UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Technology

Martin Appo

Hardware-agnostic compliant control ROS package for collaborative industrial manipulators

Master's Thesis (30 ECTS)
Robotics and Computer Engineering

Supervisor: Assoc. Prof. Karl Kruusamäe

Tartu 2018

Abstract / Resümee

Hardware-agnostic compliant control ROS package for collaborative industrial manipulators

Industrial robotics today is moving towards using lightweight collaborative robots to make it possible for small and medium sized enterprises to integrate robots in their manufacturing environment. However, there is still very few collaborative robots seen in the industry and the main reason is that programming of the robot is still too expensive and time-consuming, since there are too few ready solutions available today for controlling co-robots. The solution would be more available open source, maintainable, extendable and usable high-quality code for controlling co-robots. This thesis concentrates on developing such complete software bundle on ROS for compliant control for industrial collaborative manipulators.

CERS: T120**,** T121, T125, T130

Keywords: collaborative industrial robots, Robot Operating System, compliant control

Riistvarapaindlik ROSi tarkvarapakett tööstuslike robotite mööndlikuks juhtimiseks

Tänapäeva tööstusrobootika liigub üha enam väiksemate robotite integreerimise suunas, et ka väiksema ja keskmise suurusega ettevõtetele oleks automatiseerimise tehnoloogia kättesaadavam. Sellegipoolest on väiksemate ettevõtete tootmisliinidel näha väga harva kasutusel roboteid. Ühe põhilise põhjusena võib välja tuua, et robotite programmeerimine on väga kallis ja ajamahukas, kuna pole olemas kergesti kättesaadavaid valmislahendusi juhtimaks koostöövõimelisi roboteid. Probleemi aitaks lahendada avatud lähtekoodiga, edasiarendatava ning lihtsasti hoomatava tarkvara olemasolu. Käesolev magistritöö keskendub nimetatud nõuetele vastava tarkvarapaketi arendamisele, mis täidaks tööstusroboti mööndliku juhtimise funktsioone.

CERS: T120**,** T121, T125, T130

Võtmesõnad: koostöövõimelised tööstusrobotid, robotitarkvara raamistik ROS, mööndlik roboti juhtimine

Contents

1 Introduction

Industrial robotics has been solving economic and social problems in manufacturing processes since the 1960’s. The usage of industrial robots eliminates routine jobs and increases production quantities. However, the field has been evolving in fairly narrow direction, meaning that industrial robots are mainly used by large enterprises and for manufacturing electronic devices or in automotive industry. Conventional industrial robot today can be characterized with being heavy, expensive, single task focused and dangerous for humans resulting in high installation costs and limited flexibility in changing manufacturing environment. This is one reason, why industrial robotics have not been widely integrated in small and medium enterprises (SMEs) – they need flexible, safe for humans by design and high return on investment (ROI) machines. In the recent years, developing collaborative robots has been the focus in industrial robotics. Different robot manufacturers are coming up with safe, lightweight and flexible solutions that match the needs for SMEs. Besides the fact that co-robots are safe to work side by side with humans, they have potential to work in direct contact with them. This concept is called physical human-robot interaction (pHRI) and it is rising trend. It is discovered that productivity can be increased by having robot and human working together, since robot can be responsible for precision and repetitive tasks, on the other hand, human can perform flexible and complex actions, thus they are compensating each other.

However, we are long way from having co-robots actively working together with humans in SMEs. The concept of pHRI is actively developed in research labs and academy, but have not yet reached to actual manufacturing processes. Reasons for that is still high investment costs resulted by lack of flexibility in software and low availability of ready and packaged software implementing such functionalities.

The goal for current thesis is to assemble a hardware-agnostic software bundle providing compliant force control for industrial serial collaborative manipulators. The final results show that it is possible to merge together different software packages and have them work together while maintaining hardware independency and properties of maintainable and flexible software.

The content of this thesis is organized into 5 chapters. Chapter 2 (Motivation of the work) gives overview of the current state of industrial robotics including challenges in collaborative industrial robotics. Chapter 3 (Practical task setting and aim) sets the initial practical goal of current thesis. Chapter 4 (Literature review and previous work) gives overview of human-robot cooperation including necessary forms of manipulator control methods. Furthermore, previous work will be discussed and how the previous projects fit into current thesis' software. Chapter 5 (Software implementation) will first give overview of the software project in wider plan and subsequently will go into more technical details about the implementation. Chapter 6 (Testing and demonstrations) represents the testing plan and results of the software. Furthermore, demonstrations of use cases are illustrated and explained.

2 Motivation of the work

In this chapter, the general problems that industrial robotics will solve are reviewed. Brief statistics about the recent past and predictions about near future are given to emphasize the importance of industrial robots in modern days.

Today’s most challenging prospects for industrial robot’s development, maintenance and deployment, are discussed. It is implicated that the field of robotics is far from being thoroughly advanced. Furthermore, the solution to the complications today is proposed by introducing lightweight industrial robots. Additionally, the application and open challenges associated lightweight industrial robots are highlighted.

2.1 Industrial robotics today

Two main type of problems in the world that are driving the advancement of robotics and automation are economic and social issues. In [1] both of these problems are being addressed as follows. In economics, robotics can solve the problem of continuously rising manufacturing labor costs while increasing productivity and reducing the price of goods, thus raising the overall standard of living. The social issues, involving dissatisfaction among workers performing jobs that are potentially harmful, dangerous, unpleasant or routine, can be alleviated by giving this kind of jobs for robots to handle. In addition to elimination of these low-level jobs, field of robotics creates new, creative, flexible and more pleasant careers. The Industrial Federation of Robotics (IFR) estimates that between 2000 and 2008 the robotics industry created 8-10 million highly qualified jobs and it is prediction that between 2012-2020, another 4 million jobs will be created [2].

According to a recent report [3], the demand for industrial robots has accelerated considerably since 2010 due to continuous trends towards automation and innovative technical improvements in industrial robots. The total worldwide stock of operational industrial robots at the end of 2016 increased by 12% to about 1.8 million units [3]. The main driver of the growth has been the electronics and automotive industries as these fields make up 80% of all robot installations today [4].

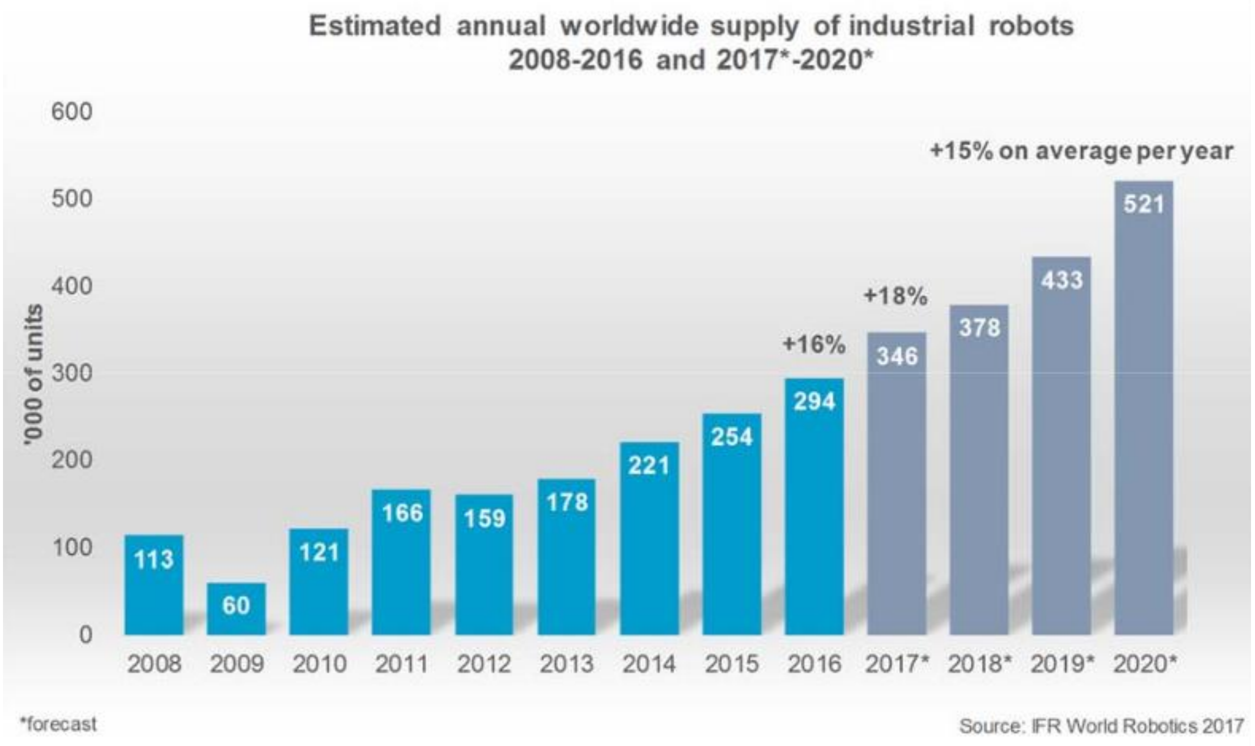


Figure 2.1. *Statistics and estimated annual worldwide supply of industrial robots* [3].

The statistics (Figure 2.1) shows that developing robotics systems, especially industrial robots, is globally important. It is estimated that the worldwide stock of operational industrial robots will reach to about 3 million units at the end of year 2020 [3].

2.2 Open challenges in industrial robotics

As robots have been in industry and under rapid development since the 1960’s [5], one may say that industrial robots is a solved problem. It is, of course, an outdated opinion, because large-scale production comprises only a minor part of needed capability in modern day’s society - many more processes could be automated. Today’s most discussed challenges include overcoming barriers that are preventing robots from being more widely used, especially in small and medium-sized enterprises (SMEs). The range of applications could increase and expand to SMEs, if robots were easier to install, program and reprogram, and safer to work with [4]. It is extremely important, since SMEs represent more than 99% of the businesses in the EU [6] and currently only a small portion of them use robots in their factory lines [2].

The main repressive properties that prevent deployment of industrial robots in wider range of SMEs can be described as follows:

- <u>Cost of a robotic system.</u> As stated in recent report [7], the high investment costs of industrial robots and automation systems have negative impact on utilizing them in SMEs. As traditional industrial robots are built for specific task and are not mass-produced, they are very expensive, thus unreachable for smaller companies.
- <u>Safety regulations</u> for conventional rigid industrial manipulators require huge efforts for meeting the safety requirements resulting in high installation and operation costs. Many SMEs have to cope with frequently changing production processes, thus robots and humans have to share the same workspace, which would be impossible with traditional industrial robots. Presently, humans are being kept safe by isolating them from robots.
- <u>Usability and flexibility problems</u> imply that a system does not have simple and intuitive way of reconfiguring or reprogramming. Thus non-experts cannot use and maintain robotic cells effortlessly, however involving software specialists for this kind of duty would be misuse of their time and company resources. As production is experiencing a paradigm shift from mass production to mass customization of products [8], the robots should adapt to handle more product variation with smaller cycles and batch sizes. Yet, typical industrial robots are statically placed and continuously repeating predefined sequence of actions. As conventional industrial robot’s setup requires safety fences and are very specialized systems, setting up a robotic system takes about three months involving the work of robot software engineers with high costs [9].

- Programmability for robotic manipulators has not been advanced much since the industrial robots have been in use. Most common method for controlling a robot today is controller-specific languages [10]. These programming languages are usually very low-level and specific to robot or robot-family. As controller-specific languages do not have any universal standard between them, every new robot in a factory, will significantly affect the deployment cost and time of a robot.

2.3 Importance of lightweight industrial robots

Lightweight industrial robots are expected to revolutionize many assembly and manufacturing processes and are the topic of a significant research effort [11]. These robots are also called collaborative robots (“co-bots” or “co-robots”). Their main aim and advantage is to operate alongside or in direct contact with humans. Co-robots offer a number of significant advantages over their conventional industrial counterparts:

- low installation costs due to the lack of safety cages, this also results in overall space savings;
- simpler programming of robots, which allows faster and more agile reallocation of robot’s function;
- short payback period thanks to low cost and high productivity.

This kind of robots enable new opportunities for SMEs which would not be able to afford traditional costly and complex industrial robot. The fact that lightweight robots have been researched long time and the advances in recent years with high future potential shows that collaborative robots are state of the art technology in modern day. Consequently, research and development focused on these kind of robots is in many ways significant priority in today’s robotics.

The problems with conventional industrial robots in modern days, described in previous chapter (2.2), can be relieved or solved completely by utilizing lightweight robots in the automation of production lines. Several major advantages of lightweight industrial robots can be highlighted:

- Cost efficiency. Since co-robots are mass-produced as universal robots, the investment costs for SMEs can be reduced significantly. The integration costs are also lower because no extra safety measures have to be installed.
- Safety is certainly one keyword that is already considered in early stage of designing a collaborative robot. These robots are designed to not need safety fences or other external systems for keeping humans safe. The collision-safety is already integrated and can be divided into two: collision avoidance and collision detection [12]. However, there is no sensor system that guarantees avoidance of collision and at the same time having sufficient reliability without restrictions to the speed of motion. Therefore, the tendency is to accept collisions while keeping them harmless for humans by using collision detection.
- Flexibility and usability is improved for collaborative robots as they are designed lightweight and compliant with surroundings. As co-robots are light by design, the installation cost and -time is reduced. Meaning the ability to quickly change the robot's location and adapt to changing manufacturing conditions. More flexibility in production process can be achieved by involving humans alongside with robots on worktime, which is possible only considering the aforementioned safety features of collaborative robots. The usability of robots for SMEs is a major research subject and considerable advances have been made in the field [13], [9], [10].
- Generic procedural languages can be used for programming co-robots. It means high-level multi-purpose code (e.g. C++ and Python) that is more comprehensible for wider range of engineers. As a result, the transition between different robotic systems or personnel is more effortless for SMEs, thus requiring less time and resources. Most lightweight robot systems today already come with Robot Operating System (ROS) [14] support, which is significant advantage, being one big step closer to hardware agnosticism.

2.4 Main challenges in collaborative robotics

Even though, involving collaborative robots in industry reforms the whole manufacturing process and resolves number of problems in industrial automation, there are still hurdles to tackle in relation to co-robots. A chapter of Springer Handbook of Robotics [15] describes several software related long-term challenges that modern days co-robotics is still confronting:

Human-friendly task specification [8]. The current situation is that robots can perform very narrow and specific tasks. The challenge is, to get the robots to recognize more goal-focused tasks. Taking care of these problems can account for up to 80% of the total programming time.

Embodiment of engineering and research results*.* New technical solutions today are often developed from scratch. This is important problem in the field, because robotics employ exceptionally wide variety of technologies and need constant upgrading of the systems. Therefore, packaging and sharing re-usable solutions over wider community would be step closer to resolving the problem. Unfortunately, there is still long way to go for efficient robotics engineering in terms of reuse of technologies and know-how.

Open and extendable systems*.* Software systems need to be open to permit extensions by a third party, on the other hand, they need to be closed for modifications, such that the correctness of functionalities can be ensured. Software engineering and architecture design is a key challenge here. There is need for frameworks that are less restrictive compared to the ones that are available today.

Intuitive human-robot interaction (HRI). Although, collaborative robots are a major advancement in the HRI field, there are still improvements that can be made. Issues, such as teaching human and robot to work together comfortably and managing different variations and exceptions in robots work during runtime are still under continuous research.

Hardware agnosticism. Despite the fact that compatibility with ROS is a leap towards hardware-independent systems, the situation today in collaborative robotics and HRI related software is not gratifying. There are number of proposals of hardware-agnostic software architectures (e.g. [16], [17], [18]), however to the best of author’s knowledge, open-source, ready and usable projects are not available today.

Current thesis is thriving towards to resolving aforementioned problems by constructing open-source hardware-independent ROS software package for force-controlling lightweight industrial manipulators. Concurrently taking into account the state of the art practices in software development, re-using open communities developed components and sharing the results for wider audience.

3 Practical task setting and aim

3.1 Constructing a hardware-agnostic software package

Main goal of this thesis is to assemble a ROS (Robot Operating System) software package for controlling a serial-chain collaborative robotic hand in Cartesian space using force-torque (FT) and positional data from the driver of the robot. The software bundle can and should re-use existing functionalities (sub-packages) to take the full advantage of the built-in modular approach of ROS framework. Binding different software packages, which were originally not meant to work together or with particular robot, can be challenging in many cases. ROS certainly helps to overcome many hurdles but we also need to bear in mind that there really cannot be fully hardware-independent software - some configurability is nearly always involved in switching between different hardware systems and also, a list of ground boundaries on hardware need to be defined.

3.2 Improving existent solutions

In order to combine already developed ROS packages and make them function together, improvements have to be done to existing solutions. Furthermore, profound understanding of how current software packages work is prerequisite for enhancing and binding different modules. Extending existing solutions includes adding configurability for improving hardware-agnosticism, creating proxy modules for interconnection between components and fixing bugs as well as compilation errors due to outdated framework versions, etc. In addition to these technical modifications, various of functional adjustments has to be done to extend the stability and compatibility of contact control framework that will be discussed in section 4.4 (Previous work).

3.3 Test and confirm fulfillment of requirements

To confirm the software capabilities, all the functions will be tested and demonstrated in methodological manner. The testing plan is constructed and every testing routine will incorporate simulation phase, partially simulated phase (i.e., F/T data comes from real robot but movements are simulated) and full hardware testing. Moreover, the hardware-agnosticism will be affirmed by setting up and running the software on two different serial manipulator robots: 6-DOF Universal Robots UR5 and 7-DOF Franka Emika Panda. Final demonstrations of actual use cases are executed on Franka Emika Panda manipulator.

4 Literature review and previous work

In this paragraph, human-robot cooperation and different forms of it are discussed. Consequently, more definite form of human-robot cooperation, physical human robot interaction (pHRI), is highlighted. The compliant control methods and concepts that are developed and advanced over the years, are introduced with examples of use cases. Ultimately, previous projects that current thesis draws on, are discussed.

4.1 Human-robot cooperation

At the present time, safe and flexible cooperation between robot and human is considered as a new way to achieve better productivity and quality when performing complicated activities. Researchers are actively publishing journals [19], and arranging conferences [20] that are specifically aimed at the field. Despite the ongoing research activity, there is still a long way to go to entirely integrate human-robot interaction (HRI) in an actual industry applications [21].

4.1.1 Categorization of interactions

Springer Handbook of Robotics [22] describes interaction between human and robot by dividing it into three distinctive groups: *supportive, collaborative* and *cooperative.*

- In supportive interactions, out of the three types of HRI, the robot has the smallest amount of physical contact with human. Main purpose of a supportive robot is to provide human with the tools, materials and information to optimize the human’s task performance or objectives.
- Collaborative robots, however, operate on the specific task with the workload divided between robot and human by turn-taking. Robot and human separately complete the parts of the task best suited to their abilities. For instance, robot pre-processes a material and human finishes the needed operation.

- Cooperative interaction includes direct physical contact with human or indirect contact through a common object. These actions encompass tasks, such as lifting and carrying, kinesthetic teaching or coordinated material handling. As this thesis mainly focuses on physical Human Robot Interaction (pHRI), cooperative interactions are discussed further.

4.2 Physical Human Robot Interaction (pHRI)

The pHRI is an emerging field in robotics that aims to combine the complementary properties of both partners, in order to facilitate current manual operations [23]. The idea behind this integration is to combine the advantages of both the human and the robot in the same production cell. Workers add the flexibility to the production cell, as they can easily adapt with many unexpected situations, yet industrial robots are reliable in terms of speed, payload, and accuracy [24]. That means, pHRI is essential component for achieving safe and functioning cooperation between human and robot. Furthermore, pHRI expects the mechanical structures of a co-robot to be lightweight, compliant and with external sensing capabilities, to enable unwanted collision detection and handle intentional contacts [25]. Thus, one main problem related to the concept is to distinguish between accidental collisions and intentional contacts between robot and environment.

There are various methods for solving the unintentional collision problem. Until recently, this problem was solved by forbidding humans and robots to share a common workspace by constructing protection fences and using different sensors to detect human in danger zone [21]. Yet, forbidding humans and robots to work side by side eliminates the physical contact factor of pHRI. Therefore, alternative methods have been developed to retain safety as well as close coexistence. The most conventional method is recognizing abnormalities in currents of driving motors of the robot [26]. Another, mechanical approach, is to solve this problem with series elastic actuators [27]. These sensorless methods provide safe coexistence of robot and human while detecting or avoiding unwanted collisions, whereas these methods are not convenient for detecting intentional low-force contacts between end-effector (EEF) and its surroundings. For more precise force sensitivity, the robot should be equipped with one or more Force/Torque Sensors (FTS) [28]. The mechanical design with FTS enables detection of intentional touch. Often, mechanical safety measures and sensor-based methods are combined on single robot to achieve compliancy and environment awareness.

4.3 Compliant control

In some robotic applications, the interaction forces on a robot can be negligible, as well as instantaneous mechanical work done by the manipulator is negligible. For instance, these applications include spray-painting and welding. Contrarily, there are numerous of operations that require dynamic interaction with the environment, thus controlling and measuring the interaction forces becomes of high importance. Examples include drilling, routing, boring, grinding, bending, fettling – essentially any task requiring modifications to be done on the environment.

A common term for environment dependent actions is called compliant control [29]. Categorization of different compliant control methods is important to acquire clear understanding of dependencies between force, velocity and position of robot. Figure 4.1 describes the manipulator control methods as position control, force control and the relation between the two. Explicit position control of a robot manipulator is control of the position and orientation of the robot’s EEF without considering the forces applied on the EEF. Explicit force control, on the other hand, does not consider EEF position while moving the EEF, instead it only responds to forces and torques applied on it.

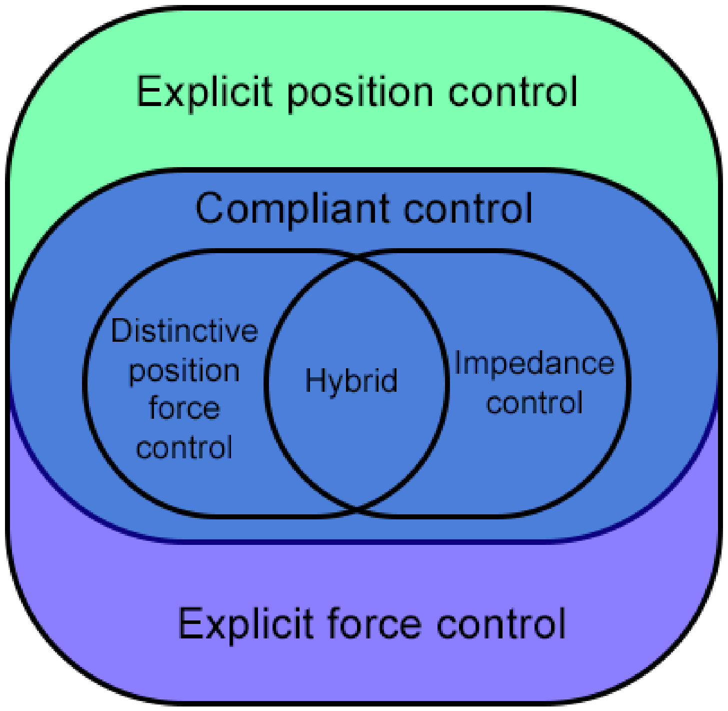


Figure 4.1. *Categorization of manipulator control methods*. Derived from [30].

At the intersection of explicit position control and explicit force control there is compliant control (Figure 4.1), combining the advantages of both control methods. Position control and force control are supplementary to one another, since small variations in relative position generate extreme changes in contact forces when objects physically interact. Knowledge and manipulation of these forces can induce tremendous increase in effective positional accuracy.

Compliant control can be divided further by taking into account the relations between position and force data considered on controlling distinctive directions of EEF. The methods are explained separately in subsections 4.3.1 - 4.3.3.

4.3.1 Compliant motion

Compliant motion is the simplest form of including elements of position control and force sensing methods in manipulator's control. The manipulator achieves compliant motion in case when it maintains compliance with environment while tracking a trajectory of movement. It can be achieved programmatically [29] as well as mechanically [27]. These methods are called active compliant control (*force control*) and passive compliant control respectively. The force control has major advantages over passive compliance referring to flexibility. For instance, using force control, a single manipulator can execute different tasks without the need of hardware modifications.

An example of compliant motion would be EEF moving towards stiff object (such as a table) and stopping the movement when it reaches into contact with the table (Figure 4.2).

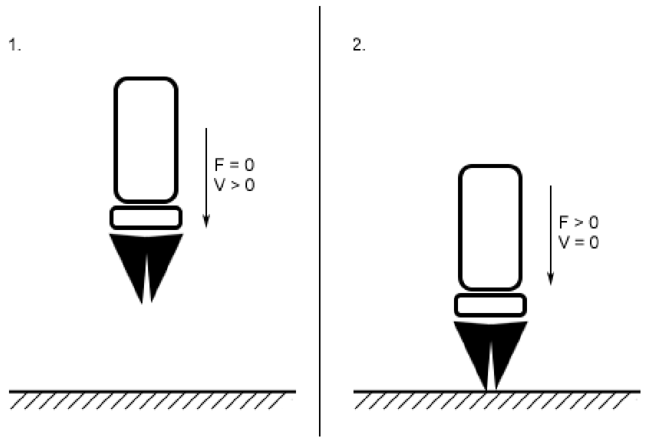


Figure 4.2. *Compliant move – robot's EEF moves towards table. (F – force applied on EEF, V – velocity of EEF).*

4.3.2 Position/force control

The problem of controlling manipulator by considering position of EEF as well as forces applied on it was formalized by Mason in 1981 [29]. In order to approach compliant control, he discussed concepts of simpler control modes: *explicit position control* and *explicit force control.* The conceptual extremes of these two modes can be explained as follows: if the manipulator tip is fixed to a stiff surface, no positional freedom exists but the manipulator will have complete force freedom. The opposite extreme would be when the manipulator is in free space. Mason introduces the conception of surfaces in configuration space (C-surfaces), which are intermediate states between explicit force control and explicit position control. That means partial positional and partial force freedom – force control is performed along the C-surface normal and position control along the tangent hyperplane of the C-surface. The C-surface concept has been already shown to be useful in designing force-position control software by Merlet [31].

On the basis of Mason's work, Raibert and Craig developed hybrid position/force control [32]. They formulated a control method that partitions all possible EEF directions into an explicit position-controlled subset and explicit force-controlled subset. By using this method, it is relatively intuitive to assign different rules or constraints on distinctive directions in order for the robot to complete a goal.

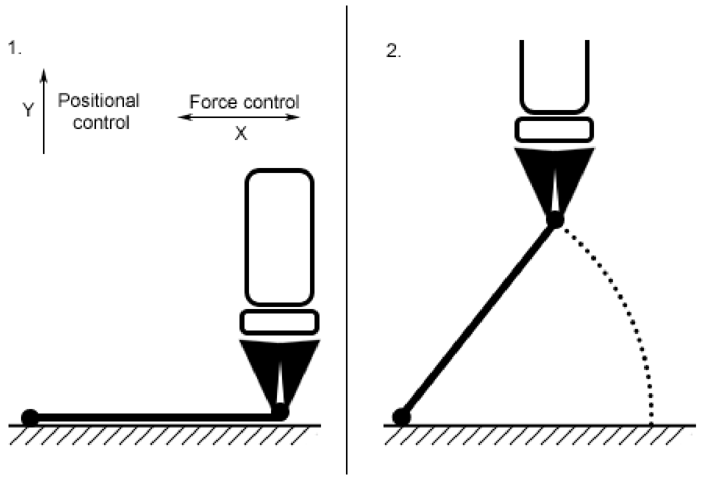


Figure 4.3. *An example of positon/force control – opening a door.*

A simplified 2-dimensional example of hybrid position/force control application is shown in Figure 4.3. In this case, manipulator's EEF has positional control in Y axis and force control in X axis, thus the robot can perform an action of opening a door while allowing the mechanical properties of the door control the EEF position in X axis. With these simple rules, we can generalize the action of opening any door, despite its opening radius. For instance, this kind of action abstraction cannot be achieved by using only explicit positional control.

4.3.3 Impedance control

Another type of compliant motion control which is an extension of the hybrid position/force control is called impedance control. The term impedance control is introduced by Hogan, who states clear distinction from conventional compliant control: "*A distinction between impedance control and the more conventional approaches to manipulator control is that the controller attempts to implement a dynamic relation between manipulator variables such as end-point position and force rather than just control these variables alone*" [33]. Therefore, instead of partitioning possible directions into explicit position-controlled and force-controlled subsets, impedance control takes both into account on single direction. The most trivial example of this motion control method would be the virtual *spring law*, discussed in subsection 4.4.1, where velocity of EEF is dependent on displacement of initial position and force applied in the direction. The mentioned law simulates spring's damping and stiffness properties on EEF in robot's controller level.

4.4 Previous work

There are several open-source projects involving compliant control for industrial robots that claim to be hardware-agnostic and working out-of-the-box. However, the situation is not as perfect as it may seem – to the best of author's knowledge there is no open software for compliant control with clearly defined requirements for a robot and sufficient documentation. To improve these circumstances, the existing projects should be examined, analyzed and improved.

In this subsection, the previous works that this thesis directly relies on are discussed. The main motivation for choosing the two works is that they are

- open source projects
- built on ROS
- aimed towards hardware agnosticism
- relatively recent releases.

4.4.1 GCCF: A Generalized Contact Control Framework

A thesis by R. A. von Sternberg presents a generalized contact control framework (GCCF) [30] that enables compliant control of a robot manipulator by processing FT data and outputting velocities in Cartesian space. The thesis highlights the importance of packaging software to expand the task space of industrial robots. Notably a field of compliant control is discussed and a general hardware-agnostic framework is proposed. The framework meets architectural requirements for integrating it in current thesis. Key elements are hardware agnosticism, ease of integration, modularity and extensibility. The GCCF is implemented in C++ language and on ROS platform. Its main function is to process received FT data depending on chosen contact control rule and then publish velocity information about how fast should the EEF move. The framework makes available three force-control laws for the user, which are formulated in following equations:

$$V_{c,follower} = \frac{1}{b}F \quad (4.1)$$

$$V_{c,spring} = \frac{1}{b}F - k(d_{travel} + d_{offset}) \quad (4.2)$$

$$V_{c,compliant\ move} = V_{desired} + \frac{1}{b}F \quad (4.3)$$

F - The force sensed in controlled dimension.

d_{travel} - The distance the EEF has moved.

V_c - Output velocity.

$k, b, V_{desired}, d_{offset}$ - User supplied constants that determine the behavior of the law.

All of these control laws can be applied on every axis in Cartesian space (X, Y or Z) of robot's EEF independently.

The follower law is described with equation 4.1 and is the simplest of the three. The EEF will react to forces by moving in the direction that the force is pushing it. The law takes parameter b as input, which controls the velocity linearly and acts as a scaling factor.

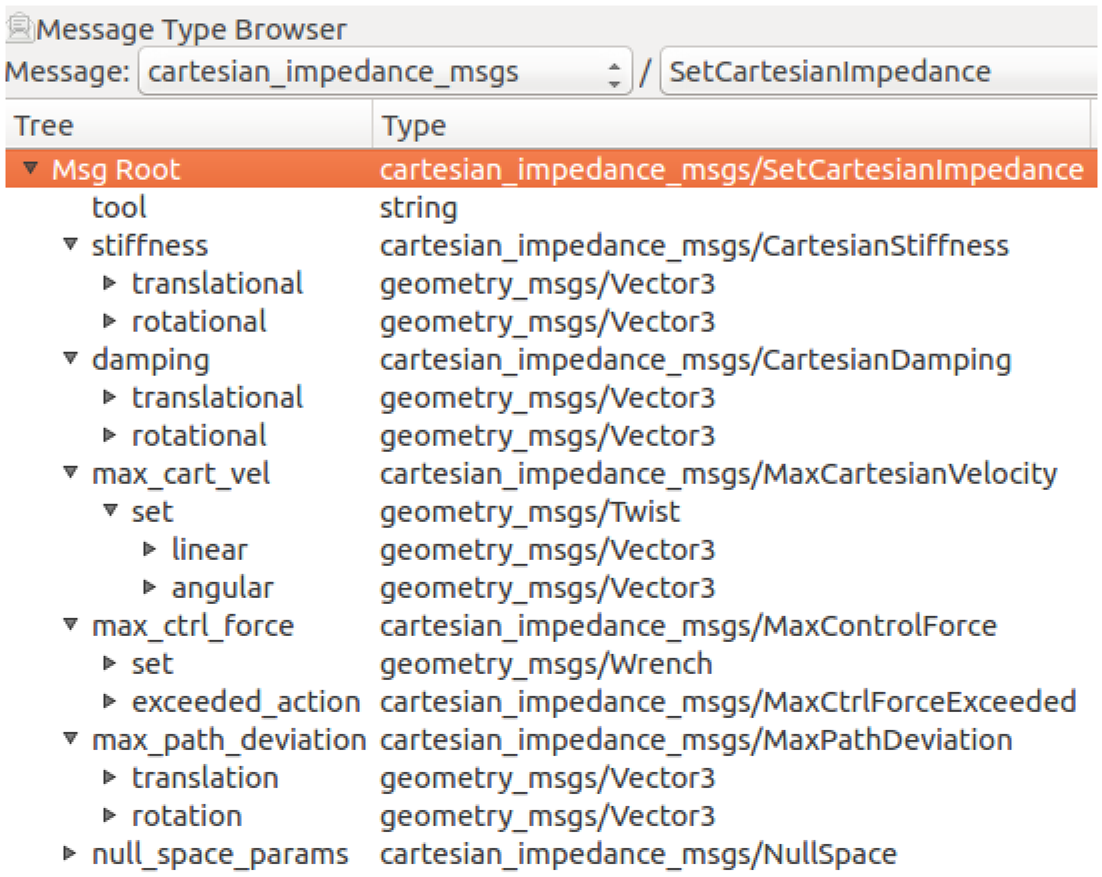
Equation 4.2 describes the spring law. This law enables robot's EEF to act as a virtual spring, where the spring resting length (d_{offset}) and damping constant (b) can be determined by user. The main concept of the law is that the further EEF is pushed from its original position, the more force it will apply in order to retain the original position.

Equation 4.3 is the compliant move law*.* In this equation, a desired velocity is opposed by the external force at EEF. This law can be used for moving the robot into a desired position while avoiding excessive forces. For instance, it enables EEF to move until it comes to a contact with an object.

Even though GCCF is developed as an open-source, hardware agnostic and generalized framework, it has not been elaborated since year 2016. There are some fundamental and technical problems with the software package. Firstly, an external dependency of the framework is in private repository and inaccessible for third parties, therefore, GCCF will not compile as is. Another issue is that the hardware-independency does not appear to be not tested thoroughly, as various problems emerged when implementing GCCF functionalities on robots used during current thesis. Furthermore, there is no straightforward configurability, i.e. configuring the framework to run on specific robot needs changes in the source code. Additionally, the contact-control law algorithms could be improved and optimized. And finally, GCCF is a stand-alone ROS package with no executable nodes [34], which means that no other ROS nodes can use its functionality through ROS communication patterns (e.g. services or messages).

4.4.2 ROS Interface for Impedance/Force Control

To interface GCCF into ROS ecosystem, the Google Summer of Code (GSoC) Project - *ROS Interface for Impedance/Force Control* [35] includes a suitable set of generic ROS messages. The GSoC project consisted of three parts. First part is set of generic ROS messages including necessary parameters for Cartesian impedance/force Control. An original set of messages that came with the interface are depicted in Figure 4.4. These messages contain parameters that are compliant with GCCF control laws. However, there is no option for choosing which control law should be applied on which direction of EEF. To make it fully compatible with GCCF, improvements need be made.



Message Type Browser

Message: cartesian_impedance_msgs / SetCartesianImpedance

Tree	Type
▼ Msg Root	cartesian_impedance_msgs/SetCartesianImpedance
tool	string
▼ stiffness	cartesian_impedance_msgs/CartesianStiffness
▶ translational	geometry_msgs/Vector3
▶ rotational	geometry_msgs/Vector3
▼ damping	cartesian_impedance_msgs/CartesianDamping
▶ translational	geometry_msgs/Vector3
▶ rotational	geometry_msgs/Vector3
▼ max_cart_vel	cartesian_impedance_msgs/MaxCartesianVelocity
▼ set	geometry_msgs/Twist
▶ linear	geometry_msgs/Vector3
▶ angular	geometry_msgs/Vector3
▼ max_ctrl_force	cartesian_impedance_msgs/MaxControlForce
▶ set	geometry_msgs/Wrench
▶ exceeded_action	cartesian_impedance_msgs/MaxCtrlForceExceeded
▼ max_path_deviation	cartesian_impedance_msgs/MaxPathDeviation
▶ translation	geometry_msgs/Vector3
▶ rotation	geometry_msgs/Vector3
▶ null_space_params	cartesian_impedance_msgs/NullSpace

Figure 4.4. *Outline of the messages provided by ROS Interface for Impedance/Force Control.*

During the GSoC project, a graphical user interface for setting parameters for Cartesian impedance/force control was developed and tested. Since the GUI was developed explicitly for Cartesian Path Planner Plug-In for MoveIt [36], it cannot be used for generic applications. Finally, an already existing ROS driver for KUKA iiwa 7 R800 robot was updated, to accept impedance/force control messages. The fact that specific robot's driver had to be modified for testing and developing this project, indicates that there is actually no hardware-agnosticism.

5 Software implementation

5.1 Objective

The main objective of the software implementation is to make GCCF more usable, configurable and hardware-agnostic. Furthermore, the ROS messages of GSoC project should be integrated with GCCF to provide connection between GCCF and ROS ecosystem. In addition, different robot-specific packages, which are including configurations, drivers and helper-packages, have to be constructed in order to test the functionality on specific robots. Finally, a link between user and robot control software has to be made, meaning graphical and/or physical user interface.

5.2 Requirements

This section describes a number of requirements that the software package under development should fulfill. Identifying the requirements for software system is essential part of development process, as requirements describe the final objective of the product.

5.2.1 Functional requirements

The purpose of the software under development is to make it possible to use compliant control on a serial open-chain manipulator co-robot according to various rules. Moreover, the human input for the robot should be easily understandable in the form of Graphical User Interface (GUI). The requirements that describe needed functionalities of the software are following:

<u>General movement control.</u> To control the robot in the first place, it needs jogging capabilities, meaning that the software should be able to control robot’s EEF in

Cartesian space relative to known reference frame. That includes capabilities for forward and inverse kinematics solving.

Manual control abilities through some external device (e.g. keyboard, joystick). This is needed for testing the robot, for cooperation capabilities and other applications, where manual position control of the robot's EEF is needed.

Pre-processing force-torque data. Capabilities for filtering potentially noisy raw data from robot's low-level controller and/or from driver. As various manufacturers implement their robot's FT sensitivity differently, the filtering functionality should be configurable.

Compliant, position/force and impedance control rules. The implementation for converting or generating jog commands (velocity commands) from preprocessed FT data depending on different predefined control rules. The rules include compliant move, follower rule and impedance (spring) rule which are described in subsection 4.4 of this thesis.

Graphical User Interface (GUI). The user should be able to assign mentioned control rules on distinctive directions and axis of manipulator's EEF through GUI. It should provide functionality for changing different contact control rule parameters depicted in Figure 4.4 at runtime.

5.2.2 Technical requirements

Technical requirements list different prerequisites on general capabilities of the software, e.g. architectural decisions, safety concerns, usability convenience assets and compatibility details. The requirements are brought out as described below:

ROS support. In addition to fulfilling general ROS package infrastructure standards [37], interconnection capability with other ROS packages is necessary. Thus, every added module must contain some interface for communicating with it through some available ROS communication pattern, such as topics, services and/or actions [38].

Performance specification. As the compliant position control happens in real time, the performance of the calculations is crucial. Although, the bottleneck of the performance might also be the controlling computer, the best effort should be made to minimize the use of computational resources, e.g. avoiding or limiting the use of thread sleep-functions, memory allocation and exponential complexity. For a robot manipulator to act as expected, the trajectory and forces processing recalculation cycle should be around 50 Hz.

Hardware agnosticism. Hardware independency is one of the most important aspects of software in robotics. The fact that single software can be used with many different manipulators advances its value significantly, as software development is relatively high cost and time consuming.

Software configurability. The software should be reconfigurable to facilitate hardware agnosticism. Different parameters that define robot-specific variables can be changed externally without recompiling the code.

Safety requirements. Despite the fact that most co-robots have safety functionalities integrated in controller level, another layer of safety features in higher level of code helps to assure that the system is safe to use. Safety features include ability of setting maximum force and torque parameters on individual axes as well as maximum velocity that the EEF of the manipulator could reach. Additionally, revival strategies from erroneous situations need to be defined.

5.2.3 Requirements for the robot

There are wide variety of robotic systems that are obviously similar to each other. For instance, consider serial manipulators – all this type of robots share common functionalities and are conceptually same. Therefore, it would be logical that some software module responsible for certain common functionality should work on different robots. It is impossible to achieve ideal hardware independence, although it is possible to define certain rules that a hardware system has to follow, so it would be compatible with the software. The software developed in the current thesis can be applied on a robot that meets the following requirements:

1. it is a serial open kinematic robotic manipulator;
2. it has ROS support, meaning that there should be a compatible driver [39] for controlling the robot through well-known standardized ROS messages [40];
3. compatible with ROS MoveIt! [41];
4. force-torque (FT) sensing capability and the driver should publish corresponding ROS wrench messages [42] containing FT data.

5.3 Software architecture

Software architecture can refer to two related, but distinct, concepts. Architectural structure is a classification of subsystems and description how the subsystems interact with each other. In contrast, technical architecture (or architectural style) defines different lower level techniques and concepts that underlie a given system. Unfortunately, in many existing robot systems, it is difficult to specify clear boundaries between different approaches and subsystems, meaning that the architectural structure is blurred and different architectural styles are tied together [43]. Consequently, clean and well-conceived architecture of software can have significant influence on the quality of the code.

5.3.1 Technical architecture

In current subsection, various lower level architectural decisions, used in the development of the software, are stated and justified.

Robot Operating System (ROS). ROS is widespread, active, and matured framework for developing robot software systems. Considering ROS's advantages, such as various communication patterns between nodes, asynchronous nature and modularity, the decision of choosing ROS was straightforward. Also, enormous community and open source code base encouraged the choice even further. For current project, Kinetic distribution [44] of ROS is used since it is officially recommended release as of May 2018.

C++ programming language. The stimulus for choosing C++ for fundamental programming language for the project derives from using ROS in the first place, as the framework as well as previous related projects are developed in C++. Furthermore, due to strict real time restrictions, C++ is optimal programming language performance wise. Finally, as C++ is object oriented language, it is intuitive to write and understand for software developer with moderate amount of experience.

Publish-subscribe pattern. Due to robotic systems need for asynchronous operations, the publish-subscribe pattern for exchanging messages between nodes is used in certain cases. The patterns principle is that a node can subscribe or publish messages on previously defined specific topics and the communication takes place asynchronously.

Client-server pattern. Information transfer with a node, which provides return value after performing an action that was initiated by second node, requires client-server pattern (services and actions in ROS). This type of interaction is synchronous. Figure 5.1 contains use cases of both communication patterns described in current chapter.

5.3.2 Structural architecture

A modular approach has been taken when designing current system. In ROS, software is always organized into and distributed as packages. A ROS package is an entity that can contain library, dataset, configuration files, launch files or essentially any complete logically useful set of elements. The aim is to assign as little responsibility to a single package as possible, to maintain scalability and reusability of the packages. Moreover, the modularity of the developed software is achieved by reusing already available packages and the modular nature of ROS.

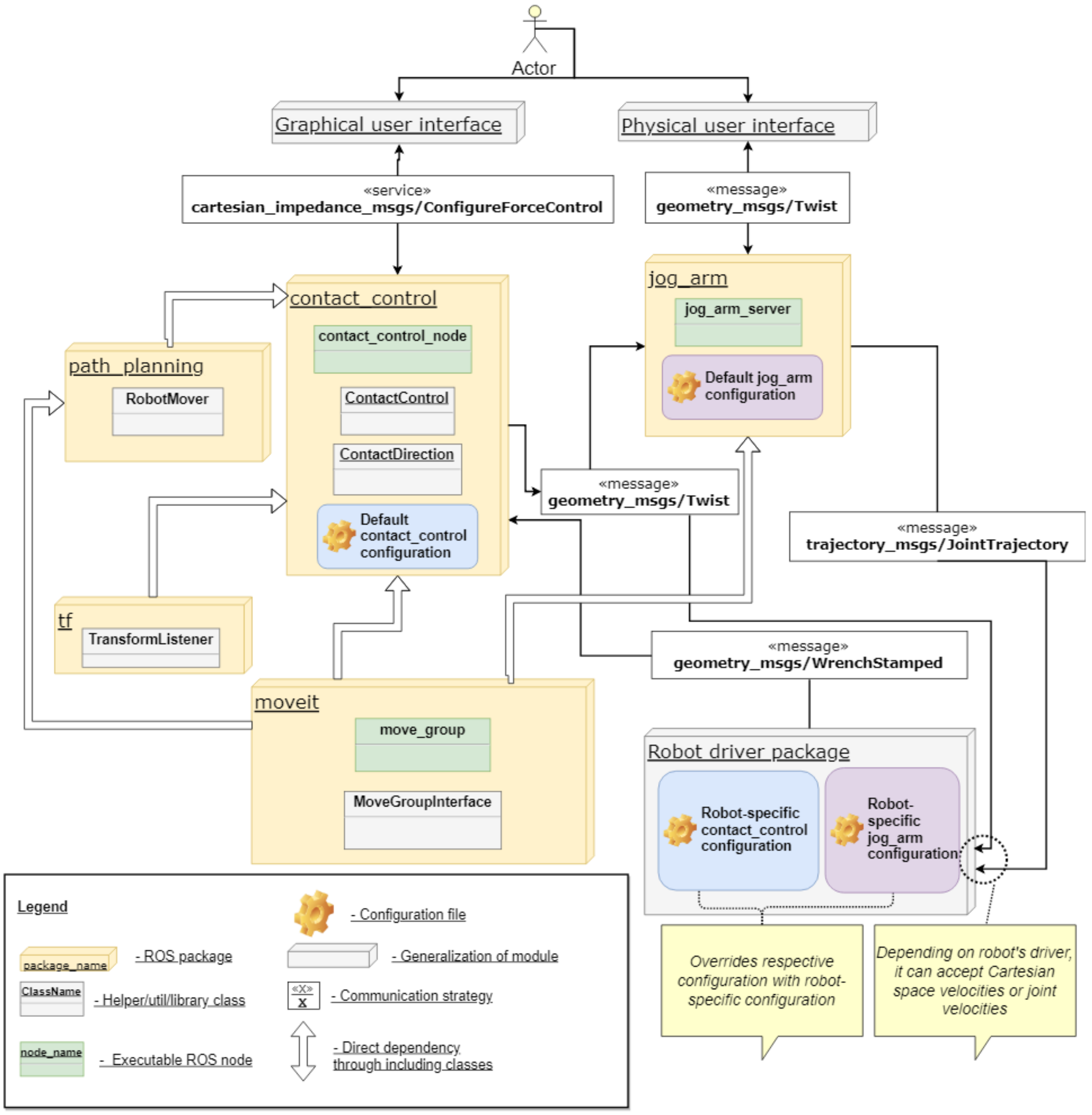

Actor
Graphical user interface
Physical user interface
«service» cartesian_impedance_msgs/ConfigureForceControl
«message» geometry_msgs/Twist
jog_arm
jog_arm_server
Default jog_arm configuration
contact_control
contact_control_node
ContactControl
ContactDirection
Default contact_control configuration
path_planning
RobotMover
tf
TransformListener
«message» geometry_msgs/Twist
«message» trajectory_msgs/JointTrajectory
«message» geometry_msgs/WrenchStamped
moveit
move_group
MoveGroupInterface
Robot driver package
Robot-specific contact_control configuration
Robot-specific jog_arm configuration
Overrides respective configuration with robot-specific configuration
Depending on robot's driver, it can accept Cartesian space velocities or joint velocities
Legend
- Configuration file
package_name - ROS package
- Generalization of module
ClassName - Helper/util/library class
«X» X - Communication strategy
node_name - Executable ROS node
- Direct dependency through including classes

Figure 5.1. *Structural architecture scheme of the software package.*

The schematic of the structural architecture of the software bundle is visualized in Figure 5.1. The arrangement describes interaction and dependencies between different modules of the system. In order to clarify the functionalities, main purpose of each component is now described. Contact control package holds contact control node, which processes force control and impedance configuration data from user and passes it on to *ContactControl* class. Contact control node acts as a middleware between GUI and *ContactControl* class by formatting and exchanging data between the two. Contact control package also includes *ContactControl* class, which processes force/torque (FT) data received from robot's driver and sends velocity commands to *jog_arm* package or directly to the controller of manipulator. The *jog_arm* package includes *jog_arm_server* node that handles kinematic calculations, in order to move manipulator's EEF in Cartesian space and continuously publishes joint trajectory messages to robot's driver. The path_planning package is helper package that can be used for path planning of the robot, for instance to set the initial pose of manipulator. MoveIt! package is used as an interface between manipulator's driver and the rest of the functionality. Essentially, MoveIt! package provides other components with *MoveGroupInterface* that communicates over ROS topics, services and actions to the *move_group* node which then provides other modules with kinematic data, such as robots current pose. The TF package [45] is used for retrieving and computing transformation between robot frames, such as fixed base frame and EEF frame. It ensures the functionality even when different robot controllers expect velocity data in different coordinate frames. Final part of the scheme is robot driver package which is always robot-specific and contains the hardware driver, MoveIt! configuration and relevant configuration files that override default configuration.

5.4 Software development process

5.4.1 Developing principles

It is crucial to define and clarify the requirements of software development process. This helps to discipline the developer to design understandable, maintainable and scalable software and makes sure that every piece of programming could be useful in future and for wider community.

Reuse of already available packages. Before starting to code a functionality, it is sensible to look into already written software. Often, there is similar functionality already implemented and it just needs some adaption or generalization. Since ROS is open community and all the code is open-source, it is easy to contribute by improving previously written code.

Maintainability. Newly written code should be maintainable, which means understandable for other developers. It is possible to follow some rules of clean code [46], including naming conventions, formatting conventions etc.

Software design rules. Software should be designed modularly, so the modules are open for extension but closed for changes in functionality. This kind of approach ensures scalability of particular software.

Documentation. Instructions for interfacing the modules and using the software in general is important for making the software open-community friendly. The purpose is to develop software for wider community and thus help robotics to improve faster in the means of reusability.

5.4.2 Development tools

Various of software development tools are essential in order to produce high-quality code and significantly speed up the development process. The appliances for software development used by the author of this thesis can be listed as follows:

- Framework for developing robotic systems. As described in section 5.3.1 (Technical architecture), the software was developed using ROS framework and its Kinetic distribution. In addition to architectural element, ROS can also be used as a development tool. It provides core functionalities, such as package management [37], code compilation tools and infrastructure [47]. Moreover, communication between nodes is standardized and the data transfer is accessible, thus debugging and monitoring of the code at runtime is practically effortless. There are number of tools available that are built for ROS that can be beneficial in system development process, including tools, described in forthcoming points.
- Simulation environment Gazebo [48] was used during the project. Newly written code might be unstable or even dangerous for robots or surrounding environment, thus it is sensible to test the code in simulation environment before launching it on real robot manipulator. Gazebo is excellent fit for this kind of task being compliant with ROS and 3D URDF models are available of both robots used in this project.
- MoveIt! is a set of software packages integrated with ROS. Along other capabilities of MoveIt! there is analytical kinematics solver, which is one key component of current thesis's software. Additionally, it provides move group interface which is used for sharing information and giving commands about robot's pose and state.
- Version control system (VCS). No considerable software projects today are developed without VCS. It is essential for various of reasons, such as tracking history of written code; keeping multiple backups of whole project; collaborating with other developers; packaging and managing different projects and their

versions into one code repository. During the development of current software project, GIT [49] was used as VCS alongside with Github web platform [50].

- Integrated development environment (IDE). IDE is graphical user interface for programming. In addition, it often offers more useful functionalities than just convenient code writing. The IDE used in the development process of current project is Clion by JetBrains [51]. Clion is focused on C and C++ writing but also offers Python support. There are numerous functionalities that Clion provides, however most convenient are syntax highlighting, automatic code formatting, GIT integration, automatic refactoring, click-navigation and references between different entities and autocomplete of lines of code.
- Operating system (OS) that was used in this project is Linux distribution Ubuntu 16.04 [52]. The OS was used for both, running as well as developing the software for manipulators. The main reason for choosing Ubuntu was ROS's prerequisites, meaning that the Kinetic version of ROS will run only on corresponding OS. Nevertheless, Ubuntu is also ideal environment for writing and managing code, as it supports various of IDEs. Again, it is perfect for controlling robot manipulators, because it is possible to install real-time kernel for low latency communication.

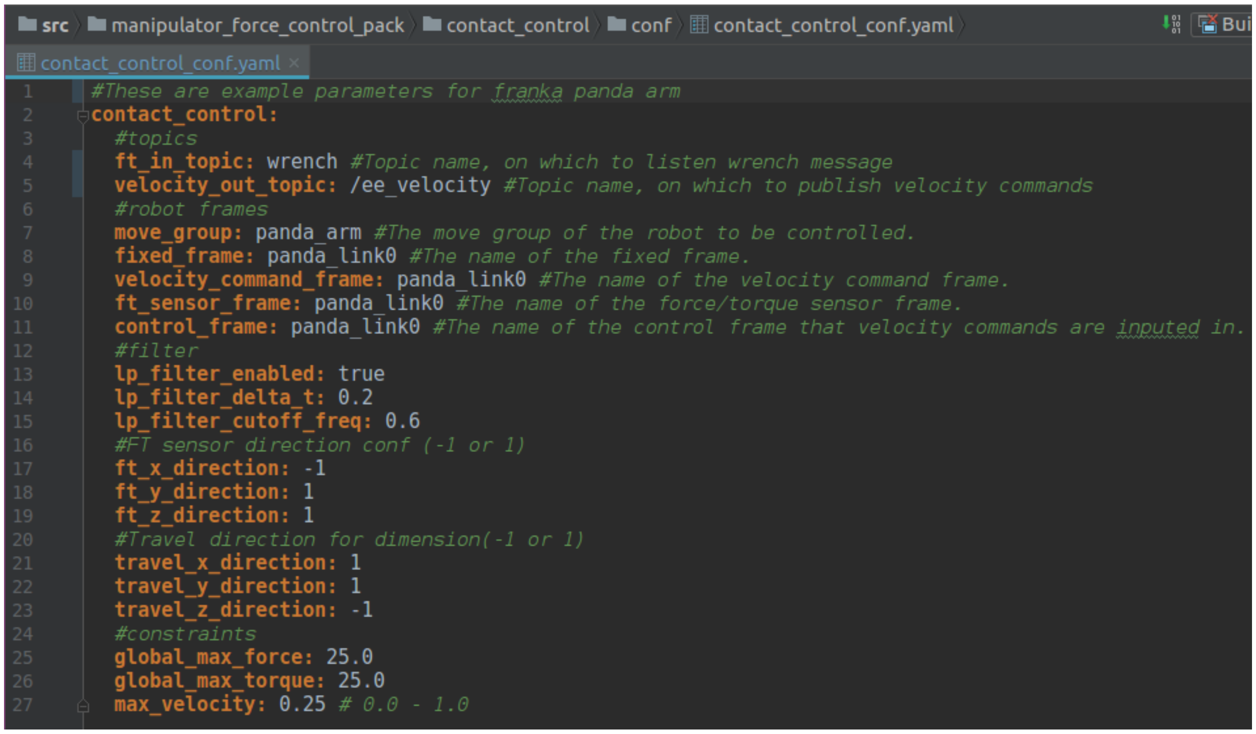
5.5 Merging and improving packages

This section introduces specific ROS packages that were used, improved and/or developed in this thesis project. General purpose and working principles are described for each software package. Additionally, problems occurred during development are mentioned along with solutions that were implemented. Finally, complete package for hardware-agnostic compliant control of manipulators is described.

5.5.1 General Contact Control framework (GCCF)

As already mentioned in section 4.4.1, GCCF is designed for processing forces on robotic manipulator and outputting EEF velocities in Cartesian space accordingly. Even though the framework was designed as hardware agnostic, it could not be used out-of-the-box. Most important modifications made during this thesis can be concluded as following:

1. Dependency on a third party *MoveInterface* was eliminated and substituted with MoveIt!'s native *MoveGroupInterface*;
2. Converting the framework package to act as a library, so it would be possible to interface it;
3. Updated software and its dependencies to ROS Kinetic (GCCF was originally designed for ROS Indigo with to end-of-life in April 2019);
4. Creating an executable server node, which interfaces the library and acts as a middleware between the GCCF framework and ROS ecosystem;
5. Improve and optimize current contact laws to be configurable and more receptive to different manipulators. The improvements include adding
    a. acceleration and deceleration of velocity,
    b. FT data thresholds for stabilizing movements,
    c. corrections of EEF travel calculations,
    d. configurable robot-specific parameters (e.g. force range and velocity range);
6. Add configurability by creating a configuration file (Figure 5.3) with different hardware-dependent variables including:
    a. force/torque data input topic, velocity output topic,
    b. move group name of the robot,
    c. different environment frames (e.g. fixed frame and EEF frame),
    d. force/torque data filter parameters and enabling,
    e. global allowed maximums of force, torque and velocity data.


src
manipulator_force_control_pack
contact_control
conf
contact_control_conf.yaml
contact_control_conf.yaml

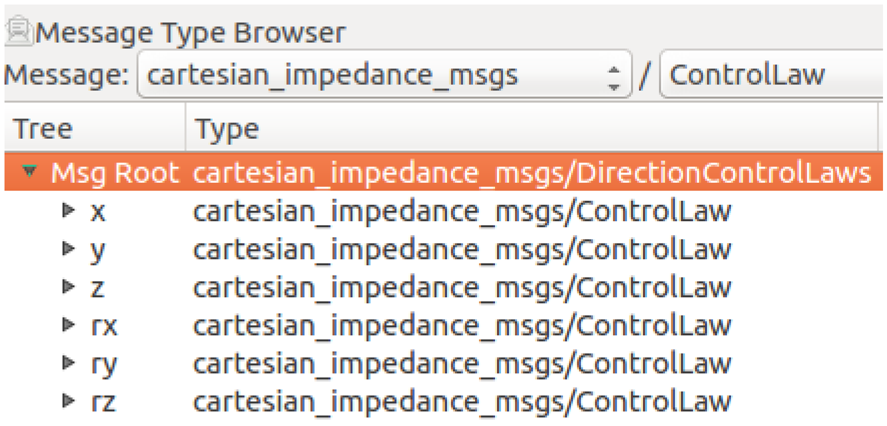
```yaml
#These are example parameters for franka panda arm
contact_control:
  #topics
  ft_in_topic: wrench #Topic name, on which to listen wrench message
  velocity_out_topic: /ee_velocity #Topic name, on which to publish velocity commands
  #robot frames
  move_group: panda_arm #The move group of the robot to be controlled.
  fixed_frame: panda_link0 #The name of the fixed frame.
  velocity_command_frame: panda_link0 #The name of the velocity command frame.
  ft_sensor_frame: panda_link0 #The name of the force/torque sensor frame.
  control_frame: panda_link0 #The name of the control frame that velocity commands are inputed in.
  #filter
  lp_filter_enabled: true
  lp_filter_delta_t: 0.2
  lp_filter_cutoff_freq: 0.6
  #FT sensor direction conf (-1 or 1)
  ft_x_direction: -1
  ft_y_direction: 1
  ft_z_direction: 1
  #Travel direction for dimension(-1 or 1)
  travel_x_direction: 1
  travel_y_direction: 1
  travel_z_direction: -1
  #constraints
  global_max_force: 25.0
  global_max_torque: 25.0
  max_velocity: 0.25 # 0.0 - 1.0
```

Figure 5.3. *Configuration file for contact control package.*

All the modifications in detail can be examined in forked software repository of the code [53]. As a result of these modifications, it is now possible to use the ROS package out-of-the-box with various of robot manipulators without changing the source code.

5.5.2 Cartesian force and impedance control messages

GSoC project was discussed in section 4.4.2 and the most useful element from the project for current thesis' goal is the generic force and impedance control messages. The messages are used in ROS service [54] which acts as an interface between user and GCCF's control laws. This way, different control laws can be applied on robot during runtime. Still, the project was not usable straightforward in this thesis context and it had to be modified. A message type was added for specifying a control law to add to corresponding axis (Figure 5.4). This enables the possibility for communicating with GCCF. The repository holding the code for the messages is accessible from [55].


Message Type Browser

Message: cartesian_impedance_msgs / ControlLaw

Tree	Type
Msg Root	cartesian_impedance_msgs/DirectionControlLaws
x	cartesian_impedance_msgs/ControlLaw
y	cartesian_impedance_msgs/ControlLaw
z	cartesian_impedance_msgs/ControlLaw
rx	cartesian_impedance_msgs/ControlLaw
ry	cartesian_impedance_msgs/ControlLaw
rz	cartesian_impedance_msgs/ControlLaw

Figure 5.4. *Outline of added message for assigning control law to every direction of EEF.*

5.5.3 Moving manipulators' EEF in Cartesian space

Controlling the EEF in Cartesian space is not a trivial task because it involves real-time kinematic calculations in order to translate between joint space and Cartesian space of the manipulator. Initially, joint level pose is received from robots' driver, forward kinematic calculations are needed for retrieving Cartesian space pose of EEF. All the velocity calculations are then done on the EEF. Inverse kinematic calculations are needed to translate new EEF velocity back to joint space velocities.

There are manipulators that already have this kind of functionality implemented on a controller level, so higher level code just has to provide Cartesian space EEF velocities. Otherwise, joint-level velocities need to be calculated on a higher level. The *jog_arm* [56] provides such higher level joint velocity calculation. It includes *jog_arm_server* node that subscribes to Cartesian velocity data and publishes joint space velocity data. The package uses MoveIt! for real-time kinematic calculations.

During current thesis, the *jog_arm* package was used for controlling UR5 manipulator. Since the package and its code was relatively fresh and not entirely tested, number of shortcomings came out during the integration. List of problems and solutions for the *jog_arm* package:

1. The ROS Kinetic branch of repository was outdated, so it did not compile. Latest updates from previous version branch were merged into Kinetic branch and compiler errors were resolved.
2. Robot in simulated environment and real robot did not react identically to the package. Additional configuration for switching between simulation and real hardware mode was added.
3. Low-pass filter is used for stabilizing joint velocities. The problem was that initial values that filter produced in its first cycles, were inaccurate. A stabilization period had to be added in order to solve this problem.
4. Several additional optimization and refactoring changes were done for smoother operation and cleaner code.

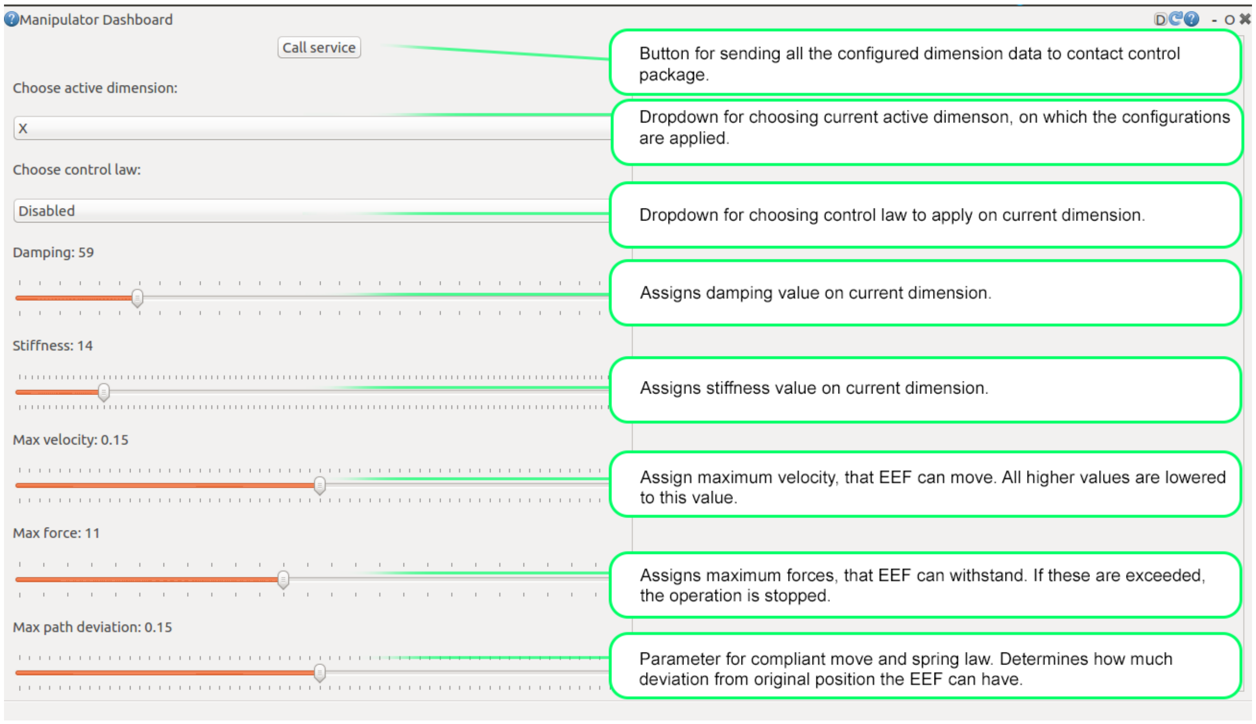
All the modifications can be viewed in detail from the *jog_arm* software repository [56].

5.5.4 User interfaces

To communicate with manipulators' control logic at runtime, user interface needs to be available. User interface can be divided into graphical (GUI) and physical (PUI) interfaces. During current thesis project, both approaches were used and developed.

PUI was mainly used for testing manipulator EEF movements in Cartesian space and manually controlling the manipulator. Two options for moving the manipulator through PUI are available – joystick and keyboard. Joystick interface functionality was already available in *jog_arm* package. For communication through keyboard, an interface had to developed, thus keyboard publisher package [57] was developed. The *key_to_twist* node in this package translates key presses into velocity commands.

GUI is required to pass compliant control rules to corresponding node at runtime. For this purpose, *manipulator_control_gui* package was developed. The GUI is developed using rqt, which is ROS a wrapper for QT toolkit [58]. The package provides understandable graphical interface for applying different control laws to different movement directions of EEF. Additionally, full configurability for every parameter is possible. All the communication between GUI and *contact_control_node* is done using ROS service calls. The GUI is shown and explained in Figure 5.5.



Button for sending all the configured dimension data to contact control package.

Dropdown for choosing current active dimenson, on which the configurations are applied.

Dropdown for choosing control law to apply on current dimension.

Assigns damping value on current dimension.

Assigns stiffness value on current dimension.

Assign maximum velocity, that EEF can move. All higher values are lowered to this value.

Assigns maximum forces, that EEF can withstand. If these are exceeded, the operation is stopped.

Parameter for compliant move and spring law. Determines how much deviation from original position the EEF can have.

Figure 5.5. *GUI for communicating with manipulator compliant control package.*

5.5.5 Hardware-agnostic compliant control software bundle

The final result of this software project is hardware-independent bundle of ROS packages to provide compliant control functionality for industrial manipulators. At the time of writing this thesis, the bundle contains six different ROS packages (Figure 5.6).

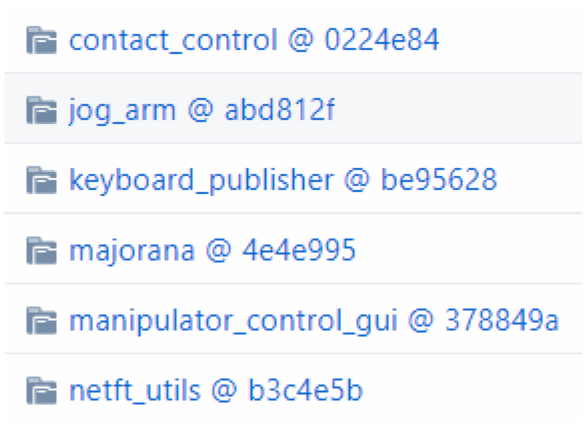

contact_control @ 0224e84
jog_arm @ abd812f
keyboard_publisher @ be95628
majorana @ 4e4e995
manipulator_control_gui @ 378849a
netft_utils @ b3c4e5b

Figure 5.6. *Compliant control repository's submodules in Github.*

The software bundle is in the form of GIT repository located under IMS Lab's (University of Tartu) robotics Github page [59]. The ROS packages are integrated into a single repository in the form of GIT submodules, which is a convenient way of nesting and managing different code repositories. The repository also provides introduction and set up guide for the software. Table 5.1 pairs a submodule to a specific functionality or project discussed in current thesis and gives functional overview of the software bundle submodules.

Table 5.1. *Overview of compliant control software bundle.*

Submodule name	**Purpose**	**Section(s) discussed in current thesis**
contact_control	Implements functionality of General Contact Control framework (GCCF)	4.4.1, 5.5.1
jog_arm	Provides functionality for moving manipulator's EEF in Cartesian space	5.5.3
keyboard_publisher	Physical user interface	5.5.4
majorana	GSoC Project - ROS Interface for Impedance/Force Control	4.4.2, 5.5.2
maniplator_control_gui	Graphical user interface	5.5.4
netft_utils	A dependency for contact_control package. It provides utils for FT data processing, such as filtering and assigning movements constraints.	-

5.5.6 Robot-specific custom configuration packages

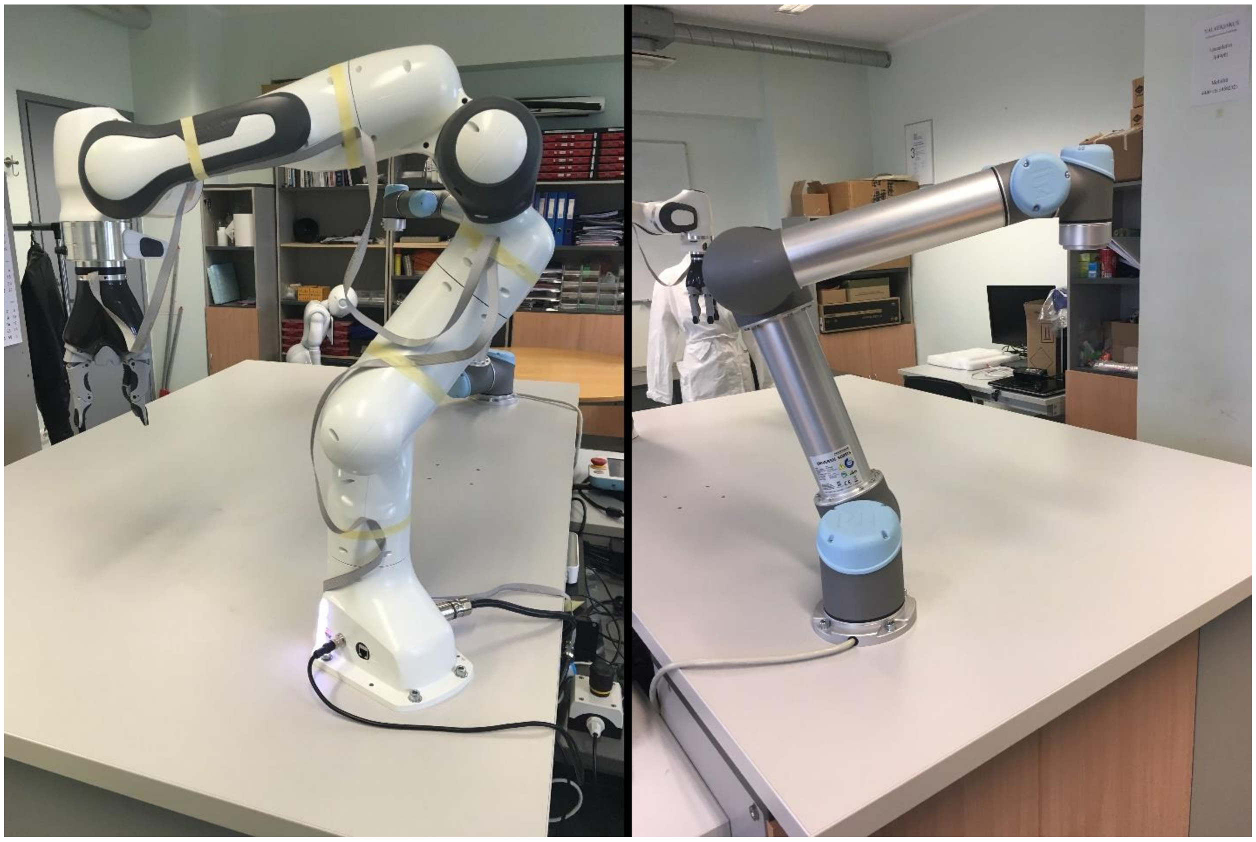
Two additional robot-specific packages were developed to override or extend default configuration, URDF models and ROS launch files. Therefore two additional repositories were initiated, one for UR5 manipulator [60] and the other for Panda manipulator [61]. These repositories include configuration files that override default configuration of compliant control package, robot controllers and EEF jogger. Also, customized URDF models are constructed and the real environment (e.g. table that the robot is fixed on) for the robot is included so that collision detection functionality would work. Finally, extended ROS launch files are included for launching different controllers, drivers and compliant control functionality and load custom configuration all at once.

6 Testing and demonstrations

6.1 Setup of demonstrations

6.1.1 Hardware setup

Tests and demonstrations were executed on two different serial manipulators: 6-DOF Universal Robots UR5 and 7-DOF Franka Emika Panda (Figure 6.1).



(a) (b)

Figure 6.1. *The robot manipulators on which the testing and demonstrations were executed: (a) Franka Emika Panda, (b) Universal Robots UR5.*

Universal Robots (UR5)

UR5 (Figure 6.1(b)) is produced by Danish company Universal Robots, which has been producing co-robots since 2009 [62]. UR5 belongs to UR family along with UR3 and UR10 (the numbers designate the payload in kg). The UR5 is a 6-DOF robot with 5 kg of payload and positional repeatability of ±0.1 mm [11]. It has integrated passive compliance - safety force and torque limits, estimated from joint actuator currents, are assigned to joints and these cannot be exceeded even if robot has more power [63]. For UR5, these limits vary between 28 Nm and 150 Nm [63].

Franka Emika Panda

A robot manipulator named Panda (Figure 6.1(b)) is produced by Germany company Franka Emika. The 7-DOF robot has a payload of 3 kg and ±0.1 mm positional repeatability [64]. Panda has separate torque sensors in all 7 joints, therefore it provides more accurate and stable FT data than UR5 [64].

6.1.2 Software setup

Along with the software bundle developed in current thesis, there are robot-specific control packages and ROS utility software, that facilitates the testing of the software bundle.

Universal Robots UR5

UR5 can be controlled through URScript API [65]. ROS drivers, using this API, have been developed by the ROS community and are available for integrating the robot into ROS ecosystem. The ROS drivers package [66] contains UR5-specific configuration, which is used for testing the compliant control software on the manipulator.

Franka Emika Panda

Franka Emika provides open source Franka Control Interface (FCI) for controlling Panda [67]. Besides low-level C++ library, the interface contains ROS integration package [68] along with MoveIt! configuration and URDF models that are used for interconnecting Panda to ROS ecosystem.

Common utility software

In order to facilitate execution of tests and demonstrations, different ROS provided software was used. RViz [69] is robot 3D visualization (Figure 6.2) and communication tool that was used for monitoring robot's current pose, FT data and executing path planning commands.

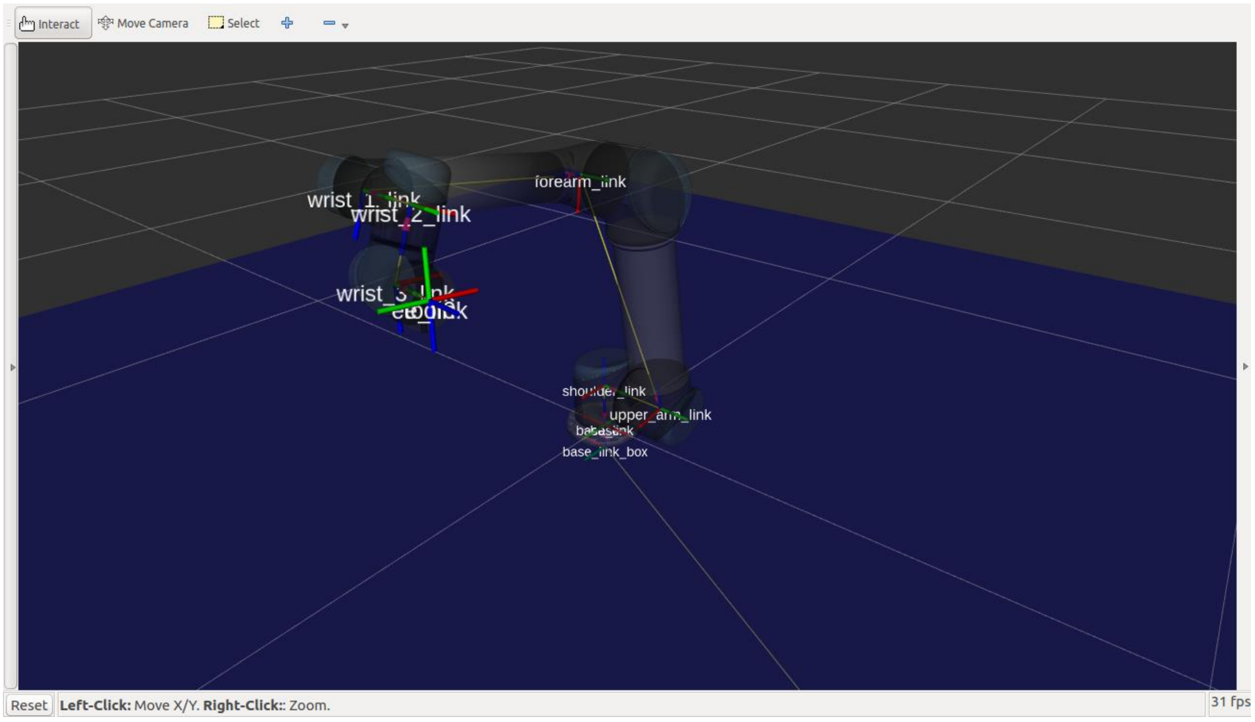


Figure 6.2. *A screenshot of RViz, robot visualization environment. An UR5 robot is visualized with the layout of its serial kinematic chain.*

Second tool that was used is Gazebo simulation environment (Figure 6.3). Virtual simulated robot was loaded into Gazebo, which then was controlled with the software. The software controlled the virtual robot as it was real one. Before any test case was executed on the real robot, it was first tested in simulation environment to ensure consistency of basic functionality.

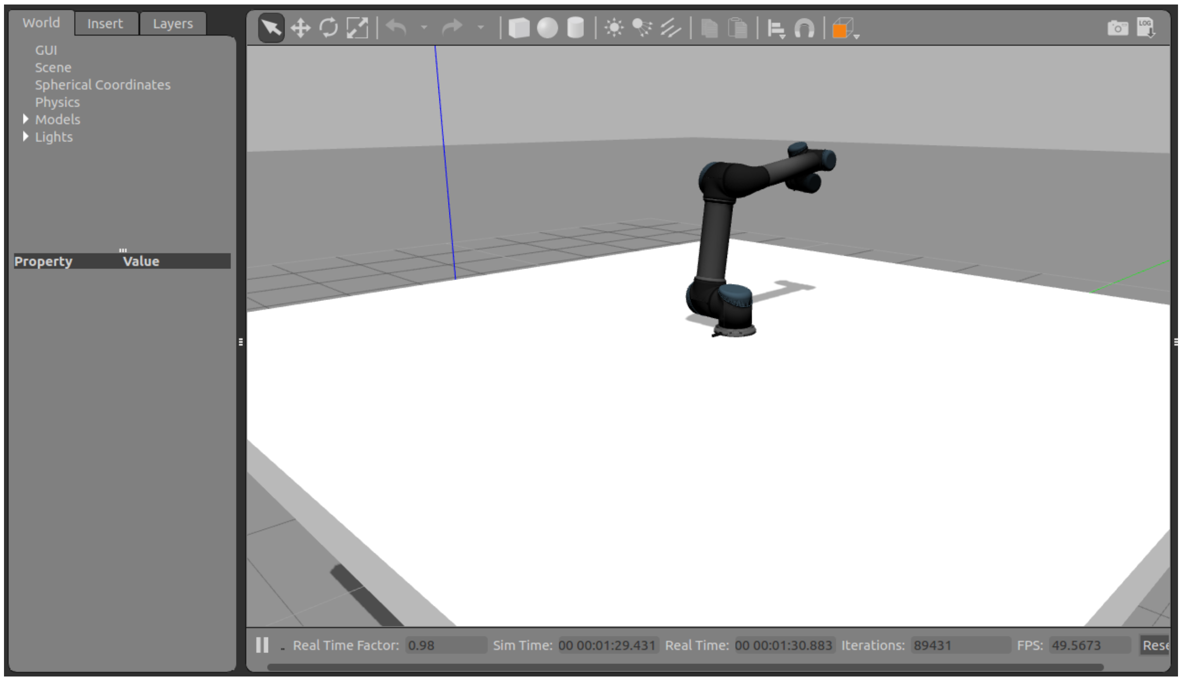
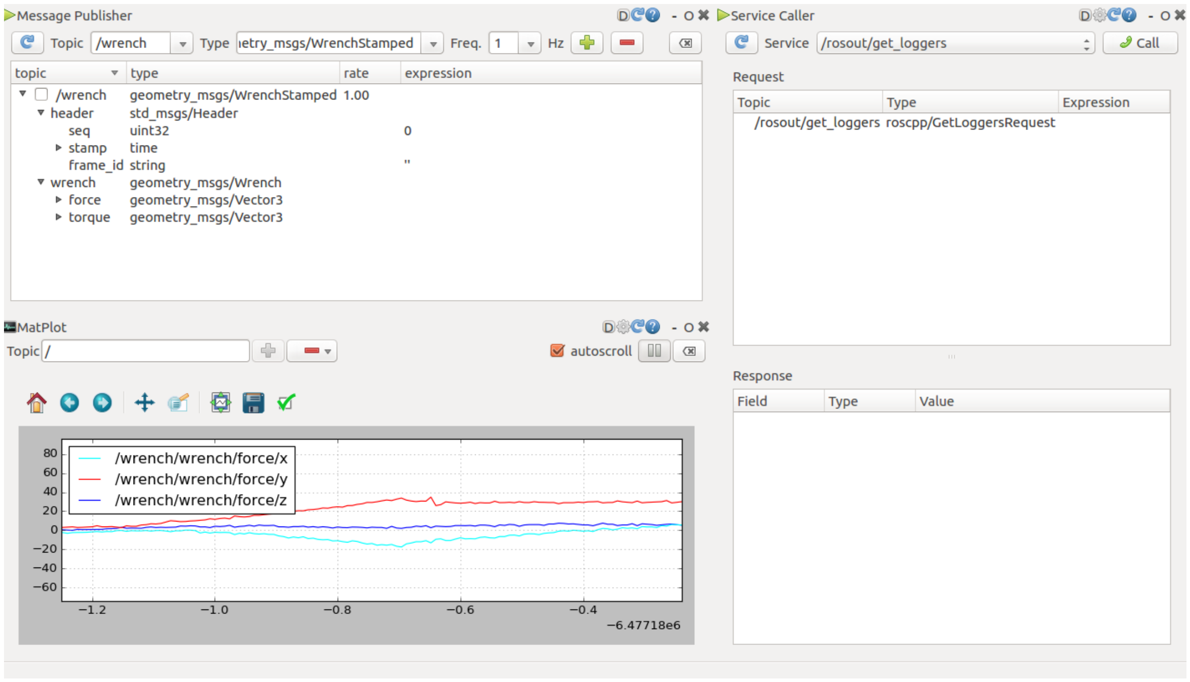


Figure 6.3. *A screenshot of Gazebo with a virtual UR5 robot loaded.*

Furthermore, number of different rqt plugins (Figure 6.4) [58] were used during testing:

- Plot – for plotting and visualizing sensor and velocity data on graph;
- Message publisher – for manually publishing various of data for testing purposes (e.g. FT data and EEF velocities);
- Topic monitor – for monitoring published messages in real time;
- Service caller – for manually communicating with nodes that use services;
- Node graph – for getting overview of interconnections between ROS nodes;
- TF tree – for viewing robots' transformation tree.



topic	type	rate	expression
/wrench	geometry_msgs/WrenchStamped	1.00	
header	std_msgs/Header		
seq	uint32		0
stamp	time		
frame_id	string		''
wrench	geometry_msgs/Wrench		
force	geometry_msgs/Vector3		
torque	geometry_msgs/Vector3		

Topic	Type	Expression
/rosout/get_loggers	roscpp/GetLoggersRequest	

Field	Type	Value

Figure 6.4. *A screenshot of rqt application with three plugins loaded, message publisher plugin, plot plugin and service caller.*

6.2 Initial testing

In order to effectively and safely test the software and its compatibility with hardware, a testing plan was prepared. The plan includes 5 different test cases (Table 6.1). The test plan is designed to cover all needed functionality while iteratively getting closer to real use scenario, where real robot is using data from GCCF. The testing steps are following:

1. Test GCCF in fully simulated environment. That means, a simulated robot is loaded into Gazebo and the FT data is published manually using rqt plugin. The purpose of this test is to confirm that GCCF functionality works.
2. Test jogger on a real hardware. Meaning manually publishing velocity messages using keyboard interface. The aim of this test is to assure that the controller can move its EEF linearly in Cartesian space.

3. Test GCCF package on simulated robot so that FT data comes from real robot. Goal of this test is to determine, how the real manipulator may respond to signals coming from GCCF while using real FT data.
4. Try GCCF on a real hardware while publishing FT data manually. Purpose of this test is to see, how the robot responds to velocity commands published by GCCF.
5. Full hardware test with real time FT data and actual robot. The test confirms, whether the full package works together with hardware.

Table 6.1. *Overview of testing iterations.*

Test iteration	**Robot used**	**FT data source**	**Velocity data source**
1.	Simulated	Manual publishing	GCCF
2.	Hardware	-	Manual publishing
3.	Simulated	Robot driver	GCCF
4.	Hardware	Manual publishing	GCCF
5.	Hardware	Robot driver	GCCF

6.3 Proof-of-concept with UR5

The listed test cases (Table 6.1) were executed on UR5 manipulator or its simulated version. Every iteration was carefully monitored and analyzed. Considering the results of testing, monitoring, analyzing, and programming, the five iterations were cyclically repeated until satisfactory results were achieved.

In the final test case, follower law of GCCF was used, meaning that robot’s EEF should move in the direction where it is pushed. In the end it was observed that the UR5 robot’s EEF started to move even if no forces were applied to it. However, when applying relatively high force on the EEF, it started to move towards desired direction. Another problem that was identified was that when no force had been applied, the EEF started to resonate. That is because FT data computed from the robot’s joint currents are influenced by robot joint movements. As a result of such circular dependency the robot starts oscillating. Furthermore, force sensitivity of UR5 robot is rather inaccurate and unstable (Figure 6.5), thus very high thresholds for processing FT data had to be configured resulting in very stiff EEF compliancy.

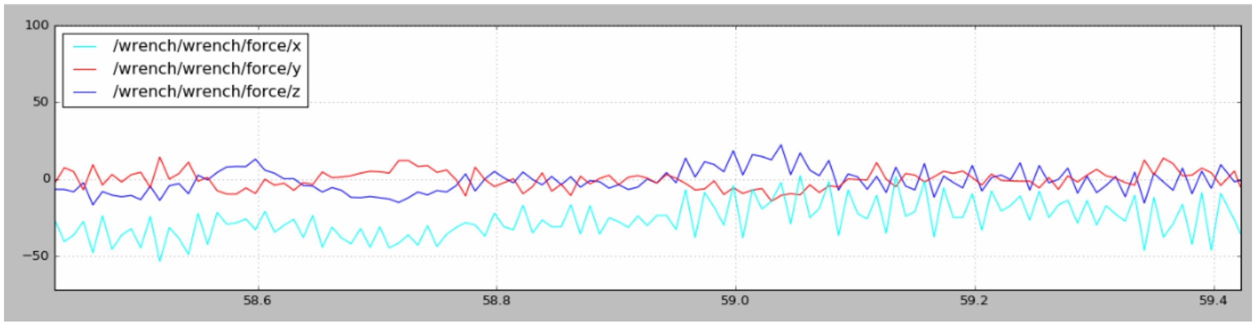


Figure 6.5. *A timeline of an actual FT data flow from UR5 controller. Forces (N) on EEF different axis are presented.*

Since named problems are highly hardware-dependent, one can still say that the testing was successful and at this point, it was demonstrated on a proof-of-concept level that the developed software was working properly.

6.4 Final demonstrations

The purpose of final demonstrations is to show that the goals of the software project are achieved:

1. the different control laws of GCCF are working;
2. software can be run on multiple different manipulators with only changing the configuration;
3. the software bundle is relatively easy to set up;
4. the software is convenient to use.

The set of demonstrations are done with Franka Emika Panda manipulator, because this robot has better force sensitivity than UR5, due to dedicated FT sensors in every joint. For every demonstration case, the compliant control configuration command is sent to GCCF through the service that uses force and impedance control messages.

6.4.1 Follower law

<u>Configuration:</u>

Dimension: Z

Control law: follower law

Damping parameter: 30

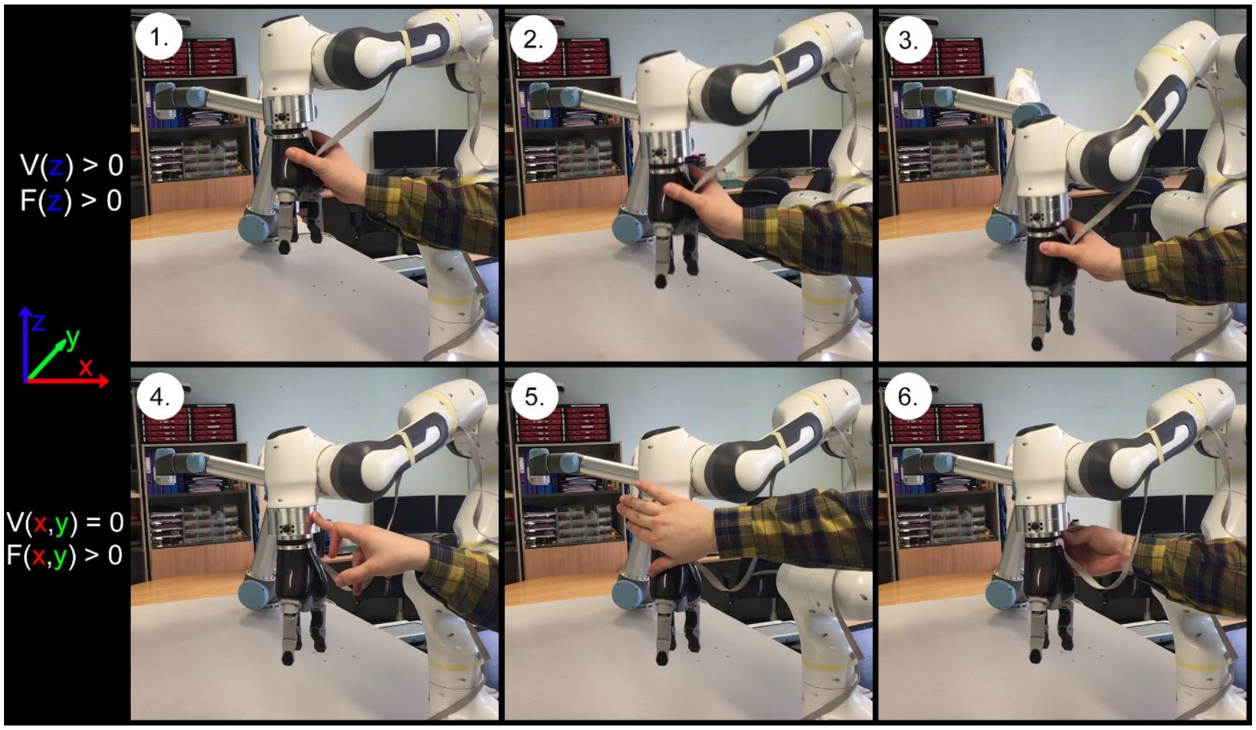


Figure 6.6. *Follower law demonstration.*

The follower law demonstration is shown in Figure 6.6. In subfigures 1–3, there is force applied on Z axis of EEF and it is demonstrated that the robot is compliant in the Z direction. In subfigures 4–5, force is applied in X and Y axis and the EEF will not move. This demonstration shows that it is possible to assign follower law on distinctive directions.

6.4.2 Compliant move law

<u>Configuration:</u>

Dimension: Z

Control law: compliant move

Maximum path deviation: -

Maximum force: 0.5 N

Maximum torque: 0.5 N

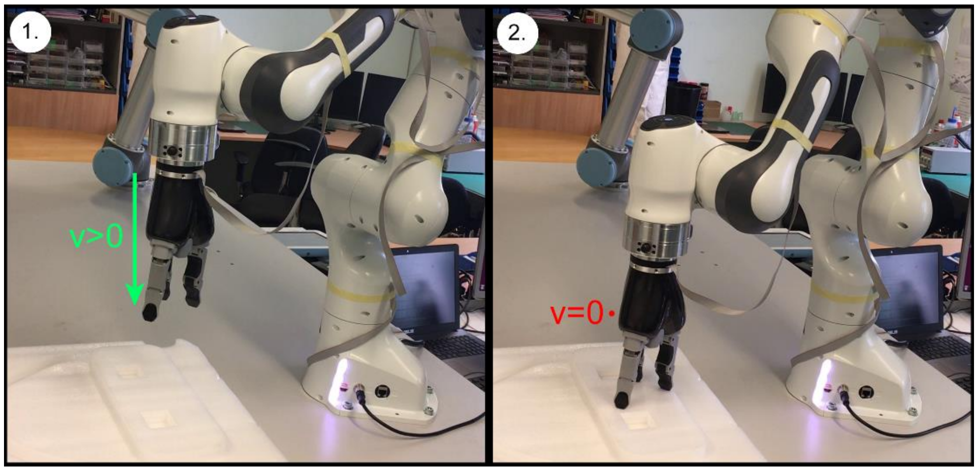


Figure 6.7. *Compliant move law demonstration.*

Compliant move law is applied on Z axis of EEF (Figure 6.7). This causes the EEF to move towards the obstacle. Since maximum force and torque parameters are provided, the robot stops the movement, when it reaches the obstacle.

6.4.3 Spring law

<u>Configuration:</u>

Dimension: X

Control law: spring law

Damping parameter: 30

Spring parameter: 100

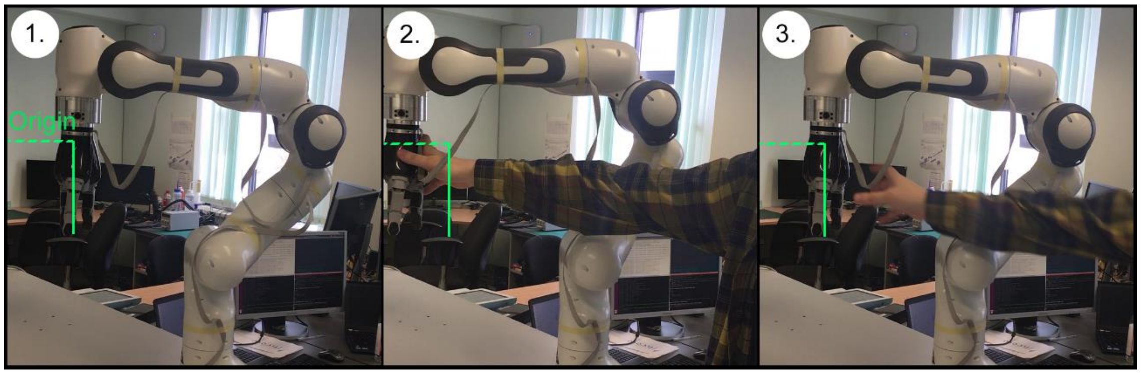


Figure 6.8. *Spring law demonstration.*

Spring law (impedance control) is applied on EEF X axis. On the Figure 6.8, the original position is marked with green line (subfigure 1 in Figure 6.8). When the force is applied on the direction, the EEF moves away from its original position (subfigure 2). When the force is released from EEF, the robot moves back to its original position (subfigure 3).

6.4.4 Hybrid control

Configuration:

Dimension: X	Dimension: Z
Control law: compliant move	Control law: spring law
Maximum path deviation: 30 cm	Damping parameter: 30
Maximum force: 0.5 N	Spring parameter: 100
Maximum torque: 0.5 N	

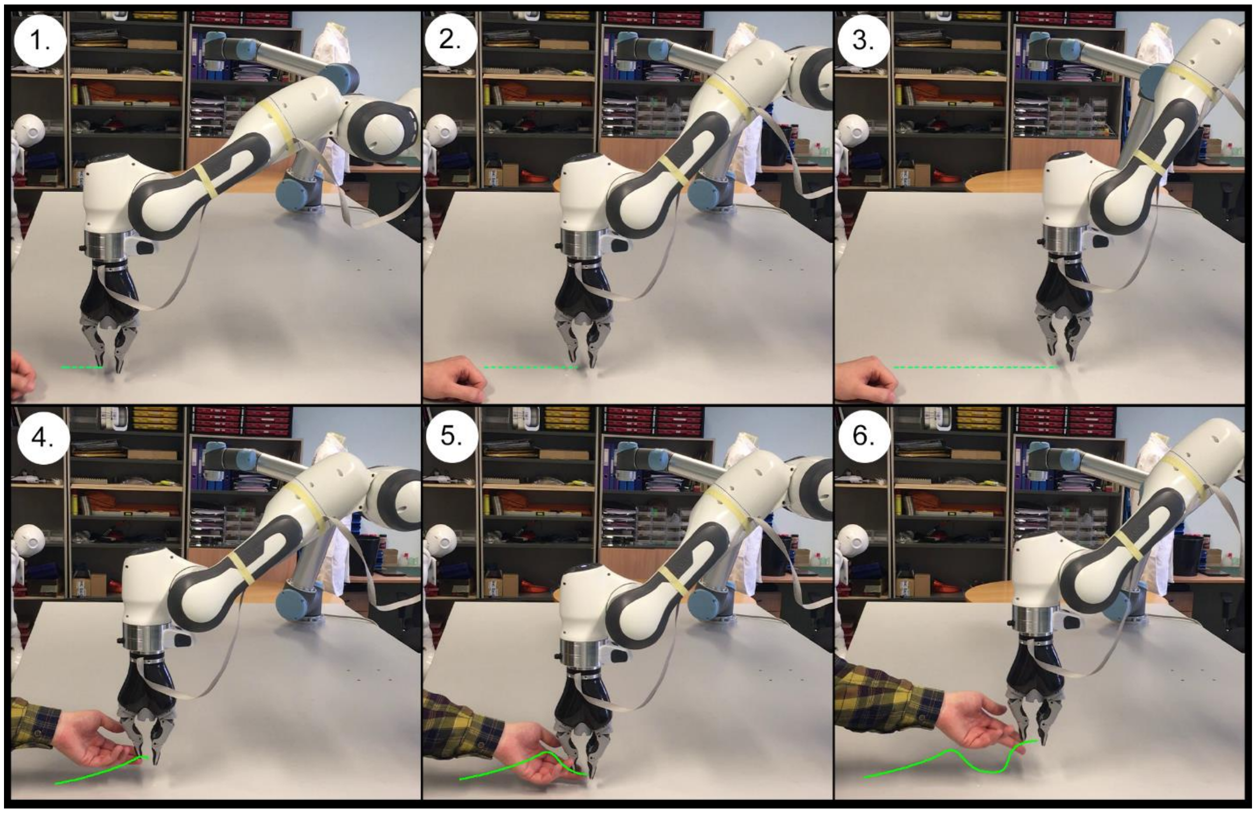


Figure 6.9. *Hybrid control demonstration.*

Hybrid control demonstration (Figure 6.9) shows that it is possible to assign different control laws on different dimensions. The compliant move law is configured for X axis of EEF and spring law is configured for Z axis. Initially, the robot moves its' EEF in straight line (subfigures 1–3 in Figure 6.9). However, when some forces are applied on Z axis, the EEF will follow these forces (subfigures 4–6) while continuing to move along X axis.

7 Conclusion and future work

As a result of this thesis, a ROS software bundle for compliant control was developed and is available for community from the code repository in Github along with documentation for setting it up [59]. The software bundle contains different packages that were either modified previous work or developed from scratch. Important improvements on the previous work include making the software hardware-agnostic and configurable, developing new user interfaces, optimizing control laws and updating submodules to work with newer versions of ROS. The software was demonstrated to be operational and hardware-agnostic by testing it on two different manipulators: Universal Robots UR5 and Franka Emika Panda. Resulting software meets described prerequisites and is ready to be used and it is developed in a modular way that supports extendibility.

Since the software is open source and built as hardware independent, it is available for using and improving for developers and users worldwide. There are many improvements that could be introduced to the software bundle, e.g.

- extending hardware-independency by running the software on more different manipulators;
- developing new, more complex contact control laws, such as gravity compensation or balance maintaining;
- developing generic middleware that would manage the logic when different control laws should be applied;
- different general optimizations, code refactorings and maintenance that would come up when the software is engaged in practice.

It is important that the software package is kept under constant maintenance and improvement which can be achieved only by employing the software in real applications. Therefore, future work also includes promoting the software for wider community in order for it to grow and spread faster, thus resulting in better code quality and extended functionalities. Final goal is to get this bundle to grow naturally, include developers worldwide and to be taken into use on many different robot manipulators.

8 Summary

The main goal of this thesis was to construct ROS software bundle for compliant control of collaborative industrial robots. The software should provide easy usage for end user, extendibility for other developers, and hardware independency in order to use it on a wide range of manipulators.

Firstly, the overview of the field was given to emphasize the importance of industrial robots in general, co-robots and the concept of physical human-robot interaction. Open challenges in industrial robotics and in collaborative robotics were analyzed and concluded.

The literature review introduced fundamental concepts of human-robot collaboration, physical human-robot interaction, and compliant control. Additionally, description of previous work in the field was given and the motivation for chosen project was explained.

Software implementation chapter specifies the functional and technical requirements for the software bundle. Furthermore, the architecture of the software is presented. Also, the development principles and process are discussed and finally improvements as well as new developed functionality is explained.

The testing and demonstrations chapter covers software testing plan and execution along with final demonstrations with their goal and results.

9 Kokkuvõte

Riistvarapaindlik ROSi tarkvarapakett tööstuslike robotite möödlikuks juhtimiseks

Martin Appo

Käesoleva magistritöö põhieesmärgiks oli ROSi tarkvarapaketi arendamine tööstusliku roboti möödlikuks juhtimiseks (ingl "*compliant control*"). See tähendab, et manipulaator allub välistele jõududele ning reageerib vastavalt eelseadistatud juhistele. Arendatav tarkvara peab pakkuma lihtsasti arusaadava kasutajaliidese. Lisaks on tähtis, et tarkvara oleks edasiarendatav ka järgmiste arendajate poolt ning riistvarast sõltumatu, et seda saaks kasutada mitme erineva roboti juhtimiseks.

Esiteks antakse selles magistritöös valdkonnaülevaade, et tuua esile tööstusrobootika tähtsus tänapäeval. Lisaks tutvustatakse koostöövõimelisi roboteid ning selle kontseptsiooni olulisust väiksematele ettevõtetele. Analüüsitakse kaasaegseid valdkonna-spetsiifilisi probleeme ning pakutakse välja võimalikud lahendused.

Kirjanduse ülevaade tutvustab põhilisi alustõdesid inimese ja roboti koostööst, füüsilisest inimese ja roboti ühistööst ning möödlikust roboti juhtimisloogikast. Lisaks kirjeldatakse ära tarkvarapaketi põhjaks valitud projektide põhimõte ning põhjused, miks konkreetsed projektid valiti.

Tarkvara implementatsiooni kirjeldavas peatükis tuuakse välja funktsionaalsed ja tehnilised nõuded tarkvaralahendusele. Lisaks esitatakse ning kirjeldatakse ära tarkvara arhitektuur. Eraldi käsitletakse arendustöö printsiipe ning protsessi. Viimaseks kirjeldatakse detailsemalt, mida konkreetselt kasutatud tarkvarapakettides muudeti ning mida juurde arendati.

Viimane peatükk katab tarkvaratestimise plaani ning olulisemad tulemused. Veel kirjeldatakse lahti ning visualiseeritakse lõplikud demonstratsioonid riistvaral.

10 References

[1] D. Nitzan and A. C. Rosen, “Programmable Industrial Automation,” *Trans. Comput.*, vol. C-25, pp. 1259–1270, 1976.

[2] P. Gorle and A. Clive, “Positive Impact of Industrial Robots on Employment,” *Iinternational Fed. Robot.*, no. January, p. 70, 2013.

[3] W. Robotics, “Executive Summary - World Robotics (Industrial & Service Robots) 2014,” *World Robot. Rep. - Exec. Summ.*, pp. 15–24, 2014.

[4] M. Hägele, J. N. P. Klas Nilsson, and R. Bischoff, “Industrial Robotics: The Main Driver for Robotics Research and Application,” in *Springer Handbook of Robotics*, 2nd ed., Berlin: Springer-Verlag, 2016, p. 1386.

[5] V. D. Hunt, “Economics of Robotics,” in *Industrial Robotics Handbook*, New York: Industrial Press Inc, 1983.

[6] A. Airaksinen, H. Luomaranta, P. Alajääskö, and A. Roodhuijzen, “Statistics on small and medium-sized enterprises,” 2015. [Online]. Available: http://ec.europa.eu/eurostat/statistics-explained/index.php/Statistics_on_small_and_medium-sized_enterprises. [Accessed: 25-Apr-2018].

[7] A. Jäger, C. Moll, and C. Lerch, *Analysis of the impact of robotic systems on employment in the European Union - 2012 data update*. 2016.

[8] M. R. Pedersen *et al.*, “Robot skills for manufacturing: From concept to industrial deployment,” *Robotics and Computer-Integrated Manufacturing*, vol. 37. pp. 282–291, 2016.

[9] E. Dean, K. Ramirez Amaro, F. Bergner, I. Dianov, and G. Cheng, “Integration of Robotic Technologies for Rapidly Deployable Robots,” *IEEE Trans. Ind. Informatics*, vol. 14, no. 4, pp. 1691–1700, 2017.

[10] G. Biggs and B. Macdonald, “A Survey of Robot Programming Systems,” *Proc. Australas. Conf. Robot. Autom.*, pp. 1–3, 2003.

[11] R. Bogue, “Europe continues to lead the way in the collaborative robot business,” *Ind. Robot An Int. J.*, vol. 43, no. 1, pp. 6–11, 2016.

[12] J. Heinzmann and A. Zelinsky, “Quantitative Safety Guarantees for Physical

Human-Robot Interaction,” *Int. J. Rob. Res.*, vol. 22, no. 7–8, pp. 479–504, 2003.

[13] M. Kraft and M. Rickert, “How to Teach Your Robot in 5 Minutes: Applying {UX} Paradigms to Human-Robot-Interaction,” *Proc. {IEEE} Int. Symp. Robot Hum. Interact. Commun.*, pp. 942–949, 2017.

[14] “ROS,” 2018. [Online]. Available: http://www.ros.org/. [Accessed: 26-Apr-2018].

[15] M. Hägele, K. Nilsson, J. N. Pires, and R. Bischoff, “Outlook and Long-Term Challenges,” in *Springer Handbook of Robotics*, 2nd ed., Berlin: Springer-Verlag, 2016, pp. 1416–1418.

[16] J. Laaksonen, J. Felip, A. Morales, and V. Kyrki, “Embodiment independent manipulation through action abstraction,” *Proc. - IEEE Int. Conf. Robot. Autom.*, pp. 2113–2118, 2010.

[17] R. H. Andersen, L. Dalgaard, A. B. Beck, and J. Hallam, “An architecture for efficient reuse in flexible production scenarios,” *IEEE Int. Conf. Autom. Sci. Eng.*, vol. 2015–Octob, pp. 151–157, 2015.

[18] R. H. Andersen, T. Sølund, and J. Hallam, “Definition and Initial Case-Based Evaluation of Hardware-Independent Robot Skills for Industrial Robotic Co-Workers,” *Proc. 41st Int. Symp. Robot. (ISR/Robotik 2014)*, pp. 101–107, 2014.

[19] “Journal of Human-Robot Interaction,” 2018. [Online]. Available: http://humanrobotinteraction.org/journal/index.php/HRI. [Accessed: 15-Apr-2018].

[20] “Human-Robot interaction conferences,” 2018. [Online]. Available: http://humanrobotinteraction.org/category/conference/. [Accessed: 15-Apr-2018].

[21] A. Tellaeche, I. Maurtua, and A. Ibarguren, “Human robot interaction in industrial robotics. Examples from research centers to industry,” *2015 IEEE 20th Conf. Emerg. Technol. Fact. Autom.*, pp. 1–6, 2015.

[22] S. Haddadin and E. Croft, “Physical Human-Robot Interaction,” in *Springer Handbook of Robotics*, 2nd ed., Berlin: Springer-Verlag, 2016, pp. 1836–1867.

[23] A. Kouris, F. Dimeas, and N. Aspragathos, “A Frequency Domain Approach for Contact Type Distinction in Human-Robot Collaboration,” vol. 3, no. 2, pp. 1–6, 2018.

[24] A. R. Sadik and B. Urban, “A Holonic Control System Design for a Human & Industrial Robot Cooperative Workcell,” *Proc. - 2016 Int. Conf. Auton. Robot Syst.*

Compet. ICARSC 2016, pp. 118–123, 2016.

[25] M. Geravand, F. Flacco, and A. De Luca, “Human-robot physical interaction and collaboration using an industrial robot with a closed control architecture,” *Int. Conf. Robot. Autom.*, pp. 4000–4007, 2013.

[26] K. Suita, Y. Yamada, N. Tsuchida, K. Imai, H. Ikeda, and N. Sugimoto, “A failure-to-safety ‘Kyozon’ system with simple contact detection and stop capabilities for safe human-autonomous robot coexistence,” *Proc. 1995 IEEE Int. Conf. Robot. Autom.*, vol. 3, pp. 3089–3096, 1965.

[27] G. A. Pratt and M. M. Williamson, “Series elastic actuators,” *Proc. 1995 IEEE/RSJ Int. Conf. Intell. Robot. Syst. Hum. Robot Interact. Coop. Robot.*, vol. 1, pp. 399–406, 1995.

[28] S. Ivaldi, M. Fumagalli, M. Randazzo, F. Nori, G. Metta, and G. Sandini, “Computing robot internal/external wrenches by means of inertial, tactile and F/T sensors: Theory and implementation on the iCub,” *IEEE-RAS Int. Conf. Humanoid Robot.*, pp. 521–528, 2011.

[29] M. T. Mason, “Compliance and Force Control for Computer Controlled Manipulators,” *IEEE Trans. Syst. Man Cybern.*, vol. 11, no. 6, pp. 418–432, 1981.

[30] R. A. von Sternberg, “GCCF: A Generalized Contact Control Framework APPROVED BY SUPERVISING COMMITTEE,” The University of Texas at Austin, 2016.

[31] J.-P. Merlet, “C-surface applied to the design of an hybrid force-position robot controller,” *Proceedings. 1987 IEEE Int. Conf. Robot. Autom.*, vol. 4, pp. 1055–1059, 1987.

[32] M. H. Raibert and J. J. Craig, “Hybrid Position/Force Control of Manipulators,” *J. Dyn. Syst. Meas. Control*, vol. 103, no. 2, p. 126, 1981.

[33] N. Hogan, “Impedance Control: An Approach to Manipulation,” *IEEE Am. Control Conf.*, pp. 304–313, 1984.

[34] “Nodes,” 2012. [Online]. Available: http://wiki.ros.org/Nodes. [Accessed: 09-Apr-2018].

[35] R. Kojcev, “Google Summer of Code Project - ROS Interface for Impedance/Force Control,” 2016. [Online]. Available:

https://rosindustrial.org/news/2016/9/16/google-summer-of-code-project-. [Accessed: 30-Mar-2018].

[36] P. Hvass, “Cartesian Path Planner Plug-In for MoveIt!,” 2015. [Online]. Available: https://rosindustrial.org/news/2014/9/5/cartesian-path-planner-plug-in-for-moveit. [Accessed: 28-Apr-2018].

[37] “ROS package,” 2015. [Online]. Available: http://wiki.ros.org/Packages. [Accessed: 30-Apr-2018].

[38] “ROS communication patterns,” 2014. [Online]. Available: http://wiki.ros.org/ROS/Patterns/Communication. [Accessed: 30-Apr-2018].

[39] “ROS-Industrial Robot Driver Specification,” 2017. [Online]. Available: http://wiki.ros.org/Industrial/Industrial_Robot_Driver_Spec. [Accessed: 01-Apr-2018].

[40] “ROS Messages,” 2016. [Online]. Available: http://wiki.ros.org/Messages. [Accessed: 09-Apr-2018].

[41] S. Chitta, I. Sucan, and S. Cousins, “MoveIt! [ROS Topics],” *IEEE Robot. Autom. Mag.*, vol. 19, no. 1, pp. 18–19, 2012.

[42] “Wrench Message,” 2017. [Online]. Available: http://docs.ros.org/indigo/api/geometry_msgs/html/msg/Wrench.html. [Accessed: 09-Apr-2018].

[43] D. Kortenkamp, R. Simmons, and D. Brugali, “Robotic Systems Architectures and Programming,” in *Springer Handbook of Robotics*, 2nd ed., Berlin: Springer-Verlag, 2016, pp. 283–302.

[44] “ROS Kinetic.” [Online]. Available: http://wiki.ros.org/kinetic. [Accessed: 15-May-2018].

[45] “ROS TF,” 2017. [Online]. Available: http://wiki.ros.org/tf. [Accessed: 01-May-2018].

[46] L. Prechelt, G. B. Uml, M. Feathers, R. Jeffries, and W. C. Xp, “The essence of ‘ Clean Code ,’” 2014.

[47] “Catkin,” 2017. [Online]. Available: http://wiki.ros.org/catkin. [Accessed: 06-May-2018].

[48] “Gazebo,” 2018. [Online]. Available: http://gazebosim.org/. [Accessed: 07-May-

2018].
[49] “GIT,” 2018. [Online]. Available: https://git-scm.com/. [Accessed: 06-May-2018].
[50] “Github,” 2018. [Online]. Available: https://github.com/. [Accessed: 06-May-2018].
[51] “Clion,” 2018. [Online]. Available: https://www.jetbrains.com/clion/. [Accessed: 06-May-2018].
[52] “Ubuntu 16.04,” 2018. [Online]. Available: http://releases.ubuntu.com/16.04/. [Accessed: 06-May-2018].
[53] M. Appo, “Contact control forked repository,” 2018. [Online]. Available: https://github.com/ut-ims-robotics/contact_control. [Accessed: 07-May-2018].
[54] “ROS services,” 2018. [Online]. Available: http://wiki.ros.org/Services. [Accessed: 27-Apr-2018].
[55] “Majorana (General force/torque control messages),” 2018. [Online]. Available: https://github.com/ut-ims-robotics/majorana. [Accessed: 07-May-2018].
[56] “Jog arm repository,” 2018. [Online]. Available: https://github.com/ut-ims-robotics/jog_arm. [Accessed: 12-May-2018].
[57] M. Appo, “Keyboard publisher package,” 2018. [Online]. Available: https://github.com/ut-ims-robotics/keyboard_publisher. [Accessed: 12-May-2018].
[58] “Rqt,” 2016. [Online]. Available: http://wiki.ros.org/rqt/Plugins. [Accessed: 13-May-2018].
[59] “Manipulator force control bundle repository.” [Online]. Available: https://github.com/ut-ims-robotics/manipulator_force_control_pack. [Accessed: 13-May-2018].
[60] “UR5 specific custom configuration.” [Online]. Available: https://github.com/ut-ims-robotics/ur5_path_planning. [Accessed: 13-May-2018].
[61] “Franka Emika Panda specific configuration.” [Online]. Available: https://github.com/ut-ims-robotics/franka_panda_custom. [Accessed: 13-May-2018].
[62] “Universal Robot - our history,” 2018. [Online]. Available: https://www.universal-robots.com/about-universal-robots/our-history/. [Accessed: 13-May-2018].
[63] “Universal Robots - max. joint torques.” [Online]. Available: https://www.universal-robots.com/how-tos-and-faqs/faq/ur-faq/max-joint-torques-17260/. [Accessed: 13-

May-2018].

[64] "Franka Emika Panda brochure." Munich, Germany, 2017.

[65] Universal Robots, "The URScript Programming Language," p. 30, 2015.

[66] "Universal robot ROS package." [Online]. Available: https://github.com/ros-industrial/universal_robot. [Accessed: 13-May-2018].

[67] "Franka Control Interface Documentation." [Online]. Available: https://frankaemika.github.io/docs/. [Accessed: 13-May-2018].

[68] "Franka ROS repository." [Online]. Available: https://github.com/frankaemika/franka_ros. [Accessed: 13-May-2018].

[69] "Rviz." [Online]. Available: http://wiki.ros.org/rviz. [Accessed: 13-May-2018].