lrt -lyoubot -lyoubothelpers

CPPFLAGS +=-I/.../youbot_interface/youbot_functions/include/youbot
CPPFLAGS +=-I/.../youbot_interface/youbot_functions/include/soem
CPPFLAGS +=-I/.../youbot_interface/youbot_helper/include

LDFLAGS+=-L/.../TERRA/youbot_interface/youbot_functions/lib
LDFLAGS+=-L/.../youbot_interface/youbot_helper/lib
```

The youBot library is also precompiled for RaMstix:

```
## protected region to add additional statements on begin
LIBS+= -lyoubot_ramstix -lyoubothelpers_ramstix

CPPFLAGS +=-I/.../youbot_interface/youbot_functions/include/youbot
CPPFLAGS +=-I/.../youbot_interface/youbot_functions/include/soem
CPPFLAGS +=-I/.../youbot_interface/youbot_helper/include
```

```
LDFLAGS+=-L/.../youbot_interface/youbot_functions/lib
LDFLAGS+=-L/.../youbot_interface/youbot_helper/lib
```

Note that the programs communicating with the youBot must be executed as superuser.

E.5.1 Library Function Calls

The following function calls are done to initialize the youBot:

```
initializeEthercat();
initializeBase();
initializeManipulator();
```

After initialization, setpoints can be sent or measurement information can be received. When reading information from the youBot, "custom_pre_io_func()" must be called before the reading function calls. When writing information to the youBot, "custom_post_io_func()" must be called after the writing function calls. After initialization, the information exchange with the youBot must be periodic, with a maximum of 0.4s time interval (Spil, 2016).

As the youBot library is written in C, the library should be included as:

```
extern "C" {
#include <youbot.h>
}
#include "helpers.h"
```

If youBot function calls are implemented as part of a class (which is the case for LUNA projects), it is advised that initialization is not done in the constructor of the class. Construction requires too much time, which results in no communication with the youBot.

E.6 Running ROS Nodes on Multiple Devices

The teleoperation system of this project was able to run on two PC's (master and slave) using ROS. This was achieved by running one ROS core at the slave PC (slave of the teleoperation system). The ROS nodes that communicate with nodes running at the other PC require information, such that communication can be performed correctly. In this project these were the master and slave controller nodes. In each terminal in which a node is executed that is not at the ROS core PC or communicates with the other PC's, the following commands must precede execution:

```
# At PC's not running the ROS core:
# In the terminal of a ROS node that is placed at other PC's:
export ROS_MASTER_URI=http://[IP address of ROS core PC]:11311
export ROS_IP=[IP address of this PC]

# At the PC which runs the ROS core:
# In the terminal of a ROS node that communicates with other PC's:
export ROS_IP=[IP address of this PC]

#Check IP address of a PC:
ifconfig
```

Bibliography

Anderson, R. and M. Spong (1989), Bilateral control of teleoperators with time delay.

Artigas, J., C. Preusche, G. Hirzinger, G. Borghesan and C. Melchiorri (2008), Bilateral energy transfer in delayed teleoperation on the time domain, in *2008 IEEE International Conference on Robotics and Automation*, pp. 671–676.

Benoît, J. and G. Guennebaud (2018), Eigen is a C++ template library for linear algebra, website accessed on: 30-4-2018.
http://eigen.tuxfamily.org/index.php?title=Main_Page

Bezemer, M. M., R. J. W. Wilterdink and J. F. Broenink (2011), LUNA: Hard Real-Time, Multi-Threaded, CSP-Capable Execution Framework, in *Communicating Process Architectures 2011, Limmerick*, volume 68 of *Concurrent System Engineering Series*, IOS Press BV, pp. 157–175.

Bezemer, M. M., R. J. W. Wilterdink and J. F. Broenink (2012), Design and Use of CSP Meta-Model for Embedded Control Software Development, in *Communicating Process Architectures 2012*, volume 69 of *Concurrent System Engineering Series*, Open Channel Publishing, pp. 185–199.

Boessenkool, H., J. G. Wildenbeest, C. J. Heemskerk, M. R. de Baar, M. Steinbuch and D. A. Abbink (2018), A task analysis approach to quantify bottlenecks in task completion time of telemanipulated maintenance, *Fusion Engineering and Design*, **vol. 129**, pp. 300 – 308.

Brodskiy, Y. (2014), *Robust autonomy for interactive robots*, Ph.D. thesis, University of Twente.

Broenink, J. F., P. J. D. Vos, Z. Lu and M. M. Bezemer (2016), A co-design approach for embedded control software of cyber-physical systems, in *2016 11th System of Systems Engineering Conference (SoSE)*, pp. 1–5.

Chen, J., M. Glover, C. Li and C. Yang (2016), Development of a user experience enhanced teleoperation approach, in *2016 International Conference on Advanced Robotics and Mechatronics (ICARM)*, pp. 171–177.

Colgate, J. E., P. E. Grafing, M. C. Stanley and G. Schenkel (1993), Implementation of stiff virtual walls in force-reflecting interfaces, in *Proceedings of IEEE Virtual Reality Annual International Symposium*, pp. 202–208.

Colgate, J. E. and G. Schenkel (1994), Passivity of a class of sampled-data systems: application to haptic interfaces, in *American Control Conference, 1994*, volume 3, pp. 3236–3240 vol.3.

Corberan Ruiz, M. (2012), Haptic teleoperation of the youbot with friction compensation for the base.

Dresscher, D. (2010), Robust autonomy for the youBot.

Ellery, D. (2017), Writing reusable code for robotics.

ForceDimension (2018), Force Dimension Omega.6, website accessed on: 19-01-2018.
http://www.forcedimension.com/products/omega-7/overview

Franken, M. (2011), Control of Haptic Interaction: An energy-based approach, phD thesis.

Franken, M., S. Misra and S. Stramigioli (2010), Friction compensation in energy-based bilateral telemanipulation, in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5264–5269.

Franken, M., S. Misra and S. Stramigioli (2012), Stability of position-based bilateral telemanipulation systems by damping injection, in *2012 IEEE International Conference on Robotics and Automation*.

Franken, M., S. Stramigioli, S. Misra, C. Secchi and A. Macchelli (2011), Bilateral Telemanipulation With Time Delays: A Two-Layer Approach Combining Passivity and Transparency, *IEEE Transactions on Robotics*, **vol. 27**, pp. 741–756.

Franken, M. C. J., S. Stramigioli, R. Reilink, C. Secchi and A. Macchelli (2009), Bridging the gap between passivity and transparency, in *Robotics: Science and Systems V*, Robotics: Science and Systems V, Seattle, USA, p. 36.

Frijnts, S. (2014), Upgrading the Safety Layer and Demo of the youBot Robot.

Gumstix (2018), Overo FireSTORM-P COM, website accessed on: 18-01-2018. https://store.gumstix.com/coms/overo-coms/overo-firestorm-p-com.html

Hannaford, B. and J.-H. Ryu (2002), Time-domain passivity control of haptic interfaces, *IEEE Transactions on Robotics and Automation*, **vol. 18**, pp. 1–10.

Hashtrudi-Zaad, K. and S. E. Salcudean (2001), Analysis of Control Architectures for Teleoperation Systems with Impedance/Admittance Master and Slave Manipulators, *The International Journal of Robotics Research*, **vol. 20**, pp. 419–445.

Hewitt, J. (2005), OSI RM model.

Hoare, C. (1985), Communicating Sequential Processes.

Hogan, N. (1989), Controlling impedance at the man/machine interface, in *Proceedings, 1989 International Conference on Robotics and Automation*, volume 3, pp. 1626–1631.

Jansen, D. and H. Buttner (2004), Real-time ethernet the EtherCAT solution, *Computing Control Engineering Journal*, **vol. 15**, pp. 16–21.

Jasinska, F. (2015), YouBot Detailed Specifications, website accessed on: 18-01-2018. http://www.youbot-store.com/wiki/index.php?title=YouBot_Detailed_Specifications&action=history

Jovanovic, D., B. Orlic, G. Liet and J. F. Broenink (2004), gCSP: A Graphical Tool for Designing CSP Systems, in *Communicating Process Architectures 2004*, volume 62 of *Concurrent Systems Engineering Series*, pp. 233–252.

Keiser, B. (2013), Torque control of a KUKA youBot Arm.

KUKA Laboratories GmbH (2012), KUKA LWR Operating and Programming Instructions.

KUKA Roboter GmbH (2012), Lightweight Robot 4+ specification.

LAAS-CNRS/ONERA (2010), KUKA LWR arm actuator, website accessed on: 19-4-2018. http://www.openrobots.org/morse/doc/0.3/user/actuators/kuka.html

Lammers, B. (2017), Design and realisation of a haptic interface between a ReFlex TakkTile and an Omega 7 Haptic Device.

Lawrence, D. A. (1992), Stability and transparency in bilateral teleoperation, in *Proceedings of the 31st IEEE Conference on Decision and Control*, volume 3, pp. 2649–2655.

Lee, D. and K. Huang (2010), Passive-Set-Position-Modulation Framework for Interactive Robotic Systems, *IEEE Transactions on Robotics*, **vol. 26**, pp. 354–369.

Luper, E. (2017), Component-Based Modelling and Simulation of a KUKA LWR+4 arm.

Mersha, A. Y., S. Stramigioli and R. Carloni (2014), On Bilateral Teleoperation of Aerial Robots, *IEEE Transactions on Robotics*, **vol. 30**, pp. 258–274.

Niemeyer, G. and J.-J. E. Slotine (2004), Telemanipulation with time delays.

Ott, C., R. Mukherjee and Y. Nakamura (2010), Unified Impedance and Admittance Control, in *2010 IEEE International Conference on Robotics and Automation*, pp. 554–561.

RaM (2017), RaMstix FPGA Board Documentation: RaMstix Overview, website accessed on: 18-01-2018.
https://www.ram.ewi.utwente.nl/ECSSoftware/RaMstix/docs/index.html

Raspberry Pi Foundation (2018), Raspberry Pi 3 Model B, website accessed on: 19-01-2018.
https://www.raspberrypi.org/products/raspberry-pi-3-model-b

Ryu, J.-H., J. Artigas and C. Preusche (2010), A passive bilateral control scheme for a teleoperator with time-varying communication delay, *Mechatronics*, **vol. 20**, pp. 812 – 823, special Issue on Design and Control Methodologies in Telerobotics.

Ryu, J. H. and C. Preusche (2007), Stable Bilateral Control of Teleoperators Under Time-varying Communication Delay: Time Domain Passivity Approach, in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pp. 3508–3513.

Schaft, v. d. A. (1999), L2-Gain and Passivity in Nonlinear Control.

Seo, C., J. Kim, J.-P. Kim, J. H. Yoon and J. Ryu (2008), Stable bilateral teleoperation using the energy-bounding algorithm: Basic idea and feasibility tests, in *2008 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, pp. 335–340.

Spil, T. A. (2016), User-input device based command and control of the youBot using a RaMstix embedded board.

Stanford University (2014), Fast Research Interface Library, website accessed on: 16-4-2018.
https://cs.stanford.edu/people/tkr/fri/html/

Stramigioli, S. (1998), From Differentiable Manifolds to Interactive Robot Control, phD thesis.

Stramigioli, S. and H. Bruyninckx (2001), Geometry and Screw Theory for Robotics (lecture notes).

Streiner, D. L. (2003), Unicorns Do Exist: A Tutorial on "Proving" the Null Hypothesis, *The Canadian Journal of Psychiatry*, **vol. 48**, pp. 756–761.

Tadele, T. S., T. J. A. de Vries and S. Stramigioli (2014), Combining energy and power based safety metrics in controller design for domestic robots, in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1209–1214.

Teeffelen, v. K. (2018), Intuitive Impedance Modulation in Haptic Control using Electromyography.

van de Ridder, L. (2018), Improvements to a tool-chain for model-driven design of Embedded Control Software.

Walker, D. S., R. P. Wilson and G. Niemeyer (2010), User-controlled variable impedance teleoperation, in *2010 IEEE International Conference on Robotics and Automation*, pp. 5352–5357.

Weijers, F. (2015), Experimental evaluation of a safety aware impedance controller design.

Werff, W. v. d. (2016), Connecting ROS to the LUNA embedded real-time framework.

Wijnholt, R. (2017), Design of a real-time network channel in LUNA.

Wilterdink, R. (2011), Design of a hard real-time, multi-threaded and CSP-capable execution framework.

Woodall, W. (2018), ROS on DDS, website accessed on: 01-02-2018.
http://design.ros2.org/articles/ros_on_dds.html