Master-Slave Teleoperation with Force Feedback in Hazardous Environment

Zsolt Pasztori

Advisor: Raul M. Prades
Co-Advisor: Pedro J. Sanz

Universitat Jaume I
EMARO++

In partial fulfillment of the requirements for the degree of

Master of Science

September 3, 2018

Acknowledgements

I would like to thank the staff of the RobinLab in Castellon and the Robotics Group in CERN, without whom this thesis could not have been possible. I would like to especially thank prof. Angel P. del Pobil, prof. Enric Cervera, prof. Raul Marin Prades for helping me organize this cooperation.

I want to thank Mario Di Castro and Giacomo Lunghi, who were always helpful during my visit to CERN.

I want to acknowledge my supervisors Raul Marin Prades and Pedro Sanz for their guidance during the writing of my thesis.

Finally, I would like to thank the organization and the students of the EMARO+ program for giving me an opportunity to learn robotics in a very friendly environment. Express my thanks to Gabor Vanyi who told me about the EMARO program, and Dr. Petra Aradi and Dr. Janos Hamar who helped me with my application.

Yours, Zsolt Pasztori

Abstract

Intelligent and precise robotic systems are becoming essential for operating in harsh environments. In order to increase safety and machine availability, robots can perform repetitive, unplanned and dangerous tasks, which humans either prefer to avoid or are unable to carry out due to hazards, size constraints, or the extreme environments in which they take place.

A novel user-friendly system consisting of robotic arms that can be operated by using other robotic arms as controllers, is presented in my thesis. The system is equipped with force-feedback based haptic devices and uses high-precision gravity compensation algorithms to increase the stability during the robotic operations.

The force-feedback is realized through the utilization of impedance control and to ensure the safety of the operator, the force is exerted through virtual springs. Experiments to tune the force-feedback to the operator have been implemented, as well as custom end-effector, designed specifically for teleoperation.

Contents

List of Figures

1

Introduction

> *Robot, any automatically operated machine that replaces human effort, though it may not resemble human beings in appearance or perform functions in a humanlike manner.*
>
> – Encyclopdia Britannica

1.1 On The Importance of Teleoperation

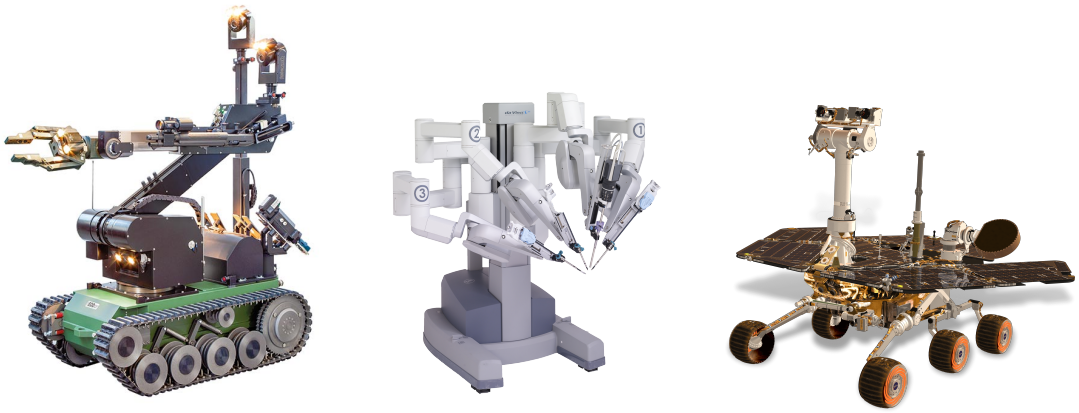
Nowadays robots are starting the come out of the ordered, rigid environment of factories and find their place in close proximity to humans. They take pictures while flying, drive around on our roads and look for exotic fishes at the bottom of the ocean. Robotic bodies are already capable enough to swim, fly and most recently there have been breakthroughs in the development of bipedal robots. Robotic intelligence is also developing rapidly, but today robots are still mostly limited by their cognitive capabilities. For a few decades there have been robotic manipulators, which can pick and place objects, but they can only do these tasks in highly ordered environments because of their lack of the simplest form of intelligence.

Luckily there is a way to overcome these limitations. Instead of putting the intelligent processor on the robot we can simply amend it with human teleoperation. Teleoperation is a form of remote control. It is the lowest level of supervisory control interaction. During teleoperation the robot's movements are determined not just by its environment and inner state, but rather the human operator. Generally teleoperation is used only in the most critical parts of control, such as manipulation of hazardous objects. For example navigation can be performed automatically by the robot at the same time (26).

Teleoperation has several engineering challenges. The operator needs to learn how to control the robot. Fast and reliable connection must be provided with the

robot, which can limit the range of teleoperation. Teleoperation and automation is not direct opposites. It can be as simple as directly controlling each motor of a robot, or simply giving the robots higher level commands. It can also be used as a stepping stone in a road towards fully automating a task. The human input and the robot output can be logged and later used for machine learning and automation of the process (14). This area of machine learning is called imitation learning.

Teleoperated robots are used in a wide variety of fields even today. They are mostly used in scenarios where the environment is too dangerous for humans, or simply inaccessible. They are used by the military for reconnaissance with drones, and bomb disposal. They are used while handling radioactive samples in laboratories and nuclear power plants. All the space missions since the Apollo program were done trough the usage of teleoperated robots. Teleoperated robots took over space exploration, because they can operate for decades without rest, without the need for resupplying .The space crafts do not have to be equipped with life support systems which makes them cheaper, and also do not endanger the lives of the crew. They are also used in robotic surgery, here the biggest advantage is their higher precision and ability to access smaller places and perform more refined motions then a human hand can ever be capable of. The most widespread teleoperated robots are used as toys. RC racing cars can be thought of as the predecessors of teleoperated robots, and also drones, such as quadcopters fit into this group.



(a) tEODor EOD Robot (b) daVinci Surgical System (c) Mars Rover

Figure 1.1: Examples of teleoperated robots

The human interface can be as simple as a keyboard, mouse for input; and a computer screen for output. A bit more ergonomic option is to use joysticks or console controllers. These methods are relatively cheap and can be used to

control different robots. The main drawback of them is that the operator has to learn which action corresponds to which input in the robot. For example pushing the button K will land the airplane. Since there is no similarity between the movement of the input device and robot operators can easily make mistakes. On the other hand the screen as the single output device also poses limitations. The operator might not be able to perfectly judge the situation and capabilities of the robot. From a screen it is hard to know how much more torque can the drone output, in order to determine how hard is it to fly against the wind. Similarly on an underwater robot, even if the pressure is highlighted as number, it is hard for the operator to judge how much deeper it can descend.

The above mentioned limitations can be eased if the controller and output device is structurally similar to the actual robot. During master-slave control the controller itself is also a robot. It does not have to be the same model, for example the controller can be a simplified, smaller, cheaper version of the actual working robot. Master-slave control is more intuitive, the operator needs a fraction of time for getting familiar with it, and also makes it easy to judge for example the joint limits of the robot. The biggest drawback of master-slave control is its expensiveness, since to control a robotic hand the user needs to buy another one. Also master-slave control is much more specialized control, the controller robot can not be changed as easily as in case of a joystick.

1.2 Problem Description

My master thesis was carried out in a cooperation with the European Organization for Nuclear Research (CERN). CERN was founded on 29 September 1954. Its main site is located in the suburbs of Geneva, Switzerland. It has more than 2500 employees at the location, but serves the requests of more than 12000 users. The organization's aim is to provide infrastructure to particle physics researchers. Several discoveries were performed in the institution, the most recent being the proof of the existence of Higgs boson. CERN has also provided break troughs in real world applications, such as medical robotics for radio therapy and 3D scanning of interior body organs.

To be able to further particle physics it is not enough to provide the state-of-the-art infrastructure, it is necessary to look further and develop some of the necessary technologies in place. Several technologies were pioneered in CERN, the most famous being the World Wide Web. Additionally to particle physics the organization carries out research in data storage and processing, superconductivity, medical imaging technologies and more recently robotics.

The European Organization for Nuclear Research, the Nuclear word in its name refers to the fact that the organization looks into the atom, the nucleus.

CERN does not operate a nuclear reactor, and it stores a negligible amount of fissile material on its premises. However some areas are radioactively contaminated. There are two sources of radiation, which can be encountered. The active source is the ionizing beam of accelerated particles. The beam is only dangerous when it comes to direct contact with materials. For this reason any maintenance on the equipment is possible only when the beam is switched off. When the beam collides with the equipment of the accelerator it can activate its materials, these materials can emit low amount of radiation even when the beam is switched off. The most radioactive parts of the accelerator are collimators, beam dump, since these come into direct contact with the beam. The dust and water surrounding the equipment can also be irradiated. Even without considering radioactivity many equipment is dangerous to carry out maintenance on. For example because of their size, weight and strong magnetic field the accelerator magnets are not safe to maintain by human personnel.

The CERN robotics group was created in order to make it safer to carry out operations at location which might be radioactive. These operations may include visual surveillance, measurement of radioactivity and performing manipulation. The group has built long term robotic systems for operating reliably in radioactive environment. It also gives assistance to maintaining equipment which might be dangerous because of radiation. Since every maintenance mission is unique, they had to develop modular a robotic system, which can be adopted quickly to different tasks. My work focuses on the development of a part of this modular system. I have developed a master-slave controller in order to control the robotic arm on-top of this robotic system.

Because each operation is unique, the staff of the Robotics Group is very likely unfamiliar with the equipment needing maintenance. Usually the people who know what to do perform the maintenance are the technicians. Conducting repairs on super-conductive magnets, or other specialized cooling equipment is a very complex task. The modular robots are controlled right now with buttons and joysticks. To learn how to control the robot requires a lot of time. In order to perform a repair, first the technicians have to explain the sequence of actions the robot operator has to perform, but there is always a chance that a complication is encountered during the intervention. This makes each operation really slow and stressful for the operator. With the help of the master-slave controlling the robots became much easier. Anyone can operate the robots with a few minutes of training. Thanks to this, technicians become able to directly conduct the operations, and the robotic staff only have to supervise them.

(a) Large Hadron Collider Tunnel (b) Beam Collimator in an Accelerator

Figure 1.2: Typical locations of interventions

The usage of master-slave teleoperation also provides much higher precision. With the help of force-feedback the operator can immediately feel if the robot arm made contact with an object, and can assess the weight of objects and the force exerted by it on the manipulator. This latter allows him to avoid damaging both the robot and the environment. With a joystick only one direction of movement can be controlled at the same time. This makes it slow and imprecise to move the robot arm. On the other hand with master-slave the movement is possible to all directions at the same time, and the speed can be changed fluently.

The robots have to be very reliable, since it is really hard to recover or fix them if they break down during an intervention, because usually the operation location cannot be accessed by people. They also have to be relatively fast, since operations have to be carried out when the particle accelerator is not working, in order to minimize the time while the accelerator have to stand down. They have to be cheap and easy to use, to minimize both personal and equipment cost. In CERN safety of the employees is a major concern. While developing my master thesis I had to pay special attention to rule out any possibility of injury for the user of the master-slave system.

During the operations there are two main difficulties. Many of the equipments are hard to reach, since the particle accelerator is a narrow space. It is also hard to navigate the robot. Caution has to be taken not to damage any equipment with the robotic hand. In the concrete tunnels teleoperation is hard, because the wireless communication is very constrained. The bandwidth is limited because there is no aerial 4G connection. There is no GPS signal. There can be high amount of packet drops because of the magnetic field of the accelerator.

1.3 Hardware Setup

To be able to easily adjust the robotic system to the changing needs of each operation a modular robotic platform was needed. For this reason the Robotics group has developed their own customizable robot base, the CERNBot. The CERNBot is a mobile robot, which moves with the help of 4 servo motors rotating 4 omni directional wheels. It has 3 degree of freedom (DOF) movement on planar surfaces. This makes it a perfect fit for navigating in constrained places. A crane can be mounted on top of it. With the crane it is possible to move vertically and reach objects higher then ground level, up to 2 meters. Finally, on top of the crane 2 robotic arms can be fitted for manipulation. The components of CERNBot (10) are mostly open source. The robot is controlled by an Intel NUC which is capable of running real-time operating systems. The Robotics group has developed the CERNRobotics Framework, it is a robotic operating system, such as ROS. The framework is written in C++. It has modules for controlling robots and manipulators, for sensory acquisition, communication interfaces and robotic vision.

Figure 1.3: Different configurations of the CERNBot

The CERNBot is teleoperated, and the control input is provided by the operator through the CERN Robotic Graphical User Interface (20). The main design principle of the Robotic GUI was modularity. It can be configured to be used with different manipulators and input devices. The output interface is a video feed, which is acquired from a movable IP camera situated on top of the robot. It can be easily installed to any computer running Microsoft Windows. The operator can carry a laptop or tablet to the location of the intervention and navigate

it with either keyboard-mouse or joystick. Thanks to my work it is now also possible to control the robotic arm from the laboratory with a master arm.

For manipulation one or two robotic arms can be fitted on top of the CERN-Bot. The choice of robotic arm is constrained by its size,weight and power consumption. Most industrial arms are built for factories, they are big and they are plated with metal, which makes them too heavy to be carried by the CERN-Bot. Usually robotic arms are not constrained by their power consumption, but CERNBot draws its power from batteries, so the less power the manipulator needs the more time can be spent on operation without recharging the robot. An additional requirement is for the robot to be freely programmable, and not need specialized controller, so the control logic of the base and the robotic arm can be handled by the same computer on-board the platform. Unfortunately, there is only a few possible candidates on the market. The possibilities are the Kinova Jaco and the Schunk Powerball.

(a) Schunk Powerball Arm (b) Kinova Jaco (c) Kuka iiwa LBR

Figure 1.4: Robotic arms of the project

Kinova Jaco is ultra lightweight robotic arm. It is capable of 6 DOF movement. It weighs only 4.4 kg and can lift 2.6 kg. It has absolute encoders in each joint, so the movement of joints are not limited by joint limits. Each joint also has built in torque sensors. The company has a very open approach towards researchers and users of their products. The robotic arm can be easily programmed through C++, Python or ROS. The disadvantage of this robot arm is that in most situations its payload is simply not enough for the operation. It can only be used for handling tools such as screwdrivers. Additionally one of the links in the arm meets in 45 degrees with the next joints instead of 90. This unusual link configuration makes it harder to map from Joint to Cartesian space. The Kinova Jaco

has a very limited choice of end-effectors, only 2 types of grippers are available as of the writing of my thesis.

For my master thesis I chose to use the Schunk Powerball robotic arm as the manipulator on top of the CERNBot. This is the slave part of the master-slave teleoperation. It has 6 DOF. Three balls contain 2 motors each. This construction makes the robot very space efficient and robust. It has low power consumption, and operates at 24V DC. The joints operate with joint limits of +-160 degrees, and they do not contain torque sensors. The robot is able to exert forces well over 100 N. The manipulator has several extensions which can be purchased from Schunk Ag. It can be fitted with force-torque sensors and there are dozens of different end-effector types available. It is possible to control it through CAN commands, so the programming is possible in C++. The main reason I chose it as my platform was that it is able to lift much heavier payload, and its mechanical setup is much more traditional. The laboratory has already had several force-torque sensors and grippers for this robotic arm.

The master robot has to satisfy very different needs. Since the master-slave teleoperation will be carried out from the laboratory the robotic arm is stationary. The size, the power consumption and even the payload is not a limiting factor. The arm will be hand-guided. Hand-guidance can be done through impedance control. In this control mode the robotic arm senses the forces exerted by the operator on its joints or end-effector. It complies with the forces and moves even with the slightest push. In order to perform teleoperation the most important is its precise movement and high precision impedance control with high speed. There are only a few robots available today which have sufficiently advanced impedance control. We chose the Kuka iiwa LBR (13) robot as the master controller.

The iiwa LBR is the first series produced sensitive robot. It was developed to work in a human-robot cooperation. The impedance control mode was created so the human workers who do their job in the same workspace are always in safety. The robot has built-in high precision force-torque sensors. When it senses that outside forces are exerted on the joints, such as during a collision, it can stop and move along them. The iiwa LBR has the same precision and durability as any other manufacturing robot. It has a hard real-time controller which can accept commands with a 20 ms delay (3) . It has an impedance mode with which the robot end effector can be hand guided, this mode already includes gravity compensation. The end-effector can be modified with the flange, which can be calibrated according to the load. The robot can be programmed only through the Kuka SunriseOS. This is a closed source programming Api, which can be accessed in Java. Unfortunately, it is lacking in its documentation, most of the functionality I used in my project is undocumented. For this reason programming it is very hard, each function has to be tested independently, and finding the right

functionality can be done only by looking through the function names. Several goals are simply impossible to achieve, because there are no available API calls for them. For example measuring directly the velocity, or acceleration is not possible. Also the timing of the calls is not traceable, for this reason a sequence of operation can have very surprising outcomes.

1.4 Outline of the Project

During my master thesis work I have developed the master-slave teleoperation system.

I have integrated 6 hardware components:

1. Schunk Powerball robotic arm as slave
2. Kuka iiwa LBR robotic arm as master
3. Schunk FTM115 force-troque sensor
4. Robotiq 2F-140 gripper
5. DS18B20 temperature sensor
6. specialized end-effector for controlling the master

I have written code in 4 different frameworks:

1. Code of the slave robot was written in C++ for the CERN robotic Framework
2. Code of the master robot was written in Java in SunriseOS
3. The code of the temperature sensor was written in C for Arduino
4. The data exploration and analysis was done in python in Jupyter notebooks

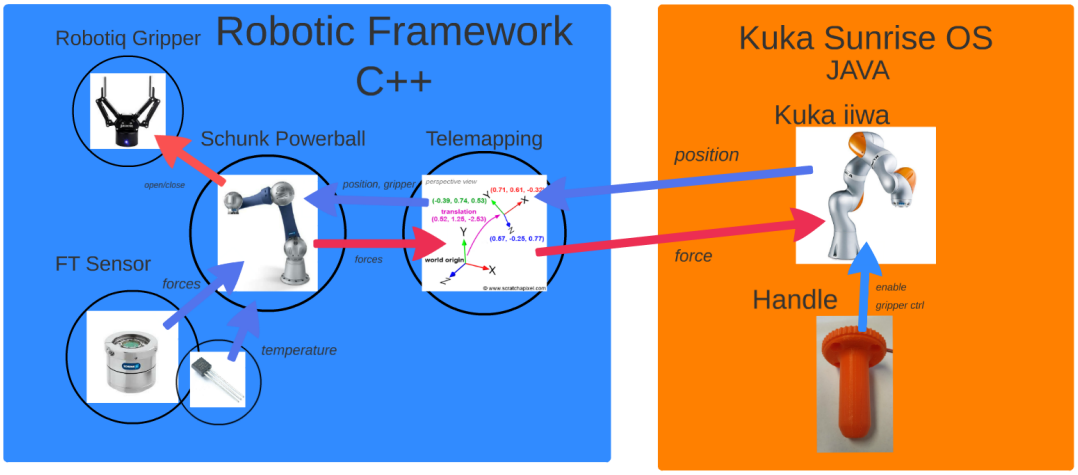


Figure 1.5: The CERNBot with different robotic arm setups

Chapter 2 describes how the movement between the two robots is mapped. In this chapter a position controller for the Schunk Powerball is illustrated. Chapter 3 describes the measurement of the force-torques on the slave robot. It gives details on dynamic filtering of measurements. Introduces a robust method for bias and gravity compensation. Chapter 4 demonstrates how hand guidance and the force-feedback is realized on the master robot. A novel end-effector specifically designed for teleoperation is also introduced. Chapter 5 evaluates the results of the project and outlines the possibilities for future work.

2

Robot Movement Planning and Execution

2.1 Telemapping Between Robot Arms

The most important component of a master-slave controller is the mapping between the movement of the two robots. The mapping should be done between the end-effectors of the two robots. When the end of the master moves the slave should mirror its movement.

Judging the velocity of a moving object from a 2D video feed is really hard. Our perception of it depends on the camera position and in some cases the camera might be moving at the same time. It is hard from a programming point of view also, since it is not possible to directly measure the speed of the master robot. The Kuka SunriseOS API does not provide a way to measure it. Most importantly when the connection is not perfect, packet drops and jitter can be expected. The control of the robotic arm based on velocity becomes dangerous. If for example the connection is dropped suddenly, the robot might keep following the last command and damage itself. This is why I decided to map the positions instead of the velocities of the two end-effectors.

Preferably the mapping should be able to generalize to any choice of the master-slave robot. This is needed in order to make the system more flexible by making it possible to exchange which robotic hands are used for the master or the slave. In the ideal case the system should work in case the operator uses a virtual reality headset instead a monitor for visual interface. In this case the operator does not see the master robot in front, but is rather put into the viewpoint of the slave robot for more precise manipulation. If on the headset only the end-effector of the master can be seen, the operator should still be able to accomplish his task.

Robot movement can be parametrized in either Joint (configuration) or Cartesian (task) space. I have implemented the mapping both joint-by-joint, and based on inverse kinematics. On the master side both the Joint and the Cartesian positions are readily available. On the slave side however only the joint positions can be accessed directly.

2.2 Mapping based on Joint positions

If we use the same robot on the master and the slave side during teleoperation the telemapping becomes very easy. The position of the robot joint have to be simply mirrored between the robots.

During my master thesis project however I used two different robotic arms. The slave robot has 6-DOF and 3 links. The master robot has 7-DOF and 6 links. Although at first glance the robot arms look very different there exist an approximate mapping between them. The Joint 3 of the Kuka robot has to be restricted. After this it can be seen that its structure almost has the same proportions as the Schunk Powerball. The mapping can be seen on Figure 2.1.

This mapping is a very simple and fast solution, and in practice it works very well too. With this solution the problems of inverse kinematics can be avoided. On the other hand the positions of the two end-effectors will only be similar, not the same. It is also impossible to generalize the mapping to other robotic arms. If later the Robotics Group wanted to use a master or slave with a different physical structure, they would have to rethink the whole project.

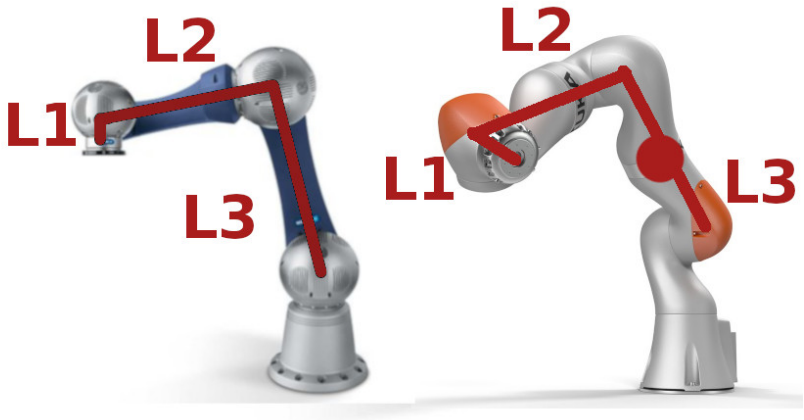


Figure 2.1: The 3 corresponding links between the robots
The red dot is placed on the blocked joint

2.3 Mapping based on Inverse Kinematics

In order to map the position of the two end-effectors, first I had to make it possible to locate the end-effector of the Schunk Powerball in the Cartesian coordinate system. The tool for this is called forward kinematics. Forward kinematics calculates the movement and position of the end-effector based on the knowledge of the joint positions. In case of serial robotic chain the forward kinematics can be calculated recursively given the knowledge of the kinematic chain and the joint positions. The mapping is linear, and can be done by multiplying the transformation matrices of the links.

The forward kinematics problem:

$$\boldsymbol{x} = f(\boldsymbol{q}) \tag{2.1}$$

- $\boldsymbol{x} \epsilon \mathbb{R}^n$ the pose of the end-effector
- $f \epsilon \mathbb{R}^m \rightarrow \mathbb{R}^n$ the linear mapping between joint and Cartesian space
- $\boldsymbol{q} \epsilon \mathbb{R}^m$ the joint coordinate vector

In case of serial kinematic chain and rotational joints the solution becomes:

$$^0T_n = \prod_{i=1}^{N} {}^{i-1}T_i(q_i) \tag{2.2}$$

- $^0T_n \epsilon \mathbb{R}^{4\times4}$ transformation matrix between the base and end-effector frames
- $^{i-1}T_i(q_i) \epsilon \mathbb{R}^{4\times4}$ transformation matrix between the frames of the i-1 and the i-th joint coordinate systems

From the master the slave robot receives the end-effector coordinates in Cartesian frame. From these Cartesian coordinates the joint positions of the slave robot have to be calculated. This is an inverse problem called inverse kinematics. It means that we try to infer the parameters of a function given a solution. It is a mathematical toolkit for finding the corresponding Joint coordinates for a given Cartesian frame. There exists inverse kinematics for velocity and position. For this application the position inverse kinematics of the Schunk Powerball arm was needed to be solved.

The inverse kinematics problem:

$$\boldsymbol{q} = f^{-1}(\boldsymbol{x}) \tag{2.3}$$

- $x \epsilon \mathbb{R}^n$ the pose of the end-effector
- $f^{-1} \epsilon \mathbb{R}^n \rightarrow \mathbb{R}^m$ the nonlinear mapping between Cartesian and joint space
- $q \epsilon \mathbb{R}^m$ the joint coordinate vector

The equation 2.3 can be:

- $n > rank(f^{-1}(x))$ overdetermined system, there exists no solution
- $n = rank(f^{-1}(x))$ the problem is well-posed
- $n < rank(f^{-1}(x))$ under determined system, there exists infinite number of solutions

The Kuka iiwa LBR robot has 7 joints, which means that m > n in general configuration. It is a redundant robot and generally can reach any pose with an infinite number of joint positions. However the degrees of freedom can reduce when two or more joints are aligned around a common axis, this is called a singular position.

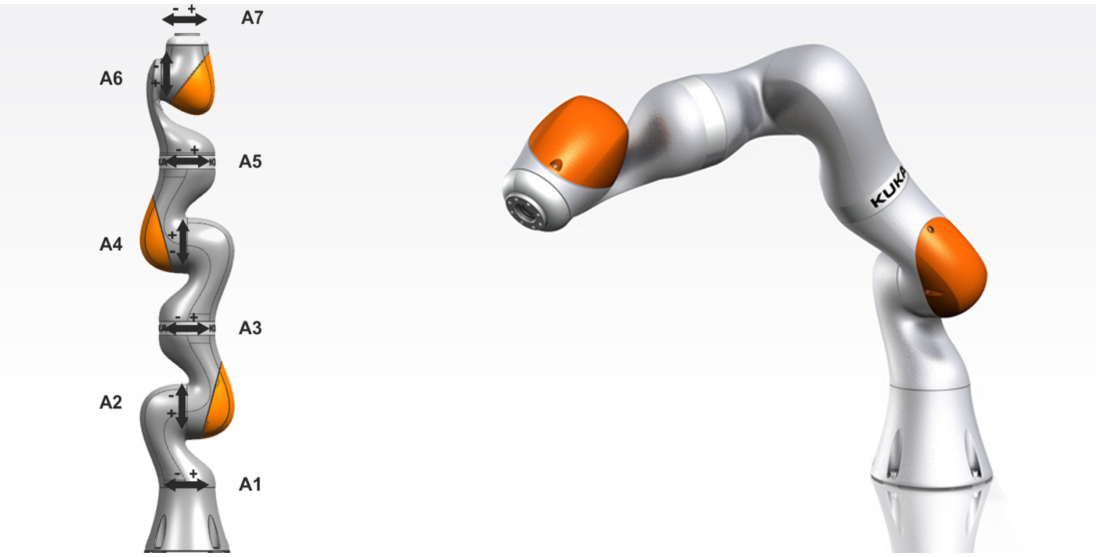


Figure 2.2: The Kuka iiwa is a redundant robotic manipulator

The Schunk Powerball robot has 6 joints. Generally there exists 2 solutions for each pose. This makes the solutions slightly easier. On the other hand the solutions have problems in singular positions and close to joint limits more often. This comes from the fact that if even 2 joints are aligned along a common axis, then the system becomes under determined, since m < n .

2.4 Robot description

In order to be able to solve the inverse kinematics problem the model of the robot needs to be specified. There are many different ways of describing a robotic arm. The most general way is using its CAD model, for example as a STEP file. CAD models however do not include the kinematics of the robot arm. There are 2 main formats which use both the CAD model and add kinematic and dynamic properties to it, these are URDF and XACRO. However to efficiently describe the movement of kinematic chains, a simplified numerical form such as Denavit-Hartenberg parameters (1), is enough.

The Denavit-Hartenberg parameters create a convention for attaching coordinate frames to the joints of kinematic chains. The convention was created by Jacques Denavit and Richard Hartenberg in 1955. The big advantage of the method is that the kinematic chain can be formulated joint-by-joint sequentially, which makes the formulation rather simple. This also enables the description of unusual kinematic chains, such as trees.

To place the reference frames, these instructions need to be followed:

1. In case of rotational joints, the z axis is the axis of rotation of the joint.
2. The X-axis is parallel to the common normal. If the two Z axis are parallel, there exists no unique common normal. In this case this becomes a free parameter . In case of the first joint usually the X-axis is specified by the manufacturer.
3. The Y-axis is oriented according to the right hand rule with respect to the Z and X axis.

Each coordinate system can be described by the 4 DH parameters:

- a length of the common normal
- α angle about common normal from old to new Z axis
- d offset along previous Z axis to the common normal
- θ angle from previous X to new X about previous Z axis

The DH parameters of the Schunk Powerball can be seen in Table 2.1. On Figure 2.3 the orientation and position of the Coordinate frames can be examined.

Link	a	α	d	θ_{offset}
1	0	$-\frac{\pi}{2}$	0.205m	0
2	0.350 m	π	0	$-\frac{\pi}{2}$
3	0	$-\frac{\pi}{2}$	0	$-\frac{\pi}{2}$
4	0	$\frac{\pi}{2}$	0.305m	0
5	0	$-\frac{\pi}{2}$	0	0
6	0	0	0.075m	0

Table 2.1: Schunk DH parameters (8)

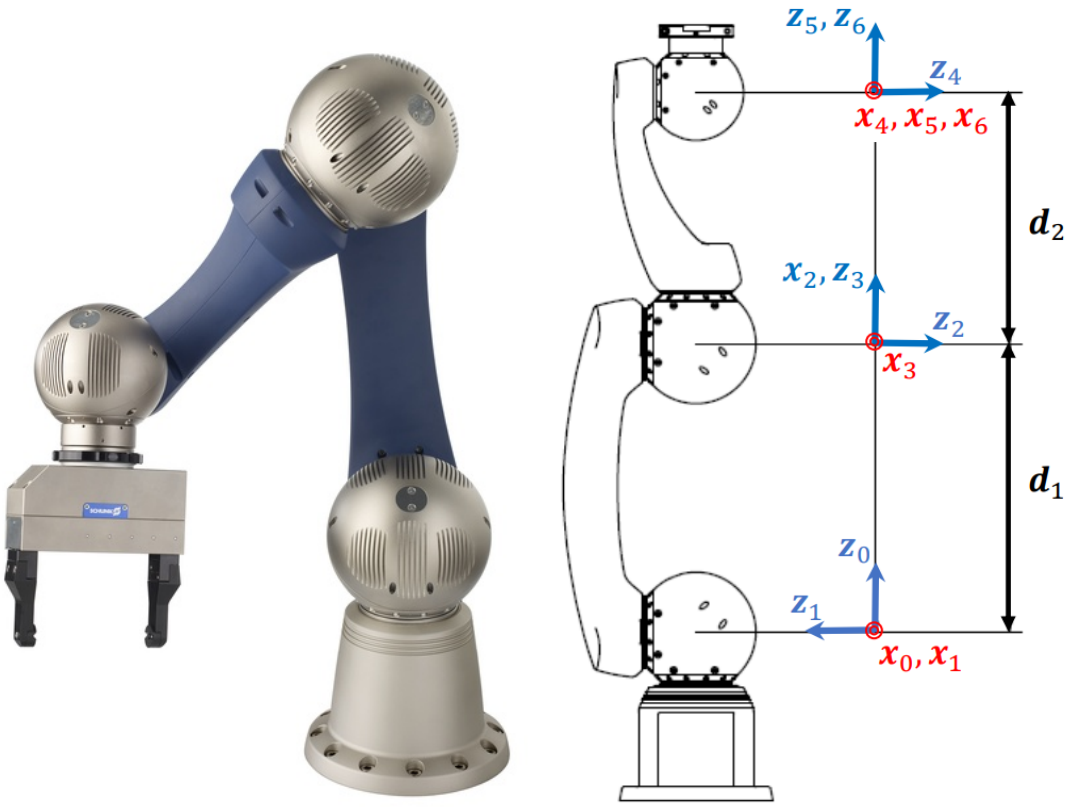


Figure 2.3: The Schunk Powerball robot and the attached Joint Coordinate Frames (12)

2.5 Solving the Inverse Kinematics Problem

2.5.1 Analytic and Numeric Solvers

Given the model of the kinematic chain there are several methods to solve the inverse kinematics problem. The two categories of solutions are analytic closed form and iterative numerical solutions.

Since the Schunk Powerball has 6-DOF there exists an analytic solution for the problem. The analytical solutions are generally fast. It can take as short as 5 microseconds for a full solution, compared to the inverse kinematic solvers for which the solve time is > 200 microseconds. Its precision is limited by the precision of the model and the basic assumptions. Theoretically it can achieve precision 5 magnitude higher then the robot positioning precision. On the other hand many times the assumptions, such as some geometric constraints, do not hold in a given configuration. This can result in huge error in end-effector position. The other main drawback of the analytical solution is the fact, that it is specific to the robot architecture, and it is not possible to change it on the go. Changing the kinematic chain can happen during operation, if a different end-effector is used, or a force-torque sensor is added between the gripper and the final joint. Also the orientation of the frames can easily change when the robot is not mounted on its standard position, but rather on a wall. The analytic solution can be formulated by hand on paper, or with the help of Openrave IKFast (11).

Numerical solutions come with the advantage of easy reconfigurability. In analytic solutions the kinematic chain and the solver is included in the same mathematic model. In numerical models the kinematic chain and the solver can be defined separately and they are only combined in runtime. For redundant chains an analytical solution is generally not feasible. On the other hand here are several methods for dealing with redundant robots trough numerical inverse kinematics. However they provide, at best, only approximate solutions. Usually inverse kinematics solvers do not provide convergence criteria, and might not return with any meaningful solutions. I will describe two numerical methods in detail: Newton-Rhapson and Levenberg-Marquardt method.

2.5.2 The Newton-Raphson Method

The Newton-Rhapson method is one of the oldest root finding methods. It is also called the Jacobian Inverse Method in robotics. In general the convergence of the method is quadratic. The basic notion is to start at a random initial point. Calculate the derivative of the function in the given point. The method approximates the derivative of the function with a line at the point. Calculate the root of the derivative. The root will be the new approximation of the solution.

One iteration of the algorithm can be seen on Figure 2.4. The method is especially good for solving linear least-square problems.

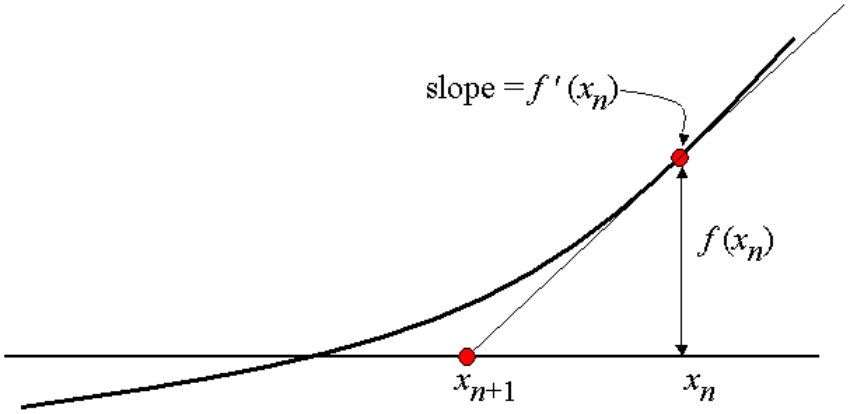


Figure 2.4: The Newton Method

$$\boldsymbol{x_d} - f(\boldsymbol{q}) = \boldsymbol{0} \tag{2.4}$$

We are looking for the root of the equation 2.4. We can make the Taylor expansion of $\boldsymbol{x_d}$ around our initial guess $\boldsymbol{q^i}$.

$$\boldsymbol{x_d} = f(\boldsymbol{q}^*) = f(\boldsymbol{q}^i) + \underbrace{\frac{\partial f}{\partial \boldsymbol{q}}|_{\boldsymbol{q}^i}}_{J(\boldsymbol{q}^i)} \underbrace{(\boldsymbol{q}^* - \boldsymbol{q}^i)}_{\Delta \boldsymbol{q}} + \text{higher order terms} \tag{2.5}$$

We assume that the higher order terms are small, so they can be neglected.

$$\boldsymbol{x_d} - f(\boldsymbol{q}^i) + J(\boldsymbol{q}^i)\Delta \boldsymbol{q} \tag{2.6}$$

$J(\boldsymbol{q^i})$ is called the Jacobian matrix. In multivariate case it is equal to the first order derivatives of a vector function. In scalar case however it is simply the gradient of the function. The Jacobian can be calculated by finite difference method or analytically.

Reorder the equation the vector of the next step becomes:

$$J^{-1}(\boldsymbol{q}^i)(\boldsymbol{x_d} - f(\boldsymbol{q}^i)) = \Delta \boldsymbol{q} \tag{2.7}$$

Instead of the inverse of the Jacobian we utilize the pseudo inverse. The pseudo inverse is the generalization of the inverse, and we can obtain it through Singular Value Decomposition (SVD).

In conclusion the algorithm becomes (21):

1. Calculate the error, given by:

$$\boldsymbol{e} = \boldsymbol{x_d} - \underbrace{f(\boldsymbol{q}^i)}_{\text{forward position kinematics}} \tag{2.8}$$

$$if \ \|\boldsymbol{e}\| < \epsilon \text{ then } \boldsymbol{q}^* = \boldsymbol{q}^i \tag{2.9}$$

2. Calculate the next approximation of $\boldsymbol{q}$:

$$\boldsymbol{q}^{i+1} = \boldsymbol{q}^i + \underbrace{J^\dagger(\boldsymbol{q}^i)\boldsymbol{e}}_{\text{inverse velocity kinematics}} \tag{2.10}$$

3. Repeat Step 1.

The algorithm has a very nice interpretation in robotics. The error can be considered the velocity in task space which needs to be carried out for unit time to get from the current end-effector position to the desired position. To calculate the error we can use the forward kinematics model. To calculate the velocity we can use inverse velocity kinematics.

The algorithm has convergence problems close to singular positions. In this case the Jacobian is ill defined and the solution might diverge. There are methods, such as dumped least squares which try to mitigate this problem.

2.5.3 The Levenberg-Marquardt Method

The Levenberg-Marquardt method is a damped least squares algorithm. I give a detailed overview of the algorithm here, because I will use it extensively also in Chapter 3.

The method is used mainly for solving regression problems, such as the least squares problem in Equation 2.13. It is a mixture of the Gauss-Newton and the Steepest Descent Method.

The basic least square problem is:

$$r_m = y_m - g(t_m, \theta) \tag{2.11}$$

- $r \epsilon \mathbb{R}$ residual/error vector
- $y \epsilon \mathbb{R}$ the data point we want to fit on
- $t \epsilon \mathbb{R}$ the independent variable
- $\theta \epsilon \mathbb{R}^n$ the parameter vector

- $g \epsilon \mathbb{R}^n \rightarrow \mathbb{R}^m$ the function class we want to fit

We want to find parameters $\theta_1, \theta_2, ..., \theta_m$ so that the cost is minimized:

$$f(x) = \sum_m r_m^2(\theta) \tag{2.12}$$

The inverse kinematics problem can be defined as a root finding/optimization problem:

$$\boldsymbol{q}^* = argmin_{\boldsymbol{q}} \left\| \boldsymbol{x_d} - f(\boldsymbol{q}) \right\| \tag{2.13}$$

- $\boldsymbol{q}^* \epsilon \mathbb{R}^m$ the optimal joint coordinate vector
- $\boldsymbol{x_d} \epsilon \mathbb{R}^n$ the desired cartesian pose
- $f^{-1} \epsilon \mathbb{R}^n \rightarrow \mathbb{R}^m$ the nonlinear mapping between Cartesian and joint space
- $\boldsymbol{q} \epsilon \mathbb{R}^m$ the joint coordinate vector

Variations of the gradient descent algorithms are extensively used in the training of deep neural networks. The advantage of gradient descent is its simplicity. For gradient descent only the first derivative of the function has to be calculated. The algorithm takes a small step towards the opposite direction of the gradient. On Figure 2.5 the gradient steps on a quadratic cost function can be seen. Another advantage is that gradient descent has stochastic variations which work very well even in a distributed manner.

The simple Gradient Descent algorithm updates the parameters of the system:

$$\boldsymbol{x_{n+1}} = \boldsymbol{x_n} - \lambda \nabla f(\boldsymbol{x_n}) \tag{2.14}$$

The gradient descent method suffers from convergence problems. It has problems close to the minima, because the gradient steps become small. Its convergence is slowed down when the gradient direction changes rapidly around a point. Also the process is dependent on a good choice of λ, if it is too small convergence can take a long time, if it is too big we can jump out of a local minima.

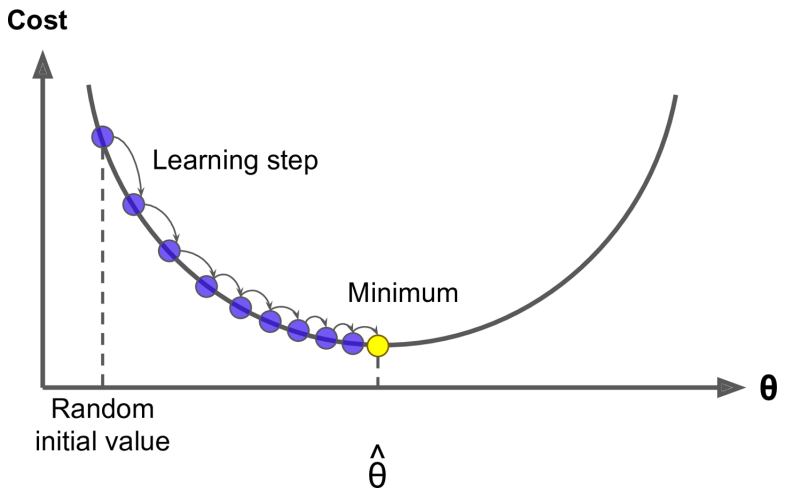


Figure 2.5: The Gradient Descent Method

The Gauss-Newton Method tries to solve the optimization problem by finding the root of the equation $\nabla f(x) = 0$. It performs the Taylor expansion the way we saw in 2.5.2. We assume that f(x) is quadratic and discard the higher order terms. For this reason the Hessian can be approximated by $\nabla^2 f(x) = J(x)^T J(x)$.

The update equation becomes:

$$\boldsymbol{x_{n+1}} = \boldsymbol{x_n} - (\nabla^2 f(\boldsymbol{x_n}))^{-1} \nabla f(\boldsymbol{x_n}) \tag{2.15}$$

The Gauss-Newton method provides rapid convergence. The reason it is not used in neural networks is because the quadratic assumptions does not stand, and the approximation of the Hessian is usually not precise enough.

The Levenberg-Marquardt method blends together the equations 2.14 and 2.15:

$$\boldsymbol{x_{n+1}} = \boldsymbol{x_n} - (\nabla^2 f(\boldsymbol{x_n}))^{-1} + \lambda I)^{-1} \nabla f(\boldsymbol{x_n}) \tag{2.16}$$

When evaluated this becomes:

$$\boldsymbol{x_{n+1}} = \boldsymbol{x_n} - (J^T J + \lambda I)^{-1} J f(\boldsymbol{x_n}) \tag{2.17}$$

The theory behind equation 2.17 is to follow the steepest descent until we get close to the canyon. In the canyon gradient descent slows down, but the Gauss-Newton part still provides quick steps. The λ parameter decides how much to take into account the gradient descent step compared to the Gauss-Newton. On Figure 2.6 the connection of the λ parameter and the step size can be seen. If λ is small Newton-Gauss part of the step becomes dominant.

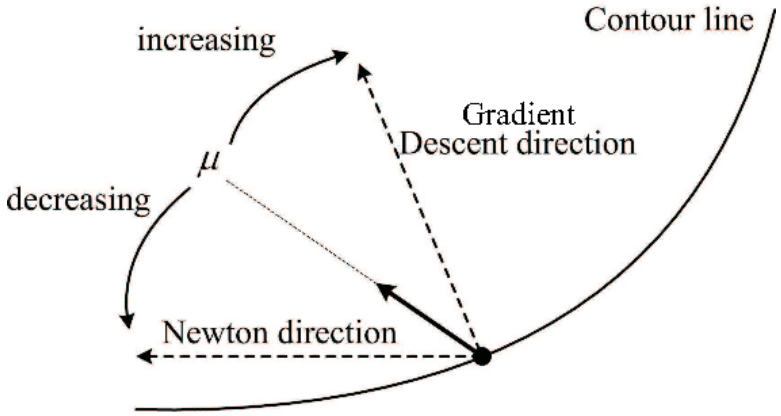


Figure 2.6: The dependence of the steps on the λ (here μ) parameter

2.5.4 The Challenges of Inverse Kinematics

The numerical optimization methods do not provide convergence criteria for non-linear cases, such as position inverse kinematics. It is not possible to predict in general whether the solver will convergence, how long the convergence will take and if it converges to the global minimum. Here the global minimum means the best joint configuration, that is closest to the desired end-effector position. Whether we reach global or local minimum depends highly on our choice of initial configuration. The closer we are to the minima the shorter time will the iterative calculation take. Since in teleoperation we move from point-to-point each displacement of the end-effector is small. This means that in general we are always close to the next solution.

The problem of local and global minimum manifest in robotics when the movement of the robotic arm is limited by its joint limits. The Schunk Powerball joint can move each joint between ± 170 degrees. When we get close to the joint limits the solution convergence to the limits if we initialize the solver from the previous position. On the other hand if we started from a more general point we could reach a much closer position from a different angle. The solver will jump to the right solution after the residuals of the function approximation become too big and it is able to escape the local minima. In this case the solution jumps from one joint configuration to another very quickly. This jump will also manifest in the dynamics of the robots, as an actual stopping of the movement when the residual rises, and a jump after that when the local minimum is stepped over. In Cartesian space the distance between local and global minimum is small, but in Joint space it is very big.

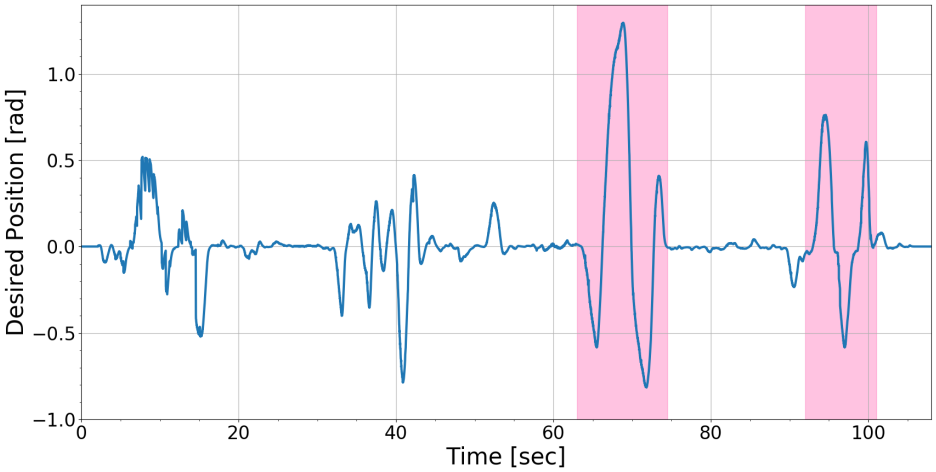


Figure 2.7: The desired joint position can quickly change around local minima Here marked purple

There are two main types of points where the inverse kinematics may fail. The first set of points is when the location is not reachable by the robotic arm. In this case a good solver should still converge to the closest reachable position. The second one is at singular positions. In singular position because of the joint configuration, the end-effector loses degrees-of-freedom of movement in task space. Because of this it is unable to follow the error vector, and the solution might diverge. In mathematical terms singularity is when the determinant of the Jacobian matrix is zero, in this case the rank of Jacobian is reduced. This can happen very easily during teleoperation. The operator can observe these states when two or more joints are aligned along a common axis. In this case around the singular position the arm starts to move uncontrollably. This is because the solution diverges around the singular point, but it diverges at each time step to a radically different solution. This behavior makes teleoperation rather dangerous, because the arm starts waving uncontrollably with its maximum acceleration.

In the Schunk Powerball arm generally there are 2 solutions of joint positions for a given end-effector pose. The difference is whether the first 3 joints are reaching the point from below or above. On Figure 2.8 two different joint configurations can be seen, which result in the same end-effector pose. Usually the solution depends on the choice of initialization. It is desired to reach always from above, so when we are lifting an object it does not collide and damage the lower parts of the arm. To ensure that the arm is reaching from above the best solution is to modify the numerical optimizer. Add a new entry to the cost function which depends on the angle between the first and second link. This method however

requires the modification of the KDL library. Another not as efficient way is to simply start the calculation from the vertical position of the arm. In this case the solution will always be approximated from above. However this makes the calculations much slower.

Figure 2.8: There are usually 2 solutions for the inverse kinematics problem

The telemapping between the two end-effectors in task space unfortunately is not one-to-one. There can be two main types of difference, mismatch of the base or mismatch of proportions. In the first case the base of the World coordinate system in the two robots is not at the same place. In the Schunk arm the base is located 205 mm under the first link. However the base of the coordinate system is located 510 mm under its first link. Secondly, the size of the Schunk arm is smaller then the Kuka iiwa. The desired end-effector positions are parameterized in mm, the positions have to be scaled according to the reach of the two robots. This mismatches of the task spaces have to be compared to each robot pairing. I have determined the best values after several experiments.

	Schunk Powerball	**Kuka iiwa LBR**
X [mm]	705	955
Y [mm]	705	955
Z [mm]	910	1305

Table 2.2: The reach of the two arms in Task Space

2.5.5 Evaluation of IK solvers

There are several numerical solvers based on the method for solving the regression problems. The most widespread methods are Newton-Rhapson, Levenberg-Marquard and BFGS.

A very thorough evaluation of the KDL inverse kinematics library has already been written by Track Labs (7). In the paper the authors have evaluated the solvers based on solution speed and solve rate. The experiments were done on several different robot architectures. The robots were given high number of points in their task space. The authors have concluded that the main problem is around joint limits. Around limits the success rate drops heavily, and also the solvers slow down remarkably. The solve rates of KDL ranged from the 21.07% for the Fanuc M-430iA/2F to the 97.85% of the NASA Robonaut2 robot.

I have only evaluated the solvers on the Schunk Powerball robot specifically for teleoperation. The main criteria of evaluation were stability, speed and ability to stay in joint limits. To evaluate the solvers I have performed the same experiments with the solvers. Moved the robot arm freely, then approached a point of singularity, and finally went close to a position which is between two local minima. Both solvers had 500 maximum steps for solving , and the precision of solution was 1e-5. Every solution attempt started from the vertical position where all joints are initialized to zero. Ran the solvers with 100 Hz frequency.

In practice it was easy to see the difference between the solvers. I have found it hard however to show the results on paper. I decided to use 4 plots for comparison. These plots can be seen on Figures 2.9 and 2.10. On the "runtime-time" plot the fluctuation of the solver runtime can be seen. The "position error-time plot" shows the mean difference between the returned position of the solver and the actual position of the arm in joint space. The "joint overrun-time" plot shows how much the joint solutions violate the joint constraints. Finally, the "joint overrun-position error" plot shows the connection of these two error types.

The Newton-Rhapson solver proved to be very unstable in operation. As can be seen on Figure 2.9 when the solution does not converge the solver returns very late. In these cases the joint positions returned by the solver are invalid. Additionally the joint positions diverge towards infinity. From the normal range of $[-\pi, +\pi]$ to the magnitude of 15000. Finally, the correlation between joint overrun and position error is very weak, the two effects can happen separately. In conclusion the Newton-Rhapson solver is very unstable both around joint limit and singularity. Although it must be noted that with a lower precision and initiation from the previous solution the solver shows only instability around singularity.

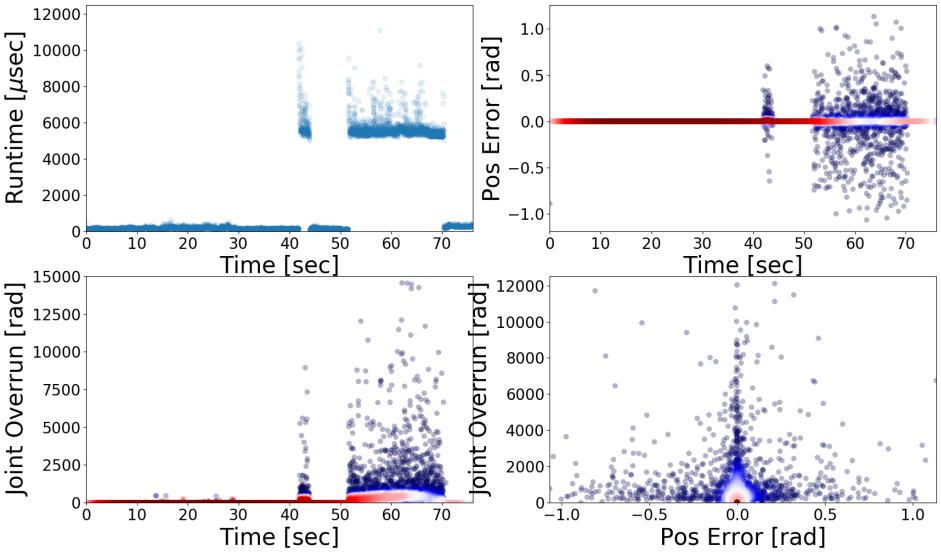


Figure 2.9: Evaluation of KDL Newton-Rhapson solver

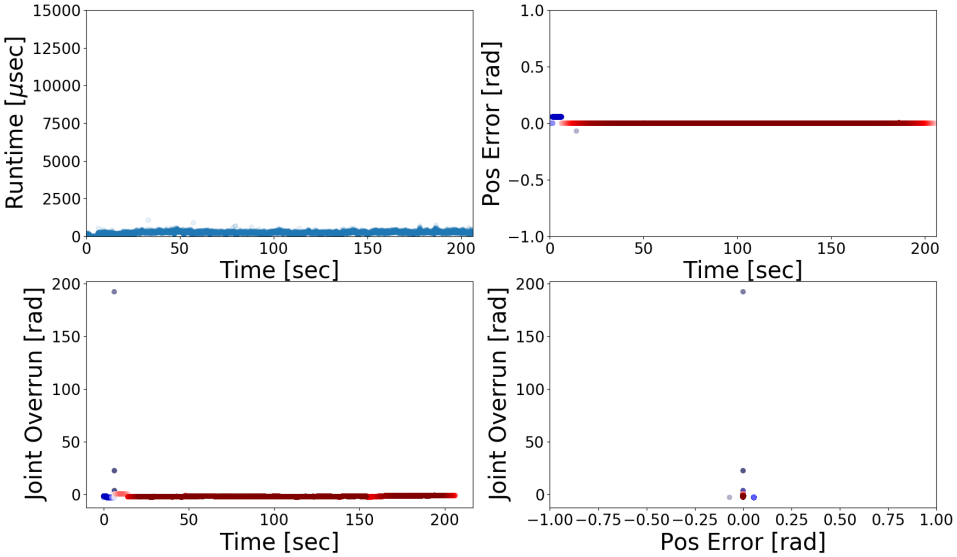


Figure 2.10: Evaluation of KDL Levenberg-Marquardt solver

The Levenberg-Marquardt solver is much faster and more stable then Newton-

Rhapson. The Newton-Rhapson solver has systematic problems with both joint limits and singularities this is why there is a spread of erroneous points on the figure. On the other hand the Levenberg -Marquardt solver does not have regions where it fails, only sporadic points where the calculations have errors. On Figure 2.10 it can be seen that there are very few outliers. The solver returns consistently at the same time, there are no points where the calculation take much longer. On the 3 error plots most of the points are concentrated in the 0 error point or line. There are only a few points which do not fit the solutions well. The solver did not have problem with singularities whatsoever. Suprisingly it even surpassed in precision the more sophisticated industry quality inverse kinematics of the Kuka iiwa around singularities. However it has small instability close to the two local minima. In practice I used this in my application. In application I have reduced the precision of the solver by one magnitude, in order to get faster results.

2.6 Position Controller

The inverse kinematics solver provides the target joint positions based on the desired end-effector position. Unfortunately, it is not possible to set the joint positions directly from the CERN Robotic Framework. Through the framework only the velocities of the joints can be controlled. In order to set the desired position I had used a soft real-time proportional-derivative (PD) controller.

The proportional-integral-derivative controller (PID) is one of the oldest controller types. The controller can be used to drive an output value of a single input single output (SISO) system to a desired value. The big advantage of the controller is that there is no need to have a model of the process in order to set it up. The controller only needs to receive the difference of the desired and measured values of the system, the error term, to calculate the control input. On the other hand it does not provide optimal control, unlike controllers based on state-space models.

The general equation of a PID controller:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt} \tag{2.18}$$

- u(t) the control input of the system
- K_p proportional gain, a tuning parameter
- K_i integral gain, a tuning parameter
- K_d derivative gain, a tuning parameter

- $e(t)$ error term in time $e(t) = x_{desired} - x_{measured}$

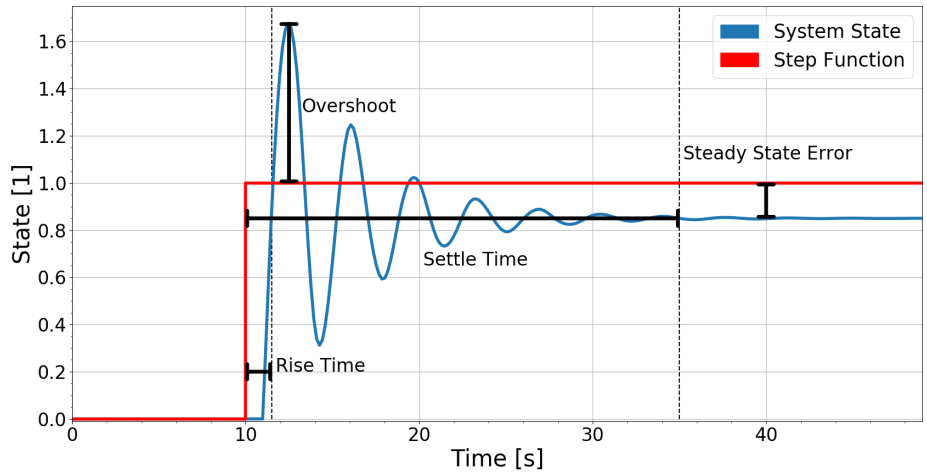


Figure 2.11: Closed-loop response of a poorly tuned PID controlled system

The PID controller has 3 independent parts: proportional, integrator and derivative terms. The proportional part creates feedback based on the magnitude of the error term. If the proportional term is too big, the system becomes unstable. If it is too small, the error reduces slowly. The proportional part only creates feedback in the presence of an error signal. When the error reduces close to zero, the feedback created by the proportional term also tends to zero. For this reason there might be a small amount constant error, called steady-state error, remaining.

The integral part creates the feedback based on the integral of the error term. The integral part is capable of driving the steady-state error to zero. Because of the accumulation of errors the integral term results in a bigger overshoot. It provides a very slow convergence to steady-state, for this reason in application it is always used in parallel with a proportional part.

The derivative term creates the control signal based on the change of the error term through time. It essentially predicts the change of error through time. It is used in order to reduce the settle time and the overshoot of the system. In itself the derivative term does not ensure the convergence to a steady-state. An ideal derivative term is not causal, its control output at time step t depends on the input at time step t+1. It is not possible to implement an ideal derivative term in practice. Derivation amplifies noise of the error signal and can create stability problems. For this reason it is not included in every PID controller, and when it

is applied it is usually used with a low-pass filter. A derivative term reduces the overshoot and the settle time of the closed-loop system.

My experiments with the Schunk Powerball arm revealed that the joints have separate velocity and acceleration limits. The joint where the limits were exceeded shuts down, and the whole arm has to be restarted after. The velocity limits are given relative to the maximum joint velocity. The acceleration limit is the difference between two consecutive velocity values in relative units.

	J1	**J2**	**J3**	**J4**	**J5**	**J6**
Velocity Limit [1]	0.45	0.5	0.5	0.6	0.8	1
Acceleration Limit [1]	0.04	0.06	0.06	0.1	0.1	0.2

Table 2.3: The relative velocity and acceleration limits of the joints

The first step of setting up a PID controller is deciding which parameters of the system we want to optimize. When setting up a control system always the most important factor is closed-loop stability. If the feedback of the controller is too big the system can become unstable. In this case the robot arm end-effector starts to oscillate with increasing amplitude even if the desired state of the system is constant. Additionally the controller has to be fast, so the slave robot does not fall behind the master robot. Finally, the controller has to reach a small, but not necessarily zero, steady error. Precision is not that important in telemapping, because the operator estimates the end-effector position error only from the video feed not from measurements. For this reason the end-effector position error in the order of 5 mm can be accepted.

During telemapping it is important to achieve small or no overshoot and oscillation. Overshoot manifest in quick vibration of the end-effector. Usually the human operator follows a trajectory with his movement. The trajectory segments are sampled with over 200 Hz. In such a small time scale the orientation of the segments changes very slightly. The slave robot gets at every time step a segment of this trajectory. The controller might be working with a higher frequency then the segments are received. The robot controller can do several time steps between segments. In the presence of overshoot the controller attempts to correct it in the opposite direction. On the other hand the movement of the next segment will be most likely in the direction of the overshoot. Because the end-effector changes the movement direction rapidly it starts vibrating. The vibration is very high-frequency, can be barely seen through a camera. Between the last joint and the end-effector the force torque sensor is located. The sensor readings however pick up this vibration, which manifests as dynamic noise of the readings. This dynamic noise however will reduce the precision of the force-feedback.

According to the above points it is clear that a PD controller is the best

choice for telemapping. Apart from the fact that the precision provided by the integrator part is not needed, implementing it would have been very problematic. Because the inputs of the system are limited the integrator would show integral windup (5). This happens, because the integrator term accumulates the error during rise time according to the ideal control output and not the actual, limited output. This effect results in the instability of the system.

The PD controller has only 2 free parameters: proportional and derivative gain. There are several ways of tuning these parameters. Since I did not have the dynamic model of the system, I could not use automatic tuning software, such as the built-in tuner of MATLAB. I used the Zigeler-Nichols Method (18) to get a first estimate of the parameters, then refined them by experimentation. The method does not require the knowledge of the system, and it is easy to perform. The proportional gain of the system has to be increased until the motor starts oscillating with constant amplitude. This is basically the border point between stability and instability. In this point the proportional gain K_u and the oscillation time period T_u has to be noted down. From these two values the gains of the controller can be set according to Table 2.4.

	Kp	$\boldsymbol{T_i}$	$\boldsymbol{T_d}$
P	0.5 K_u	-	-
PI	0.45 K_u	$T_u/1.2$	-
PD	0.8 K_u	-	$T_u/8$
PID	0.6 K_u	$T_u/2$	$T_u/8$

Table 2.4: Ziegler-Nichols method parameter selection

With the Ziegler-Nichols Method the value of P = 20, D=3. I have found these parameters to be a bit to aggressive, since the method does not take into account the velocity and acceleration limits. It resulted in too high acceleration, so in practice I reduced the values of P and D.

	$\sigma_e(J1)$	$\sigma_e(J2)$	$\sigma_e(J3)$	$\sigma_e(J4)$	$\sigma_e(J5)$	$\sigma_e(J6)$	σ_e	$Avg(\|e\|)$
P=5 ;D=0	0.042	0.069	0.167	0.264	0.132	0.059	0.146	0.062
P=10;D=0	0.041	0.011	0.025	0.162	0.055	0.141	0.093	0.028
P=15;D=0	0.059	0.04	0.072	0.171	0.083	0.099	0.097	0.025
P=10;D=0.1	0.06	0.07	0.147	0.133	0.065	0.051	0.096	0.033
P=10;D=0.3	0.082	0.06	0.065	0.216	0.048	0.131	0.116	0.035
P=10;D=0.5	0.049	0.049	0.099	0.125	0.044	0.018	0.074	0.030
P=10;D=0.75	0.044	0.029	0.077	0.277	0.037	0.103	0.128	0.037

Table 2.5: The effect of different PD values on the precision and speed of the control system

In order to find the best parameter values I have performed experiments with the robotic arm. The experiments were done through teleoperating the arm, and using the Levenberg-Marquardt inverse kinematic solver. In each experiment I have performed a predefined set of motions, which try to resemble tasks that can be encountered during a real intervention. These tasks include picking up a connector, placing it at a given area, touching the base of the robot, pushing a wire. Each experiment takes around 90 seconds and creates more then 10000 time steps of the controller. At each step I have logged the error of the joints. The standard deviation of the error signals shows the speed of the controller. The less the standard deviation the faster the controller converges to the desired value. I have also calculated the average error of the controller, which shows the precision of the movement.

In Table 2.5 the parameters of the experiments can be seen. In general the P controllers without a D term proved to be more precise. The fastest controller proved to be a PD controller it was 25% faster then the best P controller, and was only slightly less precise. The controller is mostly limited by the velocity limits, which it reaches usually in around 5 time steps. There is not too big of a difference between individual controllers because of the limits.

3

Force-Torque Sensing on the Slave Robot

Chaos is merely order waiting to be deciphered.
– Jose Saramago, The Double

3.1 On the Importance of Haptics

By far the most utilized sense we have is vision (25). According to studies up to 50% of the neural context is involved with processing the vision stimuli (6). In order to create a crude teleoperation system visual feedback should be enough. We can judge the position, velocity and size of objects with our eyes. We can also guess some of their properties, such as their weight, surface properties, friction and acceleration. These parameters we can not directly infer from images, but only based on our previous experiences of manipulating objects.

On the other hand for manipulation we utilize highly our sense of touch from a young age. With our perceptive hands we are capable of measuring weight, inertia and acceleration (19). When children learn how to manipulate objects around them, they touch anything that surrounds them in order to learn their dynamical properties. In the area where maintenance is carried out by members of the Robotics Team many unusual objects can be encountered. It is not possible to judge for example how the magnets and collimators react when they are touched, how heavy they are, since the operator most likely never encountered anything similar to them before.

There is another key difference between manipulating with human hands and through teleoperation. When we manipulate with our hands, we are perfectly used to our limbs and fingers. We know how much force we can exert, and how far we can reach. During teleoperation an untrained operator might misjudge

the strength and reach of the robotic arm, since they are not so accustomed to them. If the operator underestimates the exerted force he can easily damage the surrounding environment, if he overestimates it he might break the robotic hand. With the help of force-feedback the operators can easier predict the dynamical properties of objects and also the manipulator, which makes operations much safer.

3.2 Robotic Force Torque Sensors

In order to be able to create a force-feedback signal on the master side, the slave robot has to be able to measure the forces and torques applied to its end effector. There are three main ways of measuring forces with a robotic manipulator:

- Calculating forces from the joint currents
- Measuring torques in each joint with joint torque sensors
- Measuring with 6 DOF force-torque sensor attached to the end-effector

The Kuka iiwa LBR robot has built-in sensors in each joints. This provides it with the ability to not only precisely measure the forces, but to be able to use impedance mode. On the other hand Schunk Powerball has no built-in sensors. To be able to calculate the joint torques from the currents and voltages a very detailed dynamic model is needed. The precision of the calculation also highly depends on the non-linear elements of the load, such as the inner friction of the motor transmission, and the effect of the heating of the joints. Several experiments were carried out for sensor-less torque measurement, but there does not exists a solution for this particular robotic arm.

In order to make it possible to measure forces with the Powerball, the Schunk company has designed the FTM115 force-torque sensor. It can be connected in series between the last joint of the robot and the end-effector. The advantage of the sensor compared to other alternatives is, that there is no need for a special mechanical link. It can be easily fitted to the end-effector, between the last joint and the gripper. The signal of the sensor can be propagated through the cabling of the robot arm, so no outside wires are needed.

The force-torque given by sensor has the form:

$$\boldsymbol{F_{sensor}} = M_{calib} * \boldsymbol{SG} \tag{3.1}$$

- $\boldsymbol{F_{sensor}} \epsilon \mathbb{R}^6$ the measurement 3 forces [N] and 3 torques [Nm] in the sensor coordinate system
- $M_{calib} \epsilon \mathbb{R}^{6x6}$ the calibration matrix of the sensor, it is not time dependent

- $\boldsymbol{SG} \epsilon \mathbb{R}^6$ the raw measurement values of the sensor, they are dimensionless

In my model I have assumed that the sensory readings have 4 components:

$$\boldsymbol{F_{sensor}} = \boldsymbol{F_{bias}} + \boldsymbol{F_{gravity}} + \boldsymbol{F_{noise}} + \boldsymbol{F_{contact}} \tag{3.2}$$

- $\boldsymbol{F_{bias}} \epsilon \mathbb{R}^6$ the parasitic bias component of the measurement
- $\boldsymbol{F_{gravity}} \epsilon \mathbb{R}^6$ the force of gravity exerted by the lower parts of the sensor and the gripper connected to it
- $\boldsymbol{F_{noise}} \epsilon \mathbb{R}^6$ measurement noise
- $\boldsymbol{F_{contact}} \epsilon \mathbb{R}^6$ the force caused by the interaction between the robot and the environment, the amount we want to measure

The sensor however has the following problems:

- Very high dynamic noise
- Strong sensory bias
- Lack of gravity compensation
- Bias changes with the sensor temperature

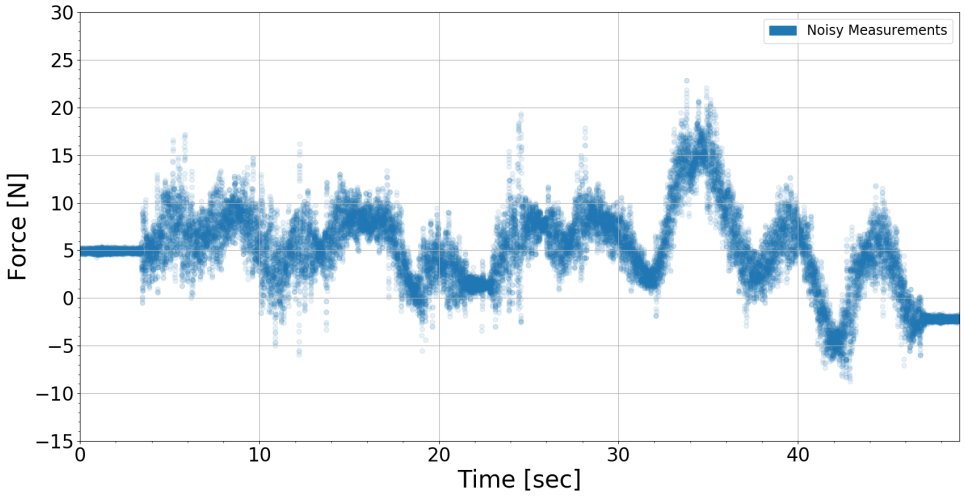


Figure 3.1: The F_z component in the Sensor Coordinate system
There is no contact force present

On Figure 3.1 the measurements can be seen, when there is no outside force. The measurements are not centered in 0, rather in 5N. This is due to the sensor bias. The high spread of points comes from the dynamic noise of the sensor. Finally, the mean of the forces changes through time because the orientation of the arm changes. The force of the end-effector is differently acting on the axis of the sensor. Fortunately these effects can be observed in different timescales. The noise has very high frequency. The biases can be compensated at the beginning of the operation. It has a close to zero frequency. The heat compensation has to be carried out slowly as the sensor heats up. Thanks to this, they can be handled separately. It is important to mention that without these processing steps the output of the sensor can not be used in practice, since the measurement can differ from the real values up to 40 N for each force component. This can lead to the measurement of 70 N force, when there is absolutely no load on the end-effector. In the next sections I am going to go through each of these points and give a solution for their compensation.

It must be pointed out, that the above definitions do not include the effects of the hysteresis. Usually, it can be assumed that the measurements are independent of each other. The history of the sensor plays no part in the output values. According to my experiments this is not the case with the Schunk FTM115 sensors. There is an error to the system, which comes from the hysteresis of the sensor. Because of its high non-linearity and dependence of temperature and other factors I was not able to address this disturbance.

3.3 Reducing Noise with Kalman Filtering

3.3.1 Analysis of the Disturbances

The force-torque sensor measurements contain additive noise. The noise has a relatively small amplitude in a static environment, introduces a sample variance of around 0.1 N. This noise consists of mostly pink noise, or so called flicker noise. Flicker noise is introduced by CMOS elements of the sensor circuitry. There is also a small amount of white noise, and noise introduced by the rectifier around 50 Hz. The latter will not be noticeable during the teleoperation, since the robot runs from a DC battery.

When the robotic arm starts moving, the joints vibrate a bit because of the rapid acceleration and decelerations. Although this vibration is small for the individual joints, because of the serial connection of links it is additively propagated to the end-effector. This vibration is measured by the force sensor. It has zero mean and Gaussian distribution. The variance is bounded, around 3-5 N. In light of this, it can be treated as colored noise. Most notably the frequency spectrum

resembles that of pink noise, as can be seen on Figure 3.2.

It is important to reduce the effect of this noise to increase the dynamic range of the sensor. When an object comes into contact the end-effector it exerts a force on it. If we contact something flexible such as a sponge the force might be small. If the contact force is smaller then the noise amplitude of the system it is not possible to distinguish whether the readings are coming from an actual contact. The more the effects of the noise can be reduced, the more delicate objects can be handled by force-feedback teleoperation. On the other hand when moving the arm, the vibration amplitude is so big, that it can destabilize the teleoperation system if the delay exceeds a threshold. The master robot can pick up the vibration and amplify them, creating an unstable positive feedback loop. In addition to greatly reducing the comfort of the operator.

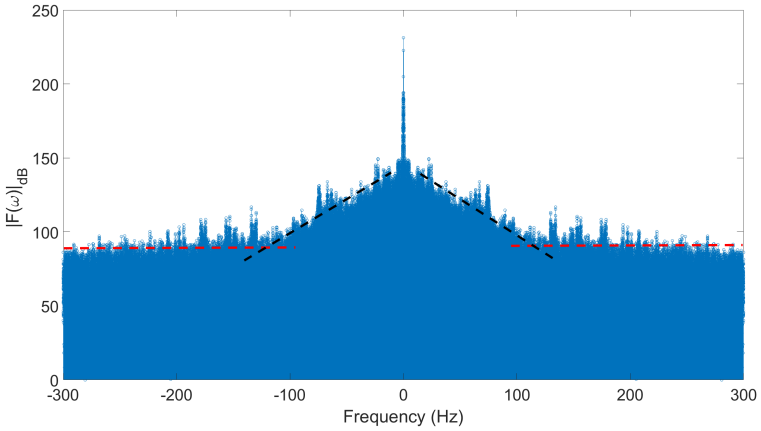


Figure 3.2: The Fourier transformation of noise during movement. Black line shows pink noise. The red line shows white noise.

In order to identify and measure the components of the noise I have carried out several experiments. Moved the robot arm with its maximum speed while keeping the orientation of the end effector constant in task space. This would result in constant force values on each measurement axis, only the dynamic movement would create noise.

The 6 components of the readings show similar characteristics, for this reason I will only plot 1 component on Figure 3.3. The other component characteristics can be found in Table 3.1.

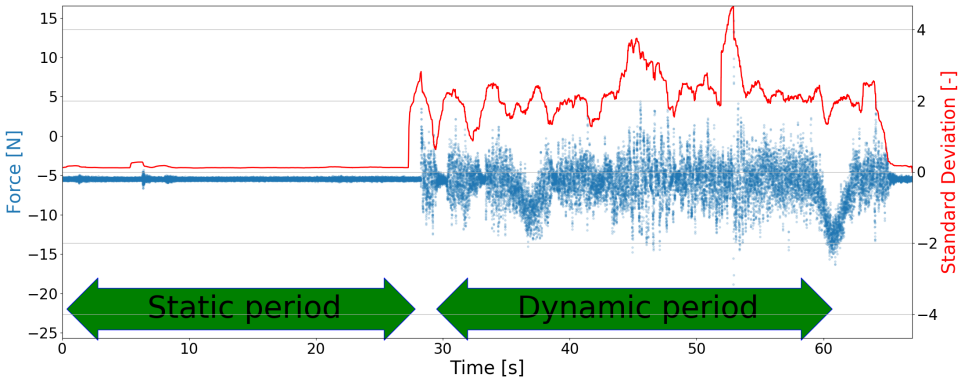


Figure 3.3: The signal in Static and dynamics measurement, with the standard deviation plotted

	F_x [N]	F_y [N]	F_z [N]	T_x [Nm]	T_y [Nm]	T_z [Nm]
Static Stdev	0.1374	0.1283	0.1344	0.0065	0.0074	0.0015
Dynamic Stdev	2.4862	2.4726	2.5888	0.191	0.2184	0.0342

Table 3.1: Standard deviation of the signal

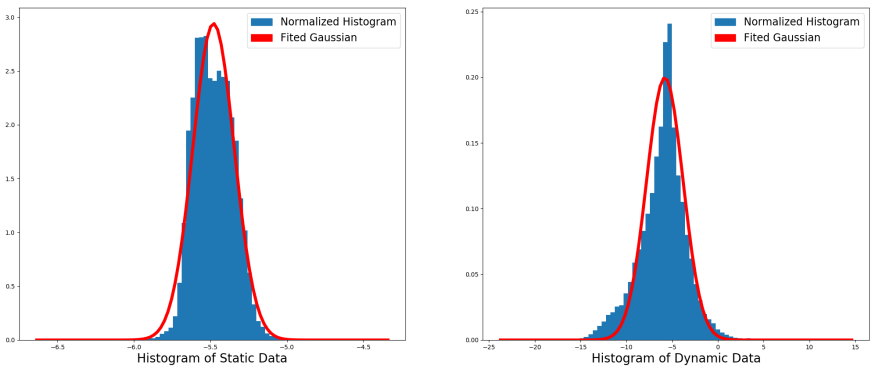


Figure 3.4: The histogram of the distribution of data points in static and dynamic case

On Figure 3.3 it can be seen that during the dynamic movement the noise is amplified. The vibration depends on the acceleration, it is not a constant effect. For signal processing purposes I will use the standard deviation of the dynamic

section for estimating the sample variance. The standard deviation is the square root of the variance. The values can be found in Table 3.1. This can be thought of as a worst case for noise amplitude.

Unfortunately, with such a big sample size standard numerical normality tests do not provide any meaningful insight. Both Shapiro-Wilk or Anderson-Darling test were created for small sample sizes, in the order 50-100 data points. With high number of data points the uncertainty rises. I applied the test to sets of randomly sampled 50 points from the data, and both algorithms failed to reject the H0 hypothesis of normality with a p value of 0.95.

In this case the visual normality test provides more insight, that is the reason why I have included Figure 3.4. I have created histograms for the static of dynamic case of the distribution of data points. For better understandability I have fit a Gaussian curve on both datasets. From the histogram it can be clearly seen that they follow normal distribution. In the dynamic case the data does not perfectly fit under a Gaussian. There are sporadic periods where the robot arm is not accelerating. These periods are concentrated around the mean, which becomes elongated. It can also be seen that the histograms are skewed towards the left. This change comes from the heat drift of the measurements. Fortunately the distribution is not long tailed, in both cases it is clearly bounded. This means that there are no too great, clear outliers, in the measurement points.

3.3.2 Introduction to Kalman Filtering

To counterbalance the effect of the noise a filter must be applied to the data measurements. In choosing and tuning the filter the two main concerns are frequency and phase response. An ideal filter completely eliminates the noise, while introduces zero phase shift. Generally however these two properties are inversely correlated. For this project I have decided to use Kalman Filter. I made this choice, because it is proven that the Kalman Filter (31) is optimal for a linear system with Gaussian-noise.

The filter was published by Rudolf E. Kalman in 1960 (16). It was developed as a solution for the discrete linear filtering problem. This method was mainly used in navigation systems of rockets in the 1960s, most notably in the guidance systems of the Apollo program rockets.

It is easy to implement, and is fast enough to be used for real-time data processing. It can be considered as a low-pass Infinite Impulse Response filter. It is a recursive state estimator, which means that it does not need to store the historical data points. Each state estimate is based on the parameters of the last and the current time step.

The Kalman filter is used to estimate state of the linear system of the form:

$$\boldsymbol{x_k} = A * \boldsymbol{x_{k-1}} + B * \boldsymbol{u_{k-1}} + \boldsymbol{w_{k-1}} \tag{3.3}$$

- $\boldsymbol{x_k} \epsilon \mathbb{R}^n$ the state of the system in time step k
- $A \epsilon \mathbb{R}^{n \times n}$ the state-transition model
- $\boldsymbol{x_{k-1}} \epsilon \mathbb{R}^n$ the previous state
- $B \epsilon \mathbb{R}^{n \times m}$ the input matrix
- $\boldsymbol{u_{k-1}} \epsilon \mathbb{R}^n$ the system input at time step k
- $\boldsymbol{w_{k-1}} \epsilon \mathbb{R}^n$ the process noise

$$\boldsymbol{z_k} = C * \boldsymbol{x_k} + \boldsymbol{v_k} \tag{3.4}$$

- $\boldsymbol{z_k} \epsilon \mathbb{R}^p$ measurement at time step k
- $C \epsilon \mathbb{R}^{p \times n}$ the observation model
- $\boldsymbol{x_{k-1}} \epsilon \mathbb{R}^n$ the previous state
- $\boldsymbol{w_{k-1}} \epsilon \mathbb{R}^n$ the measurement noise

It is assumed that both measurement noise and process noise are drawn from zero mean normal distributions.

The Kalman Filtering process can be split into two steps, estimation and time update of the state variables. During the estimation phase the state and error covariance are predicted. During the update phase, the estimates are corrected based on the measurements. The two steps do not need to be synchronous. Measurements can be omitted, or if several measurements are available, the estimation can be updated consecutively.

The time update equations:

$\hat{x}_{k\|k-1} = Ax_{k\|k-1} + Bu_k$	Predicted state estimate
$P_{k\|k-1} = AP_{k-1\|k-1}A^T + Q$	Predicted error covariance

The measurement update equations:

$$
\begin{aligned}
S_k &= CP_{k|k-1}C^T + R && \text{Innovation covariance} \\
K_k &= P_{k|k-1}C^T S_k^{-1} && \text{Optimal Kalman Gain} \\
\hat{x}_{k|k} &= Ax_{k|k-1} + K_k(z_k - C\hat{x}_{k|k-1}) && \text{Updated state estimate} \\
P_{k|k-1} &= (I - K_k C)P_{k|k-1} && \text{Updated state covariance}
\end{aligned}
$$

The matrices A,B,C,Q and R come from the equations 3.3 and 3.4.

The 6 measurement values obtained from the force-torque sensor can be considered independent. The Kalman filter can make the estimation for the values independently, or in vector form. It is easier the implement the filter for scalar value case. In matrix form however the computation is more efficient, since matrix calculation can use the SIMD units of processors. For the sake of easier understanding, I will describe the equations in case of applying the filter for the scalar case independently for each measurement.

The equations become simplified in this case. There is no control input, so the matrix A = [1] and C = [1]. In my case, the estimation and measurement updates are carried out synchronously.

The update equations simplify to:

$$
\begin{aligned}
P_{k|k-1} &= P_{k-1|k-1} + Q && \text{Predicted error covariance} \\
K_k &= \frac{P_{k|k-1}}{P_{k|k-1} + R} && \text{Optimal Kalman Gain} \\
\hat{x}_{k|k} &= (1 - K_k)x_{k|k-1} + K_k z_k && \text{Updated state estimate} \\
P_{k|k-1} &= (I - K_k)P_{k|k-1} && \text{Updated state covariance}
\end{aligned}
$$

As it can be seen the Kalman Filter is simplified in this case to an alpha filter, with variable alpha parameter. To describe how it is working, we can say that the filter follows the measurements, but the amount it changes with the measurement depends of how certain it is in the previous estimate. This certainty is encoded in the P parameter.

The rate of change of the parameter depends on the relation of R and Q. The parameter R is the measurement noise covariance, which can be approximated by the square of the standard deviation of the signal under constant load. The parameter Q shows how much do we trust our state space model. The higher the

Q the more we trust our model, the less effect the measurements will have on the predicted state values. This parameter can not be calculated explicitly, it can be estimated by for example Autocovariance Least-Sqares (ALS) (23). Here I set it through experimentation.

As Q is the only free parameter of the system, it is possible to balance time-shift and smoothing. On Figure 3.5 the effect of different Q values can be seen on the estimation. I set Q so that during the dynamic movements the vibration of the force values stays in the range of ± 0.5 N. This value is small enough in practice that it is not possible to sense it through force-feedback, but on the other hand the state estimation is not slowed down too much.

It must be noted that most of the objects the manipulator will come in contact during operation are made of metal. Additionally, the end-effectors used in operation are also made of steel. When the end-effector makes contact with another metal part the contact force skyrockets. It can go from 0 N to 100 N in less then 50 milliseconds. This is because the contact materials have a very high stiffness. No matter what phase shift the filter introduces it is impossible to create the same amount of force change with force-feedback. On the other hand this kind of jump in the force-feedback would also not be desirable.

When we touch something with our hands the forces change with a much smaller speed. Our fingers have a low stiffness, and they can damp the collision when making contact with an object. It is much more natural for the operator to encounter this effect during the operation. This damping of force-feedback is performed by the Kalman filter, with the phase shift it introduces. In the future it would be a good development to 3D print the tip of the end-effector from elastic materials. To mimic the human fingers dumping effect in practice.

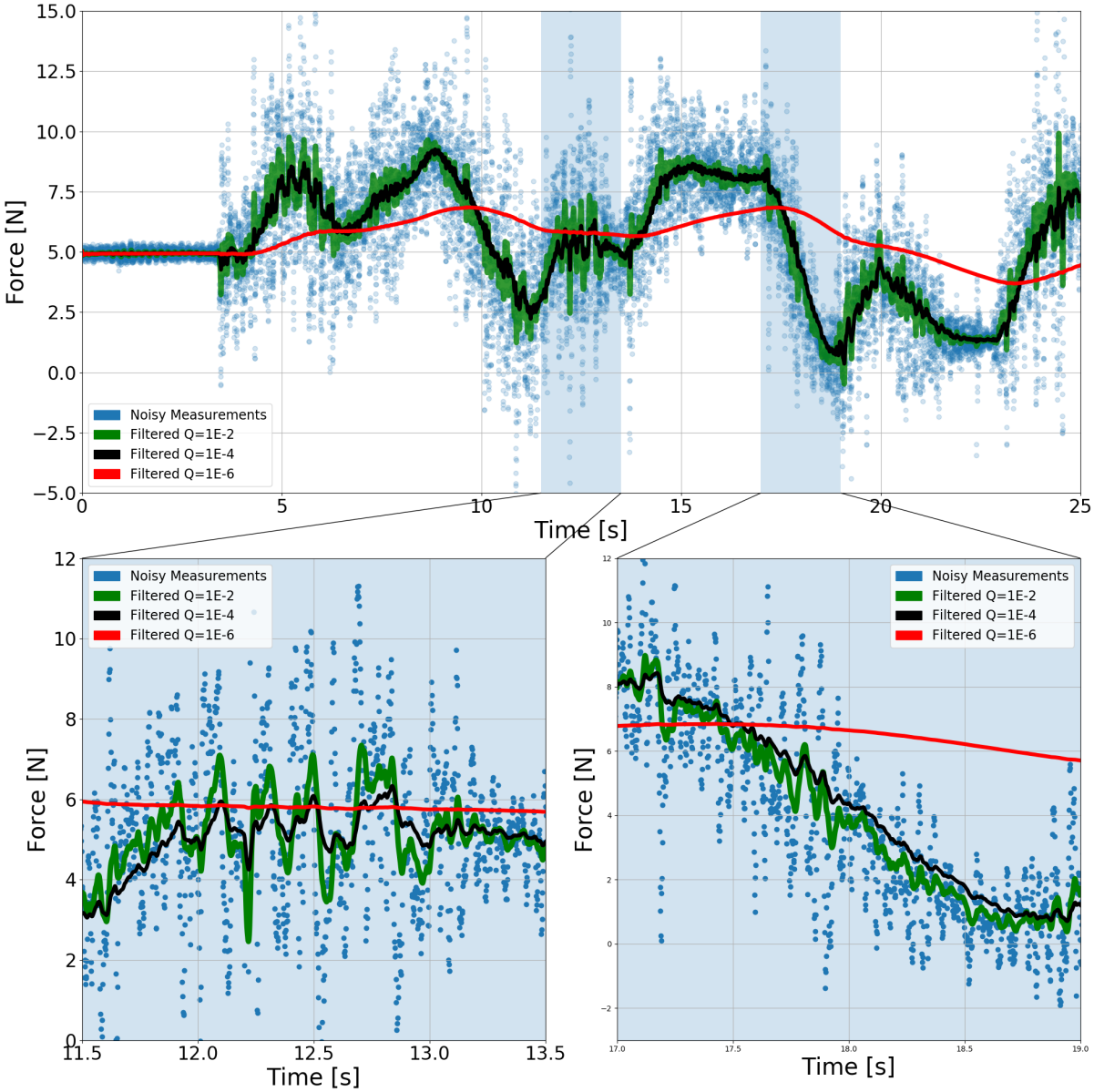


Figure 3.5: The effect of different Q values on the Kalman Filtering of F_x

3.4 Bias and Gravity Compensation

In order to be able to sense when the end-effector of the robot arm makes contact with an object, the readings of the force toque sensor has to return zero when not under load. The sensor readings have to be corrected to zero in this case. We can assume at stable position that the effect of the noise is small, and that there

is no contact force, so the Equation 3.2 becomes:

$$\boldsymbol{F}_{sensor} = \boldsymbol{F}_{bias} + \boldsymbol{F}_{gravity} \tag{3.5}$$

The bias is present in every sensory reading. The value of bias is not dependent on the orientation of the end-effector and on the force exerted on it. In a small time frame it can be considered as an additional constant. According to my experiments the bias of an individual force component can be anywhere between ±30 N. If it is not corrected, then the user can experience a compound force which can exceed 50 Newtons on the master robot. The bias is unfortunately dependent on the temperature of the sensor. I will deal with this correlation in detail in Chapter 3.4.1.

Attached to the force-torque sensor there is an end effector connected in series. This end effector can be changed depending on the intervention. It can be a parallel gripper, a screw driver or a vacuum pump. On Figure 3.6 the end-effectors used during my project can be seen. These end-effectors exert a constant force based on their weight. They also exert a torque whose size depends on the weight of the end-effector, and changes with the orientation. During the measurement process we need to account for these force and torque readings too, and we need to subtract them from the raw readings of the sensor.

(a) FTM115 (b) Robotiq 140 (c) Schunk PG 070

Figure 3.6: The force-torque sensor and two different end-effectors of the Schunk Powerball robotic arm

If the effect of the noise would be small the biases could be identified by simply aligning the robot arm vertically, since in this position the end-effector force is only exerted on the Z-axis, and the torques are negligible. From upward and downward position the bias of the Z-axis and the mass of the end-effector could

be calculated. Finally, the torques could be measured when setting the arm in a horizontal position.

There are several difficulties for doing this calibration. It is hard to separate the effects of the bias and the end-effector load, since all the biases can be identified only when there is no outside load. The measurements have to be carried out in the sensor coordinate system. It has to be also taken account, that the positioning of the arm also has a noise, which introduces an error on the bias calculation. We can assume that there are no dynamic forces on the sensor, but even then the readings have a small noise which have to be accounted for. Finally, the calibration has to be done relatively quickly, so to minimize the effect of the heat drift.

I have created an algorithm based on least-square regression to carry out the calibration. To minimize the positioning error of the hand during calibration only one of the joints are moved. During the process the robot end-effector follows an arc in the vertical plane and records the readings of the force-torque sensor. The arc takes up ±140 degrees compared to the horizontal plane. The calibration arc can be seen on Figure 3.7. At each degree 100 force torque readings are logged. This way we have readings to compensate for position and measurement noises. During the algorithm I address the forces and the torques separately.

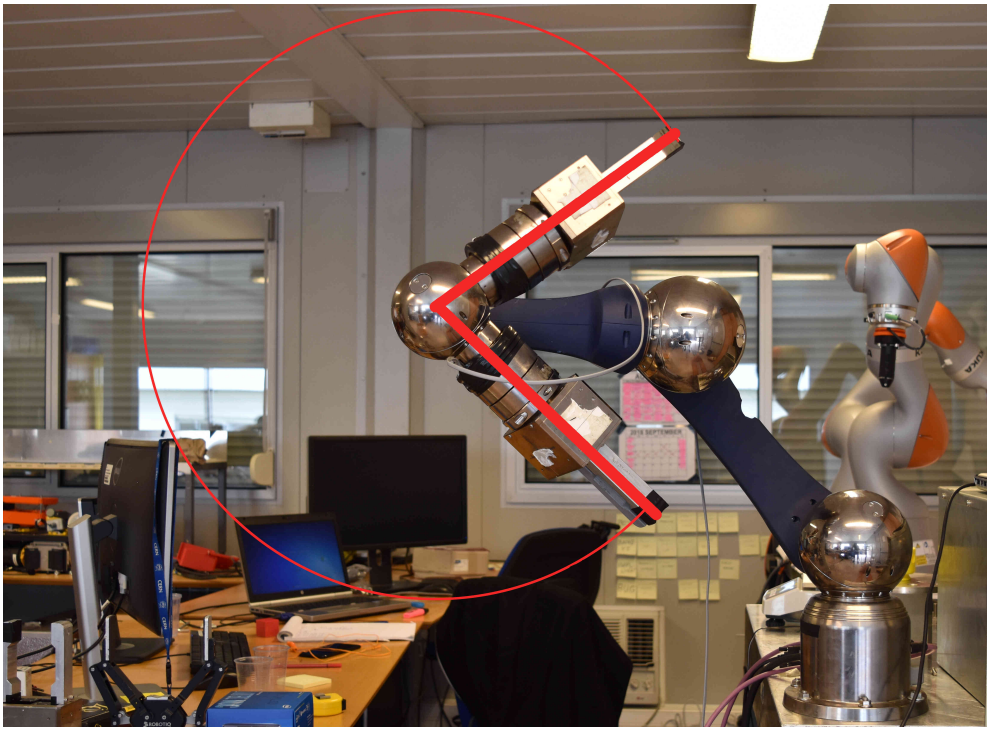


Figure 3.7: The calibration curve

3.4.1 Identifying the Force Components

The sensor provides 3 readings of force: F_x, F_y, F_z. There are 4 parameters needed to be identified, the three biases F_{xb}, F_{yb}, F_{zb} , and the mass m of the end-effector. Regardless of the orientation of the arm the end-effector mass exerts a constant force $F_g = m * g$ on the sensor. The gravitational force is always parallel to the Z coordinate of the robot base, but in the sensor coordinate system its components depend on the orientation.

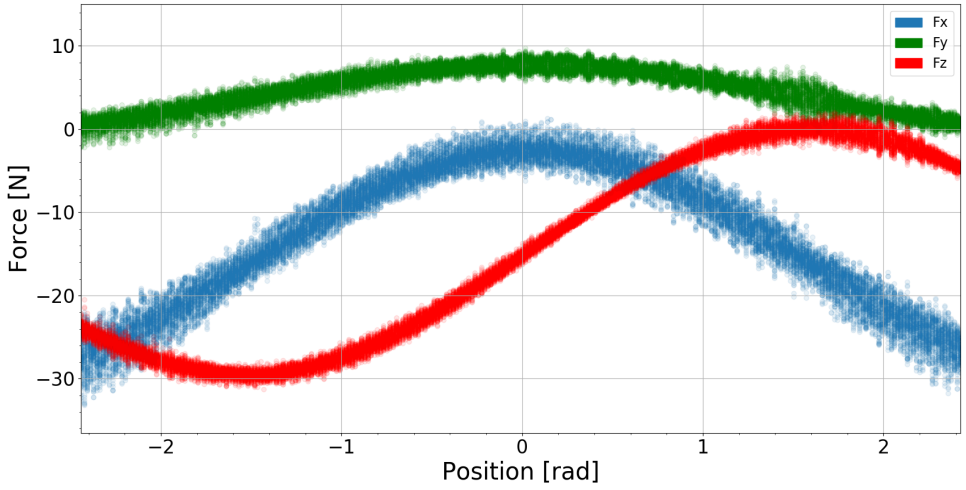


Figure 3.8: The noisy readings of the force values in the sensor coordinate system

The component of the gravity is constant size and direction in the world coordinate system. If we can identify its value in world Z coordinate, it is easy to compensate for it given the pose of the end-effector in sensor coordinates. From the Euler-theorem we know that to transform the vector between the coordinate system 3 rotation around the axis are enough. During the calibration the arm is rotating around Y axis of the robot base in the plane given by $X \times Z$. This rotation creates a force on the 3 axis of the sensor. The readings during the calibration arc can be seen on Figure 3.2. The frequency of the change of the 3 components are the same, and it is equal to the frequency of the arm movement. Because these components are the linear combination of sinusoidal signals with the same frequency, according to the harmonic addition theorem (30) the underlying function can be expressed in the form:

$$F(x) = A * sin(\omega x + \varphi) + O \tag{3.6}$$

- F(x) the force component

- A amplitude of the sine wave
- $\omega = 2\pi f$ angular frequency of the wave, known constant
- x position of the arm in radian, the independent variable
- φ phase shift of the wave
- O constant offset of the wave, this is the bias of the readings

For the optimization process it is helpful to have the Jacobian of the function. The Jacobian of the function can be expressed:

$$\nabla F(x) = \begin{bmatrix} \frac{\partial f}{\partial A} \\ \frac{\partial f}{\partial \omega} \\ \frac{\partial f}{\partial \varphi} \\ \frac{\partial f}{\partial \emptyset} \end{bmatrix} = \begin{bmatrix} sin(\omega x + \varphi) \\ A * cos(\omega x + \varphi) * x \\ A * cos(\omega x + \varphi) \\ 1 \end{bmatrix} \tag{3.7}$$

From the 3 readings we can also calculate the amplitude signal of the force. Although the amplitude of the force is constant, because of the biases the second norm of the readings will be sinusoidal. It is also important to notice that the composition of the three forces usually amplifies their noise.

$$G(x) = \sqrt{F_x^2 + F_y^2 + F_z^2} \tag{3.8}$$

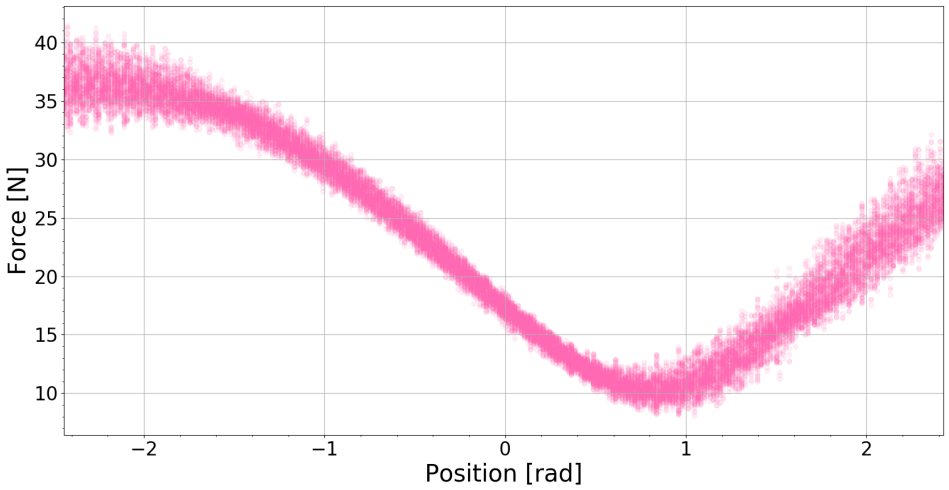


Figure 3.9: The second norm of the force readings

We have 4 sets of points. We know that each set of measurement point comes from a sinusoidal function, and the equation of gravity acts as an added constraint. We can create an $\mathbb{R} \to \mathbb{R}^4$ equation system, with the independent variable of position. It must be noted that the arc does not form a full sinusoidal wave. We only get a limited sequence because of the joint limits. In order to obtain a full wave we would need to move more then one joint, which would add a huge amount of noise coming from the positioning error.

The system can be solved with a least-square optimization algorithm, such as the Levenberg-Marquardt method mentioned in Chapter 2.5.3. For parameter initialization we can guess the amplitude of the waves from the standard deviation of the data point, in the form $A_{guess} = \sqrt{2} * stdev$. We can guess the bias as the mean of the data points, and simply set the initial phase to 0.

The equation system to be solved:

$$F_x(x) = A_x * sin(\omega x + \varphi_x) + O_x \tag{3.9}$$

$$F_y(x) = A_y * sin(\omega x + \varphi_y) + O_y \tag{3.10}$$

$$F_z(x) = A_z * sin(\omega x + \varphi_z) + O_z \tag{3.11}$$

$$G(x) = \sqrt{F_x^2(x) + F_y^2(y) + F_z^2(z)} \tag{3.12}$$

From the Equations 3.9, 3.10 and 3.11 come the biases F_{xb}, F_{yb} and F_{zb} respectively. From Equation 3.12 the actual force can be calculated simply by subtracting the offsets of the 3 component functions.

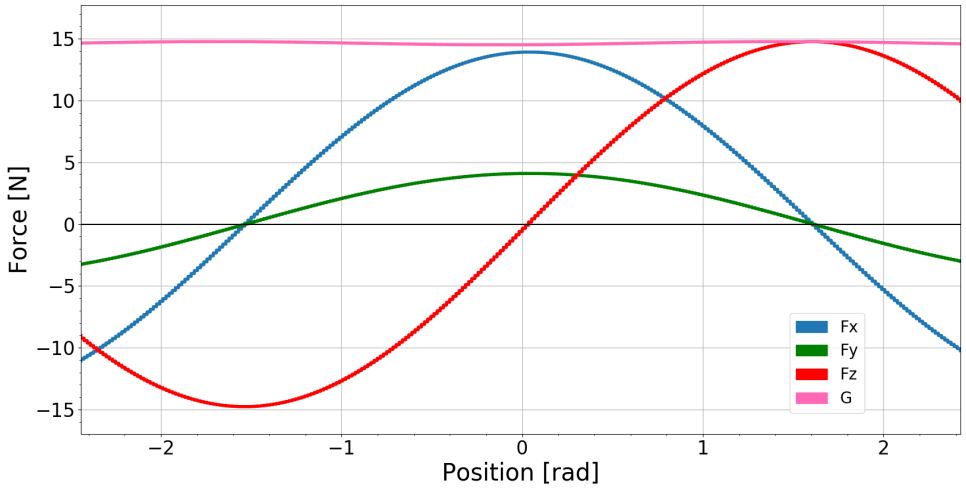


Figure 3.10: The 4 functions compensated for bias

From the bias compensated functions the G can be calculated taking the mean of the second norms of the 3 component sinusoids. It is also important to remark that regardless of the orientation of the X and Y axis of the sensor the maximum and minimum of $F_z(x) = G$. This is because the end-effector is moving in the vertical plane, as can be seen on Figure 3.10. The previous relation is true because the end-effector is truly moving in the XZ plane. The XZ plane is the most precise plane of movement in case of the Schunk Powerball, since the orientation of the plane is only modified by Joint1, which is the strongest and most stable joint. The relation of the amplitudes of the $F_x(x)$ and $F_y(x)$ depends on the angle θ they enclose with the base coordinate system when the end-effector is in vertical position, the relation is constant during the calibration.

3.4.2 Identifying the Torque Components

Torque can be considered as a rotational force. The size of the torque depends on the magnitude of the force creating it and its lever arm.

The vector equation of the torque:

$$\boldsymbol{\tau} = \boldsymbol{r} \times \boldsymbol{F} \tag{3.13}$$

The magnitude of a torque component:

$$\tau = \|\boldsymbol{r}\| \, \|\boldsymbol{F}\| \, sin(\theta) \tag{3.14}$$

- $\boldsymbol{\tau}$, the torque vector in [Nm]

- $\boldsymbol{F}$, the force vector in [N]
- $\boldsymbol{r}$, the lever arm vector vector in [m]
- $\boldsymbol{\theta}$, the angle between the lever arm and the force component [rad]

Inside the rigid body we have a single active force, the gravity, which is applied in the center of mass of the object. The torque applied on the force-torque sensor comes from the weight of the end-effector and the lever arm. The lever arm is the vector which connects the center of mass of the end-effector with the middle point of the sensor.

The torque of the body can be expressed with the force and a 3×3 inertia tensor in general case. If the end-effector is mirror symmetric 1 eigenvalue of the inertia tensor is 0. If the end-effector is rotational symmetric 2 eigenvalues are 0. Generally end-effectors are mirror symmetric, and are close to rotational symmetric. This means that center of mass is diagonal to the surface normal of the torque sensor. The torque component exerted by gravity can be modeled with a single parameter. This parameter is the distance of the center of mass and the torque sensor.

Identifying the torque components is very similar process to identifying the forces. The torque readings also have constant biases T_{xb},T_{yb} and T_{zb}. The orientation points visited during the force calibration are perfect for calibrating the torques too. So we do not need to do any additional movement, no positioning noise is added. The components of torques are also sinusoidal in the sensor coordinate system. The toque measurement readings during the calibration arc can be seen on Figure 3.11. Because the end-effector is close to rotational symmetric the T_z component is close to zero. At every orientation the Z axis of the sensor coordinate system is orthogonal to the weight. Unfortunately, here we do not have the linear constraint of Equation 3.8. The magnitude of the torque vector changes based on the rotational angle compared to the horizontal plane of the world coordinate system.

Because there is no additional constraint, the maximum torque value is estimated from a single point. This obviously makes the method less robust to calibration noise.

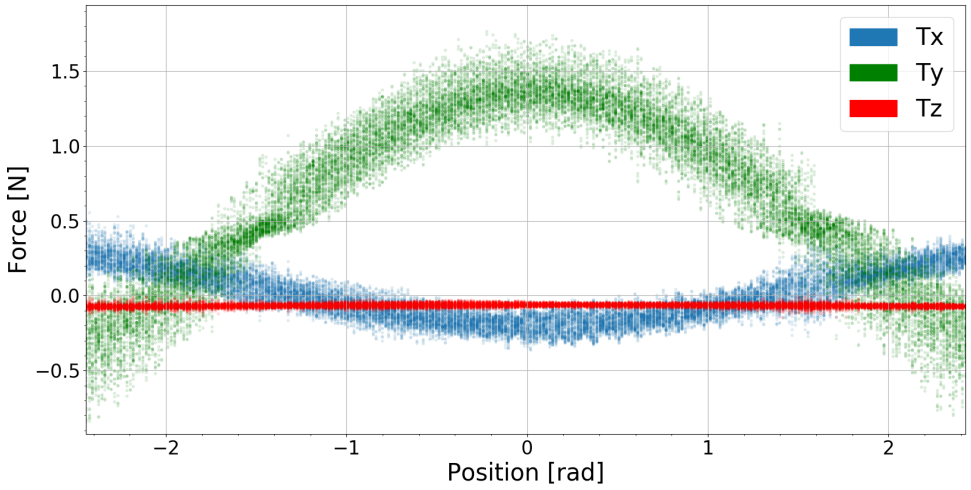


Figure 3.11: The three components of the Torque

It is important to note that torque readings are around one magnitude smaller than force readings. It is also evident that the weight of the end-effector is on the same scale as the usually encountered external forces. On the other hand, because center of mass of the end-effector is in practice very close to the base, the lever arm of the weight is small. This results in an around one magnitude smaller torque coming from the end-effector then usual contact torques. This makes torque compensation much less sensitive, since its effect does not reduce the dynamic threshold of the sensor more than 10%.

To identify the biases of the torque components we can simply fit the general sinusoid on the readings from Equation 3.6. For the least-square regression I have yet again used Levenberg-Marquardt method. The frequency of the signals is the movement frequency, and the amplitude can be initialized as $\sqrt{2} * stdev$. From the fitted sinusoids we can extract the bias, they are equal to the offset of the waves. To identify the length of the lever arm I have calculated the second norm of the torque components. The maximum of the second norm will be the maximal torque of the end-effector.

$$L = \frac{1}{G} * max(\sqrt{T_x^2 + T_y^2 + T_z^2}) \tag{3.15}$$

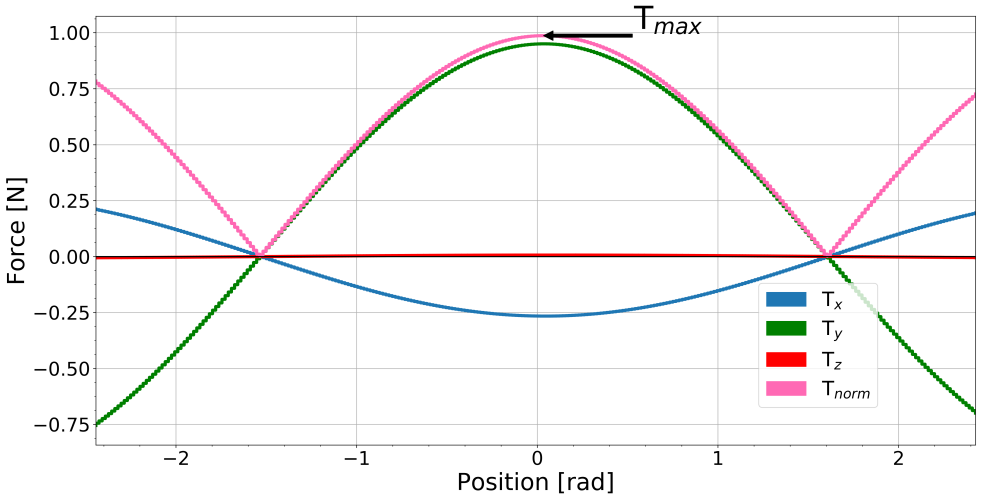


Figure 3.12: The fitted Torque components and the norm

On Figure 3.12 it can be seen, that my assumption of rotational symmetry stands. Since $T_z(x)$ has almost zero amplitude, it can be approximated by a constant. The precision of the numerical maximum value estimation depends on how close the phase of the waves are. If the phase is almost the same the estimate will be good, if the phase differs between components it means that the estimate is noisy. On Figure 3.12 the 3 component waves have the same phase.

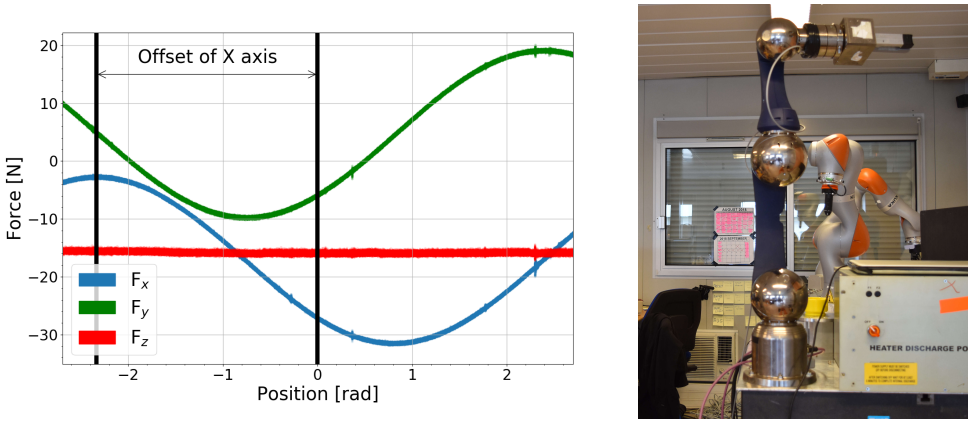
3.4.3 Compensation Based on the Orientation

After measuring and calculating the biases, the weight and the lever arm in the previous chapter we can proceed with the compensation of the measurements.

The compensation depends on the orientation of the end-effector. The orientation can be obtained from the forward-kinematics model of the robot, which I have described in Chapter 2.4. Unfortunately, the sensor coordinate frame does not coincide with the frame attached to the last joint of the robot. The axis Z of the joint and the sensor, and the center point coincide, only their XY frames are rotated.

In order to measure the rotation angle θ of the axis, I have set up an experiment. In the position depicted on Figure 3.13b, I have rotated the sensor around axis Z and measured the forces. I had to find the location in the joint frame where the X axis measurement reaches its minimum. During the force measurements the last axis is moved slowly to minimize the dynamic noise. After obtaining the data I have fit a cosine function on the F_x^i points with Levenberg-Marquardt

method. The phase shift of the cosine function is the offset between the two X axis.



(a) Force readings (b) Calibration position

Figure 3.13: Finding the offset of the X axis

The algorithm for compensating the bias:

1. Get readings, compensate bias and rotate to match the final joint and sensor coordinate systems:

$$\boldsymbol{F_1} = {}^{S}R_{J6} * (M_{calibration} * \boldsymbol{SG} - \boldsymbol{B}) \tag{3.16}$$

2. Calculate Gravity vector in Sensor coordinates

$$\boldsymbol{F_g} = R_{fk}^{T} * \begin{bmatrix} 0 \\ 0 \\ G \end{bmatrix} \tag{3.17}$$

3. Calculate torque components in sensor coordinate system:

$$\boldsymbol{\tau_g} = \boldsymbol{F_g} * L \tag{3.18}$$

4. Compensate for the end-effector forces and torques

$$\boldsymbol{F_c} = F_1 - \begin{bmatrix} \boldsymbol{F_g} \\ \boldsymbol{\tau_g} \end{bmatrix} \tag{3.19}$$

5. Transform the compensated values from the coordinate system of the last joint to the base coordinate system

$$\boldsymbol{F} = R_{fk} * \boldsymbol{F_c} \tag{3.20}$$

3.5 Compensating the Thermal Drift

During my initial experiments with the force-torque sensor I had to realize that the sensory measurements depend on time. This time dependence is highly correlated with the temperature of the force-torque sensor. According to my experiment the measurement values could change up to ±30 N for components. The output values reached steady state value after four hours without operation of the arm. With movement of the arm I was unable to reach a steady-state, but based on the data I suspect that there is a steady state for the dynamic system, simply the time constant is too big to reach it. The magnitude and the time of change was too much to simply ignore. I have contacted Schunk Ag, who have verified the sensor and found no problem. After this incident I had to perform the heat calibration of the sensor myself to be able to use it for measurements.

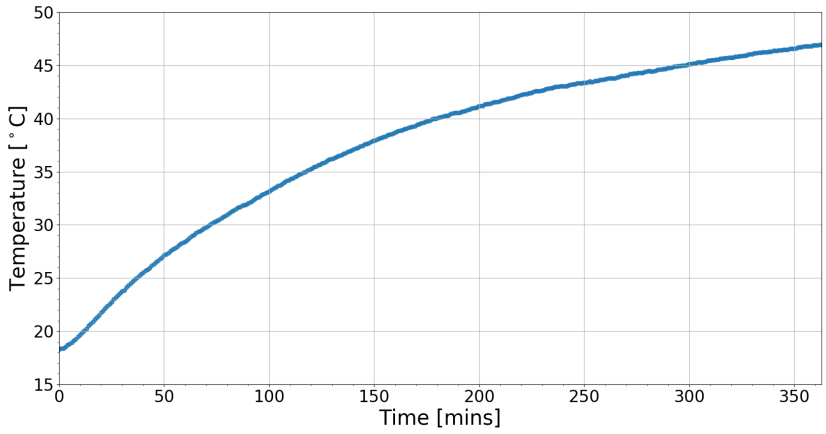


Figure 3.14: The temperature of the Force-Torque sensor

The sensor has a built-in temperature sensor, but it is only for overheating protection. It only returns whether the sensor is exceeding the safety temperature limits of 120 °C or not. For my purpose I needed detailed readings, so I had to create my own measurement circuit.

I have decided to use the DS18B20 temperature sensor (22). It can measure temperatures between -55 °C and +125°C, with a resolution of 0.06 °C. The data sheet claims that the measurement has ±0.5 °C accuracy. According to my experiments the measurement has a far better accuracy of ±0.06. The only disturbance that manifests is close around the transition state, there the output can fluctuate. The main advantage of the sensor is its 1-wire protocol. It sends the data through a single wire digitally. There is no need for a an analog-digital

converter to convert the measurement values. The precision is also good with a simple pull-up resistor, there is no need for a Wheatstone bridge.

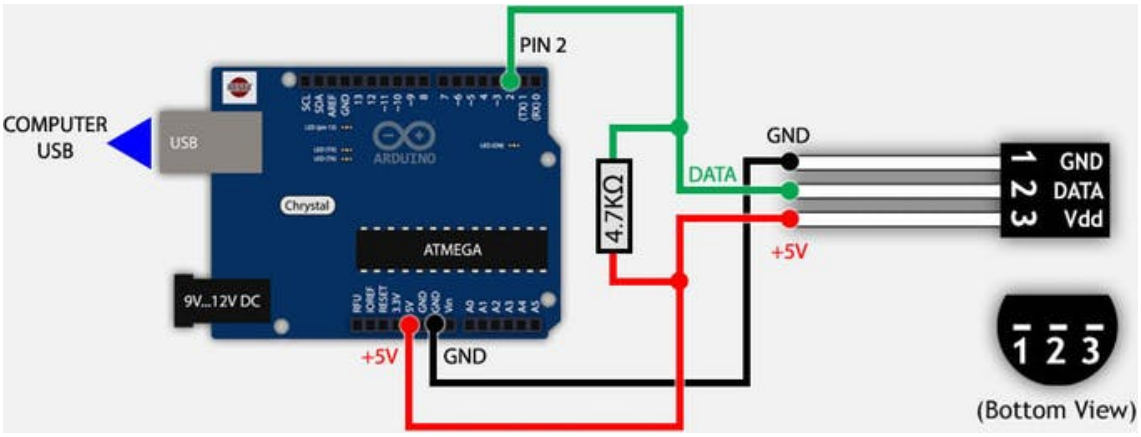


Figure 3.15: The temperature sensor set-up (17)

I have fastened the sensor on the base of the force-torque sensor with the help of silicon glue and industrial tape. Then connected the sensor to an Arduino parallel with a pull up resistor. The circuit arrangement can be seen on Figure 3.15. The Arduino reads the temperature values from the sensor, then transfers the readings onto the robot controller through serial connection with 100 Hz frequency.

In order to be able to compensate for the heat drift I have performed experiments to establish the correlation between the output values and the temperature of the sensor. In each experiment I have consecutively performed the previously described bias compensation algorithm for four hours, while collecting the data of the measurement values and the temperature. Each experiment consists of hundreds of times performing the calibration arc.

To rule out the dependence of the values on the end-effector I have done a control experiment without any end-effector, and experiments with two different end-effectors of Figure 3.6. In order to establish the stability of the readings I have performed each experiment three times. During each experiment I have waited for the whole robotic arm to cool down to room temperature, and then performed the readings until the temperature of the force-torque sensor rose from room temperature to around 45 °C. Unfortunately, there was no easy way to cool down the arm, so each experiment took four hours to perform, and then one day for the arm to cool down. The starting temperature varies between experiments by a few degrees.

For each calibration arc I have performed the fitting of the sinusoid function the way I did in Chapter 3.4.1 and 3.4.2. The sinus functions have 3 parameters:

amplitude, phase and offset. Without the heat drift the phase and offset values should be constant between all 3 end-effector experiments, but the amplitude would change between them based on the weight of the end-effector. To check how the values change with temperature I have calculated the standard deviation of the 3 parameters for the 6 components. If the standard deviation is small, it means the heat has no effect on the measurement. If the standard deviation is big then the heat can be affecting the parameters.

	F_x	F_y	F_z	τ_x	τ_y	τ_z
Amplitude	0.0167	0.0041	0.013	0.0011	0.0012	0.0001
Phase	0.001	0.0006	0.0024	0.0023	0.003	0.0024
Offset	1.2205	0.2862	8.7281	0.0326	0.0071	0.0043

Table 3.2: Standard deviation of the sinusoid parameters

From Table 3.2 it can be seen that the phase does not change for the measurements. The change of amplitude and bias has to be inspected better. I have plotted the standardized amplitude of the measurements, the mean was subtracted from each value. On 3.16 it can be seen that the amplitudes do not change considerably with temperature, the change can mostly be attributed to noise.

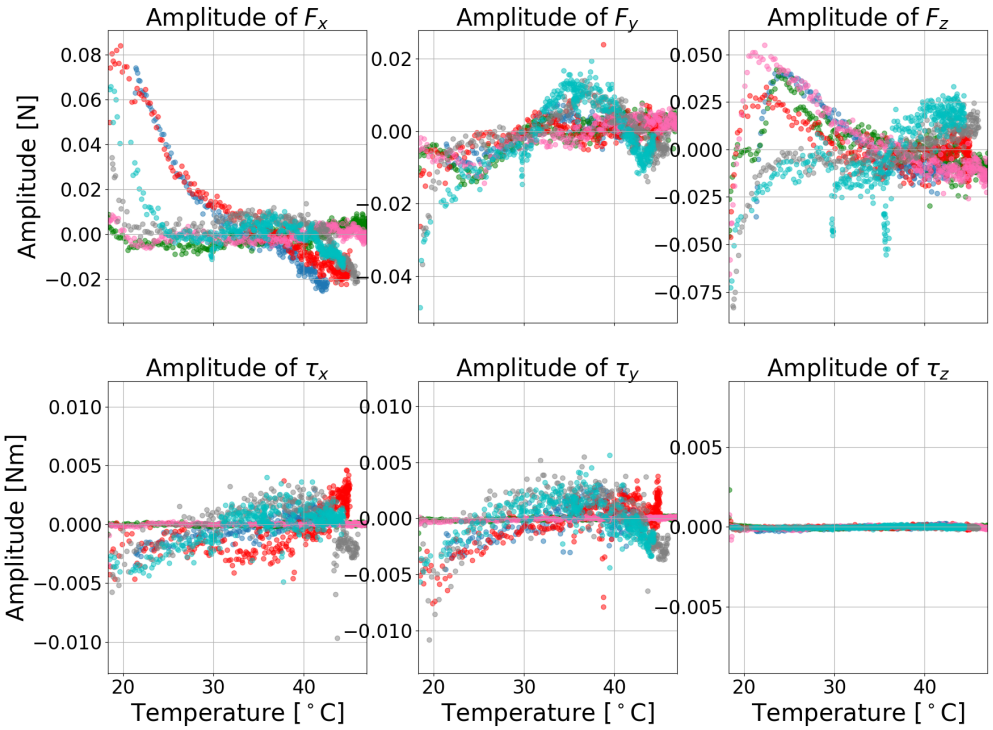


Figure 3.16: The relationship of standardized amplitudes and temperature

On Figure 3.17 the offset of the sinusoids is plotted. The offset corresponds to the bias of the sensor measurements. It can be said that the change of measurements can be attributed to the heat drift of the bias.

Since the amplitude of measurements does not depend on the temperature the calibration process can be done at any temperature. The calibration is invariant to the bias, so if the relationship of temperature and bias can be identified the measurements can be compensated during operation. This means that every time the robot arm is switched on the calibration parameters can be read from a file, and the arm does not have to perform the calibration arc. I only had to identify the relation of bias and temperature.

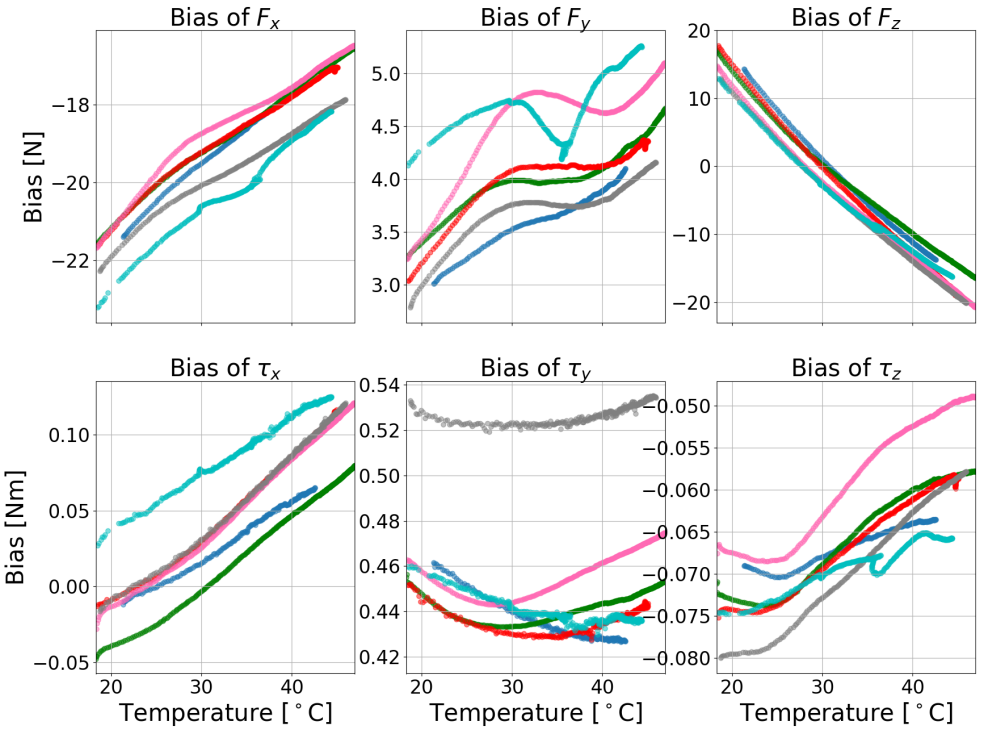


Figure 3.17: The relationship of biases and temperature

Regression analysis is the collection of methods for establishing the relationship of variables. I used regression analysis to find a model for the data. As can be seen on Figure 3.17 the bias does not depend too much on the end-effector used. I only had to establish the relationship of each bias component with the temperature. Between experiments the curves can be very different, there is not a single function which realizes the connection between heat and bias. The biggest similarity is in the trends they follow. I decided to make the heat compensation in a way that the robot arm calibration could by done only once when the robot is turned on. Later the bias can be corrected based on this single measurement with fitting a trend.

The bias of τ_y and τ_z only change slightly. The change would not be noticeable for the operator so these components do not have to be compensated. The bias of F_x, F_z and τ_x follows a linear trend. Although the bias of F_y does not follow a linear trend it can be approximated with it. This was because this component was changing too much between experiments, but to explain the overall trend a linear fit seems good.

Unlike the problems of inverse kinematic and calibration here the model is linear. It is a simple line with the equation : $y = ax + b$. There is no need for non-linear optimization methods, there is an explicit solution for the parameters. I have collected all the data points across the experiments and fit the lines by component. To evaluate how well the data fits on the regression line I have used two goodness-of-fit measures. These are Coefficient of Determination (R^2) and Standard Error of the Estimate (S).

R^2 evaluates how well the data can be explained by the regression function compared to simply taking the mean of the data points, and weights it with the variance of the data. The value of R^2 is between 0 and 1. The value 1 means that the data perfectly fits on a line, while a value of 0 means the data is not linearly correlated.

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}} = 1 - \frac{\sum_i (y_i - f_i)^2}{\sum_i (y_i - \bar{y})^2} \tag{3.21}$$

- R^2 coefficient of determination, value is between 0 and 1
- SS_{res} sum of the squared difference between predicted and measured value
- SS_{tot} variance of the data
- y_i,$\bar{y}$ measured data points, mean of data points
- f_i predicted data point

R^2 gives a normalized score of how good the fit is. Makes it possible to judge the line fit. Standard Error of the Estimate on the other hand gives a measure how far a measurement point is on average from our model. It is a much better measure of showing that during the measurements with the sensor how much variance is added because of the bias compensation.

$$\sigma_{est} = \sqrt{\frac{\sum (y_i - f_i)^2}{N}} \tag{3.22}$$

- σ_{est} standard error of estimate, a positive number
- y_i measured data points
- f_i predicted data point
- N number of data points

After fitting the lines to the data I have calculated R^2 and SSE. The results can be seen in Table 3.3. As expected the functions of $F_{bx}(T)$, $F_{bz}(T)$ and $\tau_{bx}(T)$ can be approximated well by a line, this can be seen by the R^2 values which are close to one. Although the line fits well on $F_{bz}(T)$, still the variance will be very big and the estimation of this parameter can be imprecise. This is the result of the fact that this parameter changes the most in the temperature range used for calibration.

On the other hand the line fit of $F_{by}(T)$ is not too strong. Still the bias estimation is good enough in our case. According to SSE the mean difference will be less then 0.5 N. The components $\tau_{by}(T)$ and $\tau_{bz}(T)$ does not fit the line well. On the other hand because their magnitude is very small, the error even with a bad fit will be small, less than 0.02 Nm.

	$F_{bx}(T)$	$F_{by}(T)$	$F_{bz}(T)$	$\tau_{bx}(T)$	$\tau_{by}(T)$	$\tau_{bz}(T)$
R^2	0.9795	0.4885	0.9689	0.8238	0.05827	0.667
SEE	0.1823	0.3249	1.6269	0.0173	0.01242	0.0038

Table 3.3: Evaluation of the line fits.
Using evaluation metrics of R^2 and SSE.
Columns are by the lines fitted on each force component

3.6 Evaluation of Gravity and Bias Compensation

With the help of gravity, bias compensation and kalman filtering I wanted to achieve, that no matter the orientation and the velocity of the slave end-effector, the force-torque readings should be close to zero. In this case, no matter how small the contact force is, it is possible to detect that the robot came into contact with a surface. The sensor's dynamic range would be maximized.

Of course there are lots of effects which add a noise to the readings. Some of these effects I have described in the previous chapters. It is important to know how well the compensation algorithm performs. How much is the noise amplitude around the zero point when there is no contact force. By establishing this range it is possible to declare, that when we received a reading above this threshold we have clearly made contact with an object.

To establish this threshold I have performed two sets of experiments. In the first experiments I wanted to evaluate the compensation in steady-state, without taking into account the heat drift. I performed the calibration of the sensor, and right after I have moved the end-effector to different orientation. The orientation

mostly correspond to the normal operation area of the slave robot. The speed of the movement was also set in a way to imitate the circumstances of a real intervention. Each experiment lasted 120 seconds. In this time frame the heat drift's effects are negligible. I have tested 3 different end-effectors, on 3 different temperatures.

The results are shown in Tables 3.4, 3.5 and 3.6. The mean of the measurements shows how well the bias is compensated. This happens because it can be assumed that the movements are general enough to sample from all possible orientations uniformly. The standard deviation shows the quality of the gravity compensation mostly. Again because we sample uniformly from the orientation. The standard deviation is influenced by the change of the values. As previously seen, because the dynamic noise has zero mean it is eliminated from the standard deviation. The change of the magnitude of the force comes from the change of the force components because of the orientation. This comes from the end-effector weight. Finally, the minimum and maximum values depict the efficiency of the filter. The values of the error all have normal distribution.

	F_x	F_y	F_z	T_x	T_y	T_z	**F**	**T**
Mean	-0.15	0.01	0.26	0	0	0	0.52	0.01
Stdev	0.32	0.21	0.23	0	0	0	0.16	0.0
Min	-0.97	-0.5	-0.31	-0.01	-0.01	-0.01	0.07	0.0
Max	0.75	1.03	0.84	0.01	0.01	0.01	1.09	0.01

Table 3.4: Evaluation without end-effector

	F_x	F_y	F_z	T_x	T_y	T_z	**F**	**T**
Mean	0.02	1.2	-0.53	0.07	-0.02	-0.03	1.65	0.11
Stdev	0.81	0.67	0.43	0.05	0.06	0.05	0.55	0.04
Min	-3.17	-1.37	-2.4	-0.27	-0.26	-0.26	0.1	0.0
Max	2.46	3.72	1.32	0.32	0.27	0.11	4.44	0.42

Table 3.5: Evaluation of Robotiq 140 gripper
weight is 11 N

	F_x	F_y	F_z	T_x	T_y	T_z	**F**	**T**
Mean	-0.69	0.38	-0.7	0.01	0.05	-0.01	1.92	0.15
Stdev	1.04	1.27	0.62	0.12	0.08	0.05	0.71	0.05
Min	-3.78	-2.84	-2.53	-0.38	-0.29	-0.29	0.03	0.01
Max	2.42	3.84	1.59	0.5	0.42	0.25	4.14	0.53

Table 3.6: Evaluation of Schunk PG 070 gripper
weight is 14 N

In the static case it can be concluded that the system works well. The torque values are compensated almost perfectly. From the tables it can be seen that the mean of the errors are very close to zero, around 0.5 N. This means that the bias compensation works very well. In general because the errors follow normal distribution, it can be said that 95% of measurements fall inside a two standard deviation range. This means that only 5% of the measurements have an absolute error over 1.4 N. This is a very good result if we consider that this is around 10% of the weight of the grippers.

To check how well the heat compensation of the bias performs I have made 2 experiments. During the control experiments I have not used any grippers, just the force-torque sensor. For the second experiment I have used the Schunk PG 070 gripper, because it had produced consistently a bigger error during the steady state evaluations. During both experiments I have performed the calibration of the arm on room temperature. I had made movements emulating a normal teleoperation scenario for 2 minutes. I have done this movement phase twice for each experiment. Once close to room temperature, and once when the arm heated up. This way a good estimate of the precision of the system could be established.

After the experiments I have concluded that the error again follows normal distribution. The variance, minimum and maximum points were similar to the steady state case. They do not convey any additional information, for this reason I have omitted them from further analysis. The interesting results were in the change of the mean of the error. This shows how well the calibration is done. The results can be seen in Table 3.7.

	F_x	F_y	F_z	T_x	T_y	T_z	F	T
Control at 23 C	0.41	1.12	-1.05	-0.01	-0.04	0.01	2.0	0.05
Control at 38 C	-0.01	0.64	0.96	0.01	-0.03	0	1.44	0.05
PG 070 at 20 C	-0.6	-0.17	-3.59	0.01	0.02	-0.01	4.24	0.06
PG 070 at 33 C	0.04	-0.16	-0.19	0	0.02	0	1.09	0.07

Table 3.7: Evaluation of the effect of heat bias compensation

Between the steady state calibrations there was no meaningful difference in the

value of bias. The calibration process I have outlined is indifferent of temperature. Here however it can be seen that the mean changes based on temperature. The change is between 0 and 3.6 N. This means that the bias correction is not perfect, as expected. During 3 measurements the precision is only slightly worse then after steady state calibrations. However there is one measurement where the mean error is 3.6 N.

In conclusion the heat drift compensation is not perfect. It can significantly reduce the dynamic range of the sensory measurements. However this is a hardware problem, which can not be fully mitigated. Most importantly though with the heat compensation the calibration process loses its generality. For each force-torque sensor the heat curves must be established independently before operation. As mentioned before this process can take several weeks. In my opinion the best way to solve this problem is to simply acquire a sensor which is not plagued by this flaw.

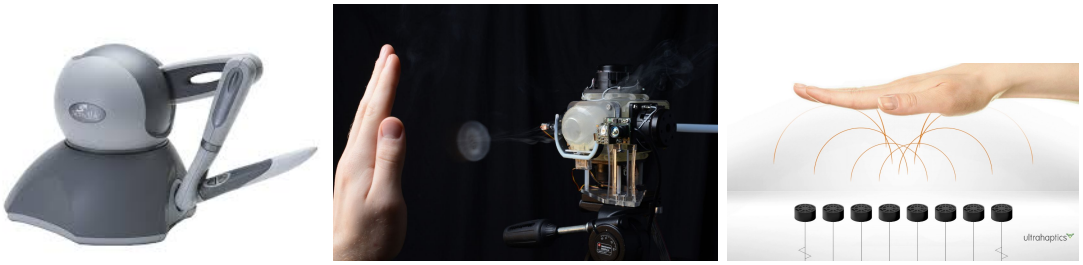
4

Creating the Force-Feedback

4.1 Creating Haptic Feedback

Haptic devices are output devices, which are meant to influence our sense of touch. Unlike the visual output devices and audio devices, such as monitors and speakers, haptic devices are not as wide spread. This can be attributed to the fact that our sense of touch in most cases does not convey as much information as vision and hearing. Additionally it is much harder to implement haptic output, since during classic tactile feedback the device has to make direct physical contact with the skin.

The most widespread haptic feedback devices use vibrations. They usually use eccentric rotating mass actuators (ERM) (4), which consist of a motor and an unbalanced weight. Another family of actuators is called Linear Resonant Actuators (LMA). LMA actuators are based on the interaction of AC current magnetic field and a permanent magnet. These motors have small size and can be fit into wrist watches and mobile phones. Their main drawback is that their haptic resolution is minimal. They can only signal a few different states, such as vibrating less or more.



(a) Phantom Omni (b) AIREAL (c) Ultrahaptics

Figure 4.1: Examples of haptic feedback devices

The most articulated way to create haptic feedback is by the usage of force-feedback. During force-feedback, trough a mechanism forces are exerted on the user. Unlike the technology based on vibrations, force-feedback can create a much detailed, lifelike sensation for the user. They have been in use in flight and car simulators for over decades. These simulators are usually big and expensive. Newer devices, such as the Phantom Omni (27) can be used for CAD modeling and graphic design. Their disadvantage is that they need sophisticated mechanisms with several degrees of freedom. Each degree of freedom needs a separate actuator.

Recently several efforts have been carried out to create non-contact haptic devices. Disney Research has created AIREAL (28), a device based on air vortex rings. A recent start-up company Ultrahaptics (9) have created a device based on the use of ultrasound vibrations. These devices are unfortunately still at an early stage of development.

4.2 Methods for Realizing Force-Feedback

4.2.1 Impedance Control of Robotic Manipulators

In order to create the force-feedback in my project I decided to use the Kuka iiwa LBR robotic arm. The robotic arm was created to work in cooperation with human workers. It is equipped with several types of impedance control mode (15).

Impedance control is a method of controlling a robotic arm, which utilizes the dynamic interaction between the environment and the robotic arm. As opposed to the normal velocity and position control, where the outside forces acting on the manipulator are simply considered disturbances, such as back EMF voltage. These effects are filtered out and not taken into account while performing movement. When there is a collision, for example the robot arm hits a worker, from the point of view of classic control the robot still tries to minimize the distance to the desired position and will simply try to force through the object of collision. This behavior makes industrial robots very dangerous to humans. This is the reason why workers have to be separated from robots with work cells in factories.

In electrical engineering notation impedance is the ratio of output voltage and input current. The mechanical impedance is an analogous measure, it relates the output of forces to the input of motion. The mechanical impedance can be modeled with springs and dampers. The spring relates the force output to the distance difference between the current position and the programmed position. The damper relates the forces to the difference between current velocity and the desired velocity.

The inverse of impedance is admittance, it relates the output motion to the forces. If the robot applies force on the environment it will result in the movement of the objects of the environment. The environment acts as an admittance with a mass.

Dynamic differential equation of motion:

$$F = M * \ddot{x} + K * \dot{x} + S * x \tag{4.1}$$

- F force exerted $[N]$
- M object mass $[kg]$
- K spring stiffness constant $[Nm]$
- S damping coefficient $[\frac{Ns}{m}]$
- $\ddot{x}$, $\dot{x}$, x the acceleration, velocity and position

During impedance mode the robot and environment creates a unified control system. the composition of the two can be thought of as a mass-spring-damper system. This system can be modeled by the relation of impedances and admittances. The impulse of the mass and the spring stores energy, while the damper dissipates it. By controlling the impedance of the robotic arm it becomes possible influence how much energy is exchanged, and in result how much mechanical work is being done. This is a very important relationship, since the energy the system exerts gets transfered during a collision from the robot to an object. By controlling the impedance we can set safety limits for operation, and hence the robot becomes capable of working in a common space with the human operator. The safety energy limits of impedance control are described by the ISO Norm 15066 (2).

There are 3 different impedance control schemes built into Kuka iiwa:

1. Cartesian Impedance Mode
2. Cartesian Sine Impedance Mode
3. Joint Impedance Mode

All 3 modes work by creating force-feedback through virtual springs and dampers. With Cartesian Sine Impedance it is possible to set additional control forces.

4.2.2 Virtual Springs and Dampers

Virtual springs are a way to create force-feedback between the user in a safe-way. The virtual spring acts as an imaginary spring between a point located on the robot and a point is space. The virtual spring creates a force at the end-effector according to Hooke's Law. The force is based on the distance between the two end points of the spring, and the spring stiffness.

Hooke's Law for linear springs:

$$F_s = -k * x \tag{4.2}$$

- F_s the spring force in [N]
- k the spring stiffness in $[\frac{N}{m}]$
- x spring displacement in [m]

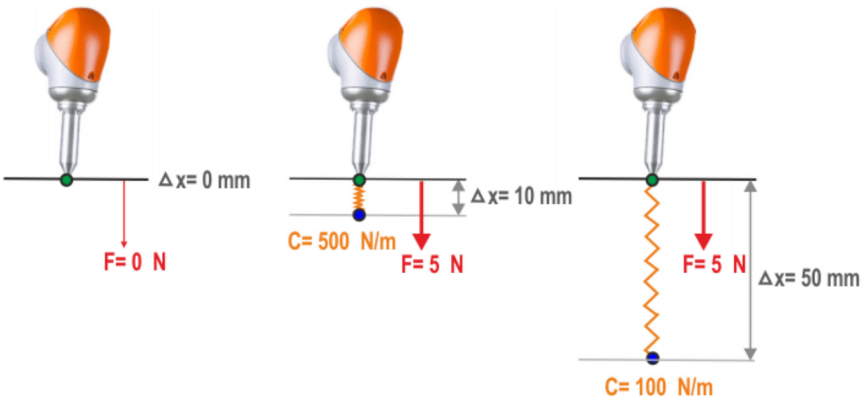


Figure 4.2: Examples for calculating the force exerted by the end-effector

The advantage of using virtual springs is their safe operation even in the close proximity of an operator. When the user modifies the path of the end-effector the end-effector applies linearly rising force-feedback. This way the impulse and the acceleration of the end-effector stays low, it can not harm the user.

The control modes allow the creation of dampers on the virtual springs. The dampers can be parametrized by the damping coefficient. The operation modes of the Kuka iiwa robot only allow the impedance control based on position. For the velocity only the relative maximum speed can be specified. Unfortunately without the ability to set the velocities the damping provides not enough resolution for my project.

The difference between Cartesian and Joint impedance modes, is that in joint mode the virtual springs are attached between the joints. In Cartesian Impedance

mode the strings are attached to the end-effector. In Cartesian Impedance mode there are 6 springs each corresponding to 1 degree of freedom of a rigid body. Since in the slave robot the forces are sensed at the end-effector and transformed into Cartesian space, the Cartesian Impedance mode was the most straight forward to use in my application.

It must be noted that Cartesian Sine Impedance mode allows the direct application of force to the end-effector additionally to virtual springs. Using forces and simply mapping them directly between the end-effectors was a very promising development option. Unfortunately however, according to my experiments, this mode is simply not reliable enough. When changing the output forces the controller always jumps to zero and only then increases the value to the desired output. So if the slave reads 16 N and then in the next time step reads 16.05 N, the Kuka robot could only do the sequence of 16 N $\rightarrow$ 0 N $\rightarrow$ 16.05 N. This causes vibrations of the end-effector and the constant switching between torques overloads the motors of the robot.

For this reason I have chosen to use Cartesian Impedance Mode and solely used virtual springs for force feedback.

4.2.3 Creating Forces with Virtual Springs

The impedance mode was mainly implemented in the Kuka iiwa for safety purposes. In the general use case of impedance mode, the programmer provides the desired trajectories of the robot arm end-effector. The robot follows the given trajectory, while the it can be slightly altered by contact of the operator or other constraints.

On the other hand impedance mode can be used to hand guide the robot. The big difference is the fact that the robot here has no predefined trajectory. The user can hold the end-effector and guide it through a path freely. Without counterbalancing the weight of the arm the operator would have to push the full weight of the arm. Since the Kuka arm weighs more then 10 kg, hand guidance would pose a real challenge to most operators. For this reason Kuka has also implemented gravity compensation in their hand guidance, which means that the operator does not have to counterbalance the weight when performing hand guidance. This makes it possible to anyone to use the robot. It can be done with even one finger.

Hand-guidance can be used for performing learning by demonstration. The path of the robot is not programmed explicitly by code, but rather is "shown" to the robot by the operator. This is a much faster way of setting a trajectory of the arm. In my use case I perform teleoperation with the same assumptions as learning-by-demonstration uses.

There are two movement types that allow hand guidance. These are: SmartServo

and DirectServo (3). I think hand guidance must be a very experimental feature of the Kuka iiwa LBR, since neither of these methods are mentioned in the official, otherwise very thorough, documentation. During my work I had to analyze the functionalities that they provide solely from experiments.

Both movement types allow hand guidance with the same underlying logic. The controller updates the desired position of the robot with a given frequency. The position of the robot can be set either by joint positions or end-effector position. When doing hand guidance the user modifies the robot position with impedance mode, and the servo controller simply sets the desired position to the actual position of the robot. This way the robot does not perform individual movement, only movement dictated by the operator. According to my measurements the main difference is their speed. Direct Servo can update the position of the robot with up to 350 Hz. SmartServo can only achieve speeds around 150 Hz. The update frequency of Smart Servo is not fast enough for the operator to perceive it as fluid movement. On the other hand SmartServo provides some additional capabilities to DirectServo, such as the ability to set maximal acceleration of the arm.

To create movement and force feedback with the KUKA robot I have used the combination of DirectServo and CartesianImpedance mode.

The algorithm at each time step is:

1. Read the current end-effector position
2. Calculate the desired output force based on the slave forces
3. Update the desired end-effector position with the current position plus a small distance towards the direction of the force
4. set spring stiffness so that: $k = \frac{F}{\Delta x}$

I have performed experiments to determine whether the real output force by the end-effector matches expectations. Used a Kern PCB 6000-1 precision balance to measure the forces exerted by the robot arm. The difference between the calculated and measured forces had a small offset, but otherwise was precise to the 0.1 N resolution.

4.3 On the Magnitude of the Force Feedback

4.3.1 Tuning the Forces to the Operator

In the previous chapter I have described how to create force-feedback. In this chapter I will try to answer the question how to set its magnitude.

The Kuka iiwa LBR can exert over 500 N. It is not safe to apply this kind of force on the operator. Additionally the Schunk Powerball robot arm can only exert around 100 N force. It is not a good idea to map the forces between slave and master robot one-to-one. The mapping of forces should depend both on the physical capabilities of the robot and the comfort zone of the operators.

In order to determine how to set the magnitude of the force-feedback I have conducted group experiments. The goal of the experiment was to determine which is the smallest force which can be noticed during force feedback. Additionally the maximal forces had to be found with which working still did not prove too hard for the operator.

During the experiments the participants had to stand in front of the master robot and grasp the end-effector. In order to have a base of comparison first they performed simple movements for a minute without force-feedback. This way they got accustomed to the dynamics of the robot arm during hand-guidance. After getting familiar with the robot, force was exerted on one axis. The force was increased gradually, every 100 ms by 0.1 newton. When the participant had noticed the force I have noted it down. The same way, when moving the arm became too uncomfortable I put it on paper as the higher limit. Each axis was measured separately one-by-one. The participant was informed which axis the force was supposed to act, since in teleoperation the operator will expect from video feed which direction the force will apply.

During the experiment I have measured the stiffness directly instead of the force. The results can be seen in Table 4.1 and 4.2.The virtual offset of the spring was constant. For the linear springs 10 mm, for the torsion springs 0.1 radian. The linear springs S_x, S_y and S_z are responsible for creating force components F_x, F_y and F_z .The torsion springs T_a, T_b, T_c are responsible for creating the torque components τ_z, τ_y and τ_x. The torque components are in reverse order because KUKA uses the ZYX Euler angle convention instead of Roll-Pitch-Yaw.

Participant	$S_x[\frac{N}{m}]$	$S_y[\frac{N}{m}]$	$S_z[\frac{N}{m}]$	$T_a[\frac{Nm}{rad}]$	$T_b[\frac{Nm}{rad}]$	$T_c[\frac{Nm}{rad}]$
1	440	630	540	3	4	6
2	1000	700	620	4.4	8	10
3	540	540	1100	10	18	14
4	930	900	1100	9	13	21
5	930	1100	1020	4	17	18
6	730	730	930	3	8	12
7	1200	930	930	4	12	17
Average	**825**	**790**	**890**	**5**	**11**	**14**

Table 4.1: The minimum perceivable stiffness

Participant	$S_x[\frac{N}{m}]$	$S_y[\frac{N}{m}]$	$S_z[\frac{N}{m}]$	$T_a[\frac{Nm}{rad}]$	$T_b[\frac{Nm}{rad}]$	$T_c[\frac{Nm}{rad}]$
1	2900	3200	2900	20	22	30
2	2100	1700	2200	15	50	25
3	3000	2600	2700	24	50	50
4	2500	2800	2800	17	39	50
5	1700	2300	2200	18	58	36
6	1600	2000	2100	18	26	28
7	3600	2400	2800	16	74	42
Average	**2485**	**2430**	**2530**	**18**	**46**	**37**

Table 4.2: The threshold for maximum stiffness

From the experiments it can be seen that people experience the force feedback in a very similar way. The mean of the measurements relates very well to any single participants individual experiences. From both the minimum and maximum values it can be seen that the 3 force components are sensed the same way. Surprisingly however people are much more sensitive to the torsion torque, which can be sensed around axis Z. All the participants except participant 1 had no previous experience with hand guidance or the Kuka iiwa robot. My theory is that if the user has more time to practice with the robot the lower threshold becomes much more apparent. The operator by time becomes more sensitive to the force feedback. However this theory is only based on a single user, it would have to be examined more thoroughly.

Surprisingly the minimal threshold of recognizing the force feedback is relatively high. It is on average around 30% of the range of measurements. I was so surprised by this finding that I had to verify with additionally experiments to decide whether the force is actually exerted on the user. This means that simply by the usage of force-feedback it is impossible to show the operator whether a contact has been made with a soft object. In the future to solve this problem vibration should be used alongside to provide this information to the user.

The high lower threshold in my case has a very beneficial effect. In Chapter 3.6 I have concluded that the zero offset error of the sensor measurements is less then 20% of the end-effector weight. This means that the operator will not notice this error through the force-feedback.

I have decided to map the forces between the minimum and maximum threshold linearly. The maximum force value was set to be the measurement of 50 N on the slave side. The minimum identifiable force feedback would be set to the contact force of 8 N. This way neither the dynamic noise, nor the residual error coming from end-effector wight will be noticeable.

4.3.2 Safety Limits for the Robotic Arms

The force-torque sensor of the slave arm can be used for collision detection. When the readings of the sensor go over a limit and the velocity of the arm goes to zero it means that the robot arm got stuck on an object. In this case instead of simple force-feedback on the master side the arm corrects its position to the opposite direction of the force. Moves away from the obstacle and avoids damaging the arm.

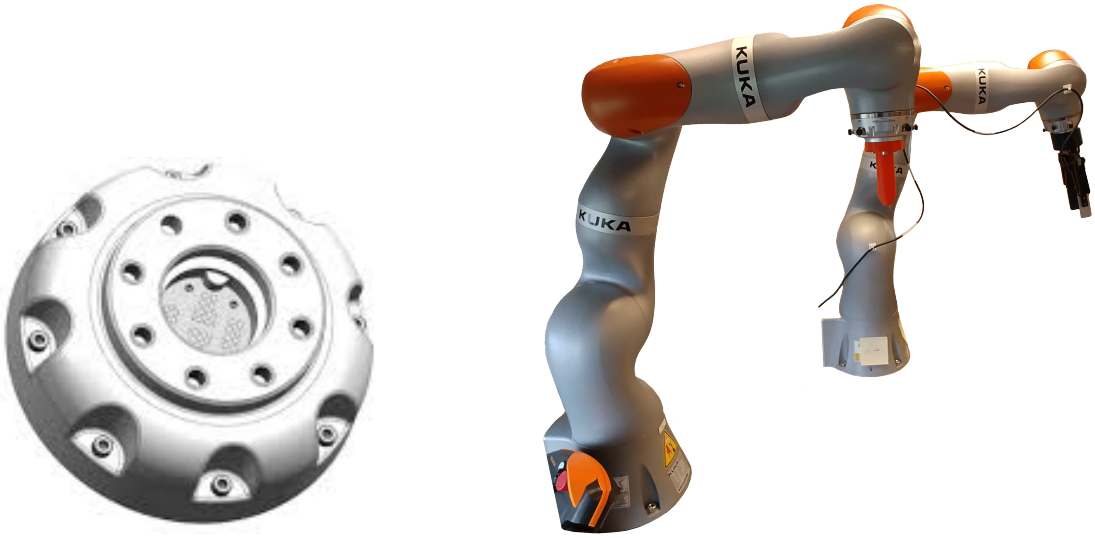
The master robot arm generally has no issues with collision. The impedance control with virtual springs does not allow damaging the arm. The master arm however can collide easily with its own joints. This is due to its mechanical design. The arm has a separate link for each joint. Around the elbow joint collision can happen easily. In this case the Kuka robot shuts down the execution of all programs, and can only be started after hard reseting the joints. To avoid this disruptions of teleoperation I have implemented a virtual wall routine. When the joint goes close to a position of collision the arm gently guides the user away from it. A virtual spring is fastened to the joint which is close to collision and it pulls out slowly the user from it.

4.4 Telemanipulation End-Effector

With the help of Carlos Veiga we have created a special end-effector for telemanipulation. This was designed to make the operations more comfortable and safer for the operator. It also makes it possible for the user to control the grippers from the master robot. The design was made from 3D printed plastic parts, sensors and an on-board micro-controller. The components can be seen on Figure 4.4.

The Kuka iiwa LBR robot arm was designed keeping in mind that the user might want to attach a special device to the last joint. The designers probably had in mind that in factories it might be useful to add a special gripper, pneumatic actuator or a welder. They made this possible by attaching the media flange (24) to the last joint of the robot. The flange can be soon on Figure 4.3. The media flange gives an interface for mechanical connections to the robot. On the sides of the flange there are digital and analog input and output connections. These connections provide also 20 V DC power supply and ground. Trough the flange connections the end-effector can directly be controlled by the robot controller computer.

The telemanipulation end-effector is connected to the bottom of the flange. It was designed to resemble and a joystick. It can be held by one hand, the operator should hand guide the master robot through it. Its length was determined so that it would fit right into an average hand.



(a) Media Flange (b) End-Effectors Connected

Figure 4.3: Different end-effectors can be interfaced with the robot through the media flange

The upper part of the end-effector is a ring. Inside the ring there are 2 mechanical switches. The user can create two different signals depending on turning the signal left or right. With the signal it is possible to control the gripper located on the slave robot. The signal can be used as boolean, to the left it opens the gripper, to the right it closes it. On the other hand during test I have decided to instead use it to set the direction of the velocity of the gripper. When the user turns the ring, the gripper on the slave slowly starts to close. This decision makes the gripper control much more flexible. It allows the user to pick up fragile items, which would otherwise just be crashed by the gripper.

With the end-effector it was possible to add another safety layer to the teleoperation. The joystick was covered with piezoresistive force sensors. The sensor used was the Flexiforce A301 (29). It is a very small and thin plastic sensor. Inside the sensor the semiconductor changes its resistance when force is applied on it. By changing the resistance the analog output voltage of the sensor changes. This change is then measured by an Arduino Micro which is located inside the joystick. The measurements values are sent through the flange to the robot controller. I have used this signal to dynamically enable and disable of the master robot's movement. The robot can only move while someone is holding the handle.

If something unexpected happens during an intervention simply letting go of the controller will stop teleoperation. It also allows the operator to take short breaks, without the need to start up the system again. When the robot movement

is disabled a virtual spring is put at the end-effector with very high stiffness. It allows the user to only change its position by a few millimeters. This method of stopping the arm is better compared to applying the brakes in the joint. If the arm is stopped by the brakes it stops very abruptly. By stopping it using the virtual springs it stops very smoothly. This smooth motion is more comfortable both for the user and the robot. Enabling and disabling the teleoperation can be done consecutively, changing state takes less then 100 ms.

(a) Joystick (b) Ring (c) Pressure Sensor

Figure 4.4: The components of the teleoperation end-effector

5

Conclusions and Future Work

5.1 Conclusion

During my master thesis work I have created a master-slave teleoperation system with force-feedback.

I have created 2 solutions for mapping the movement between the robots. Mapping by Joint-by-Joint an mapping trough Inverse Kinematics. I have tuned a controller for rapid positioning of the robot arm.

I have solved the problem of sensing the contact forces on the slave robot. Tuned a Kalman filter to mitigate the dynamic noise. Created a rapid and precise algorithm for gravity and bias compensation. Finally, attempted to counter balance the effects of the heat drift in the force-torque sensor.

I have implemented quick and robust software for the hand guidance of the Kuka iiwa LBR. Carried out experiments to determine how to map forces between slave and master robot. Realized a safe force feedback algorithm on the robot. Finally, added a specialized end-effector designed for teleoperation on the master robot.

In conclusion through my work I have added a different method of control to the CERNBot. Through this control a quick and user friendly telemanipulation became possible. Hopefully it will make the work of the robot operators easier and more comfortable. It will give the possibility to carry out more intervention in a reduce time. In the meantime not sacrificing the quality and precision of the operations.

5.2 Future Work

During my master thesis project unfortunately I was heavily constrained by hardware problems. In the future the quality of the project would increase drastically simply by acquiring a better force-torque sensor. It is not possible to solve the problems introduced by the heat drift simply from the software side. I believe that a well designed sensor should not suffer from these problems. Otherwise the contributions my algorithms make are general enough to be applied to any force-torque sensor.

It would be interesting to create an inverse kinematic algorithm specifically for teleoperation. An algorithm where the solution would be based not just on the current desired position, but also on the previous positions too. This would mitigate the effect of the changing local minima. The algorithm could leverage the novel deep learning framework implementations of gradient descent.

The calibration algorithm could be extended to measure all values of the inertia tensor of the end-effector. In my current work this was not needed, but for some special type of end-effector it might increase the precision.

The haptic feedback could be improved by incorporating vibrations to the teleoperation end-effector. When the force feedback is too small to be noticed by the operator, a small vibration of the end-effector could show that the slave robot made contact. Of course this would require the design of a slave end-effector. An ideal gripper for haptic feedback would have a soft tip, and pressure sensors built-in.

5.3 Appendix

The code and data sets for creating this document can be found at:
https://github.com/valikund/Master-Slave-Teleoperation-with-Force-Feedback
The video showcasing how the system works can be found at:
https://drive.google.com/file/d/1_dVdvX6upS7QQiuirfVqVSaVEUG8uwVf/view?usp=sharing

References

[1] FORWARD KINEMATICS: THE DENAVIT-HARTENBERG CONVENTION. Technical report. 15

[2] ISO/TS 15066:2016(en), Robots and robotic devices Collaborative robots. 65

[3] ROS-Industrial Kuka LBR-iiwa community meeting. 8, 68

[4] Haptics Solutions for ERM and LRA Actuators. Technical report, 2013. 63

[5] K. J. Astrom and L. Rundqwist. Integrator Windup and How to Avoid It. *American Control Conference*, pages 1693–1698, 1989. 30

[6] H. Barlow. Vision Science: Photons to Phenomenology by Stephen E. Palmer. *Trends in Cognitive Sciences*, 4(4):164–165, 4 2000. 32

[7] P. Beeson and B. Ames. TRAC-IK: An Open-Source Library for Improved Solving of Generic Inverse Kinematics. 25

[8] C. Bradley. Robotic Arm Calibration and Control 6-DOF Powerball LWA 4P. Technical report, 2014. 16

[9] T. Carter, S. A. Seah, B. Long, B. Drinkwater, and S. Subramanian. UltraHaptics: Multi-Point Mid-Air Haptic Feedback for Touch Surfaces. 64

[10] M. D. Castro, M. Ferre, and A. Masi. CERNTAURO: A Modular Architecture for Robotic Inspection and Telemanipulation in Harsh and Semi-Structured Environments. 6

[11] R. Diankov and J. Kuffner. OpenRAVE: A Planning Architecture for Autonomous Robotics. Technical report, 2008. 17

[12] E. T. Esfahani, A. H. Memar, and E. T. Esfahani. Modeling And Dynamic Parameter Identification of The SCHUNK PowerBall Robotic Arm DETC2015-47703 MODELING AND DYNAMIC PARAMETER IDENTIFICATION OF THE SCHUNK POWERBALL ROBOTIC ARM. 2015. v, 16

[13] G. German Conference on Robotics (6th : 2010 : Munich. *Robotics (ISR), 2010 41st International Symposium on and 2010 6th German Conference on Robotics (ROBOTIK) : date, 7-9 June 2010.* [VDE Verlag], 2010. 8

[14] A. Giusti, J. Guzzi, D. C. Ciresanciresan, F.-L. He, J. P. Rodríguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. D. Caro, D. Scaramuzza, and L. M. Gambardella. A Machine Learning Approach to Visual Perception of Forest Trails for Mobile Robots. pages 2377–3766, 2015. 2

[15] N. Hogan. Impedance Control: An Approach to Manipulation. In *1984 American Control Conference*, pages 304–313. IEEE, 7 1984. 64

[16] R. E. Kalman. A New Approach to Linear Filtering and Prediction Problems. *Transactions of the ASME-Journal of Basic Engineering*, 82:35–45, 1960. 38

[17] Konstantin Dimitrov. DS18B20 (digital temperature sensor) and Arduino - Arduino Project Hub. vi, 54

[18] M. Kushwah and A. Patra. PID Controller Tuning using Ziegler-Nichols Method for Speed Control of DC Motor. Technical report, 2014. 30

[19] S. J. Lederman and R. L. Klatzky. Hand movements: A window into haptic object recognition. *Cognitive Psychology*, 19(3):342–368, 7 1987. 32

[20] G. Lunghi, R. M. Prades, and M. D. Castro. An Advanced, Adaptive and Multimodal Graphical User Interface for Human-robot Teleoperation in Radioactive Scenarios. 6

[21] K. M. Lynch and F. C. Park. MODERN ROBOTICS MECHANICS, PLANNING, AND CONTROL. Technical report, 2017. 18

[22] I. Maxim Integrated Products. DS18B20 Datasheet. Technical report. 53

[23] B. J. Odelson, M. R. Rajamani, and J. B. Rawlings. A New Autocovariance Least-Squares Method for Estimating Noise Covariances *. Technical report, 2003. 41

[24] K. Roboter GmbH. Robot Option Media Flange For Product Family LBR iiwa Assembly and Operating Instructions. Technical report, 2016. 71

[25] L. S. Roque, K. H. Kendrick, E. Norcliffe, P. Brown, R. Defina, M. Dingemanse, T. Dirksmeyer, N. J. Enfield, S. Floyd, J. Hammond, G. Rossi, S. Tufvesson, S. Van Putten, and A. Majid. Vision verbs dominate in conversation across cultures, but the ranking of non-visual verbs varies. *Cognitive Linguistics*, 26(1):31–60, 2015. 32

[26] T. B. Sheridan. *Telerobotics, automation, and human supervisory control*. MIT Press, 1992. 1

[27] A. J. Silva, O. A. D. Ramirez, V. P. Vega, and J. P. O. Oliver. PHANToM OMNI Haptic Device: Kinematic and Manipulability. In *2009 Electronics, Robotics and Automotive Mechanics Conference (CERMA)*, pages 193–198. IEEE, 9 2009. 64

[28] R. Sodhi, I. Poupyrev, M. Glisson, and A. Israr. AIREAL. *ACM Transactions on Graphics*, 32(4):1, 7 2013. 64

[29] Tekscan. Flexiforce A301. Technical report, 2018. 72

[30] E. W. Weisstein. Harmonic Addition Theorem. 45

[31] G. Welch and G. Bishop. An Introduction to the Kalman Filter. 38