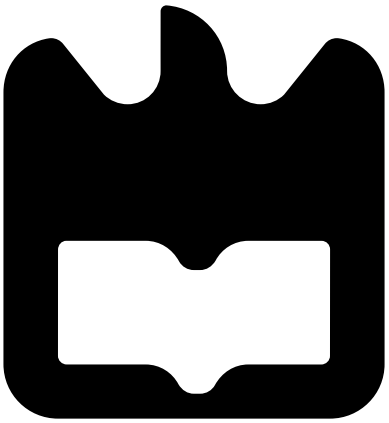
Universidade de Aveiro
2021

Fábio Miguel
Correia Alves

Braço Robótico e Humano Desempenham Tarefas de forma Colaborativa

Collaborative Tasks between Robotic Manipulator and Human



Universidade de Aveiro
2021

Fábio Miguel Correia Alves

Braço Robótico e Humano Desempenham Tarefas de forma Colaborativa

Collaborative Tasks between Robotic Manipulator and Human

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Doutor José Nuno Panelas Nunes Lau, Professor associado do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro, e do Doutor Manuel Bernardo Salvador Cunha, Professor auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

Este trabalho foi suportado pela ANI (POCI-FEDER) no contexto do Projeto AUGMANITY POCI-01-0247-FEDER-046103

This work was funded by ANI (POCI-FEDER) in the context of the Project AUGMANITY POCI-01-0247-FEDER-046103

o júri / the jury

presidente / president

Professor Doutor José Alberto Gouveia Fonseca

professor associado do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

vogais / examiners committee

Professor Doutor Pedro Mariano Simões Neto

professor auxiliar do Departamento de Engenharia Mecânica da Universidade de Coimbra

Professor Doutor José Nuno Panelas Nunes Lau

professor associado do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

agradecimentos / acknowledgements

Agradeço ao Professor Nuno Lau e ao Professor Bernardo Cunha toda a disponibilidade e esclarecimentos ao longo do desenvolvimento deste trabalho. Não só a eles mas também a todos os membros do IRISLab, pelas oportunidades de aprendizagem e crescimento que me proporcionaram ao longo do meu percurso académico.
Ao meu parceiro de investigação Marcelo Fraga, tanto pelas longas e arduas discussões como por toda a camaradagem que criamos. Foi quem mais me provou que 2 cabeças pensam melhor que 1, e que quando eu quero, consigo ser incompreensivelmente teimoso.
À Margarida, que primeiro por acaso, e depois com muito gosto, me acompanhou desde o início deste percurso, e cujo conhecimento que me transmitiu, não só académico, foi imprescindível para me tornar na pessoa que sou hoje.
À Fernanda Rocha, que sem querer, juntou o melhor grupo de idiotas que alguma vez pisou esta terra. Foram 5 anos de amizade e maluqueiras, com algum estudo intensivo pelo meio, que irei levar para a vida.
Finalmente, um obrigado muito especial aos meus pais, pelo apoio incansável ao longo destes anos. Irei ter um trabalho muito árduo para vos recompensar por tudo o que me proporcionaram até agora.

Palavras Chave

colaboração humano robô, robôs colaborativos, guiamento manual, transferência de ferramentas, prevensão de colisões, arquitetura de tarefas, controlo de movimento em tempo real

Resumo

A colaboração humano robô tem-se tornado um tema proeminente em ambientes industriais ao longo dos anos. O aparecimento de robôs colaborativos e as vantagens que eles trazem para as linhas de montagem causam a necessidade de explorar seu hardware e criar novos sistemas que permitam a execução segura de tarefas colaborativas entre humanos e robôs. Essas vantagens incluem as suas reduzidas dimensões e custos, os seus materiais de construção mais leves, a existência hardware de assistência de força/precisão e o cumprimento de requisitos de segurança, que os permitem ser considerados como colaborativos. Nesta Dissertação, propomos um conjunto de ferramentas e técnicas que possibilitam a execução de tarefas colaborativas entre um humano e um braço robótico. Essas técnicas incluem compensação de peso acoplado à garra do robô, identificação de obstáculos dinâmicos no ambiente e a criação de novas tarefas de forma simplificada e abstrata. Essas técnicas podem então ser unidas para criar várias tarefas colaborativas. Exemplos, implementados e testados, são a transferência de objetos entre humano e robô, manipulação da posição do robô através de interação direta com a sua garra, manipulação de objetos acoplados dinamicamente e a execução de uma tarefa industrial com prevenção de colisões. Todas essas tarefas foram agrupadas numa máquina de estados que permite ao utilizador interagir dinamicamente com o robô, mudar a tarefa em execução, enquanto recebe feedback constante. Estas tarefas foram implementadas e testadas num ambiente real com um Universal Robots UR10e. O cenário consiste num espaço de trabalho partilhado onde o humano e o robô podem interagir fisicamente e realizar as tarefas propostas.

Keywords

human robot collaboration, collaborative robots, hand guiding, tool transfer, collision avoidance, task level architecture, real time motion control

Abstract

Human Robot Collaboration has become a prominent subject in industry settings over the years. The insurgence of collaborative robots and the advantages they bring to assembly lines have caused the need to exploit their hardware and create new systems that can enable the safe execution of collaborative tasks between humans and robots. Such advantages include their reduced size and cost, their lightweight construction, their force/precision assistance hardware and their safety requirements, that must be met in order to be considered collaborative. In this Dissertation, we propose a set of tools and techniques that enable the execution of collaborative tasks between a human and a robotic manipulator. Such techniques include dynamic payload compensation at the end effector of the robot, identification of moving obstacles in the environment, and the creation of high level task plans. This techniques can then be joined together to create various collaborative tasks. Examples of such, both implemented and tested, are the transfer of objects between human and robot, precise hand guiding of the robot at the end effector, hand guided manipulation of dynamically coupled objects, and the execution of an industrial task with collision avoidance. All of this tasks were grouped in a global state machine that allows the user to seamlessly interact with the robot, change the task in execution while receiving constant feedback.

These tasks were implemented and tested in a real setting with an Universal Robots UR10e. The setup consisted on a shared workspace where the human and the robot could physically interact and perform the proposed tasks.

Contents

Contents i

List of Figures v

List of Tables vii

Acronyms ix

1 Introduction 1

1.1 Motivation . . . 1

1.2 Objectives . . . 2

1.3 Outline . . . 2

2 Collaborative Robotics 5

2.1 History . . . 5

2.2 Human Robot Collaboration . . . 6

2.2.1 Hardware and Design . . . 7

2.2.2 Safety . . . 7

2.2.3 Programming . . . 8

2.3 Technological Background . . . 9

2.3.1 Robotics Middleware . . . 9

2.3.2 Universal Robots UR10e . . . 11

2.3.3 Motion Planning . . . 13

2.3.4 Perception . . . 14

2.4 Related Research . . . 14

2.4.1 Proposals on HRC and Collaborative Tasks . . . 15

2.4.2 Solutions to Specific HRI Problems . . . 16

3 Force Torque Sensor Compensation 19

3.1 Force Torque (FT) Sensor Correction . . . 19

3.1.1 Noise Filtering . . . 19

3.1.2 Observed Behavior . . . 20
3.1.3 Proposed Solution . . . 22
3.1.4 Results . . . 24
3.2 End Effector (EEF) Weight Compensation . . . 25
3.2.1 UR FT Sensor Controller Internal Compensation . . . 25
3.2.2 Force Theoretical Model . . . 26
3.2.3 Results . . . 27
3.2.4 Adapting the FT Theoretical Model . . . 29
3.3 Real Time Correction and Compensation of FT . . . 30

4 Dynamic Obstacle Avoidance **33**
4.1 Obstacle Detection . . . 33
4.1.1 Hand Eye Calibration . . . 33
4.1.2 Robot Segmentation . . . 34
4.1.3 Obstacle Segmentation . . . 35
4.2 Artificial Potential Fields . . . 36
4.2.1 Attraction . . . 36
4.2.2 Repulsion . . . 37
4.2.3 Controller . . . 37
4.3 Real Time Obstacle Avoidance . . . 38

5 Collaborative Tasks **41**
5.1 Hand Guiding . . . 41
5.1.1 HGFT to EEF Velocity . . . 41
5.1.2 EEF Velocity to Joint Speed . . . 42
5.1.3 HG Architecture . . . 43
5.2 Object Transfer . . . 43
5.3 Object Manipulation . . . 44
5.4 Collision Free Execution of an Industrial Task . . . 45
5.5 Collaborative State Machine . . . 45
5.5.1 State Transitions . . . 47
5.5.2 Visual Feedback . . . 47
5.6 Software Tools for Human Robot Collaboration (HRC) . . . 48
5.6.1 rqt_ur10e . . . 48
5.6.2 rqt_sami . . . 49

6 Experiments and Results **51**
6.1 Collaborative Setup . . . 51
6.2 Collaborative Tasks . . . 52

6.2.1 Interaction Test 52
6.2.2 Hand Guiding 52
6.2.3 Object Manipulation 53
6.2.4 Collision Free Execution of an Industrial Task 57
6.3 System Stability 62

7 Conclusion **63**
7.1 Overview 63
7.2 Future Work 64

References **67**

List of Figures

2.1 Joint configuration of the UR10e 11

3.1 Comparison of raw and filtered FT measurements 20
3.2 Real time test of the variation on FT relative to the position of the Wrist3 joint 21
3.3 Variation of FT relative to time 21
3.4 Variation in FT relative to interaction with the EEF 22
3.5 5 EEF testing poses 23
3.6 Average FT values in the 5 testing poses 23
3.7 Result of the correction node applied in a real time test 24
3.8 Distribution of the correction function error on FT measurements 25
3.9 Behavior of the FT sensor with parametrization of payload 26
3.10 Visualization of the generation of theoretical force values 26
3.11 Visualization of the generation of theoretical torque values 27
3.12 Comparison of FT measurements from real sensor and analytical model 28
3.13 Distribution of analytical model error on FT measurements 28
3.14 Visualization of the 57 poses in which the tests were performed 29
3.15 Comparison of FT measurements from real sensor and adjusted analytical model 30
3.16 Distribution of the adjusted analytical model error on FT measurements 30
3.17 ROS architecture of the compensation of FT in real time 31
3.18 Result of the compensation architecture applied in a real time test 31

4.1 Visual representation of the conversion from URDF links to robot skeleton 34
4.2 Raw point cloud and categorized point cloud with ROI 35
4.3 Final result of the obstacle segmentation algorithm 36
4.4 Trajectory execution with cartesian attraction vector 37
4.5 ROS architecture of the obstacle avoidance in real time 38
4.6 Collision avoidance setup in the Gazebo simulator 38
4.7 Collision avoidance offline and final trajectory 39

5.1 ROS architecture of the HG task 43

5.2 State architecture of the object transfer task 44
5.3 State architecture of the object manipulation task 44
5.4 State architecture of the industrial task 45
5.5 Collaborative state machine 46
5.6 Complete ROS architecture of the system 46
5.7 Result of the compensation architecture applied in a real time test 47
5.8 LED status feedback on the gripper tool 48
5.9 ROS rqt_ur10e interface for easy robot control and monitoring 48
5.10 ROS rqt_sami interface for easy robot control and monitoring 50

6.1 Views of the experimental shared workspace 51
6.2 Succession of steps of the HG test 52
6.3 6 different EEF weight measurement poses 54
6.4 Results of the weight measurement test in each position 54
6.5 Result of the compensation architecture applied in a real time test 55
6.6 Succession of steps of the object manipulation test 55
6.7 Simulation scenario with obstacle and offline generated trajectory 57
6.8 Execution of the trajectory with and without obstacle 58
6.9 Simulation scenarios with multiple obstacles 58
6.10 Succession of steps of the collision avoidance test 60
6.11 Collision avoidance in real scenario with multiple speed settings 61

List of Tables

2.1 Universal Robots UR10e technical specifications 11

4.1 Internal parameters of the CEC algorithm 35

6.1 Results of the interaction test 52
6.2 Results of the HG performance test 53
6.3 Detailed results and statistics from the weight measurement tests 54
6.4 Results of the object manipulation performance test 56
6.5 Parameters for the base obstacle avoidance test 57
6.6 Parameters for the random placement of obstacles 58
6.7 Results of the obstacle avoidance test 59
6.8 Minimum EEF distance to obstacle relative to its speed 61

Acronyms

Cobot	Collaborative Robot
EEF	End Effector
TCP	Tool Center Point
CoG	Center of Gravity
HRC	Human Robot Collaboration
HRI	Human Robot Interaction
DoF	Degrees of Freedom
2D	Two Dimmensinoal
3D	Three Dimmensional
GUI	Graphical User Interface
LED	Light Emiting Diode
RGB	Red Green Blue
RGBD	RGB Depth
LED	Light Emitting Diode
RAM	Random Access Memory
FOV	Field of View
ISO	International Standards Organization
ISO/TS	ISO Technical Specifications
TCP	Transmission Control Protocol
IP	Internet Protocol
XML	Extensible Markup Language
OEM	Original Equipment Manufacturer
IO	Input Output
ROS	Robot Operating System
URDF	Unified Robot Description Format
UR	Universal Robots
UR10e	Universal Robots UR10e
ROI	Region of Interest
CEC	Conditional Euclidean Clustering
ECE	Euclidean Clustering Extraction
RRT	Rapidly-exploring Random Trees
OMPL	Open Motion Planning Library
APF	Artificial Potential Field
RTDE	Real Time Data Exchange
FT	Force Torque
LPF	Low Pass Filter
FIR	Finite Impulse Response
MAF	Moving Average Filter
HG	Hand Guiding
HGFT	Hand Guiding Force Torque
SM	State Machine
JS	Joint Speed
DLS	Damped Least Squares
PCL	Point Cloud Library
XYZ	Cartesian Coordinates
RPY	Roll Pitch Yaw

CHAPTER 1

Introduction

In this chapter, we contextualize this Dissertation and present its topics. We start with a concise motivation that enlightens the broad necessity of Human Robot Collaboration (HRC). Then, we specify the goals we want to achieve, and what exactly has been created in this work. Finally, the thesis outline is given, with a short summary of each chapter contents.

1.1 Motivation

An industrial robot is a machine that allows efficiency, strength and automation on the production line. It can be seen as a substitute for the human worker for their ability to perform repetitive and tedious manufacturing tasks autonomously with high accuracy and precision. By default, and for safety reasons they are intended to work separately from humans, in their own environment and are usually programmed to stop if a human worker enters its space.

Despite their great abilities, there are some tasks that are either too complex, or too dynamic to fully automate and require the continuous presence of a human, to either supervise or assist the robot. This need of shared execution of tasks with robots, resulted in the emergence of Collaborative Robots, also known as Cobots. These are industrial robots generally built with lightweight materials, equipped with Force Torque (FT) assistance and speed limitation hardware, and overall, are easy to set up and program. Altogether these features promote safe and efficient interactions with humans.

Starting with a cobot, to reach the collaborative execution of a task, there is a gap which this Dissertation aims to fill, by proposing and developing a set of tools, techniques and workflows that not only leverage the hardware of cobots, but also use external sensors. Such techniques are then used to create and deploy multiple tasks which, when grouped and integrated in an interaction based state machine, result in the promotion of a shared collaborative environment where humans and cobots can safely work together.

1.2 Objectives

This Dissertation has as its main objective the promotion of shared execution of tasks, between humans and industrial robots, specifically targeting collaborative robotic manipulators. To do so, it aims to develop the following set of techniques:

- Interaction and communication with the cobot through touch;
- Manipulation of the cobot by physically Hand Guiding (HG) it;
- Compensation of payload coupled to the End Effector (EEF);
- Detection of dynamic obstacles in the environment;
- Motion planning aware of dynamic obstacles;
- Creation of high-level task plans;

Combining theses techniques, this Dissertation also aims to achieve the following collaborative tasks:

- Precise and responsive HG at the EEF level;
- Transfer of objects between human and cobot, in both directions;
- Lift assistance and precise manipulation of heavy objects;
- Collision free execution of an industrial task;

Finally, all techniques were arranged in a real time concurrent architecture, and all tasks grouped in a state machine where the user interacts or changes the current task by physically interacting with the cobot, while receiving visual feedback through its gripper attachment.

Besides the main objectives, efforts were made to increase the ease of use of a robotic manipulator, with the development of 2 Graphical User Interfaces (GUIs), which help control and monitor the status of the cobot and the task being executed.

1.3 Outline

This Dissertation is divided into 7 chapters.

Chapter 2 (Collaborative Robotics) enlightens the theoretical background that supports this work. It starts with the historical context and various definitions on the HRC field. Some technologies used for the development of cobotic applications are outlined and specific details are given on the hardware used in this work. Then, it gives an analysis on existing research proposals performed in this field.

Chapter 3 (Force Torque Sensor Compensation) explains how a 6-Axis FT sensor located at the cobot EEF can be leveraged for precise HG applications. It also presents a payload compensation system based on a theoretical FT model, that is used to separate forces caused by the human from forces caused by attached payload. Finally, it describes a real time architecture for correction and compensation of FT.

Chapter 4 (Dynamic Obstacle Avoidance) demonstrates how a depth camera can be integrated with a cobot and make its motion planning collision aware. It outlines techniques for point cloud segmentation that identify moving obstacles in the environment. Then, it presents a collision avoidance system that takes into account the identified obstacles when

executing predefined trajectories. Just like the previous chapter, it also describes a real time architecture that implements these models.

Chapter 5 (Collaborative Tasks) uses the previous architectures to build the proposed collaborative tasks. Then, it aggregates all tasks in a collaborative state machine where a human can seamlessly execute and switch between tasks through touch interaction. It also showcases 2 GUIs designed for the development of robotic applications.

Chapter 6 (Experiments and Results) demonstrates the performance of the tasks created though various metrics obtained from experiments made, both in simulated and real environments.

Chapter 7 (Conclusion) gives the final regards about the work developed and elaborates on future work.

CHAPTER 2

Collaborative Robotics

In this chapter, we explore the theoretical background of this work, as well as related research performed in this field. We start by giving some historical context to see when and why collaborative robots started to emerge. Then, we define the field of HRC detailing all of its subfields and characteristics. Afterwards, we outline the current technological trends used to develop such systems and give a thorough description of the Universal Robots UR10e (UR10e) system. Finally, we give light to several research efforts performed not only in the general field of HRC, but also in specific problems directly correlated to this work.

2.1 History

Since the 1970s, industrial robots and manipulators have been part of production lines in many sectors of the industry. The main reason is that they can complete certain tasks faster and more efficiently than humans, since they are built with performance and precision in mind. Furthermore, a lot of manufacturing processes rely on repetitive and laborious tasks that for a human worker can become tedious and exhausting. According to the International Federation of Robotics[1], in the year 2020 there were an estimated 2.7 million industrial robots in operation worldwide [1].

This level of automation guarantees uninterrupted and efficient productivity in assembly lines, since robots do not take breaks or get tired, but at the same time they limit its flexibility and adaptability. When a manufacturing process, typically of customized products with smaller lot sizes, prioritizes flexibility and changeability over automation, these machines are useless and there is a need to involve a human worker in the process. An industrial robot is usually hard coded to do one task at a time, has limited abilities for handling complex or limp objects and cannot make decisions, therefore the close linkage of human and robot in these scenarios should result in the best of both sides. This collaborative approach can have several advantages compared to full automation, particularly when a robot can be guided by a human and simultaneously provide power assistance to him [2].

[1] https://ifr.org/

Traditional industrial robots have heavy structures with fixed installations, only interact with humans during programming and are otherwise separated from them through perimeter safeguarding, stopping their motion if any obstacle breaches it. These characteristics prevent them to be used in collaboration or even coexist with human workers. Given these restrictions and needs from the industry, came the concept of Collaborative Robot (Cobot), firstly coined in 1996 by J. Edward Colgate and Michael Peshkin [3] as *'a robotic device which manipulates objects in collaboration with a human operator'*. In their work, it was a simple device with a single joint that assisted the human operator by setting up virtual surfaces which could be used to constrain and guide motion.

Years later, companies like KUKA and Universal Robots (UR) made commercially available the first industry ready cobots, denoted as industrial collaborative robots. They were light-weight, flexibly relocated and easy to teach and program, even by non-experts. Most importantly, they were equipped with power, force and speed sensors and limiters, allowing safe execution of tasks near humans, therefore allowing a shared collaborative environment. Nowadays, cobots have evolved in such a way that they are replacing industrial robots in assembly lines, or from another perspective, industrial robots are being designed with collaboration in mind.

2.2 Human Robot Collaboration

HRC is a subsection of the general field of study called Human Robot Interaction (HRI) which according to research [4, 5] is defined as *'a general term for all form of interaction between humans and robots*' or *'the process of conveying human intentions and interpreting task descriptions into a sequence of robot motions complying with robot capabilities and working requirements*'. It is an umbrella term used to describe a multidisciplinary field that includes knowledge and understanding from human-computer interaction, robotics, artificial intelligence, design and psychology.
HRI can be divided in several sub-categories based on the following four criteria [6, 7]:

- **Workspace:** It is the overlapping space in the working range of human and robot;
- **Working Time:** The time the participants are working inside the shared workspace;
- **Aim:** The objective, focus and goal of each participant regarding the task at hand;
- **Contact:** Meaning intentional physical contact between the participants;

Using these four criteria, HRI can be divided in:

- **Human-Robot Coexistence** (workspace and working time): Defined by the capability of simultaneously sharing the workspace between humans and robots, but operating in dissimilar tasks and not interacting with each other. They do not have a common goal and do not share contact therefore do not need to be synchronized. Robot abilities often rely only on collision avoidance.
- **Human-Robot Cooperation** (workspace, working time and aim): It is an upgrade over the previous category. Now humans and robots also share the same purpose in the

given task. Cooperation also requires synchronization, which means that either exists a common language of communication, through instructions, gestures or voice, or machine vision is used for the robot to know when it is its time to act.

- **Human-Robot Collaboration** (all four criteria): The final stage of HRI, where humans and robots, who simultaneously share the same workspace, work together to perform a complex task interacting physically with one another. With FT sensing hardware a robot can interpret human motion and intention, and react accordingly.

With HRC defined and identified inside the broader field of HRI, some of its requirements and characteristics are going to be drawn in order to proceed with a full understanding of its context.

2.2.1 Hardware and Design

The design and composition of a cobotic system can be one of the most challenging problems in this field [8, 9]. Industrial cobots generally operate in complex working conditions and must be able to carry motion effectively, sometimes in crowded environments, while facing unexpected events such as the arrival of a human operator. This is one of the reasons that cobots are usually designed with 6 to 7 Degrees of Freedom (DoF). Another is that an higher number of DoF provides the cobot with increased flexibility and dexterity in complex manipulation tasks.

Because cobots are meant to work alongside humans, reduced weight of the moving parts is one of the main factors in cobot design [10]. Even so, in environments where collisions are inevitable, the risk of interaction with cobots can be reduced due to their increased sensorial apparatus [11], such as the use of proximity-sensitive skins or FT sensors to detect collisions, the increased energy absorbing properties of protective layers, the limits on robot velocity and maximum strength and force, and in some scenarios the placement of airbags around the robot [12].

According to recent reviews [6, 7], research is still being develop on numerous different ways to improve the design of cobots in order to make them reliable, dependable and most importantly safer.

2.2.2 Safety

The general design and composition of a cobot has been shown, but is not enough to guarantee safe HRC. According to ISO 10218:2011 [13, 14], and later more widely explained in ISO/TS 15066:2016 [15], specific requirements for cobotic systems need to be met for them to be considered safe. These standards define four classes of safety requirements for industrial robots in collaborative environments:

- **Safety-rated monitored stop:** The robot is stopped upon access of the human to the collaborative workspace. Most robot manufacturers offer a safety controller that assures the standstill of the robot. The robot can then resume the task once the human has left the collaborative workspace. This mode is mostly used when the cobot works

alone, but occasionally a human operator can enter its workspace. In a broader view, the robot does not move while the human is present.

- **Hand-guiding:** The human uses a hand-operated device, usually located at the EEF of the robot, to transmit motion commands to the robot system. This type of operation implies a direct physical interaction with the robot, where the human must have full control over its movement. The position of the human within the collaborative workspace must be defined and a safety controller for delimiting the robot speed is required. Graphic support through icons or 3D simulation is helpful for intuitive programming of the robot.
- **Speed and separation monitoring:** There is constant monitoring of the relative speed and distance between robot and human. The robot must maintain a minimum safe distance and speed to the human in order to be able to stop any dangerous motion if contact with the human is imminent. When the separation distance decreases to a value below the minimum, the robot stops. When the human moves away from the robot, the robot can resume motion automatically. External vision sensor data might be needed to achieve this level of monitoring since few manufacturers equip cobots with such capabilities.
- **Power and force limiting:** The robot system should be designed to sufficiently reduce risk to a human by allowing direct, physical interaction without an additional safety controller. This is done through the design of the robot system by limiting collision forces so that in the event of a contact between the humans and the robot, biomechanical tolerance limits are not exceeded.

A traditional industrial robot can be adapted to meet this collaborative modes and requirements, however it would need additional safety devices such as laser sensors, vision systems, or controller modifications. For this reason, a commercial cobot that has this features built-in requires no further hardware costs and can be a more attractive solution.

2.2.3 Programming

Industrial robots have the job of carrying out pre-programmed, repetitious tasks in order to promote productivity and efficiency. Their software platform is responsible for enabling HRI and conveying human intention on how certain tasks should be executed. Intuitive robot programming is an important issue that HRI deals with, since traditional approaches are either unintuitive or time-consuming. In the early days of industrial automation, robots could only be programmed by experts in actuators, controllers and hardware programming. Nowadays, most industrial robots are shipped with teach pendants. These are handheld devices containing buttons, switches and, in some cases, a touchscreen. They are currently the most common programming method as they require little to no training in robotics or programming in general, they display the robots commands in a nontechnical fashion and allow for online editing of such commands. They also allow for walk-through programming where an operator physically moves the robot through a desired task. Although these methods seem intuitive and easy to learn, they are only applicable for certain groups of tasks since they have low accuracy requirements, and for complex tasks can become cumbersome to use.

Finding accurate, intuitive and efficient ways for robot programming is also a problem for HRI. One possible and currently established technique is the generation of robotic skills, which are pre-programmed software packages that only need to be parametrized by the user. A recent study [16] has introduced a software architecture combining generation of robotic skills, named action blocks, that also allowed for process control, and a set of strategies and approaches for a fast and intuitive parametrization process.

Further research in this area is constantly looking for better user interfaces for robot programming [6, 17]. A few examples are:

- **Virtual Reality:** Where tasks are completed intuitively as if the human is present at the remote working environment. It guarantees safe programming and is flexible on the level of workspace constraints or task complexity that may exist. The main problem is that it requires previous knowledge of the working areas to construct the virtual environment, and is not suitable for loosely structured working conditions where there may be constant changes to the environment.
- **Augmented Reality:** Where virtual elements, mainly computer-generated graphics, are projected into the real world so that the user can perceive certain elements of robot programming in real-time. Such elements might include objects, robot motions and trajectories, and simulated collisions with the real environment. This technique can also help operators determine the best location for the robot before final installation.
- **Program by Demonstration:** Where a human performs a task manually and in parallel, the robot is observing, following and learning the task in real-time. It allows any user to program a robot by just giving a demonstration of the sequence of operations to be carried out and shifts the burden of robot programming from robot experts to task experts. It also makes possible to program more than one robot simultaneously. A significant problem with this approach is that jerks and inaccuracies in human demonstrations can lead to unsatisfactory execution of tasks by the robot that is learning them.

Other approaches on robot programming are also moving forward towards multimodal interfaces with the inclusion of gestures, voice, eye gaze and facial expressions serving as high-level inputs to control and program the robotic system.

2.3 Technological Background

In a cobotic system, the interaction between the robot, the user, the external sensors and software modules usually rely on several different technologies, all working together to provide a seamless and efficient experience. Below, divided in various categories is explained which technologies are going to be used in this work and why.

2.3.1 Robotics Middleware

A middleware is a piece of software that enables cohesive, structured communication between different software modules. It is informally described as "software glue". It has been

shown that a cobotic system is comprised of different components, that usually are handled by different software modules, known as their drivers. These modules may be written in different programming languages and implement different communication protocols. As such, the job of a middleware is to hide the obvious heterogeneity resultant of a system with these characteristics, and provide functions and services that not only enhance the communication between the entities, but also serve as a flexible, interoperable and central repository of information shared by them.

The Robot Operating System (ROS) [2] [18] is an open-source robotics middleware. With over 3.000 packages in its ecosystem, it is also described as a flexible framework for writing robot software, that comprises a plethora of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms. A key component of a cobotic system is its communication infrastructure and ROS, at its lowest level, offers a message passing interface that provides inter-process communication and implements a publisher/subscriber communication model. Each component that performs a certain task is called a ROS node, and multiple nodes communicate between them by exchanging ROS messages through ROS topics. These provide an asynchronous means of communication, since any node can publish or subscribe to any topic at any time, provided it respects its message type. For synchronous communication, ROS offers services which can be seen as remote procedure calls that provide request/response interactions between nodes. The middleware part of ROS also provides a global key-value server where nodes can set and get configuration parameters.

When it comes to robot specific features, ROS provides libraries that help their integration in its ecosystem through collections of software drivers, that abstracts both the low-level control of hardware components, and the treatment of information generated by sensors and other peripherals. A crucial example of such functionality is the Robot Geometry Library[3] which helps keeping track of where different parts of the robot are with respect to each other and the world. Therefore, it allows the user to define both static transforms, such as a camera that is fixed somewhere in the world, and dynamic transforms, such as the pose of the EEF in a manipulator. This way, any positional data can be easily transformed between any pair of coordinate frames in the world.

Other features that enhance the development of such systems include RViz[4] which provides 3D visualization of robot description models, transforms and many sensor data such as point clouds; an extensive library of built-in plugins based on rqt[5], which is a Qt-based framework for developing graphical interfaces that easily integrates with the ROS ecosystem; a set of command-line tools that allow the user to fully control all ROS core functionality without a GUI, useful for controlling the cobotic system remotely.

[2] https://www.ros.org/
[3] http://wiki.ros.org/tf
[4] http://wiki.ros.org/rviz
[5] http://wiki.ros.org/rqt

2.3.2 Universal Robots UR10e

Although the techniques developed in this dissertation are generic, the cobot with which this work will be implemented is an UR10e. It is a 6 DoF collaborative industrial robot equipped with a FT sensor in its EEF. A visual description of its configuration is shown in Figure 2.1, detailed technical specifications are shown in Table 2.1, and the various ways that it can be controlled will be explained below, as well as the chosen method and why.

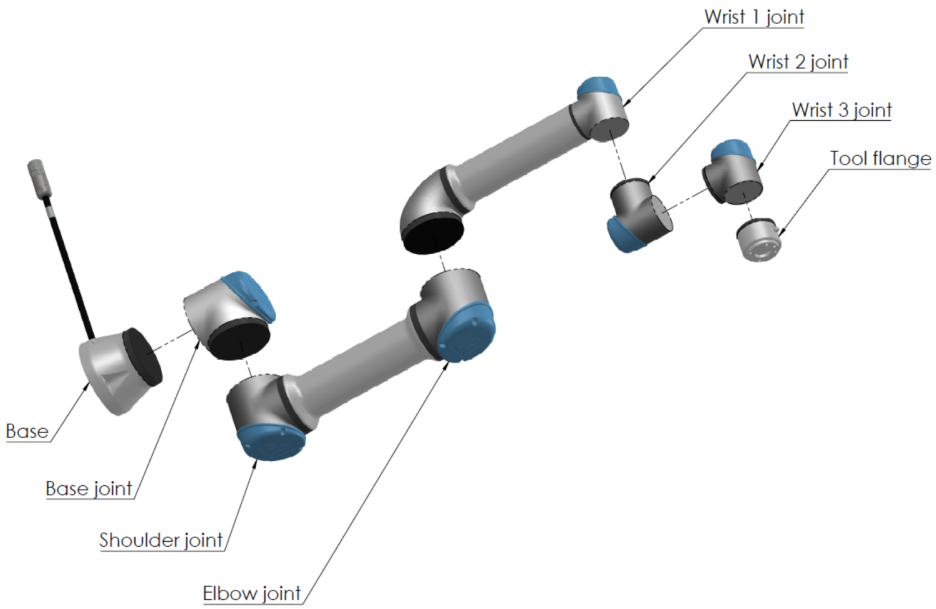


Figure 2.1: Joint configuration of the UR10e

UR10e	**Value**
Reach	1300 mm
Payload	10 Kg
DoF	6 rotating joints
Pose Repeatability	+/- 0.05 mm
Joint Working Range	± 360 °
Joint Maximum Speed	± 120 °/Sec.
Typical EEF Speed	1 m/Sec.
Footprint	190 mm
Weight	33.5 Kg

FT Sensor	**Force**	**Torque**
Range	100 N	10 Nm
Resolution	2.0 N	0.02 Nm
Accuracy	5.5 N	0.60 Nm

Table 2.1: Universal Robots UR10e technical specifications

Teach Pendant

The UR teach pendant is a handheld wired device with a 12 inch touchscreen display that allows the user to fully configure and control the UR10e. It does so using the Polyscope [19], a graphical user interface that operates the robot arm and control box, creates and executes programs. It terms of operational modes it allows compliant motion with the Freedrive mode, manual motion with arrow buttons and sliders, and automatic motion with the creation of programs using its built-in programming environment, which is seen as a programming tree

where the user adds programming nodes. These nodes can be commands telling the robot what to do or generic programming statements (if, loop, event, etc.). Polyscope allows people with little programming experience to program the robot and for most tasks it is done entirely using the touch panel without typing any cryptic commands. Given its simplicity, it is not ideal for complex tasks, or if the user needs low-level control or sensor information access. Besides the programming features, the teach pendant is also equipped with a button in the back that enables the Freedrive mode and an emergency stop red button in the front.

URScript

Parallel to the Polyscope interface, UR also provides a script level way of controlling its robots through their own programming language, called URScript [20]. This language includes variables, types, flow control statements and functions that monitor and control both IO and robot movements. Inside the Control Box of the robot there is a low-level controller called URControl. Programming a robot at the script level is done by writing a client application and connecting to URControl using a TCP/IP socket. When a connection has been established URScript programs are sent on the socket. Compared to the Polyscope interface, URScript gives the user more control over the structure of the programs and makes it easier for an experienced user to take full advantage of the robot. One downside to this approach is the user that is sending commands externally has no access to the current state of the program, since once the program is sent, it is executed immediately without feedback. The only way the user can access the state of the robot is by connecting to another one of the available client interfaces[6] that publish robot information at a fixed rate interval.

Universal Robots RTDE Interface

The Real Time Data Exchange (RTDE)[7] interface provides a way to synchronize an external application with the UR controller over a standard TCP/IP connection. This functionality is split in two stages, a setup procedure and a synchronization loop. In the setup procedure, the external application will become a client of the UR controller, over the RTDE interface by sending a recipe. This recipe should contain a setup list of named input and output fields that should be exchanged in the synchronization loop. When this loop is started, the RTDE interface sends data to the client in the same order that the client requested. On an e-Series UR cobot, such as the UR10e the RTDE interface generates output messages at 500Hz.

Researchers at University of Southern Denmark have developed a C++/Python library for controlling and receiving data from a UR robot using the RTDE interface, called ur_rtde[8]. It makes available three distinct interfaces:

- **RTDE Control Interface:** Primarily used for moving the robot and utility functions. It requires a control script to be running on the robot, which is uploaded automatically.

[6] https://www.universal-robots.com/articles/ur/interface-communication/overview-of-client-interfaces/

[7] https://www.universal-robots.com/articles/ur/interface-communication/real-time-data-exchange-rtde-guide/

[8] https://gitlab.com/sdurobotics/ur_rtde

- **RTDE Receive Interface:** Used for receiving data from the robot.
- **RTDE IO Interface:** Used for setting digital/analog IO and adjusting the speed slider of the robot.

The Control Interface allows for non-blocking commands making the flow of the external program able to continue while the robot is moving. The separation of Control and IO in different interfaces also allows to change the speed of the robot while it is moving.

Although it takes programming knowledge and skills, from all the available ways to control a UR cobot, the use of the RTDE interface has proved to be the most advantageous one.

Universal Robots ROS Driver

The goal of this driver is to provide a stable and sustainable interface between UR robots and ROS, that both enhances the control of the robots using ROS paradigms such as its controller interface, and makes available to the ROS environment all data regarding the robot. By using ROS compatible manipulators, perception sensors, peripherals and motion planners the user makes sure all components speak the same language and interoperate regardless of OEM brands or communication protocols. Further implementation details can be found in [21].

In terms of features, this driver uses the RTDE interface for communication; uses the speed-scaling of the robot for slowing down trajectory execution accordingly; serves as a replacement for the teach pendant since it offers ROS services for most of its interactions, such as start, stop and even recover the robot from safety events; uses on-the-robot interpolation for joint-based trajectories, which helps if the application can not meet the real-time requirements of the RTDE interface; and many more, extensively documented in the UR github repository[9].

This driver is currently being updated to both include with new functionality and support new UR software updates. Recent important features include joint velocity-based control that lets the user directly control the speed of each individual joint, which is very helpful for visual servoing, real-time motion planning or other kinds of control that require speed control rather than position control, and cartesian position-based and twist-based control that lets the user execute trajectories along cartesian paths.

The UR ROS Driver will be the chosen method of interfacing with the UR10e for the amount of extra functionality it provides and the overall advantages of developing software in the ROS ecosystem.

2.3.3 Motion Planning

Motion planning is a computational problem with the objective of planning motions for complex bodies from a start to a goal position. It breaks down a desired movement task into discrete motions that satisfy movement constraints while avoiding collision with known obstacles.

The MoveIt Motion Planning Framework [22, 23] is an easy-to-use open source robotics manipulation platform for developing commercial applications, prototyping designs, and

[9] https://github.com/UniversalRobots/Universal_Robots_ROS_Driver

benchmarking algorithms. Its main strength is the ability to generate high DoF trajectories trough cluttered environments and avoid local minimums. It also provides robot description and kinematics implementations for multiple robot manipulators. One of its features that is relevant to this Dissertation is the implementation of the Open Motion Planning Library (OMPL), which aggregates many state of the art motion planning algorithms. One of which is the Rapidly-exploring Random Trees (RRT) Connect planner which will be the default motion planner used in this work.

In this work, MoveIt will be used mostly for offline position-based trajectory planning. One of the objectives is to control the robot in real-time, with velocity-based commands sent at high frequency. As of writing, current MoveIt releases do not support such features, having only plans to implement them in future versions. Further details on the solution of this problem will be presented in Section 4.2.

2.3.4 Perception

To fully achieve collaboration in a shared environment, the cobot must be able to perceive its surroundings. Between the various ways to give the cobot this ability, the majority of them rely on external sensors, such as stereo or depth cameras, and lidar sensors. All of them give the system a 3D representation of their field of view through the generation of point clouds.

The Point Cloud Library (PCL) [24] is a standalone, large scale, open project for 2D/3D image and point cloud processing. It is split into a series of modular libraries ranging from filters, recognition, segmentation and visualization. It will be extensively used in this work to achieve the obstacle avoidance task which will be further explained in Section 4.1.

To be able to identify obstacles in relation to the cobot, a process known as camera extrinsic calibration needs to be performed. It consists on identifying in the cartesian space where the perception sensor is, in relation to the robot. Only this way, can the obstacles identified by the sensor be spatially placed in the shared environment. The *easy_handeye* ROS package[10] is an implementation of the highly cited work of R. Y. Tsai and R. K. Lenz [25] where a fully autonomous and efficient technique for 3D robotics hand eye calibration is developed. In the scenario that will be implemented in this Dissertation, where the sensor is fixed in the world, this package it able to obtain a static transform from the world frame to the sensor, by capturing images of a fiducial marker attached to the EEF of the robot. Then using the Tsai-Lenz algorithm implemented in the OpenCV library computes the result and stores it in the ROS environment.

2.4 Related Research

In the topic of collaborative tasks between a human and a cobot, there is no record of commercially available, full-featured frameworks or systems, that could allow the accomplishment of the objectives proposed in this dissertation. Even though manufacturers ship their cobots and accessories with extensible and flexible software, that allow the creation of such

[10] https://github.com/IFL-CAMP/easy_handeye

tasks, there is much work left for the user to design, program and implement the system architectures that aggregate the collaborative tasks and deliver a cohesive, seamless experience. Furthermore, there is no limit on the amount of collaboration that can exist, or on how the available tools can be used together in their final working environments. Despite these facts, extensive research efforts have, and are currently being made [6, 7] to enhance this field with structured architectures and frameworks that seamlessly allow multifaceted collaboration between human and cobot. This section serves as a survey on such proposals, and will firstly enumerate and detail general approaches to HRC and collaborative tasks. Then, it will shift to other solutions that directly correlate to specific problems faced in this Dissertation, and that also served as inspiration to the solutions implemented.

2.4.1 Proposals on HRC and Collaborative Tasks

The work in [26] proposes a collaborative framework for robotic task specification, where the authors start by stating that the lack of effective sensing and task variability creates too much uncertainty to reliably hard-code a robotic cell. The developed framework blends automated task specification with the experience and cognition of a human operator to provide a more accurate task specification. This work was implemented with ROS and was mainly focused on surface finishing tasks.

The authors in [27] deal with the problem of developing a collaborative coating cell. They start by teaching the robot its trajectory with a programming by demonstration technique, using a multicolored LED marker attached to the coating tool. Then, a 3D perception system was developed to perform initial point cloud alignment and 6 DoF pose estimation. Finally, to promote safe HRC, a zone monitoring system was employed to track the position of the operator inside the cell.

A coordination system for assembly tasks where HRC is required was presented in [28], where the authors developed a ROS based framework that intertwined a sequence of tasks modeled in a neutral XML format language, with the ability for human and a robot to coexist in a fenceless cell, where safety was guaranteed by a 3D industrial safety camera.

Regarding a polishing task, authors in [29] proposed and demonstrated the use of a robotic manipulator with an FT sensor in its EEF, whose task was to keep a workpiece in a prescribed sequence of poses while a human operator, equipped with an abrasive tool, proceeded to polish it. They also allowed the operator to change the orientation of the workpiece mounted on the robot, by physically pushing or pulling the robot body. They achieved this behavior with a control algorithm that was able to distinguish polishing forces applied at the EEF level, from the external torques acting on the robot joints due to the intentional physical interaction engaged by the human.

Research in [30] resulted in the development of a cobotic framework applied to solve a screwing task in collaboration with a human operator, using a skill-based approach. The ROS-based system used a software called Skill Based System, which treats tasks as a set of skills, which then are composed of sequences of motion primitives. The human operator was then able to execute programmed tasks or create new ones using the available skills.

A framework for the execution of collaborative tasks in hybrid assembly cells was developed in [31], where the focus was given to the human-robot coexistence for the execution of sequential tasks, in order for the automation level in assembly lines to be increased. Even though not strictly dealing with human-robot physical interaction, this work is relevant in the sense that the authors also developed a gesture based communication protocol between the human and the robot, and proved that the introduction of human-robot task allocation and execution benefitted the efficiency of the assembly process in question.

A system for end-user creation of robust task plans was developed in [32], where a Behavior Tree-based task editor integrates high-level information from known object segmentation, pose estimation with spatial reasoning, and robot actions to create robust task plans. The system was implemented on multiple cobots and performed a wide variety of tasks such as collaborative assembly, wire bending, sanding and polishing.

A multimodal HRI framework is presented in [33] where the authors used speech, hand gesture recognition, text programming, and interaction capabilities to allow the user to intuitively program the robot and take over its control at any given time.

2.4.2 Solutions to Specific HRI Problems

A method for precision HG of a cobot at the EEF level is presented in [34]. In this work, the authors are able to obtain Hand Guiding Force Torque (HGFT) from the EEF FT measurements. Then, a control scheme to govern the linear/angular motion of the EEF is described and implemented. The characteristics of the HG task in this Dissertation has many similarities with this approach, since the former was heavily inspired by the latter.

The work in [35] describes a tool compensation technique to ease and simplify the process of trajectory learning in common industrial setups. It allows the user to directly use the real tool attached to the EEF while HG the robot. It was crucial to understand the effects of coupled weight in the robot EEF and create the compensation model implemented in this Dissertation for the object manipulation task.

A state of the art on-line collision avoidance framework is proposed in [36], where the authors represent in space the human and the robot as capsules, and calculate their minimum distance and relative velocity. With these values, a repulsion vector is created. On the other hand, with a pre-established path generated offline, an attraction vector is computed and forces the robot into following the defined trajectory. With these two components, the implemented control algorithm is able to adjust the robots trajectory in real time, so that it is able to avoid collision with the human and simultaneously fulfill the task.

While studying the problem of collision prediction, the authors in [37] developed a method for robot self-identification based on a over-segmentation approach and the kinematic model of the robot. A similar method will be implemented in this Dissertation, since the vision system will be based on an RGBD camera, and the points belonging to the robot must be segmented from the raw point cloud.

The KUKA Sunrise Toolbox [38] is a MATLAB toolbox developed to interface and control KUKA iiwa robots. It contains functionalities for networking, real-time control, point-to-point

motion and physical interaction. This work heavily inspired the contributions made to the *iris_sami* ROS package and the development of the plugins *rqt_sami* and *rqt_ur10e*, where similar features were developed for UR cobots.

CHAPTER 3

Force Torque Sensor Compensation

In this chapter, we leverage the 6-Axis FT sensor located in the UR10e EEF to create a system that compensates dynamically attached payload. We start by describing the sensor interface, outlining some unexpected behavior and how it was corrected. Then, we present a payload compensation system which uses a theoretical FT model in order to separate FT caused by physical interaction from FT caused by attached payload. Afterwards, we described how the theoretical equations were adapted to better match the observed values. Finally, we describe a software architecture that implements the previous models in real time.

3.1 FT Sensor Correction

To achieve the tasks that require physical contact between the human and the cobot, the UR10e built-in FT sensor, located at its EEF will be extensively used. Its technical specifications can be found in Table 2.1. There are two ways of interfacing with the sensor, both of them managed by the URControl controller. One of them is through URScript commands, with *get_tcp_force()* used for obtaining the FT values at the TCP, and *zero_ftsensor()* that zeroes the FT measurement by subtracting the current measurement from the subsequent. The other, is using the UR ROS Driver which publishes the FT measurement in the *\wrench* topic at 500Hz, and exposes the *\zero_ftsensor* service. Relevant to this matter, the Polyscope interface has a configuration parameter used to describe the weight coupled to the TCP, called Payload. It is composed of mass, in Kg, and Center of Gravity (CoG), in cartesian coordinates, and their effect on the FT measurement will be later demonstrated in Subsection 3.2.1.

3.1.1 Noise Filtering

As seen in Table 2.1, the UR10e FT sensor has a reported resolution of 2N on force, and 0.02Nm on torque measurements. The 2 leftmost graphs of Figure 3.1 demonstrate that this fact is due to the noise generated by the hardware of the sensor.

The chosen way of dealing with the noise is to apply a Low Pass Filter (LPF) to the raw measurements. Equation 3.1 defines a 2^{th} order Finite Impulse Response (FIR) filter. Other

types of LPFs were also considered, such as a Moving Average Filter (MAF) but the results were not satisfactory.

$$\mathbf{W_{fil}} = (1 - \alpha)W_n + \alpha \left(\frac{W_n + W_{n-1}}{2} \right) \tag{3.1}$$

A ROS node with the purpose of filtering the raw measurements was developed. It subscribes to the $\backslash wrench$ topic, filters each value with Equation 3.1 and publishes them in the $\backslash wrench\_filtered$ topic. The α parameter can be changed in real time using the ROS dynamic reconfigure tool. Tests showed that an $\alpha = 0.2$ was satisfactory for noise reduction without major delay impact on the measurements. Results of this filtering technique can be observed in the rightmost graphs of Figure 3.1.

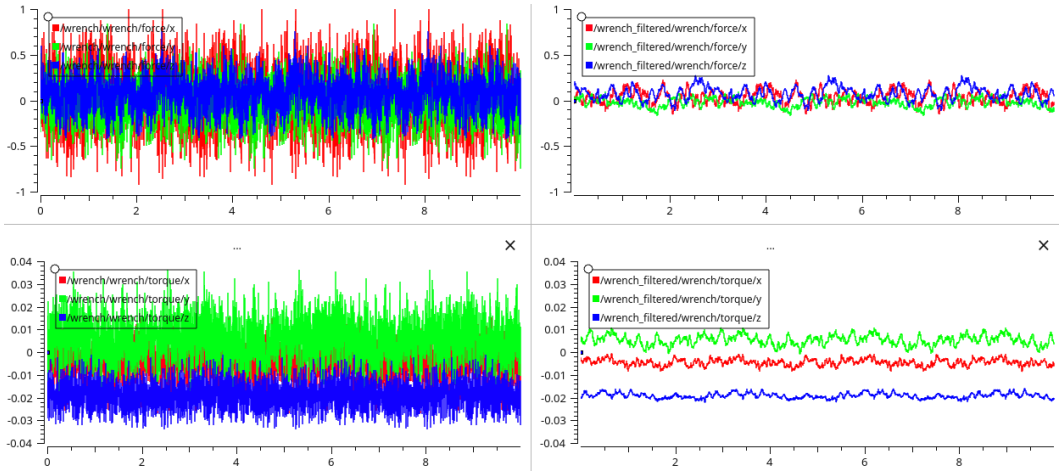


Figure 3.1: Comparison of raw and filtered FT measurements

3.1.2 Observed Behavior

Further experiments with the FT sensor measurements showed a couple of unexpected behaviors described in the following subsections.

Wrist3 Joint Positional Variation

The FT measurements present variations relative to the position of the Wrist3 joint. Figure 3.2 shows a simple real time test where the EEF is fixed in a random pose, and the Wrist3 joint is rotated from $-360°$ to $360°$. The FT measurements present clear variations that can reach 8N of difference from the correct value, which should be 0N the entire test since the EEF has no tool or weight coupled to it. Changing the pose of the EEF presents the same results.

This presents a problem for the accuracy of the HG and object manipulation tasks since an error of 8N is significant. Subsection 3.1.3 demonstrates the proposed solution for this problem.

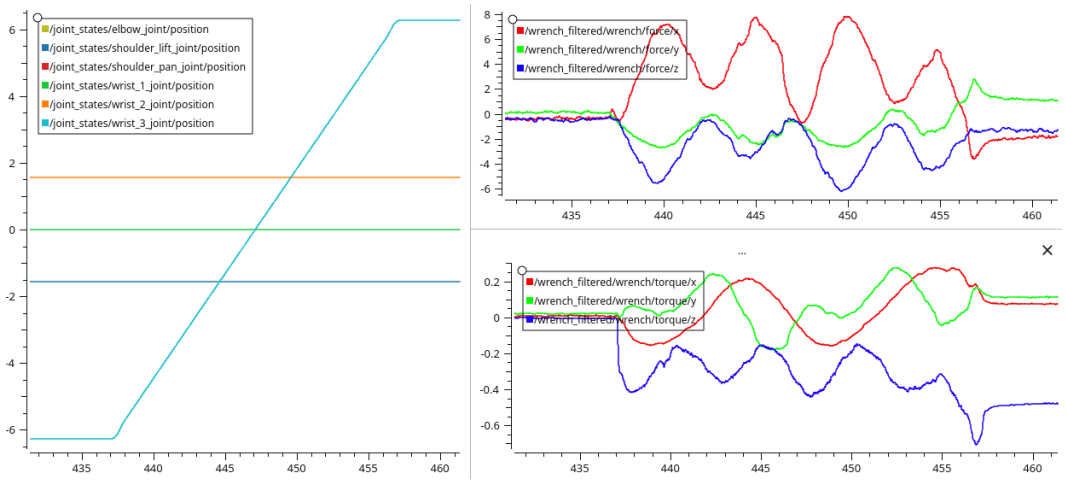


Figure 3.2: Real time test of the variation on FT relative to the position of the Wrist3 joint

Temporal Drift

When the robot is left still for long periods of time, the values of force change linearly with time, as much as 0.2N/min. It might not seem a significant value, but if an HG task were to be activated, by letting the robot stand still for a couple of minutes, it would start moving on its owm. A recording of force values over a period of 10 minutes, with a step of 200ms is shown in Figure 3.3.

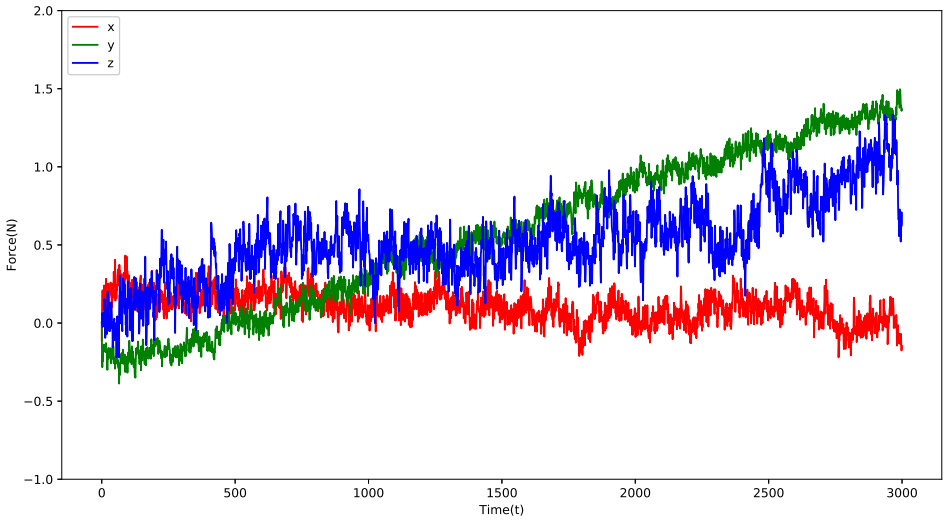


Figure 3.3: Variation of FT relative to time

Using the *zero_ftsensor* service in a recurring, timely manner would fix this problem but raise the issue on when to do so, since it might also erase important FT measurements needed for the executing task.

Variations caused by appliying FT

Applying high amounts of external FT to the sensor, will result in a variation on the measurements after its no longer applied. This behavior can be seen in Figure 3.4 where

every time there is a strong interaction with the EEF, the subsequent values of FT present variations as high as 3N and 0.3Nm.

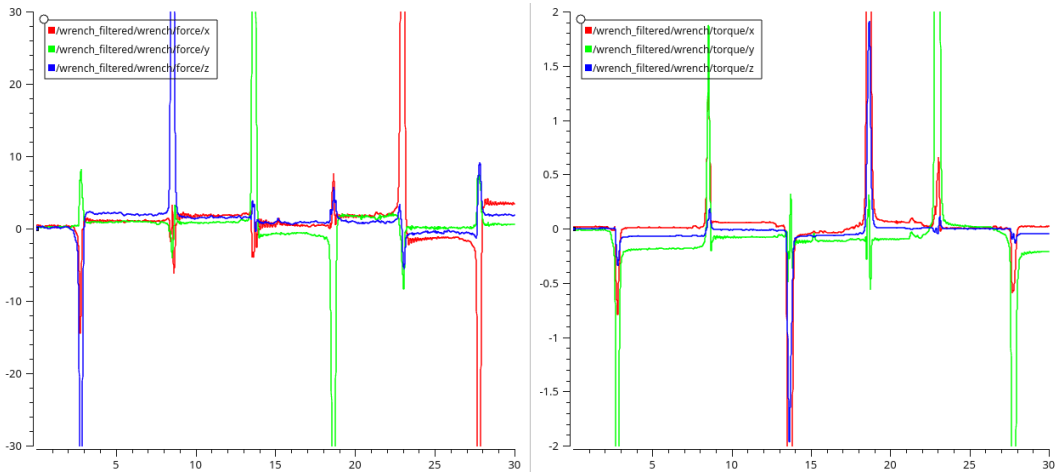


Figure 3.4: Variation in FT relative to interaction with the EEF

Similar to the previous problem, this measurements are wrongly introduced in the system, since the FT felt by the sensor should not suffer alterations due to momentarily external interaction with the EEF.

Special case of the Z-Axis

Upon coupling a gripper tool to the EEF of the robot, the force applied by screwing the mounting plate onto the EEF has effects on the values of force reported on the Z-Axis. The mounting plate has 4 screws, and even making sure they are evenly tightened, the stronger the tightening, the higher the force measurement on the Z-Axis.

Measurements of torque in the Z-Axis present momentary fixed variations depending on the amount of movement and direction of the Wrist3 joint. These variations can be seen in any of the real time tests (Figure 3.2, Figure 3.7, Figure 3.18) but in practice are irrelevant since they only occur when the robot is moving autonomously, not when the user is applying force to the EEF.

3.1.3 Proposed Solution

Regarding the variation of FT relative to the Wrist3 joint position, tests were made to prove that the pattern of variation of the measurements was independent of the EEF orientation, therefore not caused by gravitational forces. The test in question consists on the rotation of the Wrist3 joint from $-360°$ to $360°$ in steps of $1°$. Each step, the average values of FT in a time period of 0.1ms are calculated and saved. The result of the test is a matrix with shape [720, 2, 3], which means 720 values of FT in the 3 cartesian axis. The test was performed in each of the 5 positions described in Figure 3.5, and its internal behavior can be seen in Algorithm 1.

The final test results can be seen in Figure 3.6. As predicted, the variation of FT only has relation to the position of the Wrist3 joint. Furthermore, the standard deviation of the

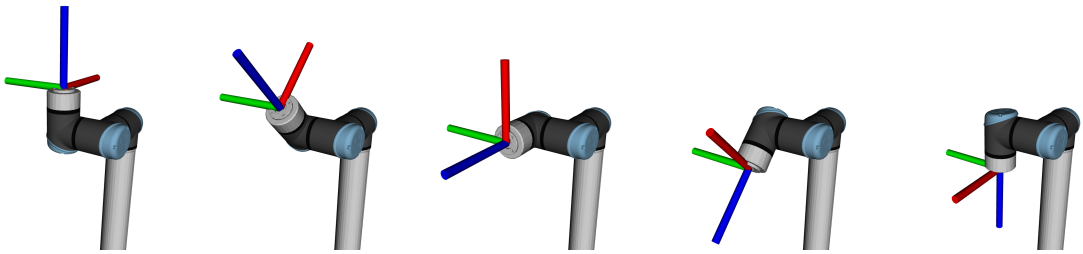


Figure 3.5: 5 EEF testing poses

```
def testWrist3(pose)
    test = Matrix(720, 2, 3)
    ur10e.movePose(pose)
    ur10e.zero_ftsensor()
    wrench = Subscriber("\wrench", size=25)

    for position in range(-360, 360):
        ur10e.moveWrist3(position)
        sample = wrench.getAverage()
        test.append(sample.force, sample.torque)

    return test
```

Algorithm 1: Recording of FT measurements in all position of the Wrist3 joint

results is very low, meaning an accurate solution based on the average values recorded can be implemented.

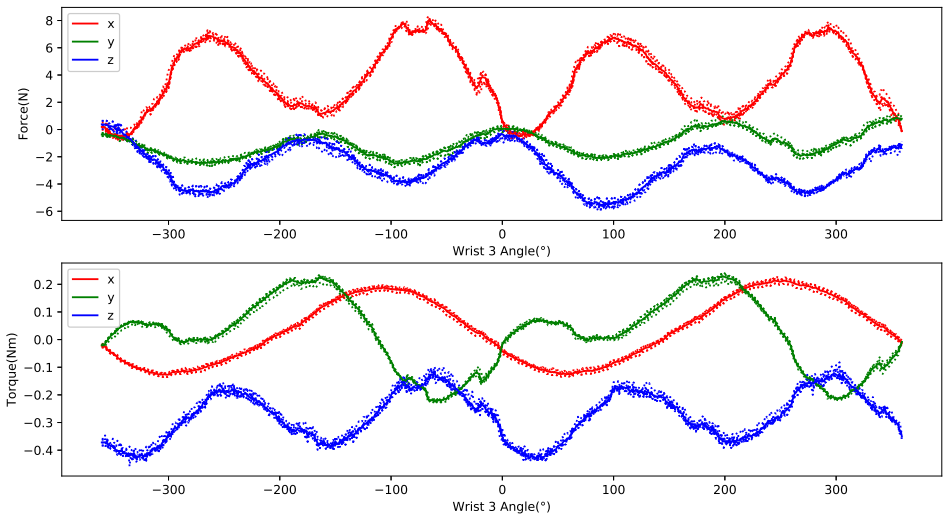


Figure 3.6: Average FT values in the 5 testing poses

As such, from the average results of the tests performed, a ROS node was developed to correct in real time the FT measurements, based on the position of the Wrist3 joint. It subscribes to the filtered values of FT from the topic *wrench_filtered*, and to the current position of the Wrist3 joint, present in the *joint_state* topic. The position of the joint serves as index to the correction matrix. The corrected values are published to the *wrench_corrected* topic.

Regarding the variations of FT based on time and extreme contacts with the sensor, those problems become irrelevant when a cobot is placed in a real use case scenario. For the drifting problem, a simple idle mode can be implemented, where the robot is suspended if no user or program interacts with it. Once resumed, the *zero_ftsensor* service can be called, making any FT drifting measurements disappear. In the case of the variation of FT due to external forces, the solution once again relies on the sensor taring service, since the system can be programmed to trigger it in multiple scenarios, such as a state change or the release of the gripper tool. Further details on the use of this service, in conjunction with a global state machine, to solve the inaccuracies of the FT sensor are presented in Chapter 5.

3.1.4 Results

With the implementation of the correction node, the results from the real time test firstly shown in Figure 3.2 are satisfactory, and presented Figure 3.7.

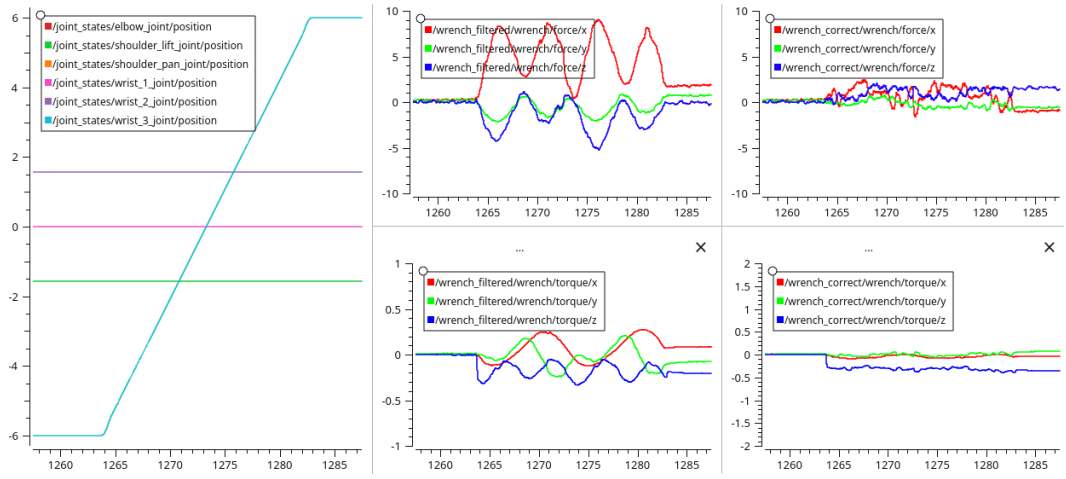


Figure 3.7: Result of the correction node applied in a real time test

The results are as expected. Since the observed pattern is caused by the position of the Wrist3 joint, and is highly repeatable, the corrected FT measurements present themselves relatively close to 0 since the EEF once again has nothing attached. Some variation is still present on the results of Figure 3.7, but it is caused by the nature of the test. Since the movement of the Wrist3 joint is continuous, vibrations and jitters are likely to cause forces on the EEF that ultimately are caught by the sensor.

A more accurate view on the results obtained is shown in Figure 3.8. Each bar in the histograms represents the probability of the correction error be within that range. The results are considered satisfactory since there is not a single component with an error higher than the resolution specifications of the FT sensor (Table 2.1).

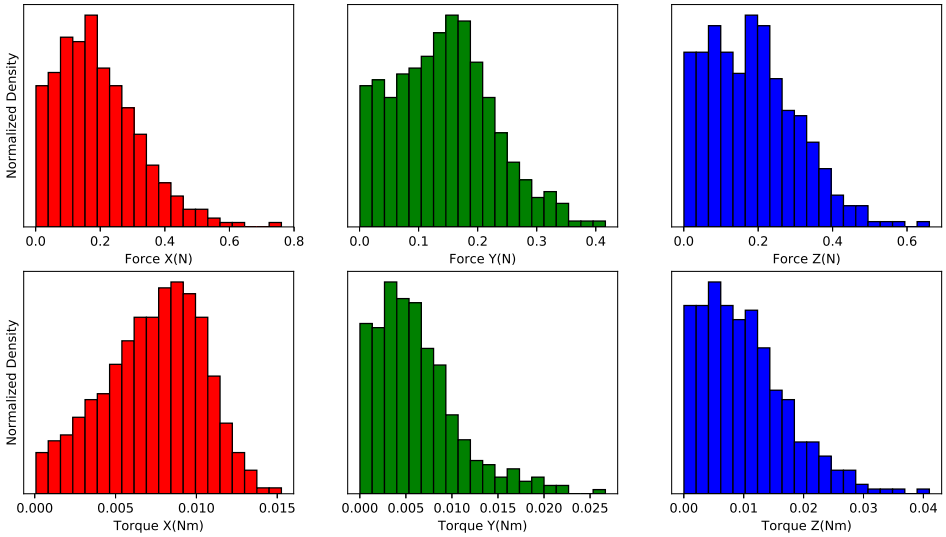


Figure 3.8: Distribution of the correction function error on FT measurements

3.2 EEF Weight Compensation

With the FT sensor properly corrected, its measurements can be used for various applications. One of the objectives of this Dissertation is the precise manipulation of heavy objects through HG. To do so, there needs to be a way to differentiate FT caused by the gravitational forces on the object or tool attached to the EEF, and FT caused by direct physical contact of the user.

3.2.1 UR FT Sensor Controller Internal Compensation

On the Polyscope interface, there is a configuration parameter that directly affects the behavior of the FT sensor, called Payload. With this parameter, the user should declare the mass and the CoG of the object attached to the EEF. In the previous tests the Payload parameter was configured to zero in the two components. The effects of this configuration parameter can be seen on Figure 3.9 where the same positional test was performed, this time with the correction function applied. The Payload parameters for this test were 1.5Kg and a CoG of 0.1m in the Z-Axis. The EEF had nothing attached to it.

Results show a direct compensation of the configured parameters. In terms of accuracy, the FT sensor was tared when the position of Wrist3 joint was 0° and once it rotated to 180°, the force measurement on the X-Axis was -30N, which appears to be correct given the configured mass. Despite this fact, there is no knowledge on the implementation of this compensation mechanism. Furthermore, the use of this parameter would required its constant update when manipulating objects. The ROS Driver enables this feature through a service, but every time the Payload parameter is updated, the sensor is tared.

For these reasons, and because of the fact that this Payload parameter might not exist in other cobotic platforms or sensors, the chosen values of Payload in this Dissertation are 0, and a theoretical compensation model will be developed with the same objective, but enabling the user to have more control over the use of the FT measurements.

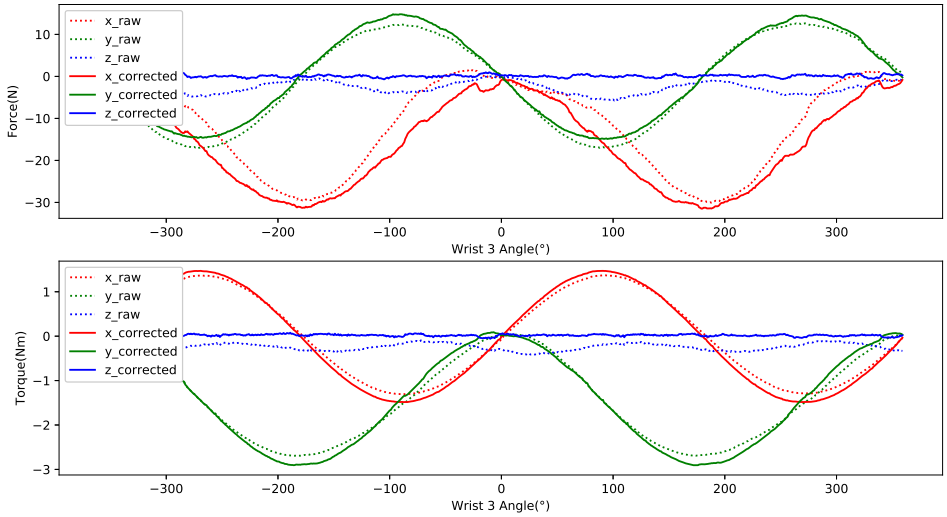


Figure 3.9: Behavior of the FT sensor with parametrization of payload

3.2.2 Force Theoretical Model

The objective of this model is to generate the values of FT that a given object or tool would cause on a perfect sensor. This way, when manipulating objects with the cobot, these generated values can be subtracted from the reported sensor measurements and easily obtain the HGFT. This model is generalized and can be used in any cobot or sensor. As inputs, it requires the orientation of the frame of the FT sensor, and the mass and CoG of the object attached to the EEF.

Force

With the orientation of the sensor relative to the world, 3 unit vectors are created representing the measurement cartesian components. Then, calculating the theoretical force is done with using the inner product of a unit vector representing gravity, and each one of the cartesian unit vectors. Figure 3.10 shows a visual representation of such method.

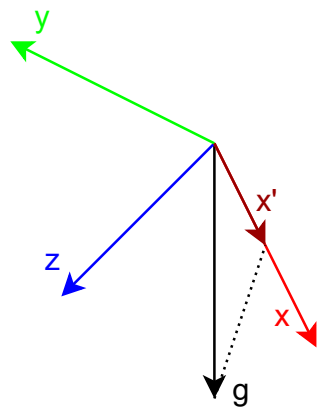


Figure 3.10: Visualization of the generation of theoretical force values

$$\mathbf{Fx} = \langle \hat{x_{eef}}, \hat{g} \rangle \cdot m \cdot g \tag{3.2}$$

Equation 3.2 generates the force in the X-Axis where $\langle \, , \rangle$ represents the inner product between 2 vectors, $\mathbf{\hat{x_{eef}}}$ represents a unit vector with the orientation of the X-Axis in the FT sensor frame, $\mathbf{\hat{g}}$ is a unit vector representing gravity, $\mathbf{m}$ is the mass of the object and $\mathbf{g}$ a scalar representing the acceleration of gravity.

Torque

Analogous to the generation of forces, torque generation also relies on the representation of the cartesian axis as unit vectors. This time, a new vector containing the direction and magnitude of the theoretical torque is created using the cross product of gravity and the CoG of the object. With this vector, torque in each axis can be calculated with the inner product, just like in the force generation. For a better understanding, Figure 3.11 shows a visual representation of this method.

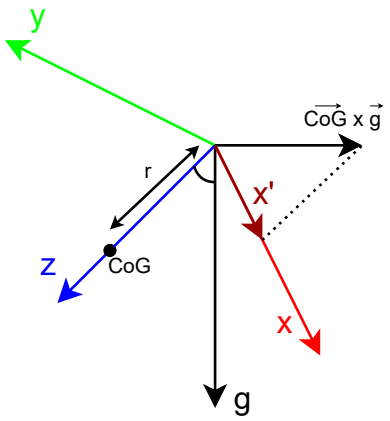


Figure 3.11: Visualization of the generation of theoretical torque values

$$\mathbf{Tx} = \langle \hat{x_{eef}}, \hat{CoG} \times \hat{g} \rangle \cdot m \cdot g \cdot r \tag{3.3}$$

Equation 3.3 generates the torque in the X-Axis where $\mathbf{\hat{x_{eef}}}$ represents a unit vector with the orientation of the X-Axis, $\mathbf{\hat{CoG}}$ represents a unit vector with the orientation of the CoG, $\mathbf{\hat{g}}$ is a unit vector representing gravity, $\mathbf{m}$ is the mass of the object, $\mathbf{g}$ the acceleration of gravity and $\mathbf{r}$ is the distance of the CoG to the origin.

3.2.3 Results

To test the accuracy of the model, the Wrist3 joint test was performed with a gripper tool attached to the EEF. The weight of the tool is 1.5Kg and its CoG is 0.045m in the Z-Axis. While the test was executing, the values from the theoretical model and the real FT sensor were recorded and can be seen in Figure 3.12.

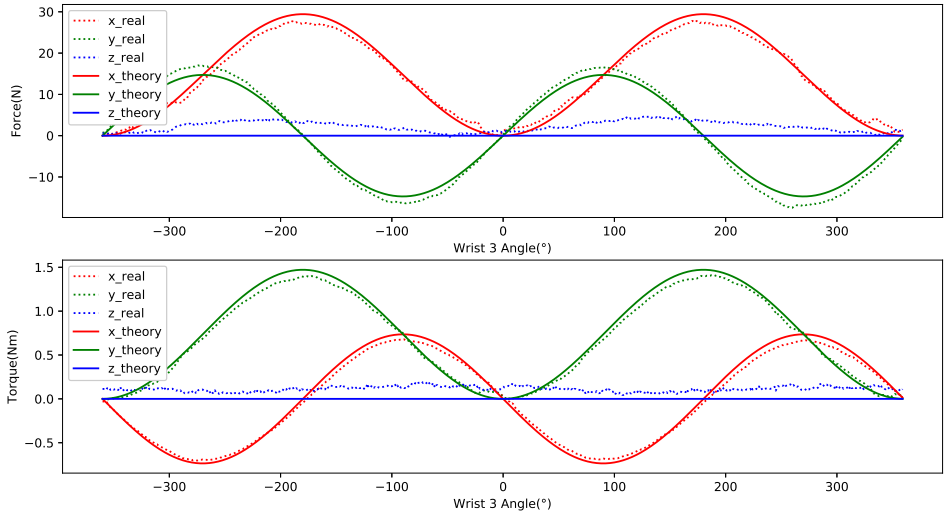


Figure 3.12: Comparison of FT measurements from real sensor and analytical model

Regarding force generation, X-Axis and Y-Axis real measurements appear to deviate from the analytical values by a constant factor. Because this factor is higher than 1 in the Y-Axis and lower than 1 on the X-Axis, it is not caused by the weight parameter, but rather by the internal behavior of the FT sensor. In the case of the Z-Axis, there are large deviations from the analytical model and it appears to be influenced by the force in the X-Axis.

Regarding torque generation, both X-Axis and Y-Axis appear accurate. The offset that exists in the Z-Axis is caused by the nature of the test, since it consists on successive rotations of the Wrist3 joint, where the FT sensor is located.

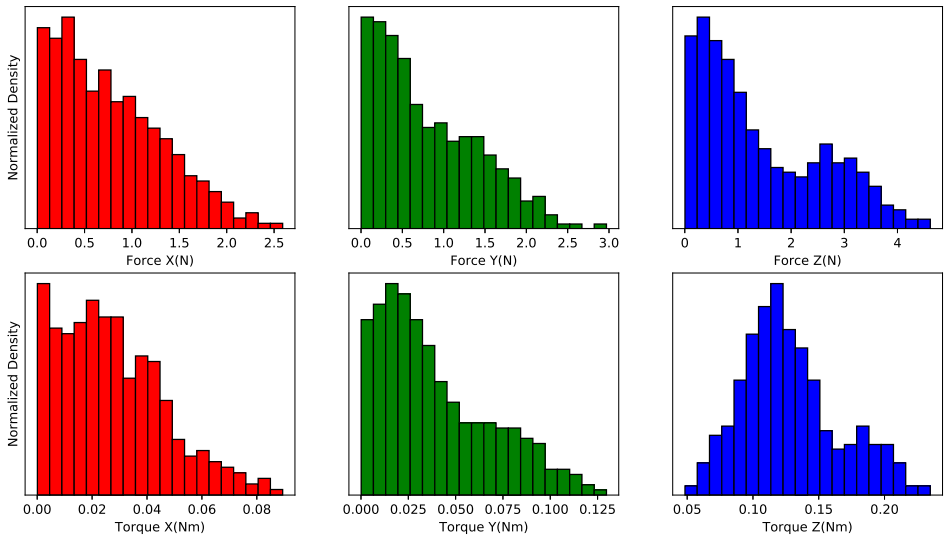


Figure 3.13: Distribution of analytical model error on FT measurements

Once again, a probabilistic view in the form of histograms can better help understand the accuracy of the model. Figure 3.13 shows that for X-Axis and Y-Axis, the FT error is well below the accuracy specification of the sensor (Table 2.1). For the Z-Axis, despite also being below the accuracy specification, it is significantly higher than the other 2 components

and can damage the precision of an HG task, since the FT threshold must be higher than the error of the least accurate component. Given this fact, efforts were made to adapt the analytical model to better match the real values.

3.2.4 Adapting the FT Theoretical Model

By running tests in a variety of different EEF orientations, both real and analytical measurements can be compared and the deviation factors can be computed. Since the the positional test relies on the movement of the Wrist3 joint, in order to create different EEF poses, a combination of Wrist1 and Wrist2 joints angles was used. With steps of 45° between $-180°$ and 180°, a total of 57 different poses can be achieved by combining these 2 wrist joints. Figure 3.14 is a visualization of the resulting EEF orientations.

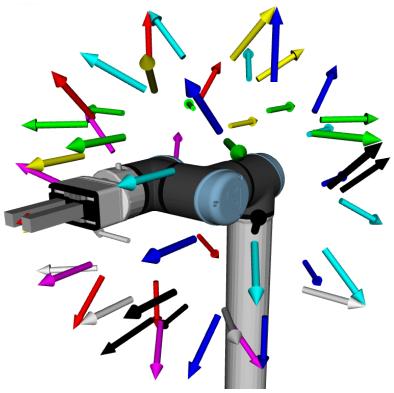


Figure 3.14: Visualization of the 57 poses in which the tests were performed

In each orientation, a Wrist3 positional test is performed, recording both analytical and real FT measurements. With the results, an optimization function based on the Damped Least Squares (DLS) method is used to find the deviation coefficients between the analytical and real measurements. After obtaining such coefficients, they are inserted in the analytical model as correction factors of the generated values. This correction is performed with Equation 3.4 and Equation 3.5.

$$\mathbf{F} = \begin{cases} x = x \cdot \alpha_x \\ y = y \cdot \alpha_y \\ z = z \cdot \alpha_z + x \cdot \alpha_{zx} \end{cases} \tag{3.4}$$

$$\mathbf{T} = \begin{cases} x = x \cdot \beta_x \\ y = y \cdot \beta_y \\ z = z \end{cases} \tag{3.5}$$

Where α and β parameters represent the correction coefficients obtained from the optimization function. When performing the same tests as in Subsection 3.2.3, the difference between the analytical and real measurements is reduced, and the error is now in the range of the noise generated by the sensor, allowing for greater precision in FT dependant tasks.

This fact is shown in Figure 3.15 where FT analytical and real measurements appear overlapped. Futhermore, Figure 3.16 details the distribution of the errors between the 2

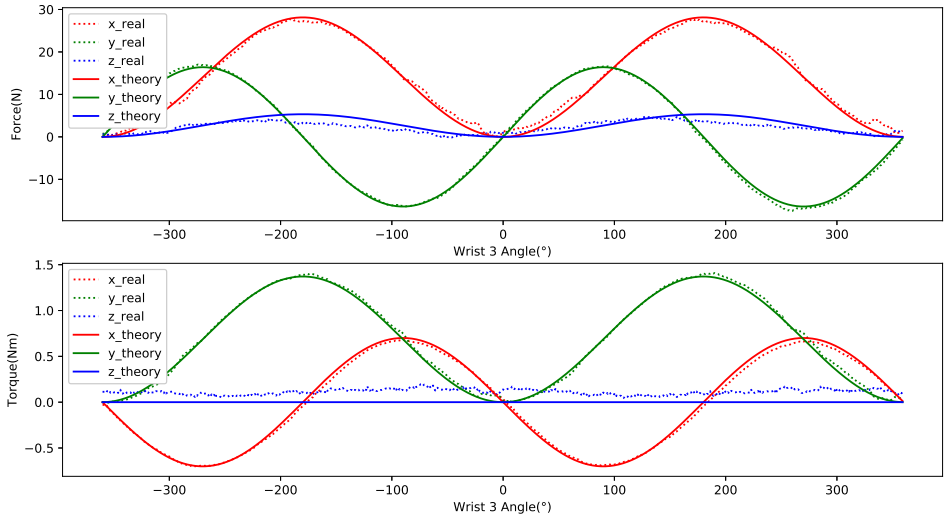


Figure 3.15: Comparison of FT measurements from real sensor and adjusted analytical model

measurements and shows what threshold interaction FT values can be used in an HG task. Given the observations, it is safe to say that a force threshold of 2N and a torque threshold of 0.2Nm are possible. These are very low amounts of FT and a system that can react to them allows precision and ease of use in any sort of manipulation task.

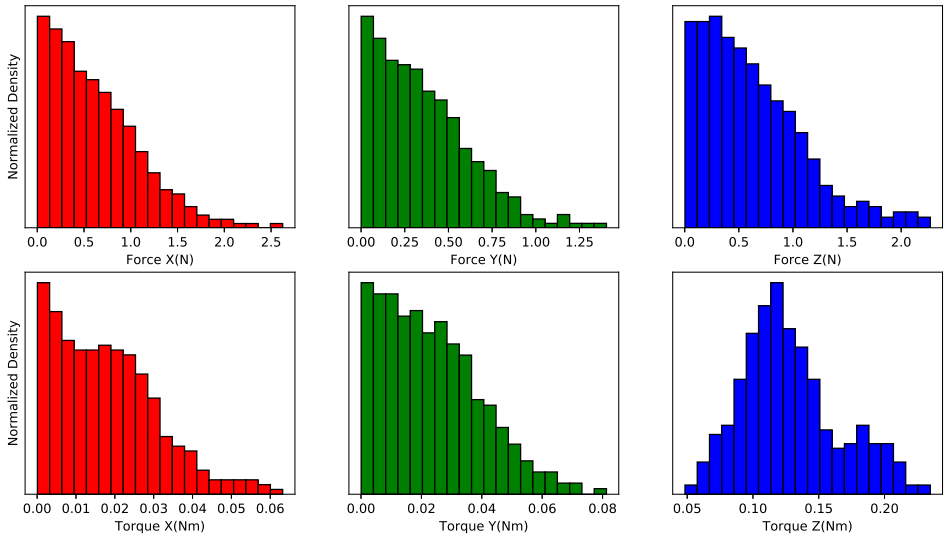


Figure 3.16: Distribution of the adjusted analytical model error on FT measurements

3.3 Real Time Correction and Compensation of FT

In order to obtain the corrected and compensated FT measurements in real time, a group of ROS nodes was developed. Some of them were already previously explained such as the Driver, Filter and Correct nodes. These nodes were arranged together according to the architecture described in Figure 3.17.

The Theoretical FT node implements the algorithms explained in Subsection 3.2.2. It subscribes to the state of the UR10e in order to obtain the EEF pose. Then calculates the

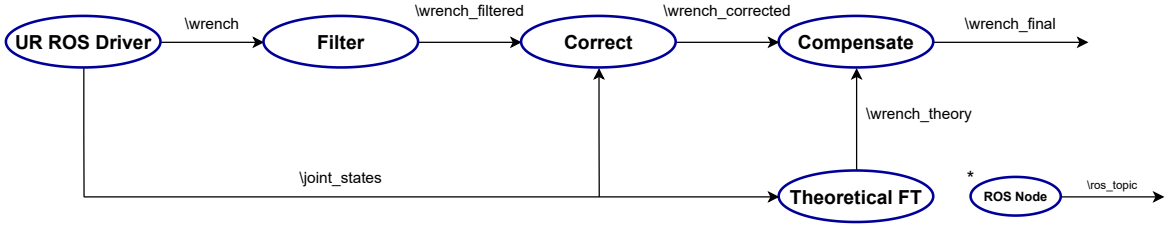


Figure 3.17: ROS architecture of the compensation of FT in real time

theoretical FT values and publishes them to the *\wrench_theory* topic. This node has 2 internal parameters, "weight" and "cog" that can be updated both from the dynamic reconfigure ROS tool and from a ROS service named *\cobot\payload_update*.

Finally, the Compensate node subscribes to both the corrected and analytical values, and publishes the final FT measurements. It provides a ROS service called *\cobot\zero_ftsensor* used to synchronize the corrected and analytical values when the user wants to zero the FT sensor. This is the recommended way to intentionally tare de FT sensor, since this way, previous information is not lost and the remaining nodes dependent of this values, can interoperate safely.

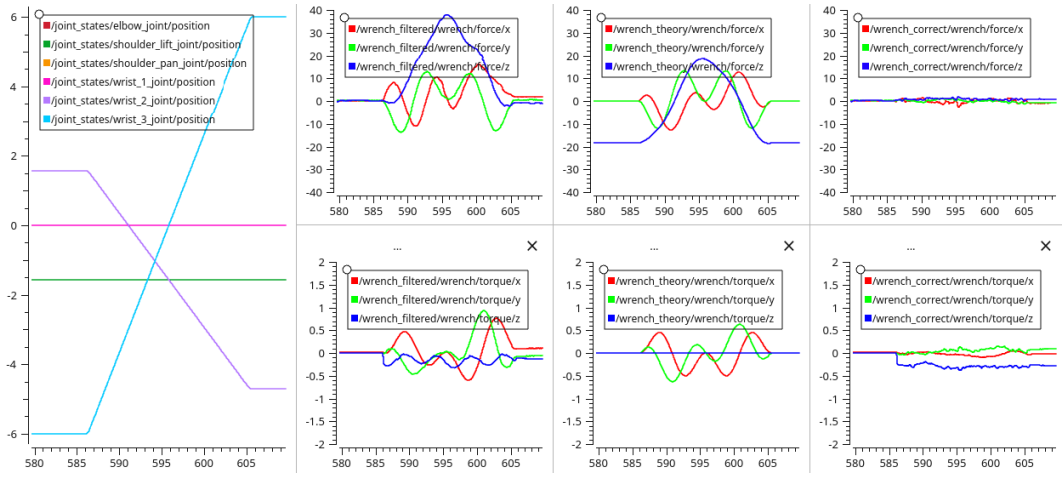


Figure 3.18: Result of the compensation architecture applied in a real time test

To test this architecture, a real time test was performed where multiple joints were rotated to provoque various orientations on the EEF, which had a gripper tool attached to it. Figure 3.18 demonstrates the execution and results of this test, where the Wrist2 and Wrist3 joints are simultaneously rotated. The graphs in the middle show both the real and analytical FT measurements. Their subtraction results in the final FT measurements, shown in the leftmost graphs, and since no external FT was applied during this test, are all close to 0.

This architecture allows for correct separation of FT caused by gravitational forces on the attached weight and FT caused by human physical interaction. This way, the cobot can have any tool or object attached, and simultaneously be manipulated by the user.

CHAPTER 4

Dynamic Obstacle Avoidance

In this chapter, we use an external depth sensor to create a real time collision avoidance system. We start by describing how the sensor information is processed in order to detect obstacles in the environment. Then, from the identified obstacles we explain how they are segmented and the distances they present to the robot. Afterwards, we outline how an attraction and repulsion vectors are created and used to generate a final velocity vector which the robot will use to avoid dynamic obstacles when executing predefined static trajectories. Finally, we describe a software architecture that implements the previous models in real time.

4.1 Obstacle Detection

In order for a robot to avoid obstacles while in motion, an external vision sensor is used to capture the shared workspace and send information to the system about the robot surroundings. In this Dissertation, an RGB Depth (RGBD) camera will be used for this purpose. It gives the system depth information on its Field of View (FOV), referenced to itself. The first step in this endeavour is to spatially and dynamically reference the camera to the robot through hand eye calibration.

4.1.1 Hand Eye Calibration

This process is able to generate an accurate transform from the robot base frame to the sensor frame. In this proposal, a fiducial marker based on the ARuco library [39] is fixed on the robot EEF. Then, using the *easy_handeye* ROS package, a set of translations and rotations are performed at the EEF level. In each movement, a sample composed of the EEF pose and the ARuco pose, identified by the sensor, is captured. A total of 3 translations and 6 rotations are performed in each axis, making a total of 27 samples. When the sampling is finished, the *easy_handeye* compute service is called and a static transform is obtained, which is both locally saved and published in the ROS environment.

4.1.2 Robot Segmentation

Since the camera will be sensing the shared workspace, there will be multiple scenarios where the robot will enter its FOV. Robot segmentation consists on a method to identify which points of the point cloud belong to the robot structure, in order to differentiate obstacles from the robot when both of them are in the sensor FOV.

The technique developed in this work starts by building a skeleton model of the robot, starting with the location of its links. The position of these links, defined in the robot Unified Robot Description Format (URDF) present in its ROS driver, does not match the real position of the robot arms and wrists, therefore, from these links, a set of points are created representing the start and end of each real link. Using this set of points and a minimum distance constant, the skeleton model is created based on Algorithm 2. The difference between the links defined in the robot URDF and the points created can be seen in Figure 4.1.

```
def ur10eSkeleton(min_dist = 0.05):
    skeleton = []
    links = ROS.TransformLibrary.getLinks()
    points = obtainPoints(links)

    for point in points:
        skeleton.append(point)
        distance = point.distance(point.next())
        num_skeleton_points = distance / min_dist
        increment = (point - point.next()) / num_skeleton_points

        for i in range(num_skeleton_points):
            skeleton.append(point + i * increment)

    return skeleton
```

Algorithm 2: Creation of a point based skeleton model of the robot

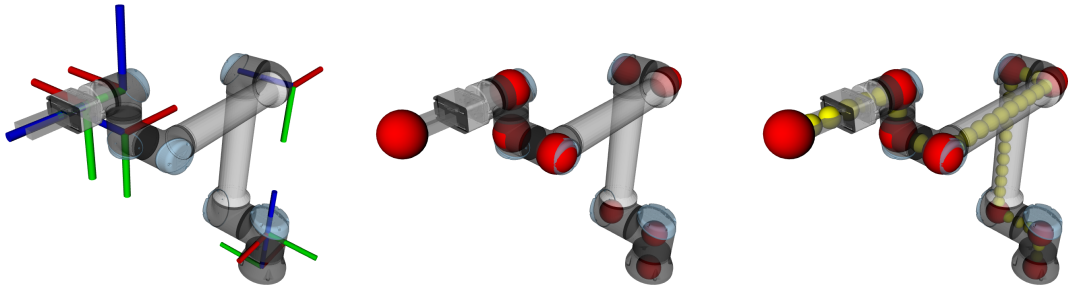


Figure 4.1: Visual representation of the conversion from URDF links to robot skeleton

From the resultant skeleton, any point from the point cloud that is closer to it than a predefined threshold, is considered belonging to the robot, therefore, not accounted for in the obstacle segmentation process. Also not accounted for, are points on the point cloud that are too far away from the robot. Based on a another distance constant, a Region of Interest (ROI) is created, and points that are further distant from the skeleton than it, are considered irrelevant. The final result of robot segmentation can be observed in Figure 4.2.

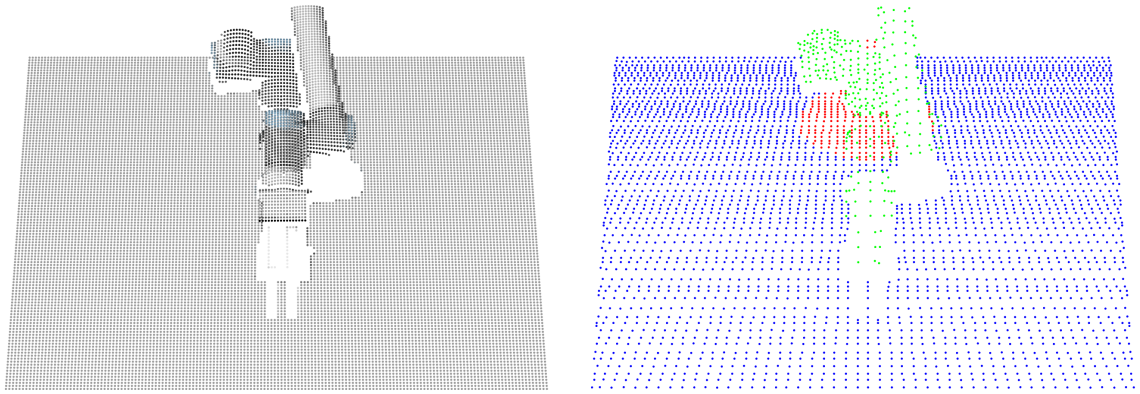


Figure 4.2: Raw point cloud and categorized point cloud with ROI

4.1.3 Obstacle Segmentation

Starting from the raw point cloud obtained from the sensor, each point is categorized. Points belonging to the robot, and points too distant from it are removed. Only points belonging to the ROI, if there are any, are considered obstacles, therefore, a new point cloud is created with these points.

To this new point cloud, a clustering algorithm is applied in order to represent the point cloud in a set of small clusters, each one to be considered an obstacle. The algorithm applied is the Conditional Euclidean Clustering (CEC), which is a region growing algorithm based on the Euclidean Clustering Extraction (ECE) algorithm. It has the advantage of allowing a customizable clustering condition and also classifying clusters as too small or too large, based on defined parameters. The parameters for the CEC implementation, used in this work, can be found in Table 4.1.

Parameter	**Value**
Leaf Size	0.3
Radius Search	0.4
Cluster Tolerance	0.5
Min Cluster Size	5
Max Cluster Size	90
Squared Distance	0.001

Table 4.1: Internal parameters of the CEC algorithm

The result of this algorithm is a set of clusters and to each cluster, its closest point to the robot is calculated, achieving the obstacle collision point. Figure 4.3 illustrates the obstacle detection final result, where 2 obstacles are present in the environment. One of them is outside the ROI, therefore ignored. The other, which is closer to the robot, is correctly identified.

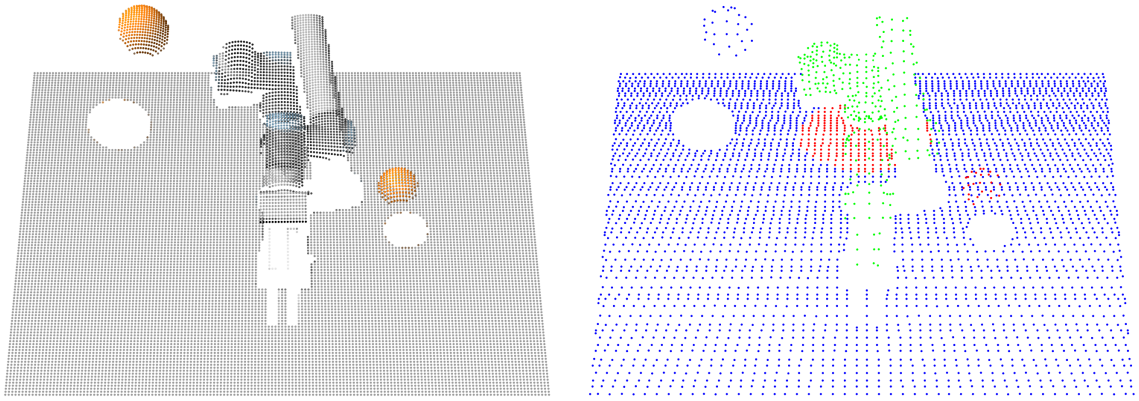


Figure 4.3: Final result of the obstacle segmentation algorithm

4.2 Artificial Potential Fields

A well known method of dealing with real time collision avoidance on robotic manipulators is to treat the robot navigation as if it was a point in a potential field, and is called Artificial Potential Field (APF). An APF is the result of the sum of 2 fields. An attraction field, converging on the goal and a repulsion field created by obstacles in the environment. Within the final field, the robot can navigate successfully while avoiding the obstacles. Adapted to the context of robotic manipulators, the robot is in an imaginary vector field and its motion is the result of attraction forces, making the robot follow a predefined trajectory, and repulsion forces that repel it away from obstacles.

4.2.1 Attraction

The attraction force is what makes the robot move to its goal. In this work, this is done by generating a trajectory offline, and in real time make the robot follow the trajectory with cartesian space velocity commands.

The first step is to generate an offline trajectory. A trajectory needs a start state and a goal state, and if no static obstacles are declared, the motion planner will generate an optimized trajectory based on those two inputs. Other factors might impact the final result but for the focus of this work, offline motion planning will not be discussed in detail. The trajectories generated with the MoveIt framework are arrays of joint positions, since it is designed to work with joint position controllers. In this work, the attraction and repulsion forces will be declared in cartesian space, therefore after obtaining the trajectory, all trajectory steps are converted to poses in cartesian space. This can be observed in the first subfigure of Figure 4.4.

After obtaining the set of poses, the robot will start executing the trajectory in its first point. For this initial position, the robot is moved with a joint position command. After the initial state, in a 500Hz loop a cartesian attraction vector, which is the difference between the current EEF position and the next point in the trajectory, is obtained with Equation 4.1.

$$\vec{\mathbf{Att}} = P_{goal} \cdot T_e^{b^{-1}} \tag{4.1}$$

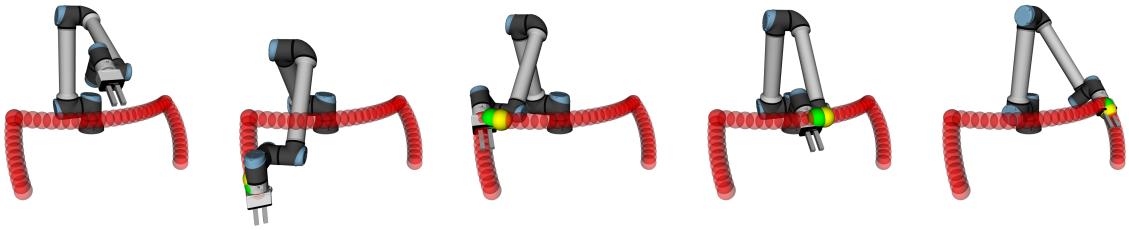


Figure 4.4: Trajectory execution with cartesian attraction vector

Where $\mathbf{P_{goal}}$ is the pose of the goal point in the trajectory and $\mathbf{T_e^b}$ a transformation from the base frame to the EEF frame. $\vec{\mathbf{Att}}$ is obtained in the form of a pose object. The position part will be the linear velocity and the orientation part will be the angular velocity. This vector is constantly being calculated based on the monitoring of the EEF and the calculation of the new goal.

4.2.2 Repulsion

The obstacle segmentation algorithm produces an array of obstacles. Each obstacle is defined by its pose. The repulsion vector is calculated using this pose and the EEF pose, since its the most likely part of the robot to colide with obstacles. Within the array of obstacles, the closest to the EEF is selected. A repulsion vector is created with this obstacle based on how distant it is to the EEF with Equation 4.2.

$$\vec{\mathbf{Rep}} = \frac{P_{eef} - P_{obs}}{\mid\mid P_{eef} - P_{obs} \mid\mid} \cdot \frac{max_d - (d - min_d)}{max_d} \tag{4.2}$$

Where $\mathbf{P_{eef}}$ is the EEF pose, $\mathbf{P_{obs}}$ is the obstacle pose, $\mathbf{d}$ is the distance between the EEF and the obstacle, and $\mathbf{max_d}$ and $\mathbf{min_d}$ are maximum and minimum distance constants, respectively. $\vec{\mathbf{Rep}}$ is obtained in the form of a linear vector since, in the repulsion vector calculation, angular velocity is not accounted for. The first part of this equation represents a unit vector with the direction of the obstacle, and the second part can be seen as a repulsion factor, which consists on a linear function that ranges from $\frac{min_d}{max_d}$ to 1. This factor causes the magnitude of the repulsion vector to increase as the distance to the obstacle decreases.

4.2.3 Controller

The APF controller sums the forces from the two components. In this work, the chosen approach is to create a final cartesian velocity vector, taking as inputs the two forces previously explained. The final velocity for the robot EEF is calculated with Equation 4.3.

$$\vec{\mathbf{V_{eef}}} = (1 - \mid\mid \vec{Rep} \mid\mid) \cdot \vec{Att} + \mid\mid \vec{Rep} \mid\mid \cdot \vec{Rep} \tag{4.3}$$

This way, when there are no obstacles, $\mid\mid \vec{\mathbf{Rep}} \mid\mid = \mathbf{0}$, therefore, $\vec{\mathbf{V_{eef}}} = \vec{\mathbf{Att}}$. When obstacles do exist, the controller will give priority to the repulsion vector as is starts to grow, since as seen in Subsection 4.2.2, the repulsion vector norm increases as the distance to obstacles decreases.

4.3 Real Time Obstacle Avoidance

Similarly to the correction and compensation of FT, a group of ROS nodes was developed and orchestrated in order to give the robot the ability to avoid dynamic obstacles while in motion and in real time. Figure 4.5 demonstrates how those nodes are arranged.

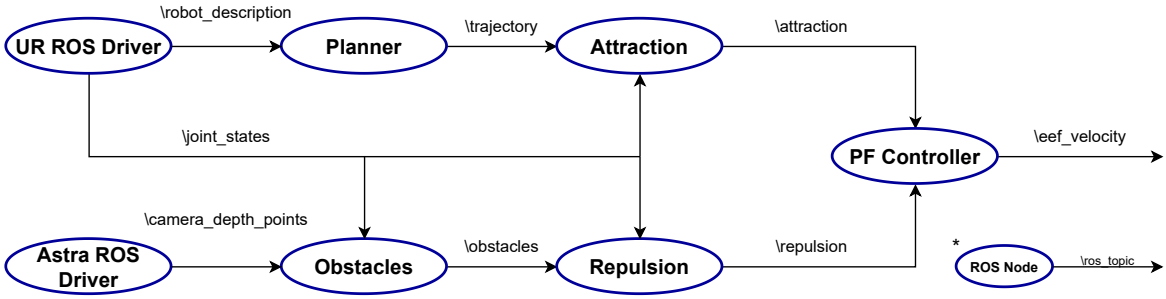


Figure 4.5: ROS architecture of the obstacle avoidance in real time

The attraction component starts at the Planner node, which uses the MoveIt motion planning interface to generate trajectories. It obtains the UR10e description from its driver and with a move group object plans and publishes robot trajectories. The attraction node listens to the trajectories published by the Planner node and acts according to what was described in Subsection 4.2.1. On the repulsion side, the Obstacles node listens to the point clouds published by the sensor driver and publishes an array with obstacle information. From that information, the Repulsion node publishes the repulsion vector, according with Subsection 4.2.2. Finally the APF Controller node joins the attraction and repulsion vectors, publishing the final EEF velocity vector.

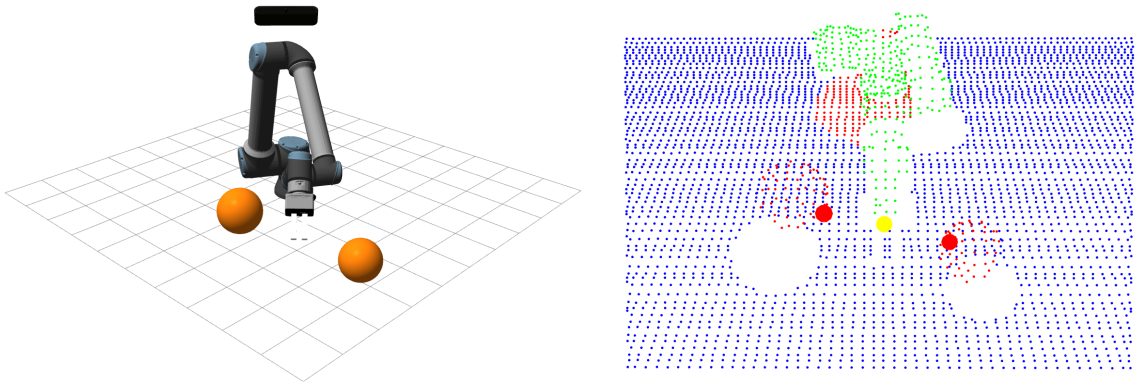


Figure 4.6: Collision avoidance setup in the Gazebo simulator

A testing environment was built in the Gazebo simulator. The UR10e was placed in the center of the world, an RGBD camera was placed on top of it, overlooking an area considered the shared environment, where 2 sphere obstacles were spawned. This setup can be seen in Figure 4.6. A trajectory was created that would intentionally colide with these obstacles.

The offline trajectory and the final trajectory that the robot performed with the collision avoidance nodes enabled can be seen in Figure 4.7. Further details on the performance of

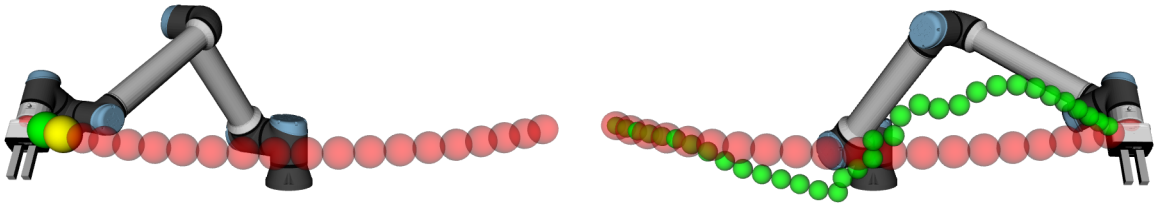


Figure 4.7: Collision avoidance offline and final trajectory

this architecture and details on the implementation on a real scenario will be outlined in Subsection 6.2.4.

CHAPTER 5

Collaborative Tasks

In this chapter, we describe the implementation of the previous models in order to create the proposed collaborative tasks. We start by outlining how each task was achieved, describing its interface, flow and parameters. Then, we present the culmination of said tasks in a collaborative state machine that allows seamless integration and execution of each task. We finish this chapter showcasing 2 GUIs that were developed with the purpose of enhancing the development of cobotic applications.

5.1 Hand Guiding

Section 3.2 explained how to obtain a compensated FT measurement in order to obtain Hand Guiding Force Torque (HGFT), which is caused by the user. This value is used to move the robot in the direction that it is applied. For this to happen, these values felt on the FT reference frame need to be converted to a velocity vector, referenced on the base frame of the robot.

5.1.1 HGFT to EEF Velocity

HGFT should only converted in robot motion when it reaches a certain threshold, otherwise, it should be zero. It should also exist a way to define the ratio of conversion between FT and velocity. To achieve this behavior, the HGFT is controlled according to Equation 5.1 and Equation 5.2.

$$\mathbf{f_i} = \begin{cases} \frac{f_i - f_{th}}{f_{div}} & f_i > f_{th} \\ \frac{f_i + f_{th}}{f_{div}} & f_i < -f_{th} \\ 0 & \text{otherwise} \end{cases} \qquad (5.1)$$

$$\mathbf{t_i} = \begin{cases} \frac{t_i - t_{th}}{t_{div}} & t_i > t_{th} \\ \frac{t_i + t_{th}}{t_{div}} & t_i < -t_{th} \\ 0 & \text{otherwise} \end{cases} \qquad (5.2)$$

Where $\mathbf{f_i}$ and $\mathbf{t_i}$ are FT components, $\mathbf{t_{th}}$ and $\mathbf{f_{th}}$ are FT threshold parameters, and $\mathbf{t_{div}}$ and $\mathbf{f_{div}}$ are ratio constants that define the conversion between FT and velocity.

Force to Linear Velocity

It is performed by multiplying the force vector by the rotation matrix between the base frame and the sensor frame. Then, the resulting vector can be used as cartesian velocity. Equation 5.3 is used to calculate the linear velocity from the HG force.

$$\vec{\mathbf{V}}_{\mathbf{lin}} = R_e^b \cdot R_s^e \cdot P_{force} \tag{5.3}$$

Where $\mathbf{R_e^b}$ is the rotation matrix of the EEF frame in relation to the robot base frame, $\mathbf{R_s^e}$ is a constant rotation matrix of the sensor frame in relation to the EEF frame, and $\mathbf{P_{force}}$ is a point, referenced in the base frame, with the force measurements as its coordinates. $\vec{\mathbf{V}}_{\mathbf{lin}}$ is obtained as a cartesian points which can directly be used as a linear velocity vector in the robot base frame.

Torque to Angular Velocity

It is performed by obtaining the difference between the sensor rotation, and another rotation with the torque measurements as its components. Both of these rotations are referenced in the base frame. Equation 5.3 is used to calculate the angular velocity from the HG torque.

$$\vec{\mathbf{V}}_{\mathbf{ang}} = (R_e^b \cdot R_s^e \cdot R_{torque}) \cdot (R_e^b \cdot R_s^e)^{-1} \tag{5.4}$$

Where $\mathbf{R_e^b}$ is the rotation matrix of the EEF frame in relation to the robot base frame, $\mathbf{R_s^e}$ is a constant rotation matrix of the sensor frame in relation to the EEF frame, and $\mathbf{R_{torque}}$ is a rotation with the torque measurements as its elements. $\vec{\mathbf{V}}_{\mathbf{ang}}$ is obtained as a cartesian rotation which can directly be used as angular velocity in the robot base frame.

5.1.2 EEF Velocity to Joint Speed

With the previous equations, the EEF velocity in cartesian space is obtained from the HGFT on the FT sensor. As said before, in this work, the UR10e will be controlled in joint space with joint speed commands. To obtain joint speeds from an EEF velocity, the Jacobian inversion method will be used.

Jacobian Inversion Method

The Jacobian matrix is used in robotics to provide a relation between joint speeds and EEF velocities in a robotic manipulator. This relation is given by Equation 5.5.

$$\dot{\mathbf{X}} = J(q)\dot{q} \tag{5.5}$$

Where $\dot{\mathbf{q}}$ is the robot joint velocity vector, $\mathbf{q}$ is the robot joint position vector, $\mathbf{J}$ is the Jacobian matrix, which is a function of the current robot joint positions. $\dot{\mathbf{X}}$ represents the EEF velocity and is obtained in the form of a 1 column matrix with 6 elements. The first part represents linear velocity, and the second represent angular velocity. The Jacobian matrix is obtained through forward kinematics equations applied on a given joint state. In this

work, the Jacobian matrix is obtained with the MoveIt motion planning framework, using the UR10e robot description.

In order to obtain joint speeds from the EEF velocities, the Jacobian matrix is inverted and Equation 5.6 is used.

$$\dot{\mathbf{q}} = J(q)^{-1}\dot{X} \tag{5.6}$$

Since the UR10e has 6 joints, its Jacobian matrix is a square matrix, and it is possible to invert it and use it according to the previous equation to convert EEF velocities into joint speeds.

5.1.3 HG Architecture

Having the joint speeds calculated, they are sent back to the ROS driver by publishing them into the appropriate topic, at 500Hz. The ROS architecture for this task is demonstrated in Figure 5.1.

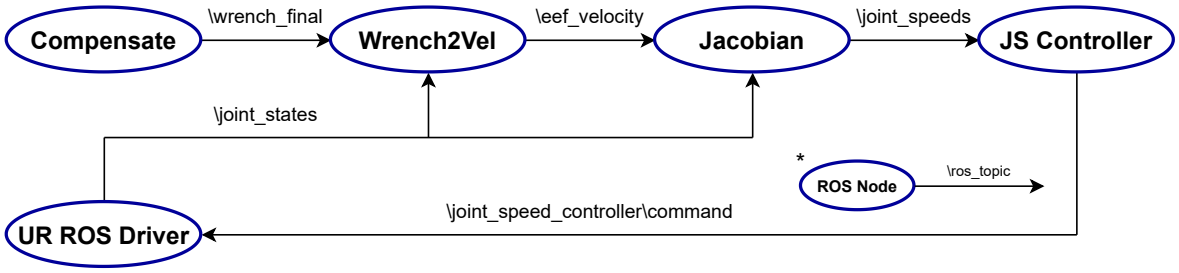


Figure 5.1: ROS architecture of the HG task

The Wrench2Vel ROS node implements the algorithms explained in Subsection 5.1.1 and the Jacobian node implements the Jacobian inversion method described in Subsection 5.1.2. The Joint Speed (JS) Controller node subscribes to the joint speeds calculated with the Jacobian matrix and publishes them in the appropriate topic made available by the UR ROS driver. This node also implements a filter to smooth the robot acceleration when rapidly changing velocity, and a ROS service that allows its control with play, pause and stop functions.

5.2 Object Transfer

The object transfer collaborative task is an example of a user customizable skill. It is very similar to the general pick and place task with the only difference being that the final destination is not a place position, but rather a deliver position where the robot will wait for user input to release the object. This task has various states, and each state has input parameters and a certain behavior. Figure 5.2 shows the arrangement of the states and their parameters.

The task starts by moving the robot to a picking position, defined in the *pick_pose*. Since this position is a predefined joint state, the trajectory can be planned with the MoveIt framework and the motion of the robot is controlled through its positional controller. This

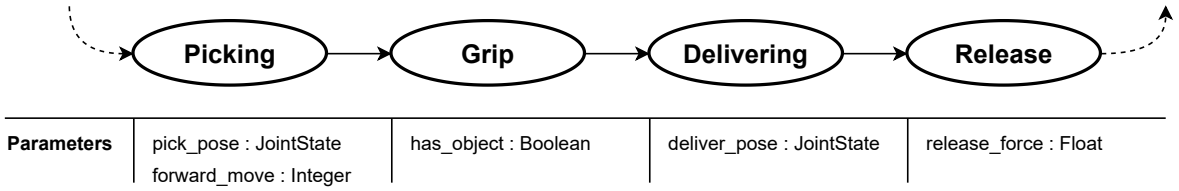


Figure 5.2: State architecture of the object transfer task

position is preferably a few centimeters distant from the object, so the next step is to move the robot forward according to the *forward_move* parameter.

The robot should reach the next state with its gripper fingers aligned with the object, therefore, the next step is to close the gripper. The *has_object* flag is a control parameter used to only proceed in the flow of the task if the gripper has gripped something. This happens if the flag is set to true, otherwise, the task would continue ignoring if the gripper has gripped any object.

The Delivering state moves the robot to another predefined position. When it reaches that position, it automatically skips to the Release state, where it stops and the system continuously evaluates the FT measured. If the *release_force* parameter is configured, the robot will release the object when a force with magnitude equal to *release_force* is applied. If this parameter is not set, which is the default behavior, the robot will release the object when its weight has been supported by the user, which programmatically means that the robot will release the object when the FT measured in the Z-Axis of the world frame is higher than zero.

5.3 Object Manipulation

The object manipulation task has the goal of allowing the user to give the robot an object and be able to HG it with precision, while the robot holds the object. Figure 5.3 shows the composition of this task states.

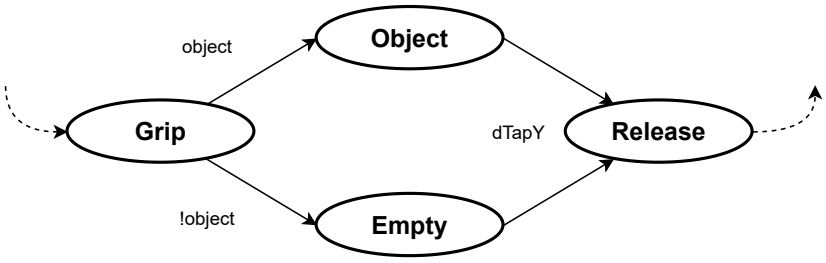


Figure 5.3: State architecture of the object manipulation task

The Grip state has been seen previously, but this time, instead of configuring a parameter to evaluate if the gripper has anything attached, this condition will be used as a way to determine the next state. If the gripper has no object, it will skip to the Empty state which has no internal behavior other that to wait for user input. This input has the form of a double tap on the EEF, in the Y-Axis direction. Further details on this communication interface with the system will be given in Subsection 5.5.1.

If the gripper has an object, the system will skip to the Object state where the attached Payload will be measured and injected in the theoretical FT model. This state starts by turning off the velocity controller and moving the robot upwards. This motion serves as a signal to the user to not touch the EEF for a while. Then, the system obtains a FT measure and from it, it calculates the object weight and CoG. With these parameters, it updates the theoretical FT model through its $\backslash payload\_update$ ROS service. Once the theoretical FT model gives confirmation, the velocity controller is turned back on, and with the updated values of payload, the user can accurately HG the system knowing that it will compensate the object weight. When the user is done with this task, he executes the same double tap as explained before and the robot transitions to the Release state.

Finally, in the Release state, the velocity controller is turned off, the robot releases the object, the theoretical FT model is configured with just the gripper tool payload, the FT sensor is tared, and just before leaving this state, the the velocity controller is resumed.

5.4 Collision Free Execution of an Industrial Task

This system allows for seamless integration of any industrial task given it provides a certain control interface. This control interface requires the existence of play, pause and stop services that will be called depending of user interaction. The structure of the states of this task can be seen in Figure 5.4.

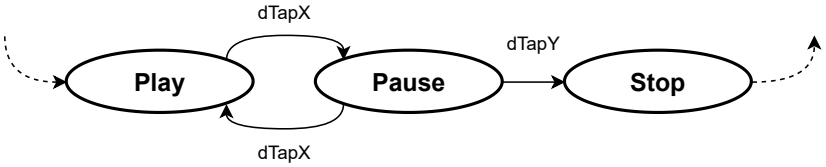


Figure 5.4: State architecture of the industrial task

In this Dissertation, the industrial task implemented is a looped transition between two positions with dynamic collision avoidance. Programmatically, when this state is activated, the behavior described in Chapter 4 is reproduced in a loop by activating the APF Controller node. The trajectory defined in the Planner node will start being executed and any obstacles detected by the Obstacles node will be avoided. While the task is executing, the user can pause it by approaching the EEF, which will start to slow down, and make a double tap on its X-Axis.

5.5 Collaborative State Machine

The tasks explained before are all arranged in a global state machine that treats each task as a submachine. The initial state is the Hand Guiding task which allows free manipulation of the robot. To execute a task, a double tap in a specific direction must be executed on the robot EEF. The behavior flow of the global state machine is described in Figure 5.5.

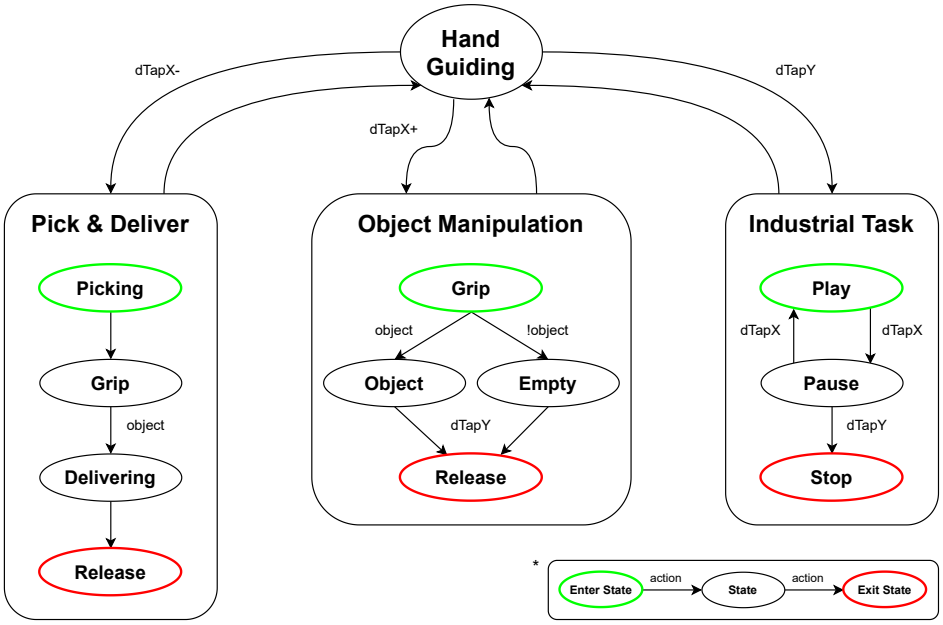


Figure 5.5: Collaborative state machine

The integration of these tasks with each other is seamless. They all start from the HG state and also return to it when finished. One of the main advantages of an arrangement such as this, is that in the execution of the different tasks the robot can be controlled by multiple entities. For instance, the Object Transfer task uses the MoveIt positional controller, the Object Manipulation uses the Wrench2Vel ROS node, and the Industrial Task uses the PF2Vel ROS node. The existence of a structure such as this state machine, allows for a cohesive and seamless change of control entities, without the user ever noticing their existence.

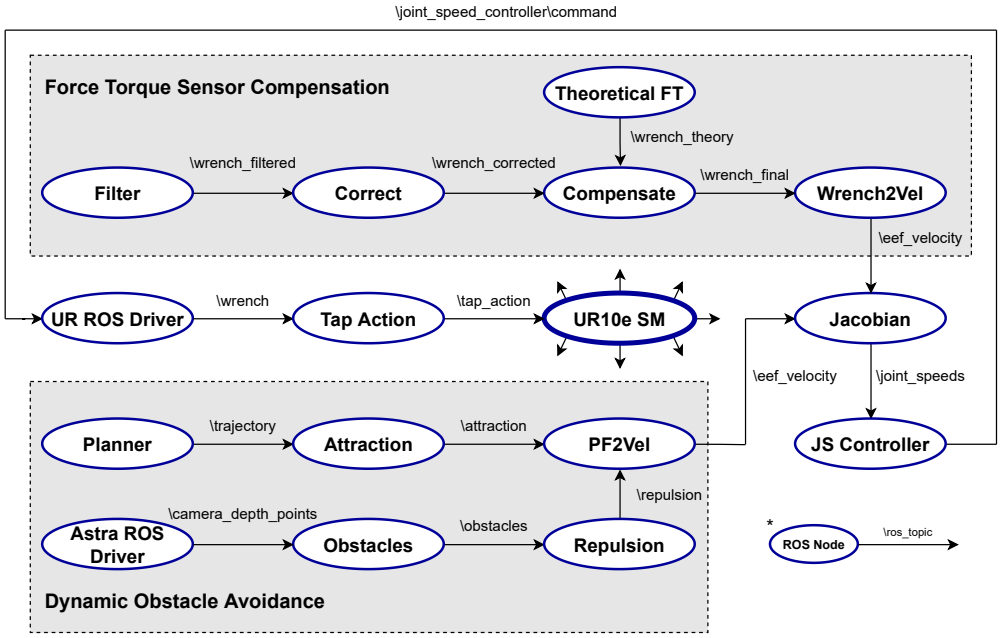


Figure 5.6: Complete ROS architecture of the system

A complete view of the ROS architecture of the global state machine, and generally, the work developed in this Dissertation can be seen in Figure 5.6. The symbolism on the UR10e State Machine (SM) means that this node interacts with almost every other node in the system, with both services calls and information collecting through topics.

5.5.1 State Transitions

The node that enables the user to interact with the system is the Tap Action node, which listens to the *wrench* topic, evaluates each FT component derivative, and according to it publishes TapAction messages. This a custom defined ROS message that contains 3 fields:

- **type** (String) - Defines the type of tap that was registered, single ou double.
- **component** (Integer) - Defines in what axis the tap occurred.
- **direction** (Boolean) - Defines the direction of the tap, positive or negative.

To generate a message of this type, the Tap Action node, derives each component of FT and once this value passes a certain threshold, it publishes a TapAction message. The effects of the derivation of FT can be seen in Figure 5.7.

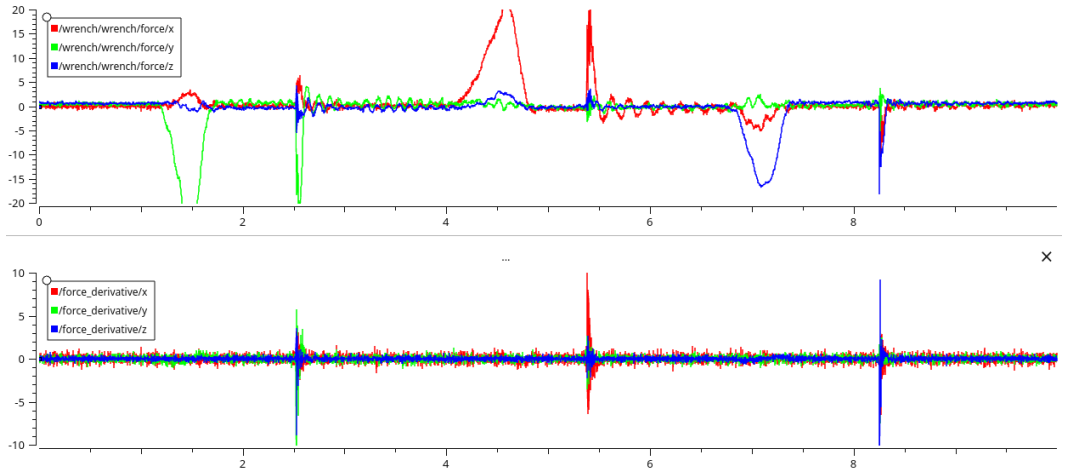


Figure 5.7: Result of the compensation architecture applied in a real time test

In the top graph, where raw FT values are plotted, a push and a tap are performed in each axis. A push is an amount of force applied slowly, and a tap is the same amount of force, but applied rapidly, like a knock on a door. By looking at the derivative graph on the bottom, it is evident that a tap can be detected just by evaluating the derivative value.

5.5.2 Visual Feedback

A means of visual feedback from the system to the user can be achieved with the gripper Light Emitting Diodes (LEDs). Its consists on a luminous ring with Red Green Blue (RGB) lights that can be programed by the user in terms of color, animation and speed. The effects that are possible to reproduce on the luminous ring are diverse, with the user being able to choose from 15 animations, 13 colors and 8 speed settings. Figure 5.8 shows some examples of these effects.

In this work, 3 colors are used to give the user awareness of the system state:

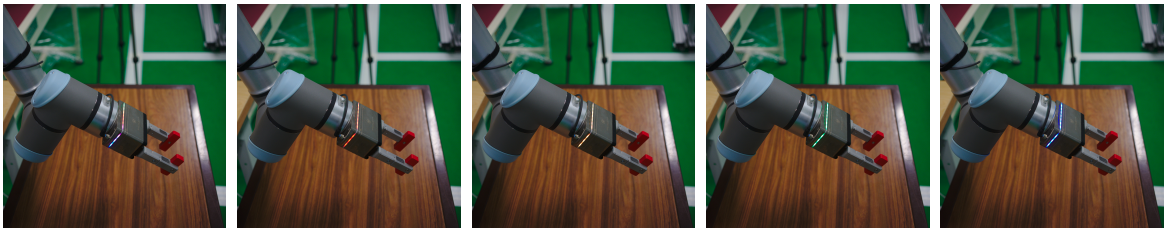


Figure 5.8: LED status feedback on the gripper tool

- **Green:** The user is free to physically interact with the system.
- **Orange:** The cobot is busy and the user should not interact with it. The system might be executing a predefined trajectory, or calibrating the weight of an object.
- **Red:** Some component of the system lost connection with the robot and the user should attend to one of the GUI in order to solve it.

With this approach, the user can always know when he should interact with the robot.

5.6 Software Tools for HRC

The collaborative state machine provides an integrated interface for robot usage and control, but in order to extend it with new features, the user needs to have programming skills and robotic knowledge. To facilitate this job, 2 rqt GUIs were developed.

5.6.1 rqt_ur10e

When working with a robotic arm, there are a group of actions that are constantly being executed by the programmer. The *rqt_ur10e* is a GUI developed for the *iris_ur10e* ROS package that gives the user a set of shortcuts for robot interaction and configuration, and can be seen on Figure 5.9. These actions are specific to the UR10e and the WEISS CRG 200 gripper, and depend on the correct execution of their ROS drivers.

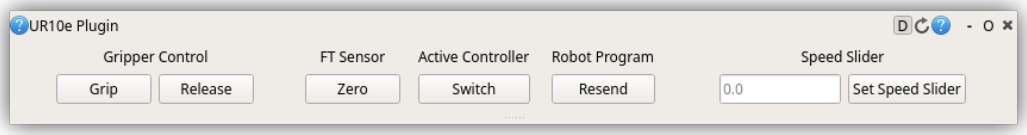


Figure 5.9: ROS rqt_ur10e interface for easy robot control and monitoring

The choice of actions implemented on this GUI were based on personal experience with the robot. They consist on gripper control, which opens or closes the gripper; common service calling on the UR10e ROS driver, with the Zero button which tares the FT sensor values and the Resend button used when the ROS driver looses connection with the robot and a new connection needs to be performed; a Switch button used to interchange between the joint positional controller and the joint speed controller; and a configuration field to control the robot speed slider value.

The standard way of executing these actions would be through the command line or by sending scripts in URScript to the robot. This plugin allows for quicker and cleaner execution of such actions.

5.6.2 rqt_sami

The *iris_sami* ROS package, where *sami* stands for Simple Arm Manipulation Interface, was developed at IRISLab prior to the development of this Dissertation. It provided easy robot motion planning through an abstraction of the MoveIt motion planning library with simple functions like *move_joints()* or *move_pose()*. With these functions, a ROS service server was developed, in which a single instantiation of the MoveIt commander was created and the user could control the robot with ROS service calls through the command line. These functions, contrary to the previous package, are robot agnostic and the only requirement is compatibility with the MoveIt library. Examples of such ROS services include:

- **/iris_sami/status:** Returns status information about the robot arm.
- **/iris_sami/alias:** Sends the robot to the alias position passed as input argument.
- **/iris_sami/save_alias:** Takes as input a string variable and saves the current robot position as an alias position.
- **/iris_sami/move:** Takes XYZ RPY arguments and moves the robot relatively referenced to the EEF frame.
- **/iris_sami/pose:** Takes XYZ RPY arguments and sends the robot to the designated global pose on the world reference frame.
- **/iris_sami/joints:** Takes an array of 6 joints positions and sends the robot to the designated joint space position.
- **/iris_sami/velocity:** Takes an array of 6 joints speeds and a time variable and moves the robot with the designated joint speeds during the designated time.

With these services the user is empowered with complex control on robot motion planning and can easily combine multiple services on shell scripts for high-level task plans. During the development of this Dissertation it was noted that the interface through command line calls is not intuitive and on a practical level could be improved with a GUI. Hence, *rqt_sami* was developed. It consists on another rqt plugin, but this time, implementing the functionality found in the *iris_sami* ROS package. Figure 5.10 demonstrates the interface.

It is horizontally divided in control functionality and status feedback. Starting at the top, there is an alias position selector where the user can send the robots to previously saved positions. Then, the area with arrow buttons allows the user to quickly move the robot in any direction, with both linear movements and rotations. These actions can be performed relative to the EEF frame or the world frame, and the buttons work in a press and hold fashion which means that the robot only moves while the button is being pressed, stopping its motion when it is released. On the right of the arrow buttons the user can call the movement services with text input fields, accordingly labeled to each function. The Send button executes the service and the Reset button clears the fields. On the status feedback section, the first text area shows relevant information about the robot such as its mode of operation, joint positions and

world pose. This information is updated at a rate of 50Hz. Finally on the bottom of the interface there is a text area where the feedback of each command is presented to the user.

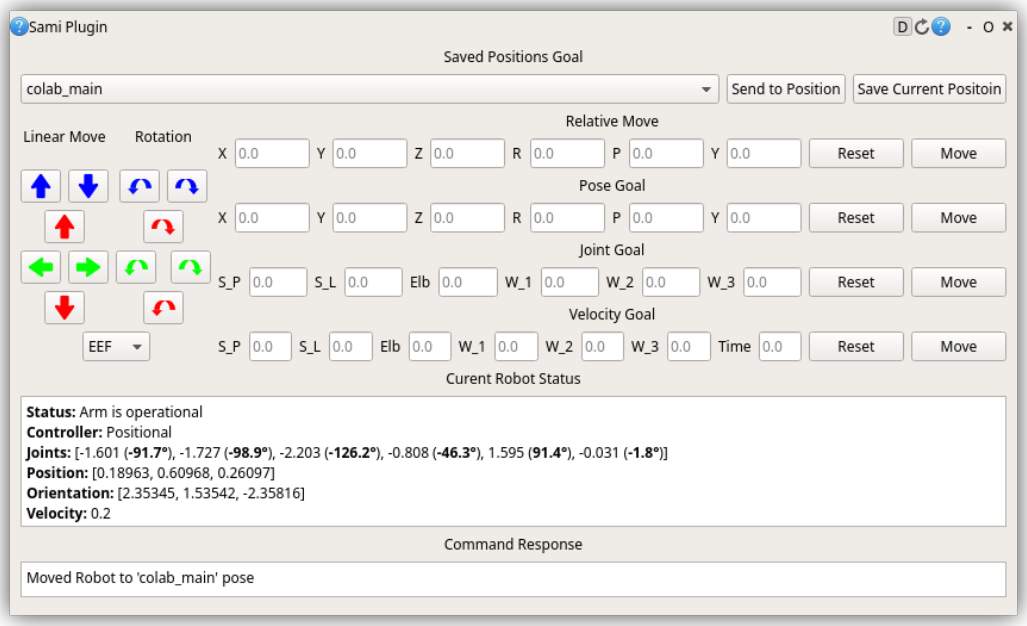


Figure 5.10: ROS rqt_sami interface for easy robot control and monitoring

The development of the tasks proposed in this Dissertation was greatly enhanced by this interface, which is the main purpose it was initially designed for.

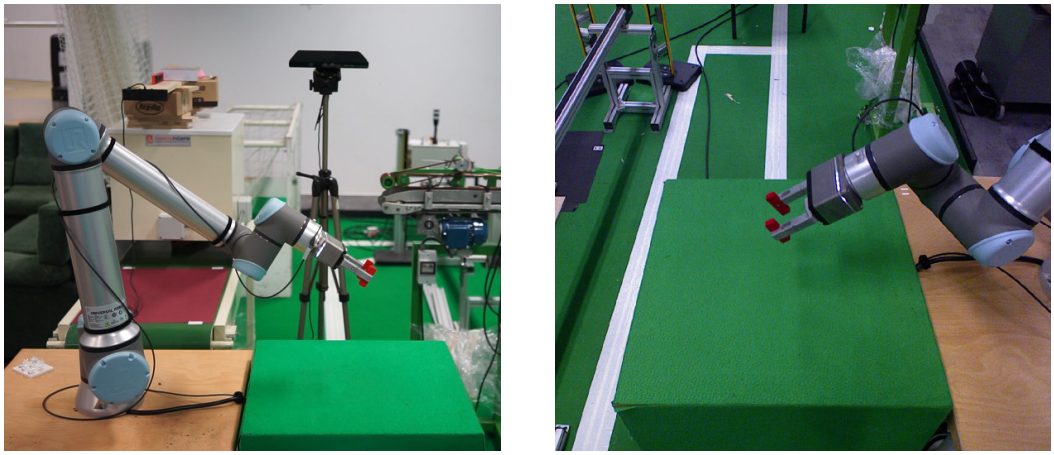
CHAPTER 6

Experiments and Results

Every collaborative task previously described was successfully implemented, but in order to validate its quality, performance, and accuracy, we created and performed a series of experiments. They result in quantitative and qualitative metrics and serve as a benchmark of the techniques and tasks developed. After briefly describing the collaborative setup, we outline each test performed on the collaborative tasks and comment on the results. In the end, we also give some remarks on the performance and stability of the overall system.

6.1 Collaborative Setup

The experimental setup consists on a UR10e, with a Weiss Robotics CRG 200 gripper attachment. The vision sensor is a Microsoft Kinect RGBD camera and the shared workspace is a simple table, approximately 50cm distant from the robot. The camera is positioned at 2m of height with a 45° downwards orientation and its FOV is shown in Figure 6.1b. The complete composition of the setup is shown in Figure 6.1a.



(a) Collaborative Setup **(b)** Kinect FOV

Figure 6.1: Views of the experimental shared workspace

6.2.1 Interaction Test

To validate the use of the EEF double tap as an interaction interface with the cobot, a test case was developed in which the user would make consecutive double taps in all the directions available and observe if they were correctly registered through the colors of the gripper LEDs. Each direction corresponds to a different color. A double tap interaction is considered a fail when the color of the LEDs does not change, or changes to the wrong color.

Double Taps	Fails	Accuracy
300	16	94,6%

Table 6.1: Results of the interaction test

In 300 interactions, 94,5% were correctly registered. The majority of the fails were due to weak execution of the double tap. There was record of a few interactions in which the direction was wrongly classified. The cause of this is due to the ambiguous direction in which the user performed the double tap.

6.2.2 Hand Guiding

The accuracy of the EEF compensation model was already discussed in Subsection 3.2.4. It has been proved that a force threshold of 2N would be possible since it is higher than the maximum error of the FT theoretical model. Despite this fact, it has also been shown in Subsection 3.1.2 that physical interaction with the sensor can cause deviations in its measurements. For this reason, an experimental test was developed to test the accuracy of the compensation model in a real scenario, and the performance of the HG task in general.

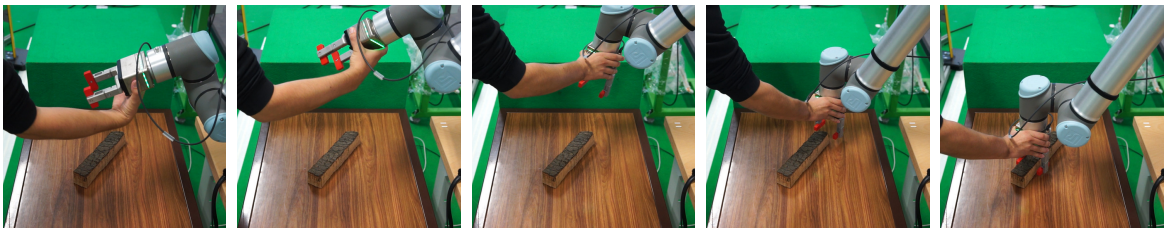


Figure 6.2: Succession of steps of the HG test

The test consists on HG the cobot with the objective of walking the gripper fingers through an object that is placed on the table. The system must compensate the FT caused by the gripper in every orientation, therefore the cobot must only move due to external forces caused by the user. On the other hand, the user must align the gripper fingers with the object without difficulty. To test the system responsiveness, the user must walk the gripper through the object in a linear motion, while the gripper fingers are aligned with it. The succession of steps for this test is shown in Figure 6.2.

The metrics for this test are effectiveness and responsiveness. Effective means that the user was able to complete the test, therefore the gripper weight was always correctly compensated

and the robot only moved according to the force applied by the user. Responsive means that the robot motion was smooth and specifically in the part that the user must align the gripper fingers with the object on the table, while performing the linear movement, the griper fingers never touched the object. The results should be interpreted as number os successes in the amount of attempts. This test was performed 10 times in multiple settings of force thresholds from the compensation model, and the results are outlined in Table 6.2.

Force Threshold	**Test Effectiveness**	**Test Responsiveness**	**Qualitative Performance**
1N	3/10	10/10	Unusable
2N	8/10	10/10	Usable
3N	10/10	10/10	Balanced
4N	10/10	9/10	Satisfactory
5N	10/10	7/10	Unresponsive

Table 6.2: Results of the HG performance test

A closer look at the results and an explanation of the qualitative performance will follow:

- At 1N of force threshold, the system is unusable. The cobot is mostly moving by itself since the force threshold is too low. This result is expected and serves as a demonstration of the consequences of the inaccuracies of the FT sensor.
- At 2N, the accuracy results are improved, but not entirely satisfactory. Due to physical interaction with the sensor, there were certain motions that would cause deviations on the FT measurements. Despite this fact, with this setting the cobot is very responsive, reacting to very low amounts of force and allowing the gripper to be easily placed exactly where the user wants it to be.
- 3N proved to be the best threshold setting. It allowed the correct execution of every test and the loss on responsiveness was not perceptible. With the gripper attachment, 3N will be the default value for force threshold on the EEF weight compensation model.
- 4N and 5N allowed the same accuracy results obtained with 3N but at the cost of motion responsiveness. In short, the user would have to exert higher amounts of force on the EEF for it to move, causing degradation on its smoothness.

The general execution and behavior of the HG task is a success. The goal of this test was not only to find the best system parameters to obtain the best possible behavior, but also to prove that the orchestration of ROS nodes and the algorithms they implement, proposed in this dissertation, is well founded and effective.

6.2.3 Object Manipulation

The object manipulation task requires the dynamic attachment of extra weight to the cobot EEF. It has been shown previously that the EEF weight compensation model supports dynamic changing of the weight and CoG parameters, but the performance of this manipulation task is directly proportional to the accurate measurement of these parameters.

The first test regarding this task should be to find both system and FT sensor performance on correctly measuring the weight of the coupled object. In order to do this, a 1Kg iron

weight will be gripped by the cobot 10 times in 6 different orientations. These orientations consist on vertically aligning each FT axis, in each direction, with gravity, in order to obtain the accuracy of each measurement component, and can be seen in Figure 6.3.

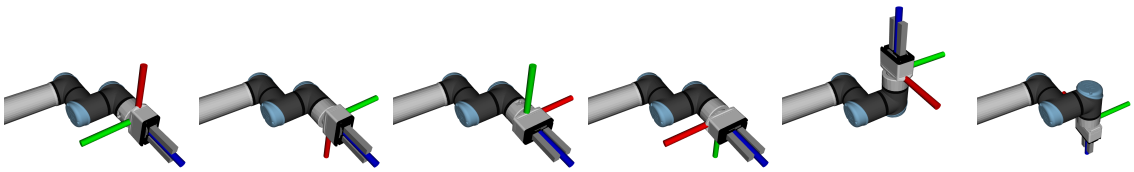


Figure 6.3: 6 different EEF weight measurement poses

The raw results are outlined in Figure 6.4. As it is evidently seen, changing the measuring axis has a significant effect on the reported measured weight.

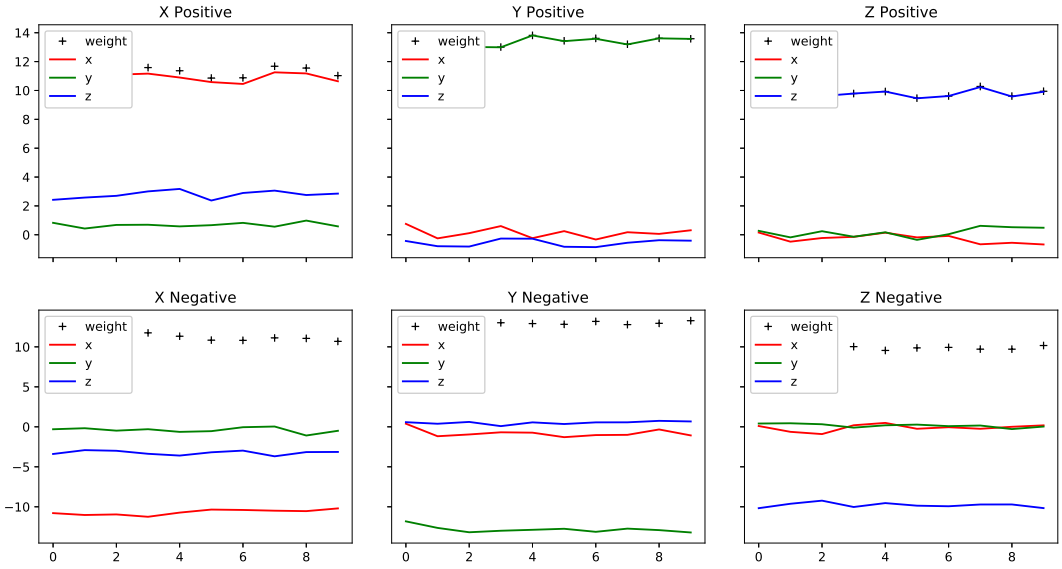


Figure 6.4: Results of the weight measurement test in each position

Table 6.3 gives a closer look at the some statistics from the weight measurement tests. All values are to be read as weight in Kg.

Weight Test	**Raw**	**DLS Correction**	**Scalar Correction**
X Positive	1.144	1.210	1.001
X Negative	1.138	1.209	1.008
Y Positive	1.369	1.229	1.025
Y Negative	1.311	1.176	0.982
Z Positive	0.992	1.219	1.016
Z Negative	0.999	1.229	1.024
Average	1.159	1.213	1.011
Std Dev	0.146	0.038	0.031

Table 6.3: Detailed results and statistics from the weight measurement tests

At first, the raw values are neither precise nor accurate showing a wrong weight average and an arguably high standard deviation. Applying the correction constants obtained from DLS optimization in Subsection 3.2.4, it is possible to improve the precision of the measurements, with a significantly lower standard deviation, but the final average weight value is still far from the correct. Applying a scalar correction with the value of 1.2 provides an accurate and precise weight measurement, with an improved and much lower standard deviation, meaning it is expected that in real use scenarios, the weight of a coupled object should only deviate, in the maximum, by this value.

Assuming the weight is correctly obtained, a real time test was performed to demonstrate that the EEF weight compensation model can sustain the same levels of accuracy independently of the weight coupled to the robot. Figure 6.5 shows a real time test where Wrist1 and Wrist2 joints are rotated causing multiple EEF orientations. Similar to previous real time tests the EEF weight is compensated with minimal error.

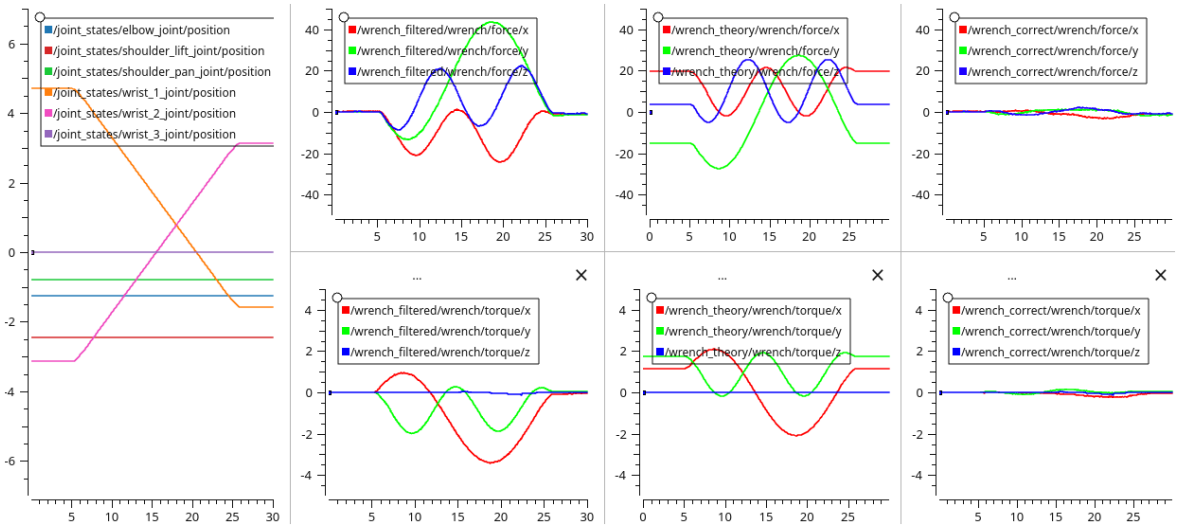


Figure 6.5: Result of the compensation architecture applied in a real time test

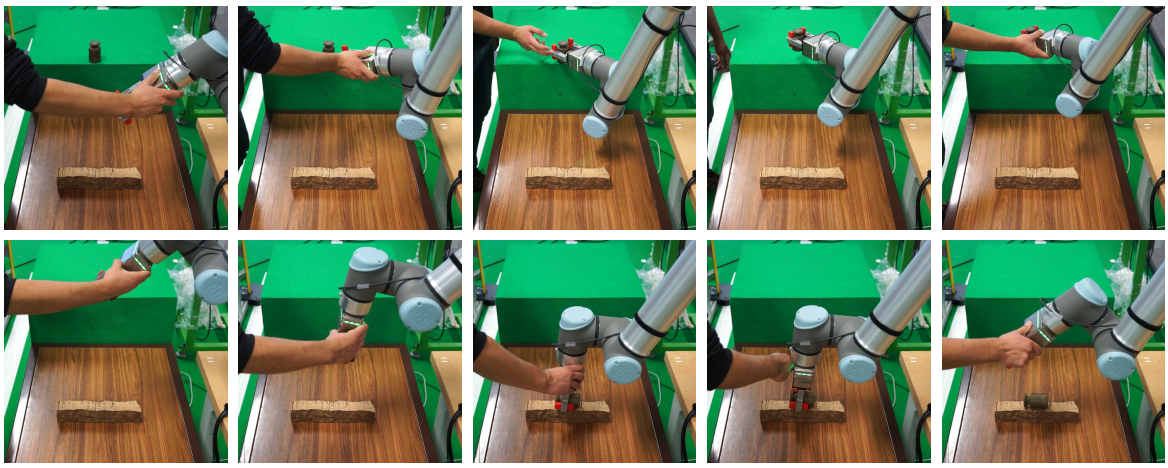


Figure 6.6: Succession of steps of the object manipulation test

For the same reasons as in the HG tests, weight compensation error is not enough to

validate this task. Furthermore, since extra weight is added, it is suspected that the deviations caused by physical interaction with the FT sensor will increase compared to the previous task. This time, the collaborative test consists on HG the cobot, aligning it with a stationary object in the environment, give the order to grip it, wait for the system to calibrate its payload and manipulate the object with both linear and angular movements, in order to release it at another location in the environment. Figure 6.6 shows the succession of steps in this test. The evaluation metrics are the same as in th HG test, and it was also repeated 10 times in each force threshold setting.

Force Threshold	**Test Effectiveness**	**Test Responsiveness**	**Qualitative Performance**
1N	0/10	10/10	Unusable
2N	3/10	10/10	Unusable
3N	7/10	10/10	Usable
4N	10/10	8/10	Satisfactory
5N	10/10	5/10	Unresponsive

Table 6.4: Results of the object manipulation performance test

Table 6.4 presents the results of the test and similarly to the previous task a qualitative interpretation of them will follow:

- Similarly to the HG task, and for the same reasons, 1N of force threshold is unusable.
- At 2N, the results improve but they are still prone to a lot of deviations, therefore not enough.
- 3N show enough improvements on the results to be considered usable, since the majority of tests are completed. Given this fact, in a real manipulation scenario, the objective would be for the robot to never fail, even if it means sacrificing on responsiveness.
- 4N would be the ideal force threshold value for this task since it allows the completion of all the tests without significant sacrifice on motion responsiveness.
- 5N presents results similar to the HG task where all tests are completed at the cost of motion smoothness.

In order for these tests to be successfully completed, the value of force threshold of the weight compensation model needed to be increased. This is caused by 2 factors:

- Increasing the EEF coupled weight increases the inaccuracies of the FT sensor due to interaction with the user, as explained in Chapter 3.
- The generation of force values from the theoretical FT model is directly impacted by the configured value of object weight, which is measured and set in real time, when the robot grabs the object. Inaccuracies on weight measurement due to incorrect handover, or simply due to the arguably weak force accuracy specification value of 5.5N, can directly impact the performance of the theoretical model, whose ultimate goal is to model the behavior of the real FT sensor.

The solution to this problem seems to rely on adapting the force threshold value of the theoretical FT model according to the current payload weight. With a 1.5Kg gripper tool, 3N

of force threshold meant manipulating the tool with a 1:5 force to motion ratio. Gripping a 1Kg iron weight, and increasing the force threshold to 5N allows to maintain the same ratio and be able to successfully and accurately manipulate the conjoint weight. The loss on responsiveness is an acceptable tradeoff for the fact that, in the end, the tests in question were effectively completed.

6.2.4 Collision Free Execution of an Industrial Task

The validity of the collision avoidance architecture has already been shown in Section 4.3, and the execution of an industrial task is a simple succession of predefined steps. To obtain performance metrics on this task and the collision avoidance architecture, a series of tests will be performed, both in simulation and in a real environment, and its results outlined.

Gazebo Simulator Environment

The first test consists on a trajectory between two poses and a sphere obstacle strategically placed on top of the trajectory. Detailed parameters of this test are outlined in Table 6.5, where position is to read as XYZ coordinates in meters, and orientation as XYZ rotation in degrees (°).

Trajectory	**Position**	**Orientation**
Start	(1.0, 0.0, 0.4)	(0, 90, 90)
End	(0.0, 1.0, 0.4)	(0, 90, 90)

Obstacle	**Position**	**Radius**
Sphere	(0.8, 0.8, 0.35)	0.1

Table 6.5: Parameters for the base obstacle avoidance test

This test was created and executed in the Gazebo simulator. The offline trajectory was obtained with the RRT Connect planner, that is implemented on the OMPL. Visual representation of the environment and trajectory is shown in Figure 6.7.

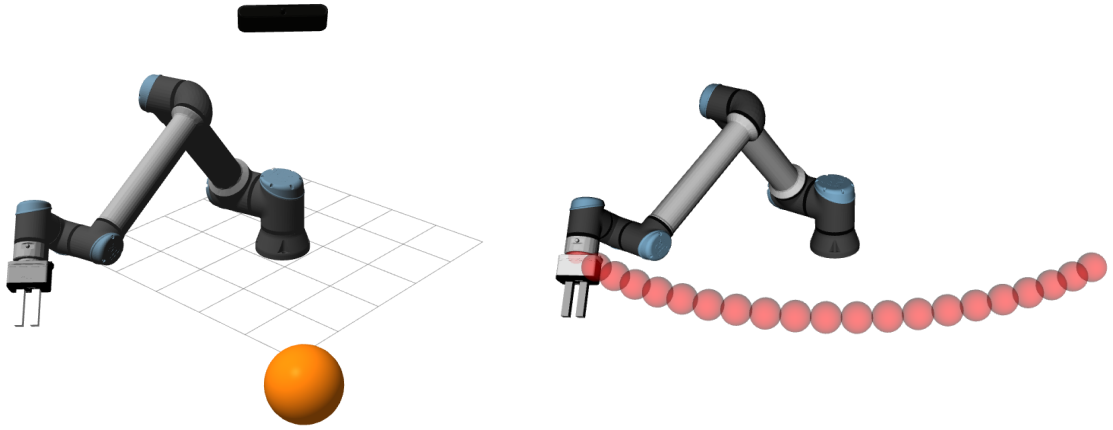


Figure 6.7: Simulation scenario with obstacle and offline generated trajectory

Once the control service on the APF controller is called with the *play* instruction, the robot starts executing the trajectory. Since the obstacle node is not detecting any obstacles, the motion vector is equal to the attraction vector, therefore the executed trajectory is the same as the predefined one. When the robot approaches the obstacle, and the maximum

distance constant is reached, a repulsion vector is generated making the final motion vector steer the robot away from the obstacle. Once the robot is enough distant from the obstacle and the repulsion vector disappears, it resumes the original trajectory. Figure 6.8 shows this behavior, where the original and adjusted trajectories are plotted with the sphere obstacle.

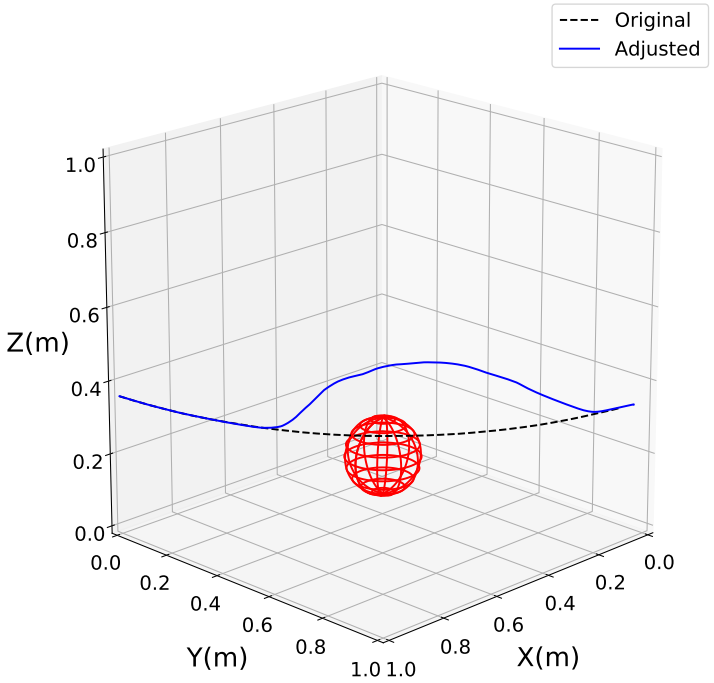


Figure 6.8: Execution of the trajectory with and without obstacle

Having performed the base test with one obstacle, a set of tests with randomly placed and varying number of obstacles was performed in order to test the robustness of the collision avoidance architecture. The number of obstacles ranged from 0 to 5 and they were randomly placed in an area defined by the parameters in Table 6.6.

Axis	**Minimum** (m)	**Maximum** (m)
X	0.5	1.0
Y	0.5	1.0
Z	0.3	0.5

Table 6.6: Parameters for the random placement of obstacles

Examples of the disposition of the randomly placed obstacles are shown in Figure 6.9.

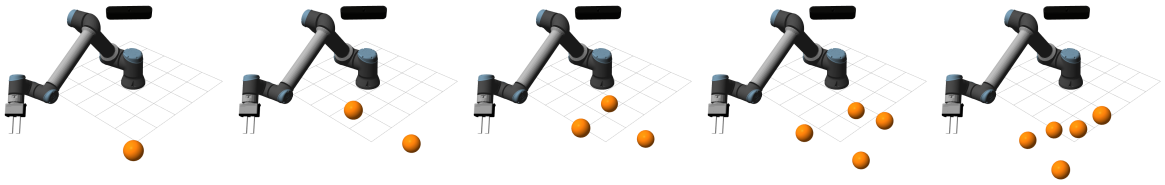


Figure 6.9: Simulation scenarios with multiple obstacles

The test consists on executing the trajectory of the previous test 10 times in each randomly

generated environment. The ability to complete the trajectory without colliding with any obstacle, and the time it took was recorded. These results are outlined in Table 6.7.

Obstacles	**Completed**	**Average Time** (s)
0	10/10	11.43
1	9/10	19.29
2	7/10	20.44
3	6/10	20.55
4	2/10	22.32
5	0/10	—

Table 6.7: Results of the obstacle avoidance test

The execution of the trajectory without obstacles in the environment is always successful, and the time is takes is proportional to the velocity configured in the robot controller. Once obstacles are introduced, the success rate decreases and the time it takes to reach the goal pose increases. For a reasonable amount of obstacles (1 or 2) the success rate is acceptable, but for an higher amount, the robot easily colides with obstacles and rarely completes the trajectory. This happens for a number of reasons:

- The current implementation of the repulsion vector only takes into account one obstacle at a time, which is the closest, therefore, the repulsion caused by an obstacle can lead the robot to colide with another obstacle.
- Due to the previous fact, the robot can also get stuck between 2 obstacles that are equally distant from it, which despite not generating a collision, it will prevent the robot to reach its goal pose.
- The testing conditions are not realistic, and the placement of more than 3 obstacles in a small area such as the one previously described is an exaggeration created to test the limits of the architecture.

To solve these issues, the collision avoidance architecture should be improved by taking into account multiple obstacles simultaneously, and calculating their distance to multiple parts of the robot, and not just to the EEF. A possible solution to the local minimum problem, would be the replanning of the offline trajectory taking into account the obstacles identified. Due to the nature of the goals proposed for this Dissertation, these endeavours were not implemented but rather classified as future work.

Real Environment

Regarding the implementation of this task in a real scenario, the executed tests and recorded performance metrics were not the same as the simulation tests. Instead of testing the ability to adapt the trajectory in a cluttered environment, focus was given to the relation between robot speed and minimum collision distance. Reasons for this choice rely on the fact that arranging various obstacles in a real scenario is more time consuming than in the Gazebo simulator. On the other hand, the joint speed control in the simulator is not possible since, currently, the UR10e Gazebo implementation only supports joint position based control.

Therefore, since the UR10e Polyscope interface allows velocity control through a speed slider variable, it is possible to correlate the speed of the robot EEF to the effectiveness of the collision avoidance architecture.

To achieve these metrics, a similar test was performed were a table served as a shared workspace, and the robot would execute a trajectory between its extremes. On the first test, there were no obstacles between the 2 points, but on the following tests an orange cone is placed in a position that would intentionally colide with the offline trajectory. Figure 6.10 illustrates both the execution of the trajectory without obstacles and with an orange cone directly interfering with the trajectory.

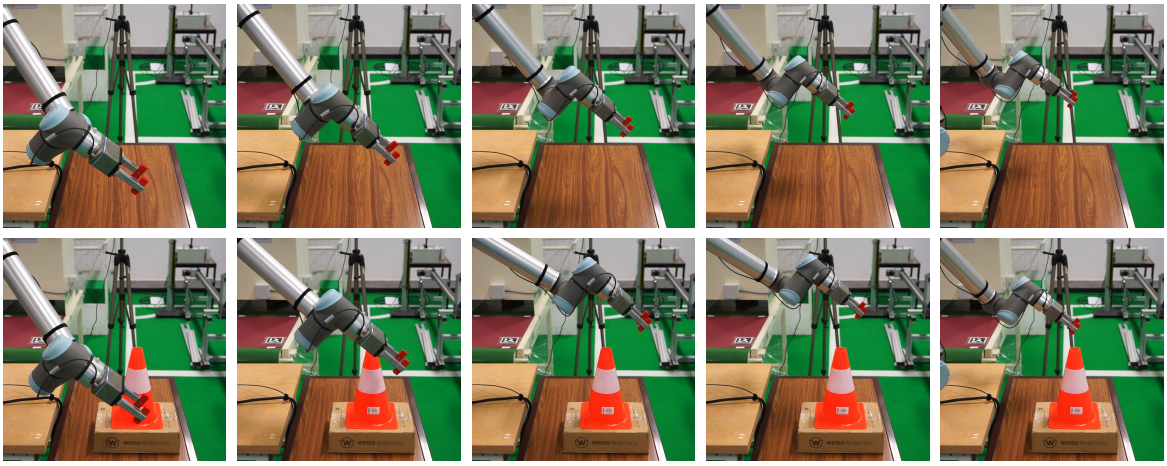


Figure 6.10: Succession of steps of the collision avoidance test

As observed, the robot is able to avoid the obstacle, complete its trajectory and keep a safe distance from the obstacle, preventing a collision with any part of the robot. This base test was performed with a speed slider of 10%. To observe how the robot speed impacts its behavior when avoiding obstacles, this test was repeated in multiple speed slider settings. Figure 6.11 demonstrates the trajectory of the EEF when performing the aforementioned tests, and it also denotes in space the points that were identified as obstacles.

When increasing the speed of the robot, the point in space where the robot starts adapting its trajectory gets closer to the identified obstacles. The maximum speed slider value was 80% due to the fact that an higher values causes a collision. Table 6.8 shows in each speed setting the minimum distance that the robot EEF was in relation to the identified obstacle at that time. The table also shows the conversion of speed slider to real EEF speed in m/s.

The configured parameter, on the collision avoidance architecture, by which a point is considered to be inside the ROI, for the execution of theses tests, was 0.3m. The results obtained are as expected since when the robot moves with a relatively low speed, the distance it keeps from the obstacle is constant and high enough to prevent collisions, throughout the execution of the trajectory. But as the speed of the robot increases, the minimum distance starts to decrease, reaching a point where this implementation is not fast enough to make the robot avoid the obstacle. This behavior happens for a couple of reasons:

- The distance between the EEF and the tip of the gripper tool is exactly 0.18m, therefore,

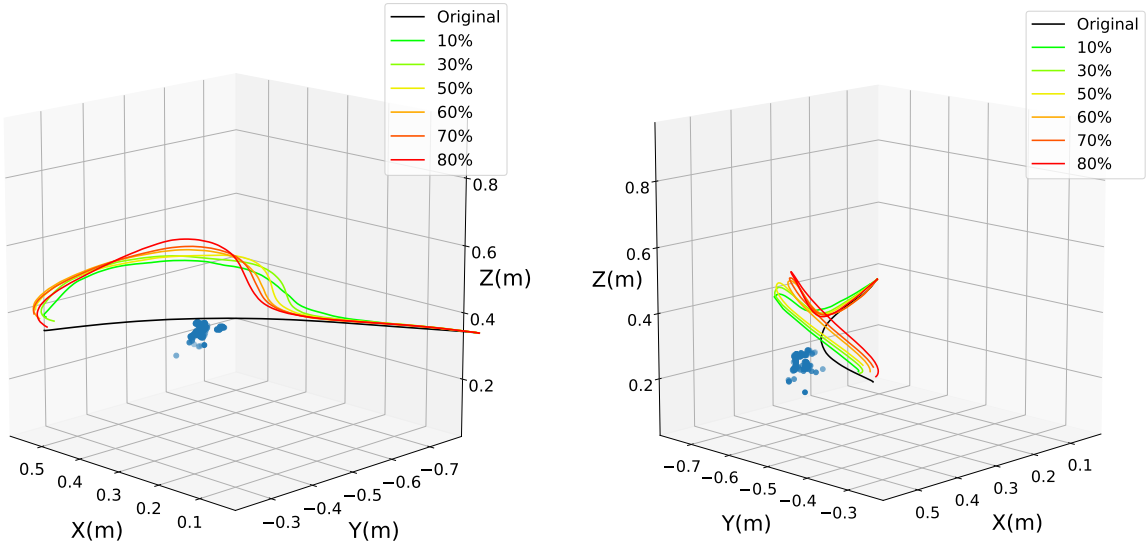


Figure 6.11: Collision avoidance in real scenario with multiple speed settings

Speed Slider (%)	**EEF Speed** (m/s)	**Minimum Distance** (m)
10	0.054	0.217
30	0.155	0.211
50	0.234	0.198
60	0.282	0.194
70	0.316	0.191
80	0.359	0.183
90	0.406	Collision
100	0,449	Collision

Table 6.8: Minimum EEF distance to obstacle relative to its speed

according to Table 6.8, when the speed slider is higher than 80%, the robot colides with the obstacle because it cannot maintain a minimum distance higher than 0.18m.

- The rate of identification of obstacles is limited, due to the fact that the Microsoft Kinect camera can only publish point clouds at a maximum rate of 30Hz. The fact that this architecture and tests were executed on a laptop also does not help performance.
- As demonstrated in Chapter 4, when identifying obstacles and calculating the repulsion vector, only the EEF position is used.

A simple fix to obtain better results in this test is increasing the ROI minimum distance constant, but in theory, 0.3m should be enough to avoid collisions at the EEF level, therefore, the performance of the robot when avoiding obstacles can be improved in various other aspects. The collision avoidance architecture should use more points to identify obstacles and calculate the repulsion vector, than using only the EEF position. Examples are the tip of the gripper tool and the wrist links. The use of a faster camera with an higher sample frequency should also be considered, since, by industry standards, 30Hz is not enough to detect dynamic obstacles. Even though the results can be improved with the aforementioned solutions, this implementation is considered successful in what it proposes, which is the execution of an

offline trajectory with realtime dynamic collision avoidance.

6.3 System Stability

A more general performance metric is the ability for the system to maintain activity for long periods of time. There are several factors that prevent this from happen. During the execution of the previous tests and any other time directly interfacing with the robot, there were some events that injured the overall experience.

For instance, at random instances, when the robot had been correctly functioning for a certain amount of time, e.g. 20 minutes, the system will lose connection with the robot. More specifically, the UR ROS driver looses connection to the RTDE interface, or in some instances, the client program that is running on the robot, requesting commands to the driver suddenly stops its execution. The cause of this problem was not studied since there was no observable pattern on the instances that is happened. The solution was to call the *\resend_robot_program* ROS service made available by the driver and the connection would resume.

Another example is the fact that, because we are dealing with a cobot and the joints are force sensitive, when applying large amounts of force on the EEF, the cobot will make a protective stop, giving the user a violation alert, explaining that a force or speed limit was exceeded. Other examples of protective stops are prone to happen when the user couples high amounts of weight to the EEF. This happens because this weight is not reported to the UR10e Polyscope interface. It is obvious that the UR10e internal controllers need to know the correct value of payload in order to correctly calculate the necessary torques to apply in each joint, but as was explained in Subsection 3.2.1, not reporting the payload parameters to the UR10e internal system is a necessary tradeoff for obtaining cohesive FT measurements on the EEF.

A final remark on system performance gives light to the fact that all components of this system were tested and implemented on a laptop with an Intel Core i7-8550U and 16Gb of RAM.

CHAPTER 7

Conclusion

We conclude this Dissertation with an overview of the work that was developed in order to achieve the proposed goals. We also give light to certain aspects that can be extended and improved.

7.1 Overview

This Dissertation proposed a set of tools and techniques that when joined together, promoted the creation and execution of collaborative tasks between a robotic manipulator and a human. The initial techniques proposed consisted on interacting with the robot through touch, converting the FT measured at the EEF into robot motion, compensate extra weight coupled to the EEF, and detect moving obstacles in the environment. With these techniques, a set of tasks was proposed that included precise HG of the robot, transfer of tools between the robot and the human, manipulation of heavy objects, and collision free execution of an industrial task.

These tasks heavily relied on the existence of FT sensing capabilities on the robot, so we started by exploring the capabilities and behavior of the FT sensor located on its EEF. Some unexpected behavior was found, such as the unreasonable variation of FT caused by the position of the last joint. This behavior and other peculiarities were properly corrected. Then, to be able to have multiple tools and objects attached to the EEF, and still be able to precisely HG it, a payload compensation model was developed based on the optimization of an analytical FT generation model. The combination of the active correction of FT measurements based on the Wrist3 Joint, and the compensation based on the measured payload, coupled to the robot, allowed the user to precisely HG the robot with multiple tool configurations and objects.

To increase the collaborativeness and safety of the system, an external vision sensor was added to the workspace in order to give the robot obstacle avoidance capabilities. To achieve this, a motion controller based on the APF method was developed. In this approach, an attraction vector was generated, that made the robot follow a predefined offline trajectory.

Then, a repulsion vector was created from the real time identification of obstacles in the environment. The combination of these 2 components allowed the robot to adapt its motion to the existence of dynamic obstacles, and still be able to complete its trajectory.

With this abilities, a group of collaborative tasks was developed and seamlessly combined in a state machine. The tasks included transfer of objects between the human and the robot, precisely HG the robot at the EEF level, dynamically couple an object and manipulate it with the same motion as in the previous task, and the execution of a predefined industrial task with the ability of avoiding dynamic obstacles.

To test and validate the performance of the proposed tasks, a number of experiments were designed and implemented in a real scenario with a UR10e cobot. The tests consisted on the execution of the various tasks with different system parameters. Some metrics were registered based on the ability to complete the tasks and the qualitative performance of the system, while doing so. The system showed correct behavior in every task and the results showed that the best possible behavior relies on proper system parametrization.

During the development of this Dissertation, there were some challenges not initially forseen when drawing its requirements. For starters, the amount of inaccuracies in the FT sensor was not expected, and for some time, it was uncertain how much the collaborative tasks could rely on it. It was only until the right tests were made and the error patterns discovered, that a possible correction was designed and implemented. Another struggle came from the fact that these inaccuracies could only be tested with the real FT sensor. Regarding the control of the UR10e, the journey to find the best method for joint velocity based control of the robot was also long and uncertain. For instance, the *iris_ur10e* ROS package was used in the initial periods of development. It consisted on a fork of the official UR ROS driver with some adicional features, such as the inclusion of the Weiss Gripper driver and description. But because this package forked an old version of the driver, joint velocity based control was not available, which raised the question of which control interface to use, given the options outlined in Subsection 2.3.2. Ultimately, efforts were made to merge the newest features of the official UR ROS driver with the implementation characteristics of the setup at IRISLab.

Nevertheless, every proposed technique was effectively implemented, and every goal successfully achieved. The final collaboration framework allows the safe execution of predefined tasks while seamlessly interfacing with the robot in many different ways. This work can be used as a whole for diverse applications, but also contributes in specific topics on the field of HRC with the development of the payload compensation model, the APF collision avoidance controller, the collaborative state machine, and the generic GUIs for robot monitoring and control. Finally, regarding these achievements and contributions, most importantly, this work helps breaking the barriers that are currently separating humans and industrial robots.

7.2 Future Work

Despite every proposed goal being accomplished, HRC is a vast field and this Dissertation only covered a small percentage of its intricacies, therefore, space for added functionality and

improved performance is plenty. In no particular order, a few key aspects in which this work can be expanded will be outlined:

- **Full Body HG:** The flexibility of manipulating an industrial robot with physical interaction can be enhanced if extended to every single joint. In this work, the HG of the robot was limited to the FT measured at the EEF. There are scenarios where being able to HG the entire robot is very useful, such as when the movement of a single joint is desired, or when the robot is reaching a singularity.
- **Extended Sensor Apparatus:** Having a single vision sensor is only sufficient for a proof of concept collision avoidance model, but is not enough for a real industry scenario. Increasing the number of sensors, playing with its dispositions and testing multiple configuration seems a natural evolution on the work developed. An interesting ideia would be to implement a vision sensor in the structure of the robot.
- **Multimodal Interaction:** The proposed work provides limited ways of conveying actions to the robot. There are multiple methods that can be added to this system, in order to increase the means of communication with the robot such as voice recognition, gesture recognition, high level task interpretation, or simply increasing the number of inputs of the system with hardware buttons and switches.
- **High Level Creation of Tasks:** The *ros_smach* library allows the creation of complex robot behavior, describing it as structured state machines. To extend the existing state machine with new functionalities, knowledge in Python programming is required. It would be useful if it was possible to create new tasks without programming knowledge. An intuitive GUI could be built for this purpose, with a drag and drop interface and a list of robot skills that the user could choose from.
- **Offline Trajectory Refactoring:** In the implemented approach to collision avoidance, the offline trajectory is static throughout its execution. There are scenarios with an high amount of obstacles where the robot is not able to complete the trajectory. This behavior could be avoided by making local changes to the trajectory or, in extreme cases, by globally replanning the trajectory, but this time, including the identified obstacles in the planning environment.

References

[1] I. F. of Robotics, "World robotis 2020 report", 2020. [Online]. Available: https://ifr.org/worldrobotics/.

[2] J. Krüger, T. Lien, and A. Verl, "Cooperation of human and machines in assembly lines", *CIRP Annals*, vol. 58, no. 2, pp. 628–646, 2009. DOI: 10.1016/j.cirp.2009.09.009.

[3] J. Colgate, W. Wannasuphoprasit, and M. Peshkin, "Cobots: Robots for collaboration with human operators", *ASME International Mechanical Engineering Congress and Exposition*, pp. 433–439, Dec. 1996.

[4] J. Schmidtler, V. Knott, C. Hölzel, and K. Bengler, "Human centered assistance applications for the working environment of the future", *Occupational Ergonomics*, vol. 12, no. 3, C. M. Schlick and J. Bützler, Eds., pp. 83–95, Sep. 2015. DOI: 10.3233/oer-150226.

[5] H. C. Fang, S. K. Ong, and A. Y. C. Nee, "A novel augmented reality-based interface for robot path planning", *International Journal on Interactive Design and Manufacturing (IJIDeM)*, vol. 8, no. 1, pp. 33–42, Aug. 2013. DOI: 10.1007/s12008-013-0191-2.

[6] A. Hentout, M. Aouache, A. Maoudj, and I. Akli, "Human–robot interaction in industrial collaborative robotics: A literature review of the decade 2008–2017", *Advanced Robotics*, vol. 33, no. 15-16, pp. 764–799, Jul. 2019. DOI: 10.1080/01691864.2019.1636714.

[7] E. Matheson, R. Minto, E. G. G. Zampieri, M. Faccio, and G. Rosati, "Human–robot collaboration in manufacturing applications: A review", *Robotics*, vol. 8, no. 4, p. 100, Dec. 2019. DOI: 10.3390/robotics8040100.

[8] B. Siciliano and O. Khatib, Eds., *Springer Handbook of Robotics*. Springer International Publishing, 2016. DOI: 10.1007/978-3-319-32552-1.

[9] F. Ferland, A. Reveleau, F. Leconte, D. Létourneau, and F. Michaud, "Coordination mechanism for integrated design of human-robot interaction scenarios", *Paladyn, Journal of Behavioral Robotics*, vol. 8, no. 1, pp. 100–111, Dec. 2017. DOI: 10.1515/pjbr-2017-0006.

[10] G. Hirzinger, A. Albu-Schaffer, M. Hahnle, I. Schaefer, and N. Sporer, "On a new generation of torque controlled light-weight robots", in *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation*, vol. 4, 2001, pp. 3356–3363. DOI: 10.1109/ROBOT.2001.933136.

[11] Y. She, H.-J. Su, D. Meng, S. Song, and J. Wang, "Design and modeling of a compliant link for inherently safe corobots", *Journal of Mechanisms and Robotics*, vol. 10, no. 1, Dec. 2017. DOI: 10.1115/1.4038530.

[12] R. Weitschat, J. Vogel, S. Lantermann, and H. Hoppner, "End-effector airbags to accelerate human-robot collaboration", in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, May 2017. DOI: 10.1109/icra.2017.7989262.

[13] "Robots and robotic devices — safety requirements for industrial robots — part 1: Robots", International Organization for Standardization, Geneva, CH, Standard, Jul. 2011. [Online]. Available: https://www.iso.org/standard/51330.html.

[14] "Robots and robotic devices — safety requirements for industrial robots — part 2: Robot systems and integration", International Organization for Standardization, Geneva, CH, Standard, Jul. 2011. [Online]. Available: https://www.iso.org/standard/41571.html.

[15] "Robots and robotic devices — collaborative robots", International Organization for Standardization, Geneva, CH, Standard, Feb. 2016. [Online]. Available: https://www.iso.org/standard/62996.html.

[16] F. Steinmetz and R. Weitschat, "Skill parametrization approaches and skill architecture for human-robot interaction", in *2016 IEEE International Conference on Automation Science and Engineering (CASE)*, IEEE, Aug. 2016. DOI: 10.1109/coase.2016.7743419.

[17] H. Fang, S. K. Ong, and A. Y.-C. Nee, "Robot programming using augmented reality", 2009. DOI: 10.1109/cw.2009.14.

[18] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng, "Ros: An open-source robot operating system", *ICRA Workshop on Open Source Software*, vol. 3, Jan. 2009.

[19] U. Robots, *Universal robots e-series user manual*, English, version 5.10, Universal Robots, Jun. 2021.

[20] ——, *The urscript programming language*, English, version 5.10, Universal Robots, Jun. 2021.

[21] T. Andersen, *Optimizing the Universal Robots ROS driver*. English. Technical University of Denmark, Department of Electrical Engineering, 2015.

[22] D. Coleman, I. A. Sucan, S. Chitta, and N. Correll, "Reducing the barrier to entry of complex robotic software: A moveit! case study", *Journal of Software Engineering for Robotics*, vol. 5, pp. 3–16, May 2014. DOI: 10.6092/JOSER_2014_05_01_p3.

[23] I. A. Sucan and S. Chitta, "Moveit", [Online]. Available: https://moveit.ros.org/.

[24] R. B. Rusu and S. Cousins, "3d is here: Point cloud library (PCL)", in *2011 IEEE International Conference on Robotics and Automation*, IEEE, May 2011. DOI: 10.1109/icra.2011.5980567.

[25] R. Tsai and R. Lenz, "A new technique for fully autonomous and efficient 3d robotics hand/eye calibration", *IEEE Transactions on Robotics and Automation*, vol. 5, no. 3, pp. 345–358, Jun. 1989. DOI: 10.1109/70.34770.

[26] S. Brown and H. A. Pierson, "A collaborative framework for robotic task specification", *Procedia Manufacturing*, vol. 17, pp. 270–277, 2018. DOI: 10.1016/j.promfg.2018.10.046.

[27] R. Arrais, C. M. Costa, P. Ribeiro, L. F. Rocha, M. Silva, and G. Veiga, "On the development of a collaborative robotic system for industrial coating cells", *The International Journal of Advanced Manufacturing Technology*, vol. 115, no. 3, pp. 853–871, Oct. 2020. DOI: 10.1007/s00170-020-06167-z.

[28] P. Tsarouchi, S. Makris, G. Michalos, A.-S. Matthaiakis, X. Chatzigeorgiou, A. Athanasatos, M. Stefos, P. Aivaliotis, and G. Chryssolouris, "ROS based coordination of human robot cooperative assembly tasks-an industrial case study", *Procedia CIRP*, vol. 37, pp. 254–259, 2015. DOI: 10.1016/j.procir.2015.08.045.

[29] C. Gaz, E. Magrini, and A. D. Luca, "A model-based residual approach for human-robot collaboration during manual polishing operations", *Mechatronics*, vol. 55, pp. 234–247, Nov. 2018. DOI: 10.1016/j.mechatronics.2018.02.014.

[30] P. J. Koch, M. K. van Amstel, P. Dębska, M. A. Thormann, A. J. Tetzlaff, S. Bøgh, and D. Chrysostomou, "A skill-based robot co-worker for industrial maintenance tasks", *Procedia Manufacturing*, vol. 11, pp. 83–90, 2017. DOI: 10.1016/j.promfg.2017.07.141.

[31] P. Tsarouchi, A.-S. Matthaiakis, S. Makris, and G. Chryssolouris, "On a human-robot collaboration in an assembly cell", *International Journal of Computer Integrated Manufacturing*, vol. 30, no. 6, pp. 580–589, May 2016. DOI: 10.1080/0951192x.2016.1187297.

[32] C. Paxton, A. Hundt, F. Jonathan, K. Guerin, and G. D. Hager, "CoSTAR: Instructing collaborative robots with behavior trees and vision", in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, May 2017. DOI: 10.1109/icra.2017.7989070.

[33] B. Mocan, M. Fulea, and S. Brad, "Designing a multimodal human-robot interaction interface for an industrial robot", in *Advances in Intelligent Systems and Computing*, Springer International Publishing, Aug. 2015, pp. 255–263. DOI: 10.1007/978-3-319-21290-6_26.

[34] M. Safeea, R. Bearee, and P. Neto, "End-effector precise hand-guiding for collaborative robots", in *ROBOT 2017: Third Iberian Robotics Conference*, Springer International Publishing, Dec. 2017, pp. 595–605. DOI: 10.1007/978-3-319-70836-2_49.

[35] C. T. Landi, F. Ferraguti, C. Secchi, and C. Fantuzzi, "Tool compensation in walk-through programming for admittance-controlled robots", in *IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society*, IEEE, Oct. 2016. DOI: 10.1109/iecon.2016.7793038.

[36] M. Safeea, P. Neto, and R. Bearee, "On-line collision avoidance for collaborative robot manipulators by adjusting off-line generated paths: An industrial use case", *Robotics and Autonomous Systems*, vol. 119, pp. 278–288, Sep. 2019. DOI: 10.1016/j.robot.2019.07.013.

[37] X. Wang, C. Yang, Z. Ju, H. Ma, and M. Fu, "Robot manipulator self-identification for surrounding obstacle detection", *Multimedia Tools and Applications*, vol. 76, no. 5, pp. 6495–6520, Feb. 2016. DOI: 10.1007/s11042-016-3275-8.

[38] M. Safeea and P. Neto, "KUKA sunrise toolbox: Interfacing collaborative robots with MATLAB", *IEEE Robotics & Automation Magazine*, vol. 26, no. 1, pp. 91–96, Mar. 2019. DOI: 10.1109/mra.2018.2877776.

[39] S. Garrido-Jurado, R. Muñoz-Salinas, F. Madrid-Cuevas, and R. Medina-Carnicer, "Generation of fiducial marker dictionaries using mixed integer linear programming", *Pattern Recognition*, vol. 51, pp. 481–491, Mar. 2016. DOI: 10.1016/j.patcog.2015.09.023.