Real Time Human Robot Interactions and Speed Control of a Robotic Arm for Collaborative Operations

Thesis · May 2018

DOI: 10.13140/RG.2.2.28723.53286

1 author:

Nick Van Oosterwyck
University of Antwerp
1 PUBLICATION **0** CITATIONS

SEE PROFILE

UNIVERSITY OF ANTWERP

Academic year 2017-2018

Faculteit Toegepaste Ingenieurswetenschappen

Master's thesis

Real Time Human-Robot Interactions and Speed Control of a Robotic Arm for Collaborative Operations

Nick Van Oosterwyck

Promoters: Dr. Ir. Cosmin Copot

Thesis to obtain the degree of
Master of Science in de industriële wetenschappen:
elektromechanica, afstudeerrichting automatisering
Antwerpen, may 2018

University of Antwerp

Acknowledgements

I would like to thank my supervisor Dr. Ir. Cosmin Copot for introducing me to this research topic and for his guidance during this graduation project. He was always available for support whenever I got stuck or had a question about my research or writing.

I also want to thank Erwin Smet for clarifying the basics of robotics and human-robot collaborations to me and allowing me to work in the robotics lab. Without his clear explanation, I would not have achieved this result.

Furthermore, I want to thank the whole Op3Mech research group, and in particular Seppe Sels, for providing and helping me with the toolbox which allowed me to communicate with the camera.

Finally, I must express my gratitude to my family and girlfriend for providing me with unconditional support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

Nick Van Oosterwyck
Wilrijk, University Of Antwerp
June 6, 2018

Abstract

The past decade has seen collaborative robots (cobots) taking a more important part in industrial environments in order to increase the flexibility, productivity and safety of the production line. Especially with high reconfigurable production systems and low product volume processes, the co-existence of humans and robots can provide an answer for these constantly changing environments.

Without extra sensors cobots will come to an emergency stop when a collision is detected leading into unnecessary stops and wear of the installation. Furthermore, this strategy cannot prevent the cobot from colliding with people in the workspace, adding extra risks to the operation. Guaranteeing the operators' safety will be the biggest issue when deploying these systems in industrial environments.

With the help of vision systems, it is possible to create a real-time adaptive environment where the robot will modify its actions to people entering the collaborative workspace. This project uses a state-of-the art UR10 collaborative robot and a Time-of-Flight Kinect 3D camera in order to create a safe and real-time environment. With the use of human skeleton tracking and object detection, the speed of the robot can be controlled proportionally to the distance to people entering the collaborative workspace. The algorithms are tested in both a virtual and real-life situation with the help of the VREP software and the set-up in the robotics lab.

This paper offers a proof-of-concept to demonstrate that the safety of the installation and co-workers can be increased when adding vision systems for human tracking and object detection. Real-time distance calculation can prevent the cobot from colliding with humans or objects and decreases the number of impacts and collisions. Proportional speed control creates a more adaptive environment leading to safer and more efficient human-robot collaboration. Additionally, the robot could move faster and the payload could be heavier because the robot system would not contact with the human operator, increasing the efficiency as well.

Keywords: Robotics, Vision, Collaborative Robot, Human-Robot Interaction, UR10, Kinect, Time-Of-Flight, Real-Time.

Contents

Chapter 1

Introduction

1.1 Background

The past decade the demand for industrial robots has accelerated considerably due to the ongoing trend toward automation and it is to be expected that this trend will only increase further. Between 2017 and 2020, it is estimated that more than 1.7 million new industrial robots will be installed in factories around the world. Industry 4.0, IoT (Internet of Things), machine learning and AI (Artificial Intelligence) will lead robotics in the coming years. [1]

Although a wide range of tasks can already be performed by robots, some operations still need some human input. The next generation robots will fulfil this need in order to combine the flexibility and problem-solving skills of humans with the strength, endurance and precision of robots. [2]

The increasing interest in these so-called collaborative robots (cobots) has heightened the need to develop new robots that are safe to use outside of their shielded environments. To date, there already exist robots that are equipped with special sensors in order to safely operate with humans. The rising popularity of these cobots is confirmed by the rising number of manufacturers with already 40 companies advertising cobots by mid-2017. [3]

However, it appears that these collaborative robots often have some limitations and are not ready to complete interactive tasks. The robots are programmed to come to an emergency stop when they collide with an obstacle, but often do not have the possibility to perform some predictive collision detection. It becomes apparent that it can be interesting to create an interactive environment, where the robot will have real-time interactions with humans and adapt its actions relative to the position of any approaching object. With the help of vision sensors, it is possible to create such an environment which is positive for the productivity, safety and possibilities of future applications with cobots.

1.2 Goals

The main objective of the project is to investigate and realize real-time interactions between human and robot through controlling the speed of the robot with respect to the distance in order to obtain a safe environment. As a proof-of-concept, a Kinect 3D camera and a Universal Robots UR10 collaborative robot will be used to create an adaptive set-up. All the algorithms are tested

in both a virtual and real-life situation with the aid of the Virtual Robot Experimentation Platform (VREP) software and the set-up in the robotics lab at campus Groenenborger.

The specific goals for this project are:

- Explore the possibilities of human-robot interaction by performing a literature survey.
- Create a kinematic model of the robot which allows the robot to be driven in both joint and cartesian space.
- Achieve a safe (speed) control over the UR10 cobot by setting up a communication between MATLAB and the controller in order to get full access of the robot.
- Process data from Kinect in real-time and create a skeleton model of the human for distance calculations and basic human pose recognition.
- To ensure the different algorithms are safe to use before implementing it on the robot, an available 3D virtual test environment (V-REP) needs to be configured in order to give a visual representation of the set-up and show a correct working of the algorithms.
- Implement a safety rated monitored stop and a speed and separation monitoring on the real set-up in the robotics lab with real-time object tracking as a proof-of-concept.
- Interpret a basic human pose command in order to create a more dynamic environment and add flexibility to the configuration.

1.3 Initial Work

A completely new Universal Robots UR10 cobot [4] was recently installed at the University of Antwerp in order to investigate new developments in human-robot interaction. As there were no previous projects performed on this robot before, no initial work has been executed on it and one part of this project was to work out a design for communication and control of the UR10.

On the other hand, the Op3Mech research group [5] at the University of Antwerp already developed a complete toolbox for communication with various industrial and non-industrial vision sensors. This toolbox was extensively used in this project for connecting the Kinect v2 camera [6] with the MATLAB software. Because of this, the time spend for setting up communication protocols could be reduced and there could be focused more on processing the data received from the camera.

Figure 1.1: Logo of the Op3Mech research group at the University of Antwerp. [5]

1.4 Outline of the thesis

The remainder of this paper is organized in the following way. First, the working principles of the two critical components of this project are explained (the Kinect camera and the UR10 cobot). After explaining the basic concepts, some tests are performed in order to find the most adequate method for object detection and control of the robot. Once this is clarified, Chapter 4 describes how data from both the robot and Kinect is combined in order to use it for distance calculations.

Chapter 5 explains the basics of human-robot collaboration and elucidates the safety aspects. Chapter 6 describes how all the previous chapters were implemented in both the simulator and real set-up and shows the results of the final algorithms. The last chapter describes the conclusions and evaluates the goals that were established in Chapter 1. Also, a few side projects are explained and some future work is suggested.

An overview of how the different chapters are related to each other is presented in Figure 1.2.

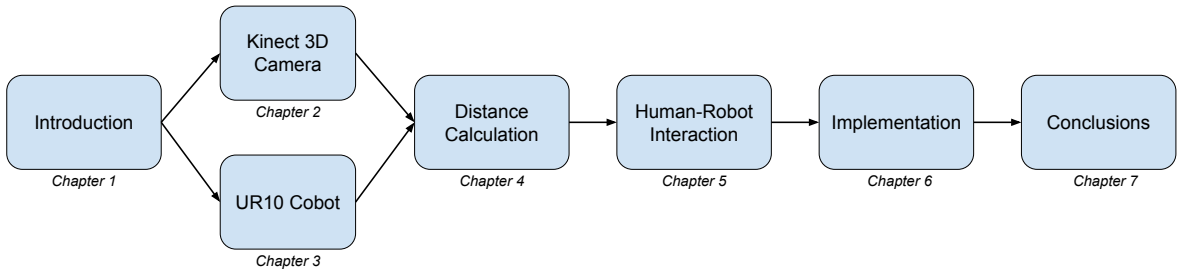


Figure 1.2: Outline of the thesis.

Chapter 2

The 3D Camera: Kinect

As mentioned in section 1.1, a vision sensor for mapping the environment is necessary in order to track any object approaching the robot. This chapter describes the Kinect v2 3D camera and gives a brief summary of the specifications and working principles of the sensor. Furthermore, the conversion to cartesian space is explained and the skeletal tracking is illustrated. Finally, two different approaches for object detection are compared and illustrated.

2.1 Introduction

To date, there are many possibilities for mapping a 3D environment from very expensive and accurate technologies to low-cost devices which are even available for consumers. The Microsoft Kinect is a perfect example of such a low-cost sensor and had a huge impact on recent research in Computer Vision as well as in Robotics. While it was primarily designed for interaction in a computer game environment, the Kinect provides a dense depth estimation together with color images at a high frame rate. [7]

There are two types of Kinect cameras available: Kinect v1 and Kinect v2, with the v2 being the most recent. Contrary to the v1 which uses pattern projection, the Kinect v2 uses Time-of-Flight (ToF) technology in order to calculate the distance. The working principles of ToF are explained in section 2.3. In this project a Kinect v2 sensor is used as it was available in the lab and will probably be the basis for the development in many future research. [8]

(a) A Kinect v1 camera (Pattern recognition) **(b)** A Kinect v2 camera (Time-of-Flight)

Figure 2.1: The different types of Kinect cameras available. [6]

2.2 Specifications

The Kinect sensor is a RGB-D camera, meaning it can capture both depth and its pixel-wise correlated color data. Therefore, the Kinect v2 holds two cameras (a RGB and an infrared (IR) camera), a IR emitter and a microphone bar as shown in figure 2.2. [9] The Kinect software maps the depth data to the coordinate system as shown in figure 2.2, with its origin located at the center of the IR sensor. Note that this is a right-handed coordinate system with the orientation of the y-axis depending on the tilt of the camera and one unit representing one meter. [10]

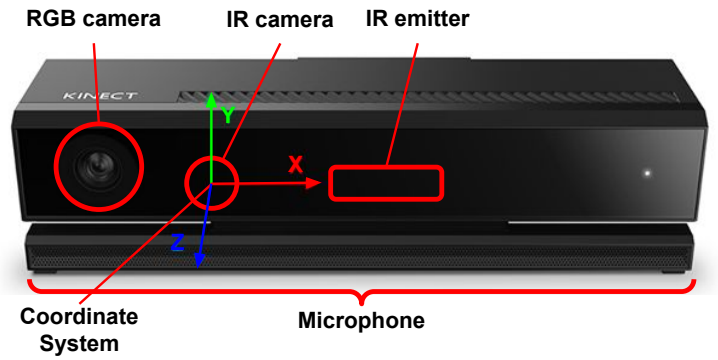


Figure 2.2: Location of the cameras, sensors and coordinate system on the Kinect v2 camera. [10]

A brief summary of the specifications of the Kinect v2 are listed in table 2.1. For example, given the specifications, a pixel size of 5 mm can be obtained at a 3 m range. [11]

Table 2.1: Technical specifications of the Kinect v2 camera [11]

Infrared (IR) camera resolution	512 × 424 pixels
RGB camera resolution	1920 × 1080 pixels
Field of view	70 × 60 degrees
Framerate	30 frames per second
Operative measuring range	from 0.5 to 4.5 m
Object pixel size (GSD)	between 1.4 mm (@ 0.5 m range) and 12 mm (@ 4.5 m range)

2.3 Distance Calculation

As stated before, the Kinect v2 uses optical Time-of-Flight (ToF) technology in order to retrieve the depth data with the infrared (IR) camera. The basic principle of ToF is to measure the time difference between an emitted light ray and its collection after reflection from an object. (Figure 2.3) There are two main technologies used in ToF cameras: pulsed and continuous wave (CW). In the first case, the delay between a sent and received pulse is measured with a fast counter synchronized with the emitted signal. Knowing this time delay, the distance d can be easily calculated with equation 2.1, where c is the speed of light and Δt the measured time difference. [12]

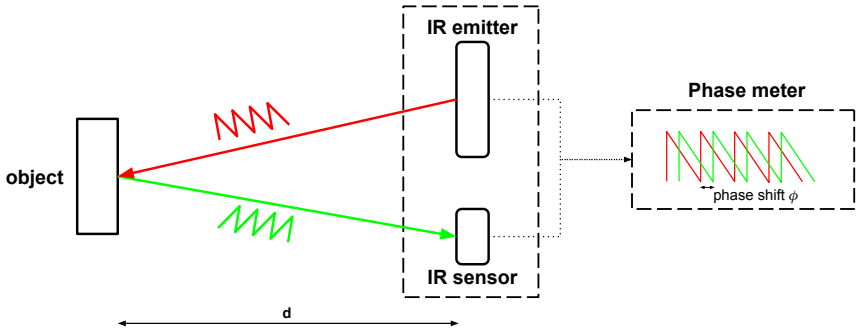


Figure 2.3: CW ToF principle used on the Kinect. The send and received signal are compared to each other in order to determine the phase change and calculate the distance.

$$d = c\frac{\Delta t}{2} \tag{2.1}$$

This method cannot be achieved with silicon components at room temperature since it would require an enormous fast counter, due to the very high speed of light. [13] Therefore, a more commonly used method is the continuous wave (CW) where a modulated light is emitted and compared with the reflected signal in order to determine the phase change ϕ and calculate the distance. The phase meter uses four different samples Q_1 to Q_4 of the reflected signal with a shift of 90 in order to determine the phase change as shown in Figure 2.4. [14] The distance d is then computed with equation 2.2, where f is the frequency of the signal and ϕ is the phase change detected by the phase meter. [12] It should be noted that, unlike other ToF cameras, it is impossible to act on the modulation frequency or integration time of the Kinect v2 sensor. [11]

$$d = \frac{c}{4\pi f}\phi \quad \text{with} \quad \phi = atan(\frac{Q_3 - Q_4}{Q_1 - Q_2}) \tag{2.2}$$

The fact that the CW measurement is based on phase shifts means that there will be an aliasing distance too. The distance where aliasing occurs is called the ambiguity distance d_{amb} and is given by Equation 2.3. [13] The Kinect uses multiple frequencies (approximately 120 MHz, 80 MHz and 16 MHz) so that this ambiguity effect is eliminated. [15]

$$d_{amb} = \frac{c}{2f} \tag{2.3}$$

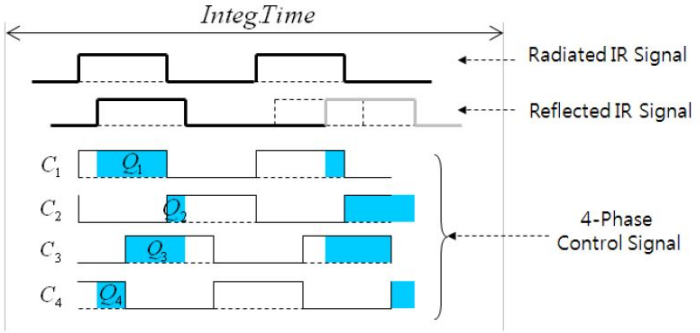


Figure 2.4: Four samples with a shift of 90 are used in order to determine the phase change ϕ. The quantities Q_1 to Q_4 represent the amount of electric charge per control signal. [14]

2.4 Conversion to Cartesian Space

The Kinect outputs the depth data in cartesian space, meaning that every point is expressed in cartesian coordinates relative to the camera coordinate system as shown in Figure 2.2. Simply using the distance d from Equations 2.1 and 2.2 as the Z coordinate is not adequate since the distance is measured along rays from the point to the center of projection. (Figure 2.5) Therefore, a pinhole camera model is presented in Figure 2.5 in order to derive a formula for the Z coordinate. This model assumes the principal point coincides with the center of the image plane. Based on simple trigonometry, the following equation is presented, where d is the distance measured according to Section 2.3, f the focal length, l the distance from the center of projection to the point on the image plane and (x,y) the coordinates on the camera image plane. [16]

$$Z = d\frac{f}{l} = d\frac{f}{\sqrt{f^2 + x^2 + y^2}} \tag{2.4}$$

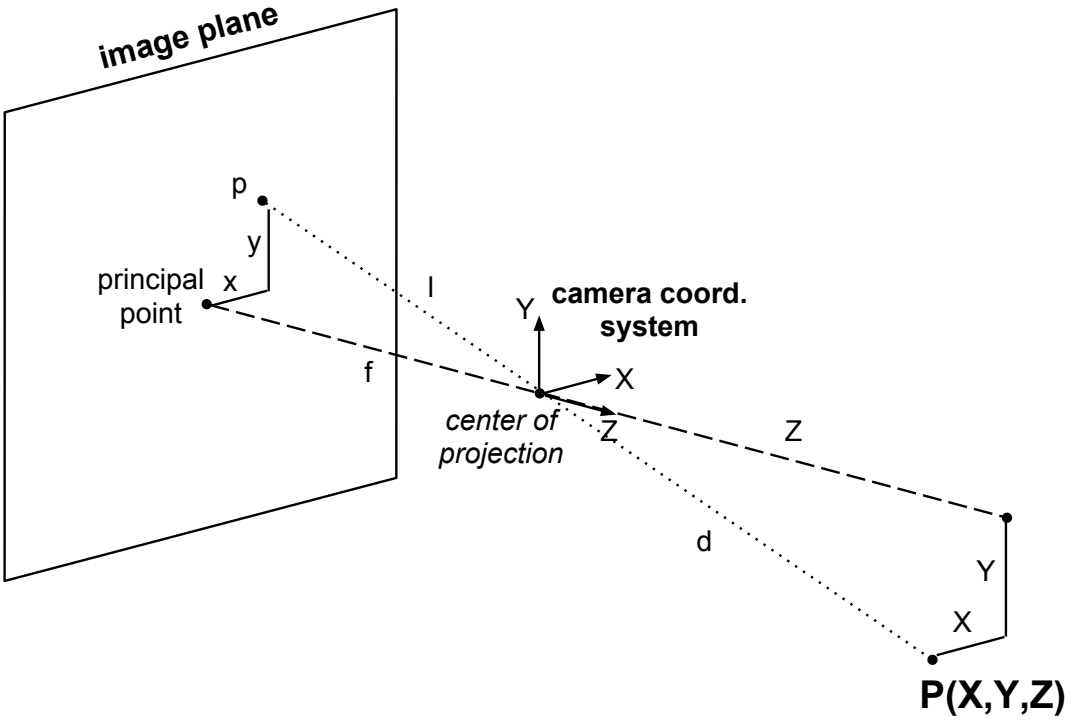


Figure 2.5: Pinhole camera model with a point P(X,Y,Z) according to the camera coordinate system and a point p(x,y) as the corresponding point on the image plane.

When the Z coordinate is known, the Kinect Software Development Kit (SDK) offers a table of (γ_u, γ_v) values for every pixel (u, v) such that the X and Y coordinate can be calculated with Equations 2.5 and 2.6. More information on how these factors are determined can be found in [16].

$$X = \gamma_u \cdot Z \tag{2.5}$$

$$Y = \gamma_v \cdot Z \tag{2.6}$$

2.5 Communication

The calculations and communication with the Kinect camera are executed in MATLAB, since the camera toolboxes of the Op3Mech research group at the University of Antwerp are developed in MATLAB too. There are also several built-in functions available in MATLAB for transforming and handling point cloud data, easing the process of developing the algorithms.

2.6 Skeletal Tracking

The Kinect camera offers a built-in solution for generating skeleton models of humans based on the depth and color data of its sensors. The algorithm calculates the location of 25 nodes of a skeleton as shown in Figure 2.6. It is important to note that the generated skeleton is a representation since a complete human skeleton is much more complex. When certain parts of the human body are not visible, the Kinect can generate a partial skeleton and predict the location of the other joints, although this algorithm is not flawless. [17]

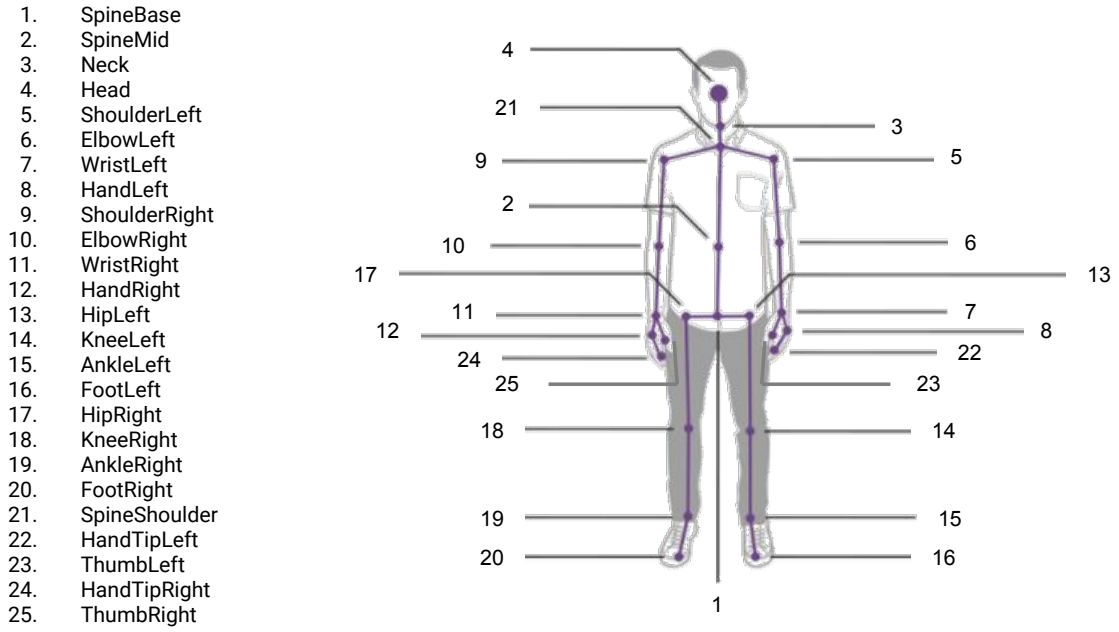


Figure 2.6: Representation of the different skeleton joints detected by the Kinect v2 camera. The indices indicates the order in which the Kinect SDK outputs the joint data.

2.7 Object Detection

In this paper, two different approaches for object detection are investigated, a point cloud and skeleton approach. When the point cloud method is applied, only points are extracted and no feature detection is performed. Therefore, the whole environment has to be defined in order to eliminate points from static objects as for example the base of the robot. An advantage of this principle is that every object entering the workspace, whether it is a human or other machine, will be detected in order to prevent collisions.

With the use of skeletonizing as described in Section 2.6, only the humans entering the workspace are identified. Every unknown set-up can be used although there is no guarantee the person will be detected since the skeleton algorithm is not flawless. Even though skeleton tracking can encounter problems of bone-length variation and self-occlusion, the calculations needed to determine the joint locations are performed on the camera hardware and therefore ease the computational effort. Since the set-up operates as a proof-of-concept, these limitations are tolerated because in industrial environments other, more reliable equipment is used. [18]

In Figure 2.7 the results for the cycle time of both the point cloud and skeleton approach on the set-up in the robotics lab are presented. It can be clearly stated that the skeleton output requires less computational time on the computer and is therefore better suited for real-time interactions in this environment. Although more detected people will increase the cycle time, this influence is not noteworthy and therefore neglected.

To improve the accuracy of the skeletal tracking and prevent dead-corners it is possible to fuse data from multiple cameras together as described in [19]. This method increases the efficiency and reliability of the object detection and can prevent people outside the range of one camera for not being detected. Since the hardware used in this project only allows to communicate with one Kinect at a time, this approach is not further elaborated although it can be implemented in future projects.

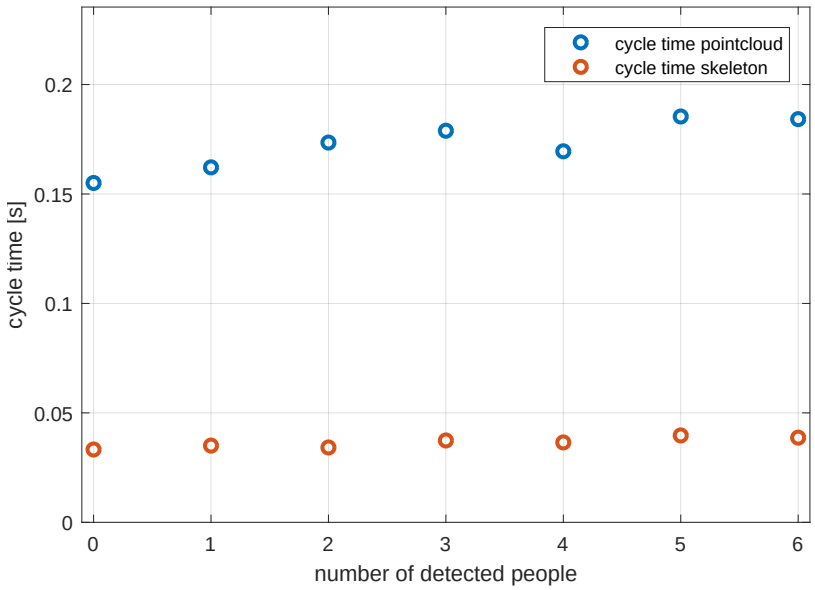


Figure 2.7: The cycles times of the Kinect for acquiring a filtered point cloud or skeleton frame.

In Figure 2.8 the results of both the point cloud and skeleton approach on the real set-up are illustrated. Starting from the raw 3D data, the filtered point cloud with only the dynamic objects is extracted. On the other hand, the skeleton tracking algorithm of the Kinect determines the location of all visible joints and only outputs the nodes of the skeleton.

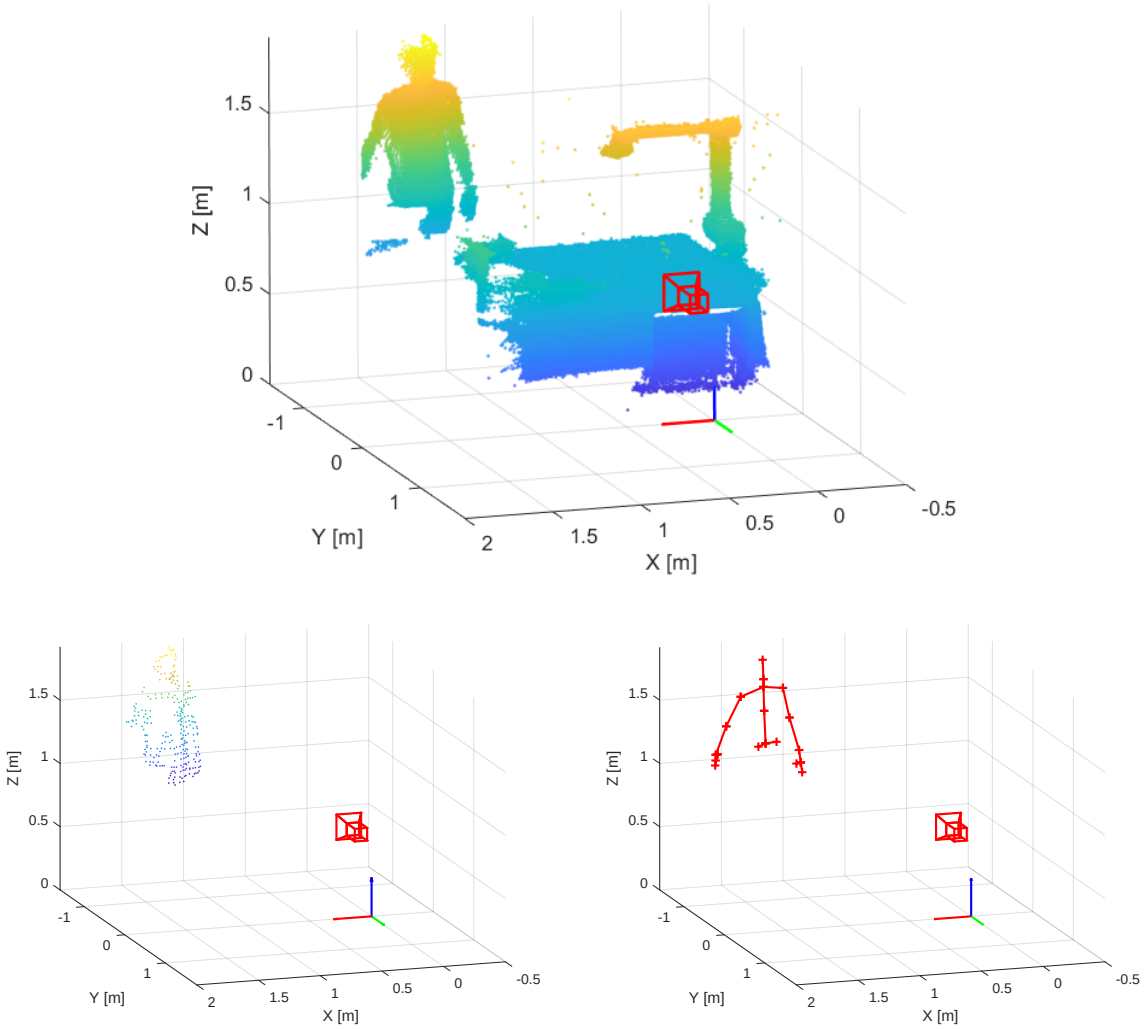


Figure 2.8: Example of object detection starting from the raw 3D data (top) to the filtered point cloud (bottom-left) and skeleton tracking (bottom-right).

2.8 Summary

The Kinect v2 camera measures distances by measuring the phase change between an emitted and received IR signal. This distance is then converted and mapped to the camera coordinate system. The calculations and transformations are executed in MATLAB allowing an easy communication with the camera. Besides the point cloud data, the Kinect also supports skeleton tracking where the location of the 25 joints of a human skeleton are derived. The skeleton tracking requires less computational time and is therefore more responsive although the algorithm is not flawless and can only detect humans entering the workspace.

Chapter 3

The Robotic Arm: UR10

The Universal Robots UR10 is a key component in this research and therefore it is interesting to look a little bit closer to the working and characteristics of this robotic arm. This chapter discusses the specifications and safety of the robot and proposes a way for communication. The speed control and stopping of the robot are investigated in order to find the best suited steering. Furthermore, a kinematic model of the robot is presented and the forward and inverse kinematics are elaborated.

3.1 Introduction

The UR10 is a product of the Danish company Universal Robots and is part of the UR family. They produce three different robotic arms: UR3, UR5 and UR10 (Figure 3.1) with the number indicating the maximum payload in kg. All three robots are collaborative robots (cobots) and are designed to collaborate with humans in a shared workspace. The robots are regarded as safe and can work without safety cages yet only after a proper risk assessment is done. The current in the joints is monitored in order to prevent the robot from exerting excessive force to its surroundings although it can still do severe harm when not handled carefully. [20]

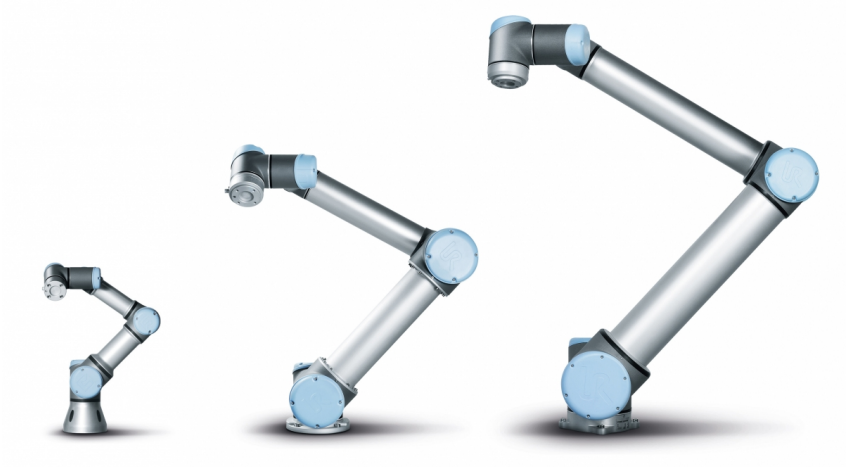
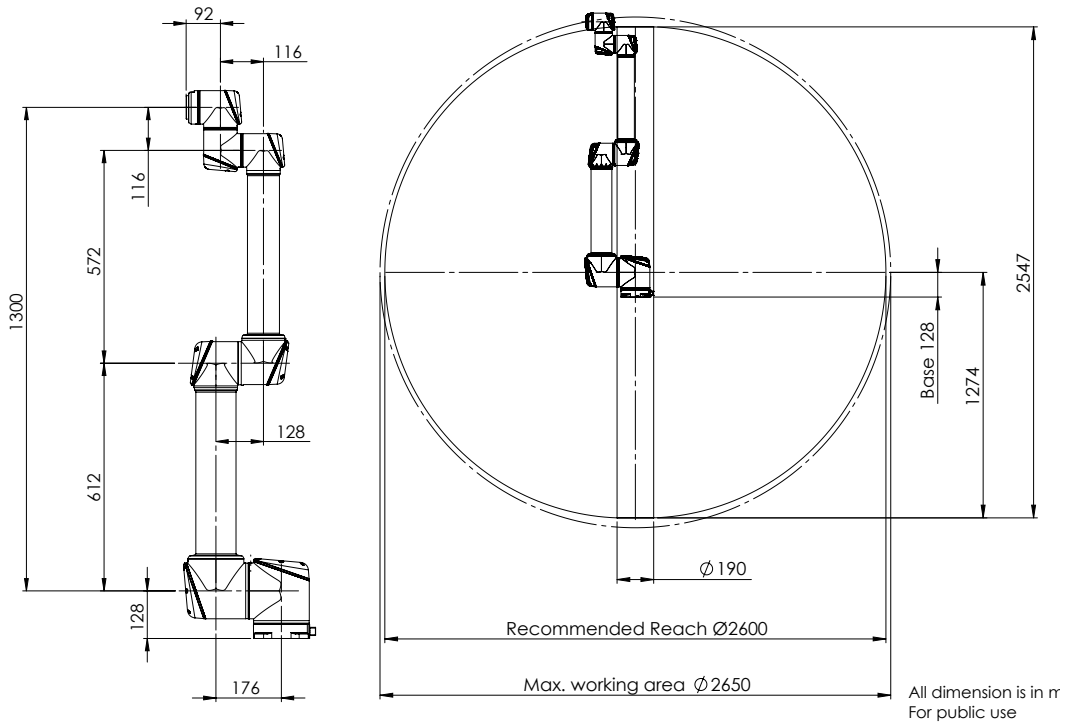


Figure 3.1: The Universal Robots UR family cobots. From left to right: UR3, UR5 and UR10. [4]

3.2 Specifications

The UR10 is a 6-DOF (Degrees of Freedom) serial robotic arm with six active rotational joints. As mentioned before, the robot has a maximum payload of 10 kg although this decreases when the distance between the center of the tool output flange and the center of gravity increases. The workspace of the robot extends 1300mm from the base joint (Figure 3.6), although the cylindrical space directly above and under the base should be avoided due to singularities. Positions at the boundary of the workspace have a high risk for singularities too. Table 3.1 gives a brief summary of the most important specifications of the UR10 robot. The robot is manufactured with aluminum and PP (polypropylene) plastic resulting in a mass of only 28.9 kg. [21]



(a) UR10 dimensions **(b)** UR10 workspace

Figure 3.2: Drawings of the UR10 robot and workspace. All dimensions in mm. [21]

Table 3.1: Technical specifications of Universal Robots UR10 [21]

Weight	28.9 kg
Maximum payload	10 kg
Reach	1300 mm
Joint ranges	± 360
Max. Speed	Base/Shoulder: 120/s
	Other joints: 180/s
	Tool/TCP: 1 m/s
Repeatability	0.1 mm
Power	Approx. 350 W

3.3 Communication

There are several ways of programming the UR10: with the teach pendant, using URScript or in the C programming language. The teach pendant only allows offline programming making it inadequate for real-time interactions. With C it is possible to make a custom control that runs directly on the controller. However, this new controller has to run instead of the original firmware which is hard to configure. URscript is Python-based programming language that is used in the background by the teach pendant too. It has the most accessible and intuitive interface and has some build in functions to move in joint space (*movej*) or in cartesian space (*movel*). Therefore, it will be used in this project to control the movements of the robot. [22]

The UR10 controller, URControl, uses Ethernet and the TCP/IP protocol to send and receive commands. The control and calculations of the robot are executed in MATLAB as the communication with the camera also goes through MATLAB (Section 2.5). Although it is possible to send commands directly from MATLAB to the controller, it is chosen to use a ROS middleware for a more versatile solution. Therefore, the ROS supported ur_moder_driver runs on a computer with a Linux operating system which is directly connected to the robot controller through TCP/IP.

The ur_modern_driver allows an easy communication with many different operating systems and publishes robot data with the use of ROS messages. As MATLAB has a built-in ROS support, this ensures a very clear and effortless communication. The ur_modern_driver also supports trajectory control, although this is not further elaborated in this project. [22] An overview of the complete communication model is given in Figure 3.3.

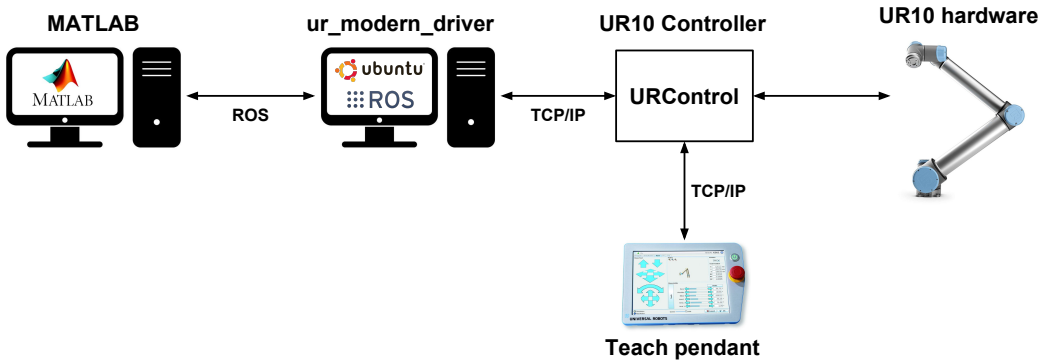


Figure 3.3: Communication model of the data transmission between the UR10 robot and the MATLAB Software. With the use of ROS-messages, the commands and data are exchanged between MATLAB and the ur_modern_driver. Via Ethernet and TCP/IP, the driver communicates with the robot controller which manipulates the robotic arm.

3.4 Speed Control

For the purpose of this project, it is essential that the robot can change its speed when executing a trajectory in order to create an adaptive environment. It is possible to use the URScript *movej* command and resend it every cycle with a new speed input. However, as stated in [23], the *movej* command handles interruptions very poorly and can cause in extreme cases a triggering of the joint torque violation safety limit. The reason for this behaviour is that the *movej* command presumes a standstill of the robot and will therefore stop the robot's movements every time a new command is initiated.

It becomes apparent that another solution for the speed control is inevitable. Therefore, the UR10 controller also offers a speed slider, which can be regulated with URScript commands and directly affects the speed at which the robotic arm moves. The speed slider accepts a factor between 0 and 1 and limits the maximum angular velocity for the robot joints. The slider is also accessible at the teach pendant. [21]

In Figure 3.4 the different test results of a sudden speed change are presented. It is clear that the speed slider handles the speed transitions more smoothly and causes less disruptions in the speed pattern than resending a *movej* command. Fluctuations in the TCP speed are caused by rounding errors and do not correspond with physical speed changes. Similar results are obtained for the other joints as shown in Appendix A.1. For this reason, the robot movements are handled by the *movej* command and speed changes in between are dealt with by altering the speed slider.

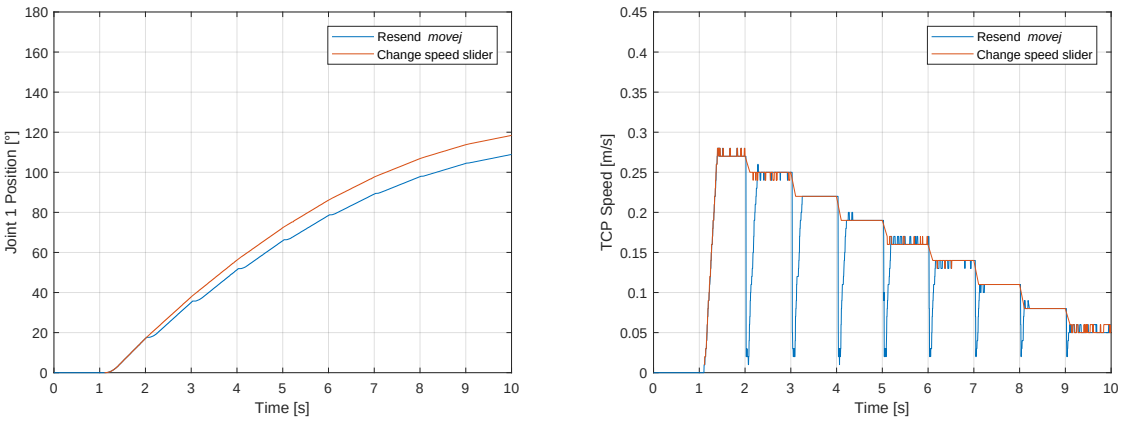


Figure 3.4: Result of a speed changes when the robot is executing a path by resending *movej* command and using the speed slider. The robot is commanded to turn 180° along the first axis after one second and its speed is decreased with 10% every subsequent second. The speed slider offers a smoother path and causes less disruptions in the speed pattern.

3.5 Safety & Stop

Before working with the UR10, a proper risk assessment according to ISO 13849:2015 [24] has to be performed in order to determine all the potential hazards and risks. Although the UR10 robot has numerous safety functions which are constantly checking the robot's movements, the robot can still do severe harm to its surroundings. For example, a cobot handling a sharp object can lead to dangerous situations showing that the cobot is not intrinsically safe. Not the robot alone but the whole collaborative operation which includes the task and space in which the task is being performed should be taken in consideration.

The robot is equipped with various safety inputs in order to connect special safety equipment. There are two stop categories available: an emergency and protective stop as described in Section 5.5. The UR10 emergency stop is designed as a stop category 1 (IEC 60204-1) and will first stop with power available to the robot before removal of the power source. The protective stop (or *safeguard stop*) on the other hand is a stop category 2 where the drive power remains available to the robot actuators. Both stop categories have performance level PLd according to ISO 13849:2015 which is the minimum for safety related parts of control equipment. [21]

The robot should be able to stop within a specific time when a certain safety limit is exceeded. Using the protective stop function of robot is the safest and most appropriate way, but requires extra switching devices and physical connections for restarting. As this equipment was not available at the robotics lab and the protective stop function is not available in the simulator, an alternative approach is chosen.

Since the speed slider is used for speed control, an alternative is to set this slider to zero when the robot has to stop. However, with this method there is no control over the responsiveness of the robot on a stop instruction. Therefore, stopping the robot with a URscript *stopj* command is selected to stop the robot with a certain deceleration that can be specified when executing the command (Appendix B.1).

A comparison of both approaches is given in Figure 3.5. The *stopj* command responds faster and gives more possibilities to alter the stop behavior since de deceleration can be adjusted. Similar results are obtained for the other joints as shown in Appendix A.2.

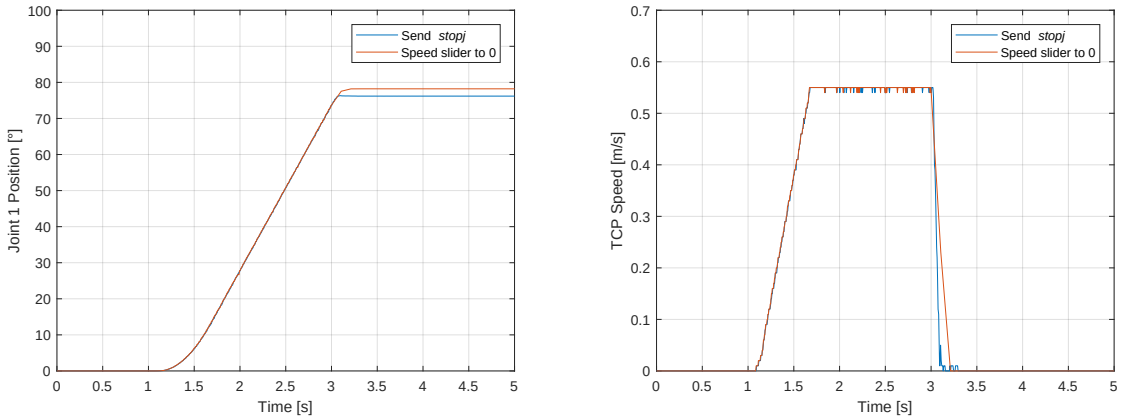


Figure 3.5: Result of a stop instruction with the speed slider and *stopj* command when the robot is executing a path. The robot is commanded to turn 180 along the first axis after one second and is commanded to stop after 3 seconds. The robot comes faster to a standstill with the *stopj* command.

3.6 Kinematic Model

Every robotic arm can be represented as a chain of links (rigid bodies) connected with each other through joints. It is possible to represent this structure with the use of only lower-pairs (singular joints) as described in [25]. A 2D representation of the kinematic structure of the UR10 is presented in Figure 3.6. When the six rotational joint angles of the UR10 are known, the location of every link is unambiguously determined. In order to work out the exact locations of the robotic arm, coordinate systems are allocated to every link.

There are at least four parameters required to determine the position of one coordinate system relative to another one. Although there are several ways to define these parameters, the Denavit-Hartenberg [26] representation with its parameters θ_i "*joint angle*", d_i "*joint offset*, a_i "*link length*" and α_i "*link twist*" has become the standard in robotics. This representation imposes additional conditions to the location of the coordinate systems as described in [27]. The resulting Denavit-Hartenberg representation can be found in Figure 3.6 and the corresponding Denavit-Hartenberg parameters in Table 3.2.

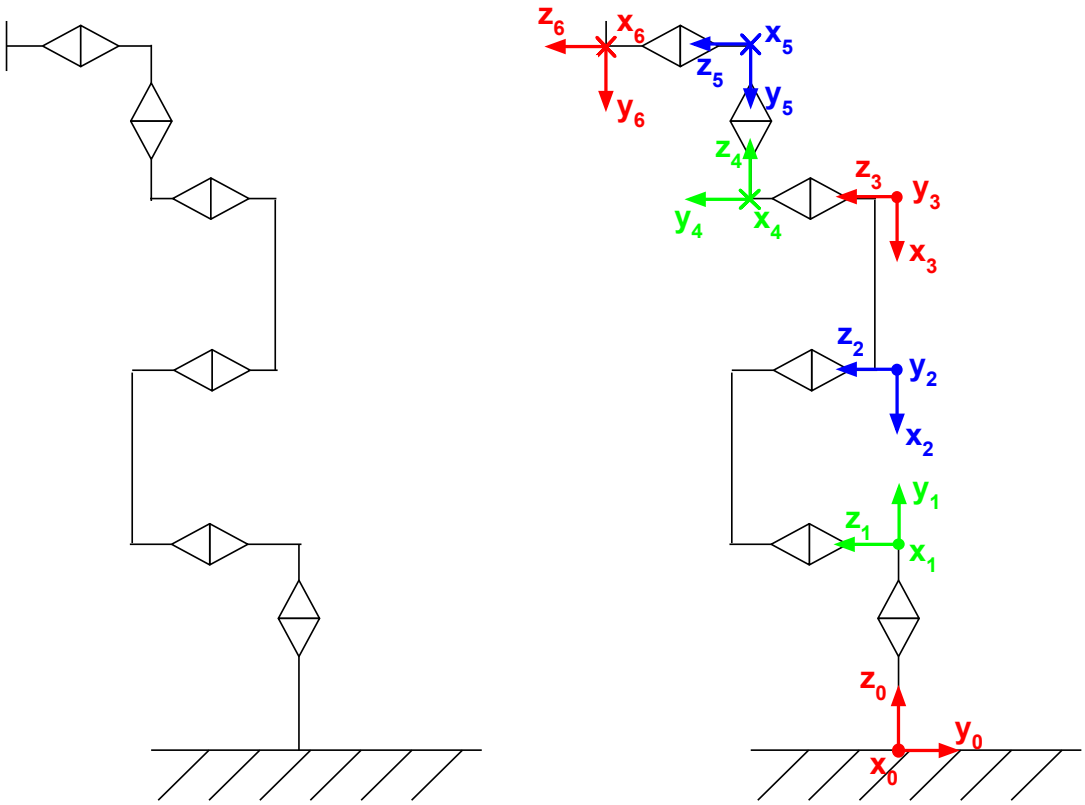


Figure 3.6: 2D kinematic representation of the UR10 with and without the DH coordinate systems.

Table 3.2: Denavit-Hartenberg parameters of the Universal Robots UR10.

Link i	θ_i [°]	d_i **[mm]**	a_i **[mm]**	α_i [°]	$\theta_{i,fig}$ [°]	**Range of** i [°]
1	θ_1	128	0	90	0	± 360
2	θ_2	0	-612	0	-90	± 360
3	θ_3	0	-572	0	0	± 360
4	θ_4	164	0	90	-90	± 360
5	θ_5	116	0	-90	0	± 360
6	θ_6	92	0	0	0	± 360

3.7 Forward Kinematics

Forward kinematics (or *direct kinematics*) are used for calculating the position and orientation of the end-effector or intermediate joints and provide a solution for translating coordinates in joint space (joint angles) to coordinates in cartesian space (x,y,z and orientation). After the kinematic model of the robotic arm is established (Section 3.6), the homogeneous transformation matrices as defined in Section 4.1 are used to express the conversion from link $i-1$ to link i, indicated as ${}^{i-1}_{i}\mathbf{T}$. Once the Denavit-Hartenberg parameters are defined, this transformation matrix can be represented as a product of four basic transformations (translations and rotations) that are used when defining the link frames. The complete transformation matrix is given by Equation 3.1, where $c\theta_i$ and $s\theta_i$ are the short hands for $cos\theta_i$ and $sin\theta_i$. [27][28]

$$
\begin{aligned}
{}^{i-1}_{i}\mathbf{T} &= \begin{bmatrix} c\theta_i & -s\theta_i & 0 & 0 \\ s\theta_i & c\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c\alpha_i & -s\alpha_i & 0 \\ 0 & s\alpha_i & c\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} c\theta_i & -s\theta_i \cdot c\alpha_i & s\theta_i \cdot s\alpha_i & a_i \cdot c\theta_i \\ s\theta_i & c\theta_i \cdot c\alpha_i & -c\theta_i \cdot s\alpha_i & a_i \cdot s\theta_i \\ 0 & s\alpha_i & c\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned} \tag{3.1}
$$

The concatenation of the multiple links can be done by multiplying the individual transformation matrices, resulting in one single transformation matrix. For example, computing the location and orientation of the end-effector with respect to the robot base frame, can be done by multiplying every link transformation of the robotic arm as in Equation 3.2. This results in a homogeneous transformation matrix where the positions vector describes the position of the TCP and the rotation matrix the orientation of the tool with respect to the base coordinate system. The homogeneous transformation matrix is further elaborated in Section 4.1. [29]

$$
{}^{0}_{6}\mathbf{T} = {}^{0}_{1}\mathbf{T} \cdot {}^{1}_{2}\mathbf{T} \cdot {}^{2}_{3}\mathbf{T} \cdot {}^{3}_{4}\mathbf{T} \cdot {}^{4}_{5}\mathbf{T} \cdot {}^{5}_{6}\mathbf{T} \tag{3.2}
$$

3.8 Inverse Kinematics

Although the forward kinematics are quite straightforward as they only require simple multiplications, the inverse kinematics of a serial robotic arm can be quite difficult to solve, especially with increasing DOFs. When calculating the inverse kinematics of the UR10, the goal is to determine the joint angles ($\theta_1..\theta_6$) for which the end-effector of the robotic arm reaches a certain position and orientation. In other words, finding the joint angles for which the transformation matrix ${}^{0}_{6}\mathbf{T}$, as specified in Equation 3.2, is identical to the desired transformation matrix of the end-effector ${}^{0}_{6}\mathbf{T}_{des}$. Furthermore, there can be eight possible configurations (Figure 3.7), increasing the complexity of the inverse kinematics even more. [30]

$$
{}^{0}_{6}\mathbf{T} = {}^{0}_{6}\mathbf{T}_{des} = \begin{bmatrix} n_{6x} & o_{6x} & a_{6x} & p_{6x} \\ n_{6y} & o_{6y} & a_{6y} & p_{6y} \\ n_{6z} & o_{6z} & a_{6z} & p_{6z} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.3}
$$

For a robotic arm with 6 DOF like the UR10, this results in solving 12 non-linear equations (9 elements in the rotation matrix and 3 elements in the position vector). However, due to the orthogonality of the vectors in the rotation matrix, only three equations of the rotation matrix are independent, narrowing the problem down to solving six non-linear equations with six unknown. [28]

There are two main strategies for solving the inverse kinematics of a manipulator: *closed-form solutions* and *numerical solutions*. Closed-form solutions use an analytical approach and can be based on geometrical properties. Numerical solutions use iterative techniques to find the desired joint angles but require more computational time. Because it is desired to keep the calculation time as low as possible in real-time interactions, it is chosen to formulate a closed-form solution for the UR10. [29]

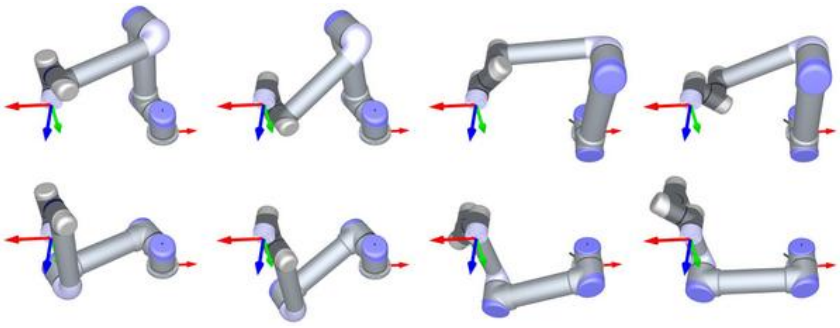


Figure 3.7: The UR10 can have 8 possible solutions for a desired end-effector pose, increasing the complexity of solving the inverse kinematics. [31]

The result of the inverse kinematic in this section is adopted from [32], although this analytical solution only delivers formulations for θ_1, θ_5 and θ_6. The joints in between can be seen as a 3R (three rotational joints) planar arm and the formulas for this are extracted from [28]. All coordinate systems as defined in Figure 3.6 are expressed relative to the robot base coordinate system 0. The location of coordinate system i is described with a position vector $\boldsymbol{p_i}$ and a rotation matrix $\boldsymbol{R_i}$. Here too, $cos\theta_i$ and $sin\theta_i$ are replaced by their short hands $c\theta_i$ and $s\theta_i$.

The desired transformation matrix corresponds with the position and orientation of coordinate system 6.

$$ {}^{0}_{6}\mathbf{T}_{des} = \begin{bmatrix} n_{6x} & o_{6x} & a_{6x} & p_{6x} \\ n_{6y} & o_{6y} & a_{6y} & p_{6y} \\ n_{6z} & o_{6z} & a_{6z} & p_{6z} \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{R_6} & \boldsymbol{p_6} \\ 0_{1\times 3} & 1 \end{bmatrix} \tag{3.4} $$

First of all, we can determine $\boldsymbol{p_5}$ by moving a distance $-d_6$ along the z-axis of coordinate system 6. The orientation of this z-axis is indicated by the third column of the rotation matrix.

$$ \boldsymbol{p_5} = \begin{bmatrix} p_{5x} \\ p_{5y} \\ p_{5z} \end{bmatrix} = \boldsymbol{p_6} - d_6 \cdot \begin{bmatrix} a_{6x} \\ a_{6y} \\ a_{6z} \end{bmatrix} \tag{3.5} $$

Once $\boldsymbol{p_5}$ is known, it is possible to calculate θ_1 with the following equation. Note that there are two solutions corresponding to configurations where the shoulder is "left" or "right".

$$ \theta_1 = atan2(p_{5y}, p_{5x}) \pm cos^{-1}\left(\frac{d_4}{\sqrt{{p_{5x}}^2 + {p_{5y}}^2}}\right) + \frac{\pi}{2} \tag{3.6} $$

From θ_1 we can determine θ_5 and θ_6. θ_5 has two solutions corresponding to configurations with the wrist "in/down" or "out/up".

$$ \theta_5 = \pm cos^{-1}\left(\frac{p_{6x} \cdot s\theta_1 - p_{6y} \cdot c\theta_1 - d_4}{d_6}\right) \tag{3.7} $$

$$ \theta_6 = atan2\left(\frac{-o_{6x} \cdot s\theta_1 + o_{6y} \cdot c\theta_1}{s\theta_5}, \frac{n_{6x} \cdot s\theta_1 - n_{6y} \cdot c\theta_1}{s\theta_5}\right) \tag{3.8} $$

The final three joints can be seen as a classical 3R planar arm. Since we have the other joints already determined, we can locate the base and end-effector of the 3R planar arm. The base location of the 3R planar arm corresponds with coordinate system 1 and can easily be calculated according to Equation 3.1 since θ_1 is already known.

$$ {}^{0}_{1}\mathbf{T} = \begin{bmatrix} c\theta_1 & -s\theta_1 \cdot c\alpha_1 & s\theta_1 \cdot s\alpha_1 & a_1 \cdot c\theta_1 \\ s\theta_1 & c\theta_1 \cdot c\alpha_1 & -c\theta_1 \cdot s\alpha_1 & a_1 \cdot s\theta_1 \\ 0 & s\alpha_1 & c\alpha_1 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{R_1} & \boldsymbol{p_1} \\ 0_{1\times 3} & 1 \end{bmatrix} \tag{3.9} $$

Finding the location of coordinate system 3 which corresponds with the end-effector of the 3R planar arm, requires an alternative approach due to unknown joint angles in between the base and end-effector. Yet it is possible to find the transformation matrix ${}^{0}_{4}\mathbf{T}$ based on the already known joint angles with the following equation, where ${}^{4}_{6}\mathbf{T}^{-1}$ is the inverse matrix of ${}^{4}_{6}\mathbf{T}$.

$$ {}^{0}_{4}\mathbf{T} = {}^{0}_{6}\mathbf{T} \cdot {}^{4}_{6}\mathbf{T}^{-1} = \begin{bmatrix} \boldsymbol{R_4} & \boldsymbol{p_4} \\ 0_{1\times 3} & 1 \end{bmatrix} \tag{3.10} $$

Analogue to Equation 3.5 we can find the position vector $\boldsymbol{p_3}$ by moving a distance $-d_4$ along the y-axis of coordinate system 4.

$$ \boldsymbol{p_3} = \begin{bmatrix} p_{3x} \\ p_{3y} \\ p_{3z} \end{bmatrix} = \boldsymbol{p_4} - d_4 \cdot \begin{bmatrix} o_{6x} \\ o_{6y} \\ o_{6z} \end{bmatrix} \tag{3.11} $$

Once these positions are known, we use the equations as stated in [29] for a 3R planar arm to solve the remaining joint angles θ_2, θ_3 and θ_4. The arm has two possible configurations, "elbow up" or "elbow down".

3.9 Summary

The UR10 is a 6-DOF serial robotic arm that can be used in collaborative workspace, yet only after a proper risk assessment is done. The robot is controlled in MATLAB to allow an easy integrating with the Kinect camera. The communication with the controller is realized through ROS with the use of ur_modern_driver for a more versatile solution. The robot movements are controlled with the URscript *movej* and *stopj* commands and speed changes in between are handled by the speed slider. A kinematic model of the robot is presented and both the forward and inverse kinematics are elaborated in order to control the robot in both joint and cartesian space.

Chapter 4

Distance Calculation

Whether it are points from a point cloud, coordinates of the skeleton joint or positions of the robot TCP, in order to evaluate this data, all points must be expressed relative to a reference frame. However, a reference frame is able to move in space which makes it very impractical for comparing it with other points as for example for distance calculations. This chapter introduces a way of describing the position and orientation of a coordinate system and how to transform its points to a chosen world coordinate system. Once the data from the camera and robot is converted, operations can be executed to calculate the minimum distance between the robot and the human being.

4.1 Homogeneous transformations

The most common way of describing points in space is with the use of its cartesian coordinates (x, y, z). In order to use homogeneous transformations, we will represent every point as a position vector $\begin{bmatrix} wx & wy & wz & w1 \end{bmatrix}^T$ with w being the weighing factor which is set to 1 in robotics. This results in the following notation for describing a point relative to a chosen reference frame: [33]

$$p = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{4.1}$$

A homogeneous transformation matrix $\boldsymbol{H}$ is a 4x4 matrix which transforms a position vector from one coordinate system to another. The matrix is composed of a rotation matrix R, a position vector p, a perspective vector f and a weighing factor w. In robotics, the perspective vector is set to $0_{1\times3}$ and the weighing factor is set to 1. The rotation matrix is further elaborated in Section 4.3. [34]

$$\boldsymbol{H} = \begin{bmatrix} R_{3\times3} & p_{3\times1} \\ f_{1\times3} & w_{1\times1} \end{bmatrix} = \begin{bmatrix} R_{11} & R_{11} & R_{13} & a \\ R_{21} & R_{22} & R_{23} & b \\ R_{21} & R_{32} & R_{33} & c \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.2}$$

4.2 Translation

As shown in Figure 4.1, a coordinate system can be translated in space, which is represented with a position vector $\begin{bmatrix} a & b & c & 1 \end{bmatrix}^T$. The orientation however remains identical, causing the rotation matrix to be an identity matrix. The resulting transformation matrix is given in Equation 4.3.

$$\boldsymbol{H} = \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.3}$$

Starting from a point p with coordinates (x_2, y_2, z_2) relative to coordinate system 2 and knowing its location referring to coordinate system 1, simply pre-multiplying the position vector of p with the homogeneous transformation matrix from Equation 4.3, will result in the coordinates of point p with respect to the axes of coordinate system 1 (x_1, y_1, z_1). [35]

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \\ z_2 \\ 1 \end{bmatrix} = \begin{bmatrix} x_2 + a \\ y_2 + b \\ z_2 + c \\ 1 \end{bmatrix} \tag{4.4}$$

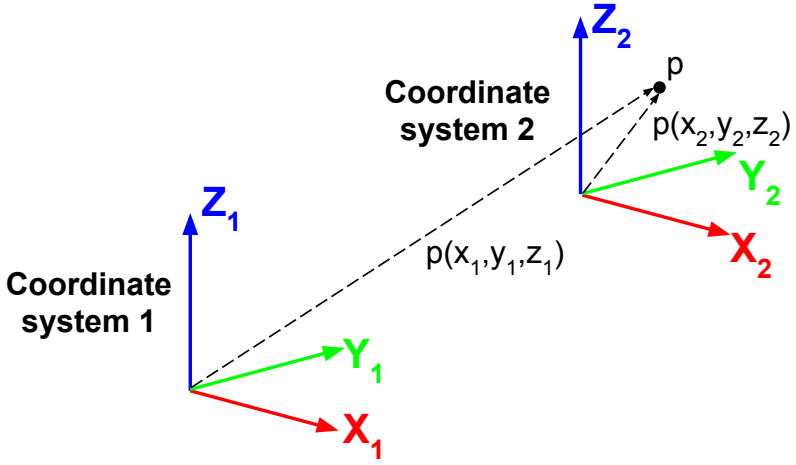


Figure 4.1: Translated coordinate system along x_1 by a, along y_1 by b and along z_1 by c.

4.3 Rotation

Besides a translation, a reference frame can experience rotations too as illustrated in Figure 4.2. These rotations are represented by a rotation matrix where every column represents the unit vector of the rotated axes relative to the original coordinate system. Equation 4.5 shows the resulting transformation matrix. [34]

$$\boldsymbol{H} = \begin{bmatrix} R_{11} & R_{11} & R_{13} & 0 \\ R_{21} & R_{22} & R_{23} & 0 \\ R_{21} & R_{32} & R_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.5}$$

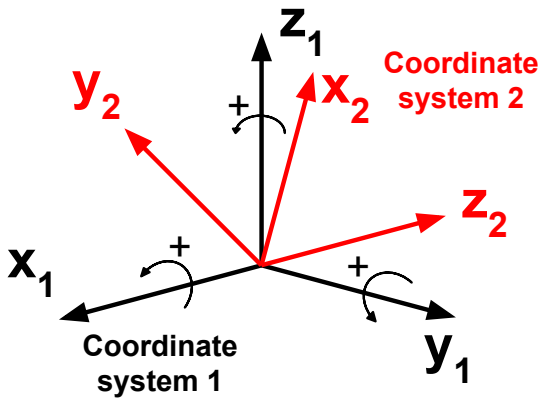


Figure 4.2: Translation of coordinate system 1 to coordinate system 2 with indication of the conventional positive angles.

As an example, we will compose the rotation matrix of coordinate system 2 in Figure 4.3. The vector representing the x_2-axis can be written as $\begin{bmatrix}0 & 1 & 0\end{bmatrix}^T$ since it lies parallel with the y_1-axis. Analogous to the previous method the vectors of the y_2 an z_2 can be formulated resulting in the following rotation matrix $\boldsymbol{R}$:

$$\boldsymbol{R} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.6}$$

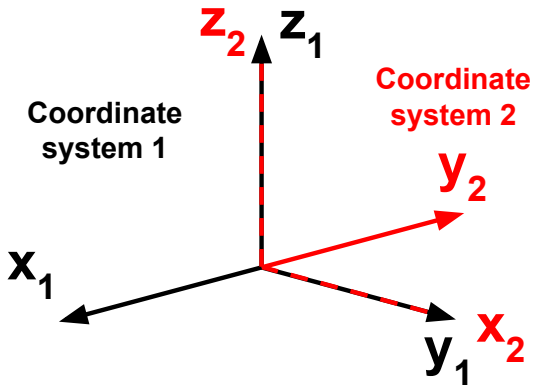


Figure 4.3: Example of rotated coordinate systems

In practice the orientation of a coordinate system is often described with angles instead of rotation matrices. By convention, we use right-handed coordinate systems with counter-clockwise angles being positive as shown in Figure 4.2. There have to be at least three successive rotations in order to be able to reach every possible orientation, although it is possible to rotate around different axes. These three angles are called 'euler angles' (α, β and γ) and the sequence of rotation is noted with the successive axes of rotation e.g. 'XYZ', 'ZXZ'... The three elemental rotations may be extrinsic (around axes of the reference coordinate system) or intrinsic (around axes of the rotating coordinate system). If the type, sequence and euler angles of the rotation are known, it is very straightforward to convert these angles back into rotation matrices with predefined formulas. [29]

4.4 Transformation

The complete transformation is a combination of the translation and rotation as described in the previous sections and results in the complete homogeneous transformation matrix. Analogue to Equation 4.4 we pre-multiply the position vector from coordinate system 2 with the complete homogeneous transformation matrix in order to get the position vector relative to coordinate system 1: [35]

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{bmatrix} = \begin{bmatrix} R_{11} & R_{11} & R_{13} & a \\ R_{21} & R_{22} & R_{23} & b \\ R_{21} & R_{32} & R_{33} & c \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \\ z_2 \\ 1 \end{bmatrix} \tag{4.7}$$

As an example we will perform a transformation of point p with coordinates (x_2, y_2, z_2) relative to coordinate system 2 in order to get its location with respect to coordinate system 1. (Figure 4.4). The rotation matrix is identical to the one in Equation 4.6. Completing the rest of the equation with the parameters from Figure 4.4 gives the result of Equation 4.8. In practice, coordinate system 1 is often a predefined and fixed reference frame in order to process data from multiple moving coordinate systems.

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 & -7 \\ 1 & 0 & 0 & 3 \\ 0 & 0 & 1 & 4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \\ 4 \\ 1 \end{bmatrix} = \begin{bmatrix} -10 \\ 5 \\ 8 \\ 1 \end{bmatrix} \tag{4.8}$$

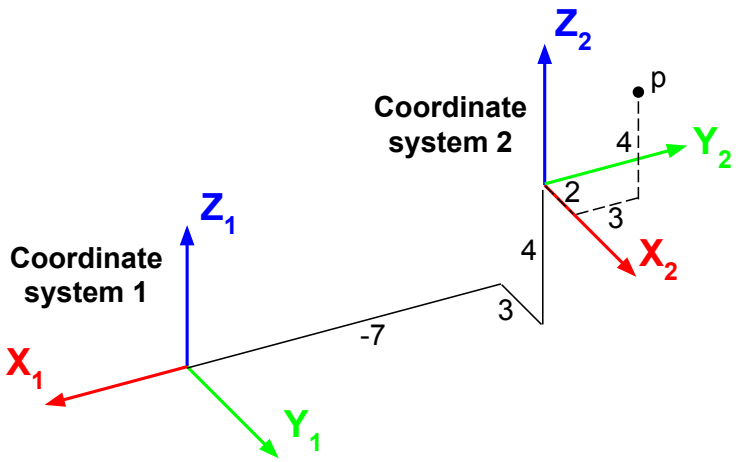


Figure 4.4: Example of translated and rotated coordinate systems.

4.5 Distance Calculation

Once data points from both camera and robot are expressed relative to a chosen world coordinate system, it is possible to calculate the closest distance between the robot and human since all data is expressed with its cartesian coordinates relative to the same reference frame. The following equation can be used to compute the euclidean distance between two points:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2} \tag{4.9}$$

An alternative approach is to calculate distances between humans and robots in depth space as described in [36]. Here the depth data from the 3D camera is not converted to cartesian space and distances are calculated to rays from the camera center. However, this method uses raw depth data and since the skeleton and point cloud data is used, it is not further elaborated.

In this project two different approaches for object detection are studied: point cloud and skeleton approach (Section 2.7). When the skeleton joints are obtained, the minimum distance is determined by calculating the euclidean distance to every joint. This calculation does not require a lot of computational power since there are only 25 joints obtained. The method is also applied on the filtered point cloud since the number of points is quite limited. There is no time difference measured between both methods for calculating the distance.

In Figure 4.5 a comparison of the distance calculation is presented. It should be noted that the calculated distance is different between both approaches since the point cloud method uses the outer shell of the body and the skeleton tracking determines the internal locations of the skeleton joints. In this example the closest distance to the TCP is calculated although other points can be easily implemented.

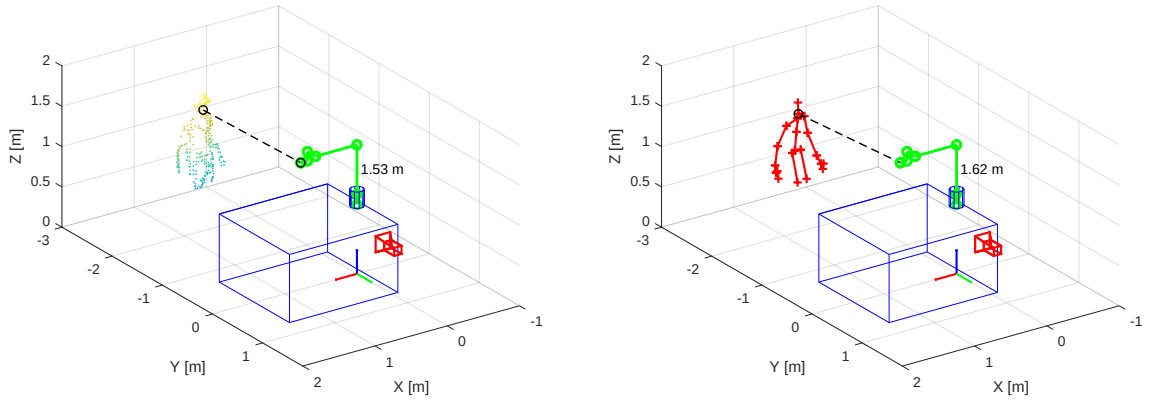


Figure 4.5: Illustration of a distance calculation with the use of the point cloud (left) and skeleton (right) approach. The robot data is used to determine the location of the TCP although other points can be used. The closest distance is calculated and the camera, WCS and robot base are added for a better representation.

4.6 Summary

The transformation of points from a reference frame to the world coordinate system is done by homogeneous transformation matrices. Points and translations are represented by a position vector and rotations by a rotation matrix. Rotations can also be expressed with euler angles and converted to the corresponding rotation matrix with predefined formulas. Once all data points are converted it is possible to evaluate the closest point by calculating the closest euclidean distance.

Chapter 5

Human-Robot Interaction

In this chapter we will have a closer look at human-robot interaction and how this can be implemented on robotic installations. First, the types of interaction and safety standards are explained. The economic viability is also discussed and the stopping functions provided by the standards are elaborated. Finally, the various suggested collaborative operation modes are explained which will be implemented on the set-up.

5.1 Introduction

Human-robot interaction is a relative new evolution in robotics changing the way we think about classical production settings. At first, robots where completely fenced and any interaction with human workers was strictly prohibited. All safety systems were designed to prevent any contact with co-workers to ensure that powerful robots could not harm human beings. When operators needed to interface with the robot, this often meant a full stop or cutting the power source. It is clear that this approach was not in favor of the productivity and efficiency rising the need for new system designs. [37]

The answer to these problems are hybrid production systems characterized by a close collaboration between the robot and its co-worker. By combining the flexibility and problem-solving skills of humans with the strength, endurance and precision of robots, a new type of production is possible which allows a lot more flexibility and allows humans to do a lot more higher value and safer work. Especially in small and medium sized productions, human-robot collaboration can have a big economic benefit even if this new approach introduces many mandatory safety aspects (Section 5.4). [38]

Although these new approaches offer a lot of possibilities, it is clear that it cannot be at the expense of human safety or as the acclaimed Isaac Asimov once stated in his "Three Laws of Robotics": [39]

> *"A robot may not injure a human being or through inaction allow a human being to come to harm."*
>
> —Isaac Asimov, *I, Robot*

5.2 Types of Interaction

The interaction of humans with robots can be characterized by two interaction parameters: space and time. (Figure 5.1) If there is no common workspace or time in which the human being and robot move, we can state that there is no interaction.

The classical applications where no human intervention is required during operation process is called coexistence. The robot is fenced off and all access is safeguarded to maintain a separate workspace. It will still be necessary for the operator to enter the robot's workspace e.g. for the purpose of maintenance work, but only when the robot is in shut-down.

When there is cooperation, the robot and co-worker use the same workspace although they will not be there at the same time. Typical applications are installations where the human loads and unloads the robot cell. Here too, safety equipment is needed to guarantee the robot will not operate when the human is nearby.

The most advanced interaction is collaboration where human and robot will use the same workspace at the same time. Reliable sensors are required for presence detection and the movement of the robot must be limited. This type of interaction will also be implemented on the set-up in the robotics lab. [40]

Application	Different workspace	Shared workspace
Sequential processing	No interaction	Cooperation
Simultaneous processing	Coexistence	Collaboration

Figure 5.1: The different types of human-robot interaction based on the the interaction parameters space (workspace) and time (processing). [40]

5.3 ISO Standards

Safety standards have been made by different institutions to ensure a safe implementation of robots in many different applications. One of these organizations is the International Organization for Standardization (ISO) which is a worldwide network of standardization organizations from 162 countries, including Belgium. Standards provide a state-of-the-art and describe agreements, specifications or criteria about a product, service or method. Although it is assumed by public opinion, standards are not prepared or agreed by governments or authorities. [41]

The standards used in this project are the ISO 10218-1:2011 [42] and ISO 10218-2:2011 [43] regarding safety requirements for industrial robots, which are both type C standards (dealing with detailed safety requirements for a particular machine or group of machines). These standards already describe collaborative operation requirements yet only very briefly (approximately eight out of the 152 pages). Therefore, new standards are in development as for example ISO/TS 15066 which is designed specific for collaborative guidance. [44].

5.4 Economic Motivation

Next to increasing the efficiency of an existing installations, human-robot collaboration can also provide an answer for production lines with low production volumes and fast changing demands. Especially for production volumes that are too big for manual assembly and too small for complete robotic installations, combined productions systems can have an economic benefit (Figure 5.2).

It is important to note that although human-robot collaboration has many benefits, it is not a solution for all production lines and existing robotic installations will still need to exist. Human-robot interaction is more an extension to existing installations although it can create new applications in which robots previously have not been used. All aspects need to be taken in consideration before deciding whether human-robot collaboration is suitable for a certain production type. [45]

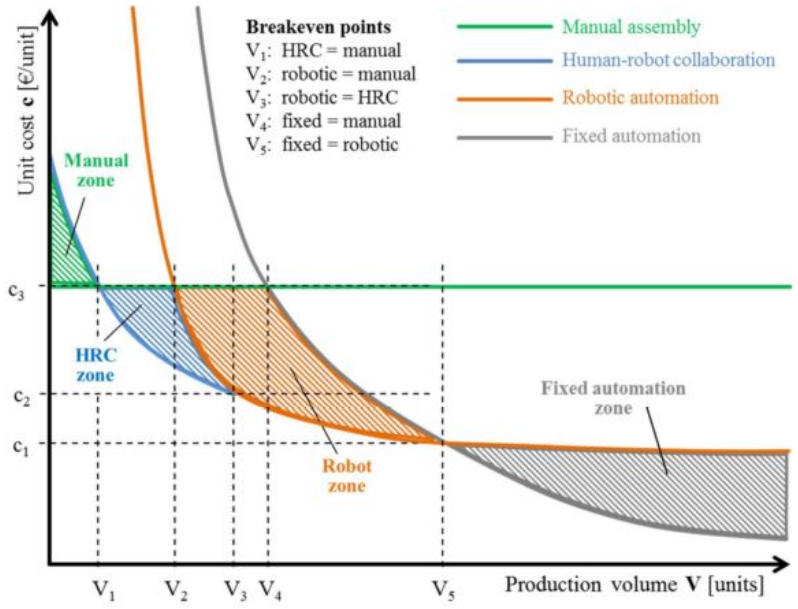


Figure 5.2: The unit cost based on the production volume for different types of automated productions. Human-robot collaboration offers an economic benefit for production size between manual and full robotic assembly. [45]

5.5 Stopping functions

When a safety limit is exceeded the robot will have to react to ensure no further damage can be done. This will often lead to a stop of the robot's movements although this can be executed in many different ways depending on the cause and type of the installation. The ISO 10218-1:2011 standard imposes two different stopping functions that have to be present on the robot: an *emergency* and *protective* stop.

An emergency stop is manually initiated and takes precedence over all other robot controls. It removes drive power from the robot actuators and can only be reset by a manual action. On the UR10 the emergency stop device is located on the teach pendant although extra stopping devices can be connected to special safety inputs.

The protective stop on the other hand can be initiated manually or by control logic. It does not have to result in the drive power being removed but does require monitoring of the standstill of the robot. A comparison of both stop categories is presented in Table 5.1. The protective stop can be used to stop the robot when a human is detected within the workspace. At the UR10 the protective stop can be controlled by specific safety inputs on the controller. [42]

Table 5.1: Comparison of an emergency and protective stop. [43]

Parameter	Emergency stop	Protective stop
Location of initiation	Operator has quick access	For protective devices
Initiation	Manual	Manual or Automatic
Reset	Manual	Manual or Automatic
Use frequency	Infrequent	Variable (can be frequent)
Purpose	Emergency	Safeguarding
Effect	Remove energy sources	Safety control of hazards

5.6 Collaborative Operation Modes

The ISO 10218-2:2011 standard offers four techniques for collaborative operations that can be implemented in collaborative applications. De different operation modes can be used on the same application although de robot should not be able to endanger people when switching between these modes. Robots configured for collaborative operation should be labelled with the symbol as shown in Figure 5.3. [34] [43] [44] The different operation modes are:

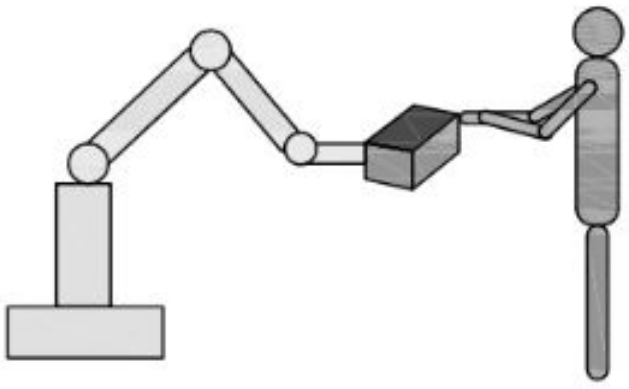


Figure 5.3: Suggested label for robots that are configured for collaborative operations. [43]

- **Safety-Rated Monitored Stop**

  If there are no human-workers detected in the workspace, the robot operates autonomously at normal speed. When co-workers enter the workspace, the robot movements will be stopped and the robot will resume its actions automatically, only after the human leaves the workspace. The Safety-Rated Monitored Stop only allows movement in the workspace of one (human or robot) at the same time and is often used in loading and unloading applications or inspection of the robot. Since the robot will not stop by cutting the drive power, the life of the power contactors is extended and the cycle time is reduced due to a faster restart.

- **Hand Guiding**

  The co-worker uses a hand-operated device to guide the robot. A Safety-Rated Monitored Stop is achieved before the operator enters the workspace and the robot can be moved manually with the guiding device. A typical application is a robotic lift assist. Hand guiding can reduce the set-up and programming time by teaching points and actions in a more natural way.

- **Speed and Separation Monitoring**

  With Speed and Separation Monitoring, the human and robot can move concurrently in the same workspace. There is always a minimum protective separation distance maintained between the robot and the co-worker and the robotic device will go into a Safety-Rated Monitored Stop when this separation distance is violated. In the rest of the collaborative workspace, the speed of the robot is lowered. A Speed and Separation Monitoring can be useful in open and unstructured environments and can increase the efficiency of the installation.

- **Power and Force Limiting**

  Physical contact between the robot and the operator is inevitable, therefore the robotic systems can be specifically designed for power and force limiting. The force the robot can exert on its environments is limited and the system will react when any contact occurs. The UR10 has already some power and force limiting functions which are constantly checking the robot's actions. The robot is also designed with rounded shapes and a low mass in order to reduce serious damage when the robot collides with humans.

For the purpose of this project a Safety-Rated Monitored Stop and Speed and Separation Monitoring are implemented on the set-up.

5.7 Summary

Human-robot collaboration occurs when the operator and robot use the same workspace at the same time and can have an economic benefit for small and medium sized production types. There are safety standards provided for handling robotic devices although new releases specific for collaborate operations are still in development. The robot should have two stop categories, emergency and protective stop, which are used in different situations. The standards also describe four collaborative operation modes from which two are implemented on the real set-up.

Chapter 6

Implementation

This chapter briefly describes how the implementation in the virtual and real environment was established. As a proof-of-concept, one of the milestones of this project was to realize a Safety Rated Monitored Stop and a Speed and Separation Monitoring (Section 5.6) on the set-up in the robotics lab. Therefore, a simulation environment had to be configured in order to test all the algorithms before implementing them on the real robot. The code structure and algorithms are described and illustrated with the use of diagrams and flowcharts.

6.1 Set-up

In the robotics lab, the UR10 robot is mounted on two worktables with a mounting profile in between. The worktables were measured and imported in the simulator (VREP), after modelling them with CAD software first. The Kinect camera is mounted on the cupboard next to the worktables as this gives the best overview of the whole configuration. A comparison of the real and modelled environment is given in Figure 6.1

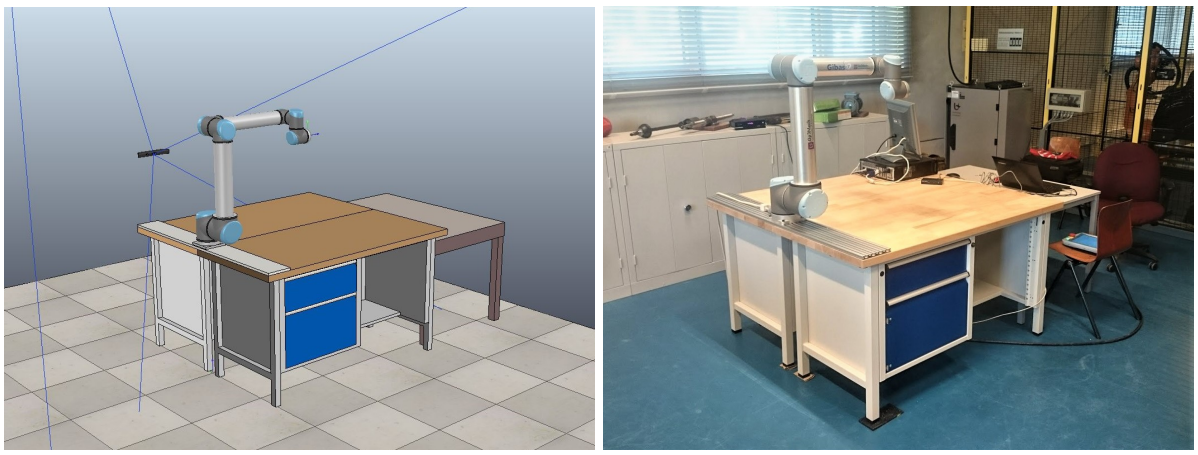


Figure 6.1: Illustration of the set-up in VREP (left) and the robotics lab at campus Groenenborger (right).

The Virtual Robot Experience Platform (VREP) was chosen as a virtual environment since it allows easy integration with MATLAB and a model of the UR10 is included in the standard installation. Moreover, the camera toolbox also supports communication with this simulator. VREP is a multi-purpose robot simulator developed by Coppelia Robotics that allows custom controllers written in several languages. Lua is the standard supported programming language and is used in this project to write the controllers for the joints. [46]

For testing, the robot is commanded to perform a pick-and-place although no objects can be grasped since there is no gripper attached. The world coordinate system (WCS) is placed directly under the robot at ground level and with its orientations as shown in Figure 6.2. Orientations are defined with intrinsic euler angles (α, β, γ) and sequence of rotation 'XYZ' (Section 4.3).

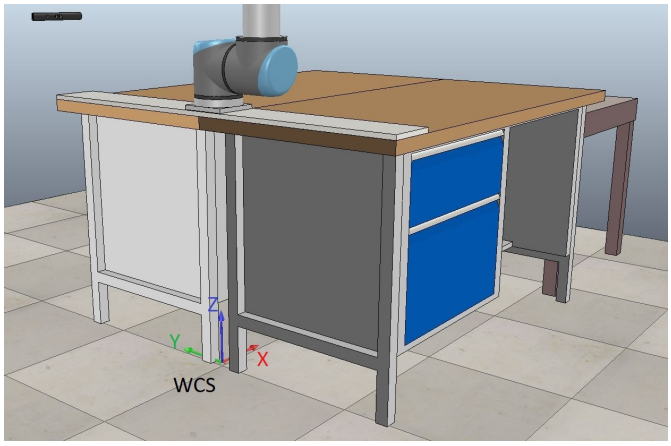


Figure 6.2: Orientation of the world coordinate system used in this project. The coordinate system is placed at ground level.

6.2 Simulator Robot Control

As the movement of the robot is controlled with *movej* commands (Section 3.4), the simulator is configured to give a good representation of this action. The *movej* function uses a limiting angular velocity v or a time t to specify the speed of the robot movement. (Appendix B.2) In VREP, every joint is controlled by a PI-controller written in a Lua script and a maximum velocity to limit the movements. Depending on the speed type of the *movej* command (v or t) the maximum velocity ω_{lim} is determined and used to limit the PI-controller of the joints. A complete flowchart of this procedure is given in Figure 6.3.

To verify the accuracy of this approach, a *movej* command is send to both the simulator and real robot. The VREP scene is executed in real-time mode in order to mimic a real environment as good as possible. Figure 6.4 gives the result for the angular position of joint 1. It is clear that the simulated robot moves very similar to the real robot and is therefore suited to verify the various algorithms. Similar results can be obtained for the other joints as shown in Appendix A.3.

Since changing the speed of the robot is a crucial part of this project, it is important that the virtual environment gives a good representation of this behavior too. The speed of the real robot is controlled with the use of the speed slider as explained in Section 3.4. However, this speed slider is not available in VREP and therefore an alternative approach for speed control is necessary.

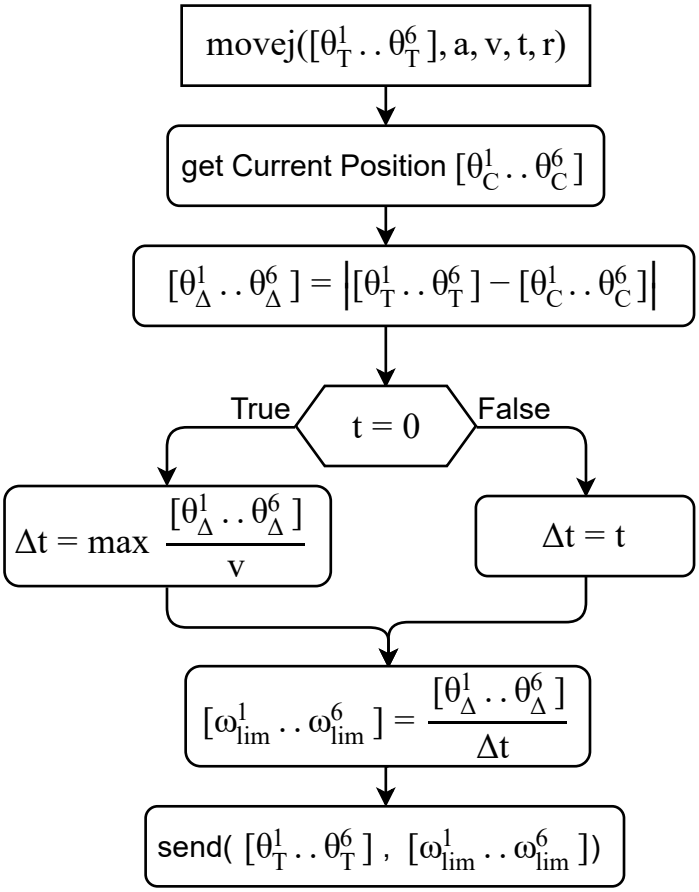


Figure 6.3: Procedure for a implementing a *movej* command in VREP. Depending on the speed type of the function (angular velocity or time) the maximum velocity ω_{lim} for the PI-controllers of the joints is determined and transmitted to the simulator.

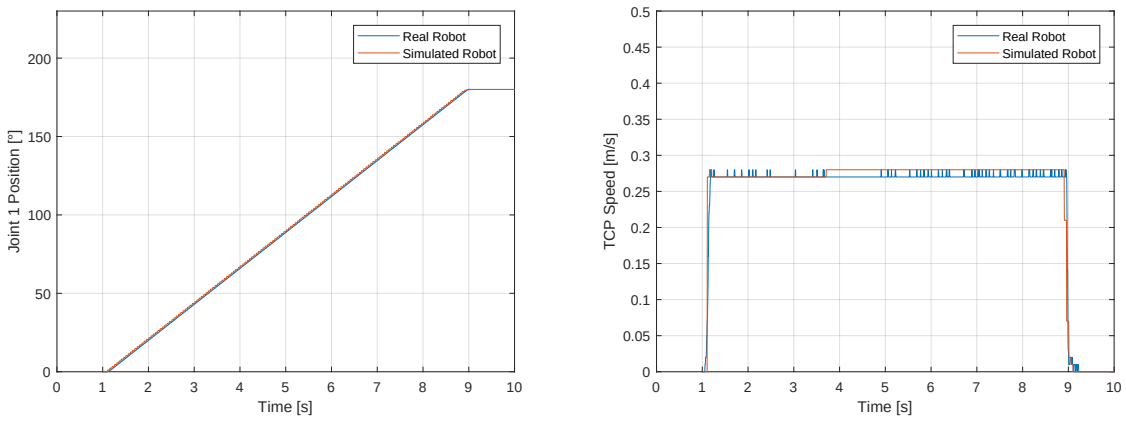


Figure 6.4: Comparison of the angular position of joint 1 between the real and simulated robot. The graphs show the similarity between the real and simulated movement.

As the speed slider is basically a factor that is used to limit the speed, we can easily integrate this in the simulator by multiplying the limiting velocity ω_{lim} with the value of the speed slider. The *stopj* command is simulated by sending the current position of the robot as the target pose in order to make the robot come to a stop. This way, the three commands *movej*, *stopj* and speed factor which are used to control the robot, can be simulated accurately in VREP.

6.3 Code Structure

As mentioned before, all the algorithms are developed in MATLAB due to its easy integration with both the Kinect camera and the UR10 robot. Furthermore, this lowers the threshold for future students to build on the existing algorithms and shortens the time needed to set up the communication protocols.

Object-oriented programming is used for organizing the code into classes. The complete code structure consists of seven classes which are related to each other as shown in Figure 6.5. Every command in the core classes has the same effect on both the real and virtual set-up. Therefore, switching between the simulator and real environment only requires a different initialization. The main controller class contains functions that need access to both the robot and camera data. A separate GUI (Graphical User Interface) class is created for monitoring the algorithms in real-time as shown in Figure 6.6.

All the algorithms, scripts and simulator models are available at the following repository: [47].

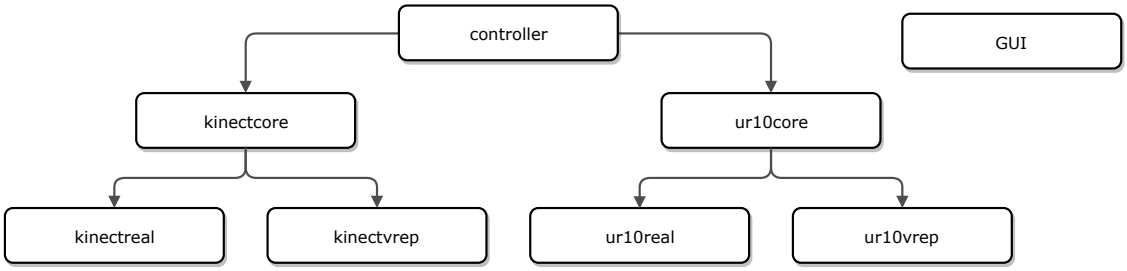


Figure 6.5: Hierarchy of the classes used in this project. Both the ur10core and kinectcore class are designed so that an executed command has the same effect on the real and simulated environment. The controller class contains functions that require both camera and robot data as for example for distance calculations.

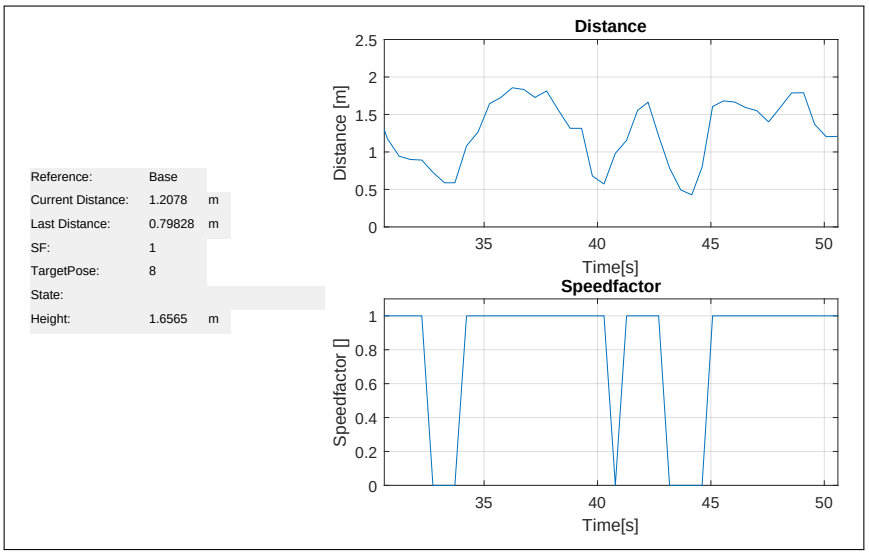


Figure 6.6: Output of the GUI class in MATLAB. The screen visualizes the parameters in real-time and plots the distance and speedfactor of a certain time frame in order to analyze the behavior of the algorithms.

6.4 Safety Rated Monitored Stop

A Safety Rated Monitored Stop as described in Section 5.6 is implemented on the set-up. In this section a flow chart is presented to illustrate the working of the algorithm. Since the workspace of the UR10 is a sphere, the stop distance d_{stop} is set to the radius of the UR10 workspace.

The cycle starts by checking the target and current position of the joints in order to send the next *movej* when a way point is reached. When the robot is not yet at its target, the distance is calculated as described in Section 4.5. The robot will only move when this distance is bigger than the stop distance and will perform a stop when this condition is not true. A flag is added in order to prevent the robot from resending *movej* command since this will lead to unintentional interruptions as described in Section 3.4. Therefore, a new *movej* command is only executed when a new target is initiated or when the robot is stopped.

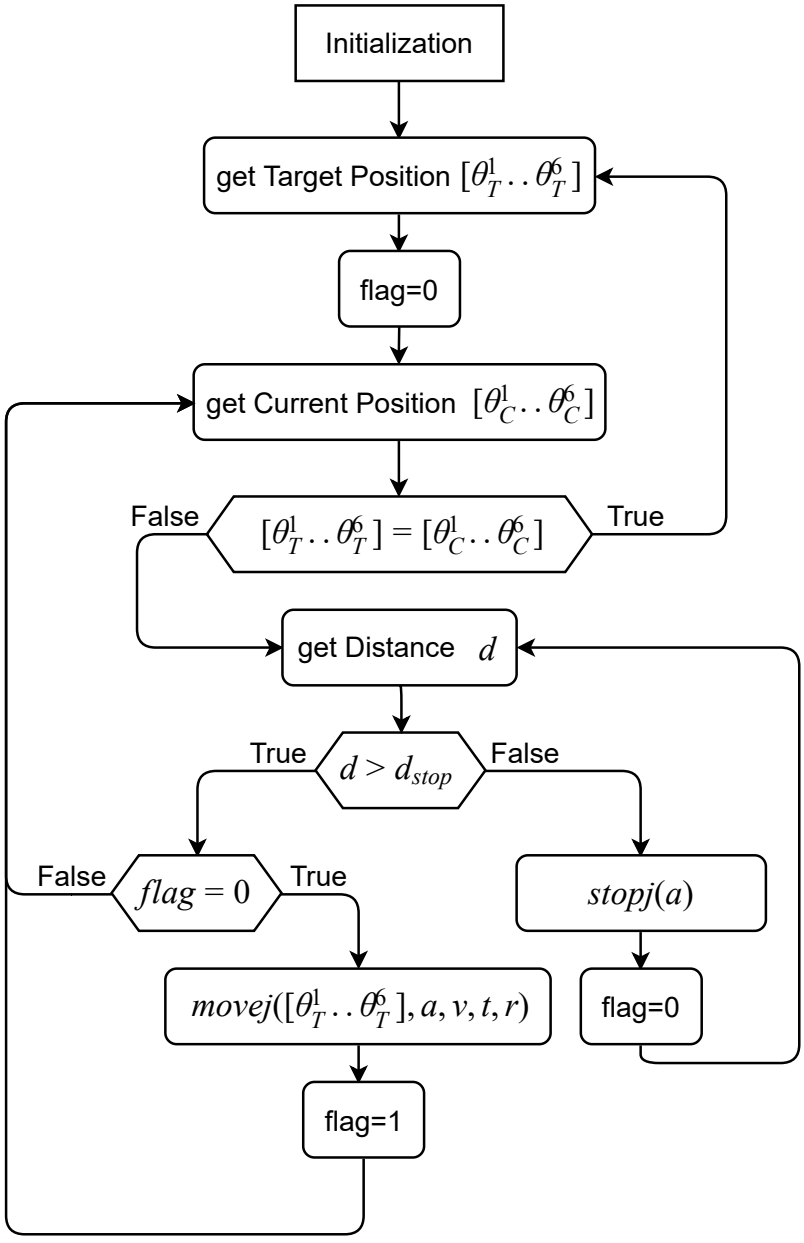


Figure 6.7: Flowchart of the Safety Rated Monitored Stop algorithm.

In Figure 6.8 the output of the algorithm is displayed. A certain distance profile is provided as an input to the algorithm to verify its behavior. Once the distance is lower than the stop distance, the robot will perform a stop and will automatically restart its movements when the distance rises above the stop distance.

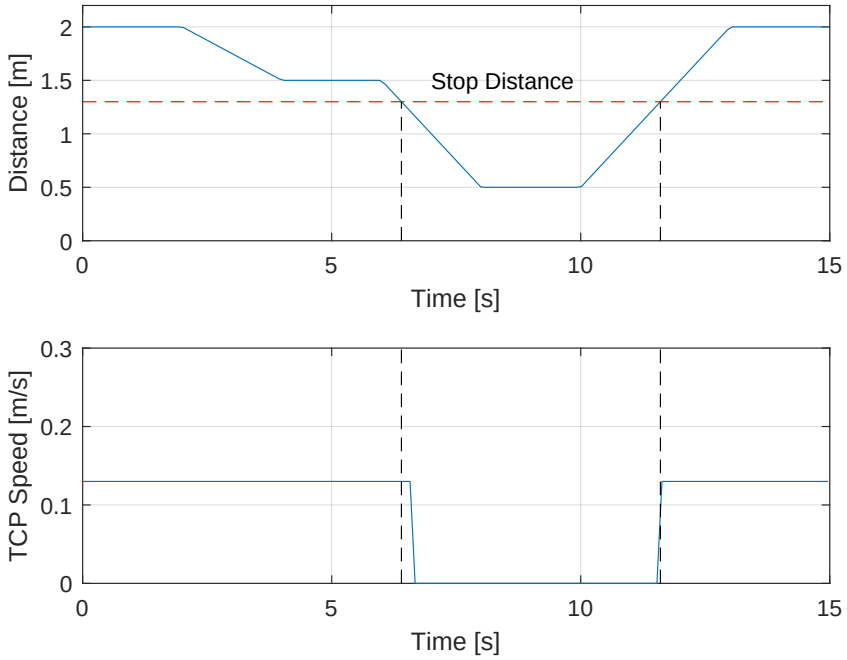


Figure 6.8: Results of the Safety Rated Monitored Stop algorithm with a certain speed profile as an input. The stop distance is set to 1.3m as this is the radius of the workspace of the robot. The robot stops when an object is within the workspace and restart its movement automatically when the object is removed.

6.5 Speed and Separation Monitoring

A Speed and Separation Monitoring as described in Section 5.6 is also implemented on the set-up. The start and stop principles are the same as within the Safety Rated Monitored Stop although an extra speed control is added. The complete flowchart of the algorithm is illustrated in Figure 6.9. Next to the stop distance d_{stop}, a slow distance d_{slow} is added which will create an extra zone where the speed of the robot is proportional to the distance between an object entering the workspace and the robotic arm. A flag is implemented here as well to prevent the robot from resending the *movej* commands.

In Figure 6.10 the output of this algorithm is displayed. The slow distance is set to 1.8m although this should be determined in real applications with a risk assessment. In contrast to results of Section 6.4, the speed of the robot is also controlled to allow slow movements near the workspace of the robot. By narrowing the safety distances of the system, the robot can still move at lower speeds when the operator is near the workspace, increasing the productivity and efficiency of the installation.

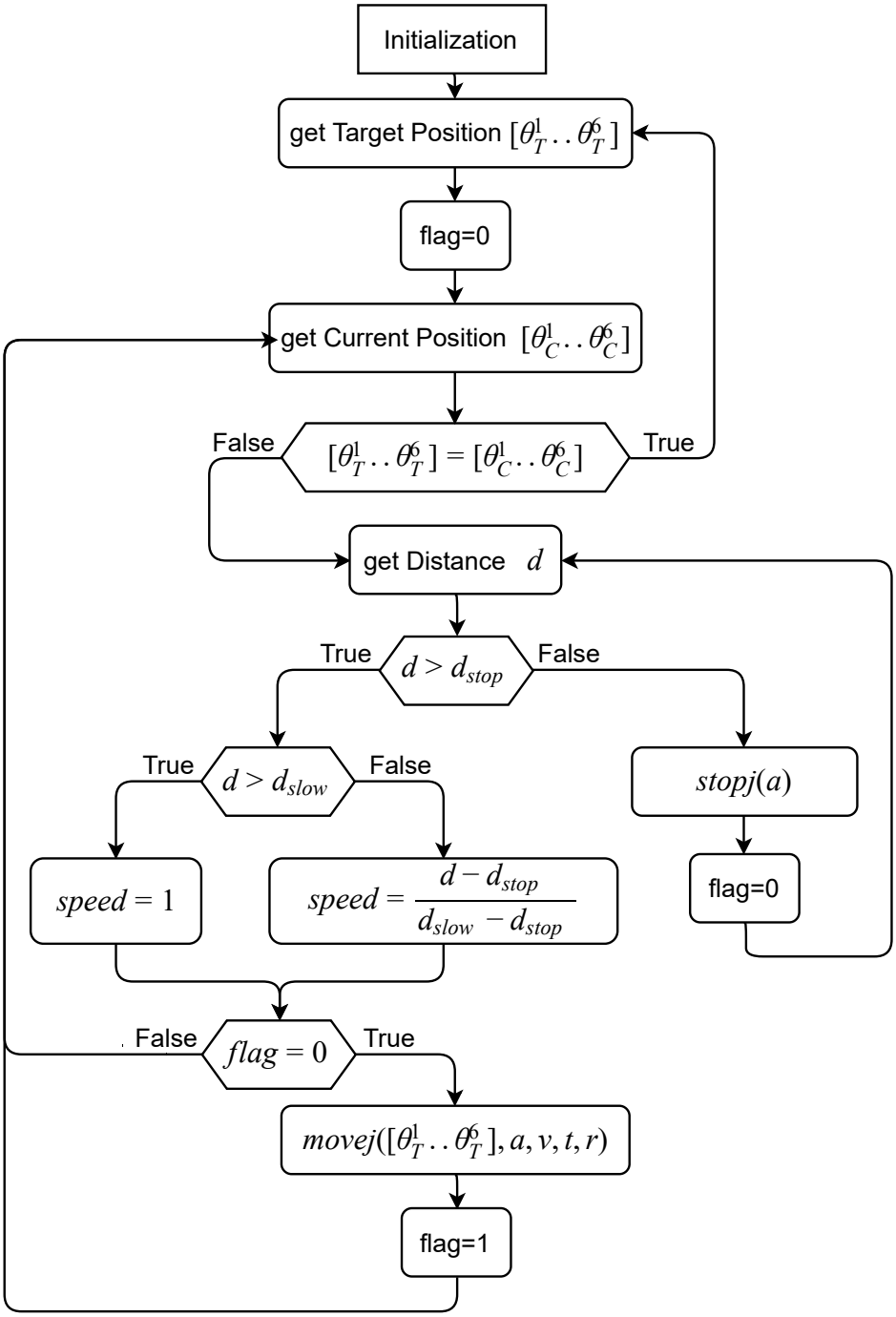


Figure 6.9: Flowchart of the Speed and Separation Monitoring algorithm.

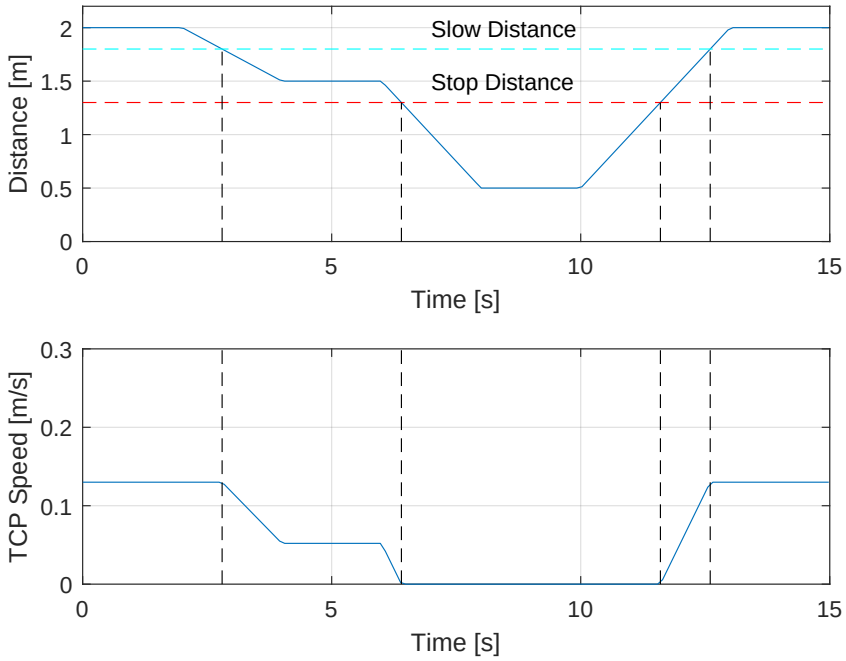


Figure 6.10: Results of the Speed and Separation Monitoring algorithm with a certain speed profile as an input. The stop distance is set to 1.3m and the slow distance to 1.8m. The robot stops when an object is within the workspace and restart its movement when the object is removed. In between the slow and stop distance the speed of the robot is proportional to the distance.

6.6 Pose Recognition

In addition, a more dynamic environment can be created by using the skeleton data for pose recognition and control of the robot. A basic stop sign detection was performed by checking the height of the hand and triggering a protective stop when this height exceeded a certain limit regardless the distance to the robot.

This could be useful in applications where people outside the collaborative workspace can prevent robot movements for various reasons in order to increase the safety of the system. It also gives the operator a faster and more natural way of stopping the robot although the reliability of this function cannot yet be guaranteed. It becomes clear that more research has to be performed to analyze and test other possible configurations.

Chapter 7

Conclusions

Consistent with other studies and the recommendations in the standards, this project shows that a safer human-robot collaboration can be achieved with the help of vision sensors and real-time object tracking. As a proof-of-concept the demo's and tests show that a collaborative workspace can be created with humans and robots safely working next to each other.

The number of impacts and collisions with co-workers can be decreased by making the robot go into a protective stop when someone gets too close. By preventing these collisions, the robots are allowed to handle bigger payloads and move faster, leading to a higher efficiency of the installation.

Besides precautionary stopping the robot, altering the speed of the robot allows movements, even when humans are close to the working area. Therefore, instead of standing still, the robot can still execute tasks at a lower speed resulting in lower process times and a bigger efficiency as well.

Additionally, human tracking opens possibilities for gesture recognition in order to create a more intuitive and dynamic environment. The robot can be stopped with basic stop signs like raising a hand from any position.

Although the results are very promising, the current set-up is not yet safe enough for an industrial implementation. Because only one camera was used, it is impossible to oversee the whole collaborative workspace. Also, the skeleton tracking functionalities provided by the Kinect camera itself can only be used on the real camera and cannot be simulated in the virtual environment.

7.1 Goals and Requirements

In Section 1.2 the goals and requirements of this project were defined. In this section we evaluate every goal and briefly describe what was achieved.

- **Explore the possibilities of human-robot interaction by performing a literature survey.**
  The result of the state-of-the-art on human-robot collaboration is given in Chapter 5. There are several approaches for a safe human-robot collaboration like a Safety Rated Monitored Stop and a Speed and Separation Monitoring.

- **Create a kinematic model of the robot which allows the robot to be driven in both joint and cartesian space.**
  The UR10 is modelled using the Denavit-Hartenberg representation. With this model it is possible to solve the forward kinematics of the robot. A closed-form inverse kinematic solution is also provided to control the robot in joint and cartesian space.

- **Achieve a safe (speed) control over the UR10 cobot by setting up a communication between MATLAB and the controller in order to get full access of the robot.**
  The ROS middleware is used for the communication between MATLAB and the robot controller. Therefore, URScript commands can be send to the robot and the speed can be altered with the speed slider.

- **Process data from Kinect in real-time and create a skeleton model of the human for distance calculations and basic human pose recognition.**
  MATLAB communicates with the Kinect through the Op3Mech camera toolbox in order to retrieve the point cloud and skeleton data.

- **To ensure the different algorithms are safe to use before implementing it on the robot, an available 3D virtual test environment (V-REP) needs to be configured in order to give a visual representation of the set-up and show a correct working of the algorithms.**
  The simulation of the set-up in VREP shows a very close correlation with the real-life situation, making it adequate for testing the different algorithms.

- **Implement a safety rated monitored stop and speed and separation monitoring on the real set-up in the robotics lab with real-time object tracking as a proof-of-concept.**
  A set-up with a safety rated monitored stop and a speed and separation monitoring is created to demonstrate the potential of human-robot collaboration.

- **Interpret a basic human pose command in order to create a more dynamic environment and add flexibility to the configuration.**
  The robot can be stopped from multiple distances simply by raising the hand in the air even when the worker is outside the collaborative workspace.

7.2 Future Work

Although the main goal of this project was achieved, there are still some aspects that can be improved or further elaborated. Human-robot collaboration is a study area on itself and there are still many features that can be further explored. A few side projects were already performed in order to investigate the possibilities for further work on the set-up. In the next paragraph some improvements of recommendations for eventual projects are discussed.

- For this project only one Kinect camera was used to oversee the collaborative workspace. It is possible to use multiple cameras in order to increase the performance of the skeleton tracking and eliminate dead spots. It is also possible to experiment with other 3D cameras to evaluate the performance of different camera types for human-robot collaboration.

- Although the robot's movement can be accurately simulated in VREP, the skeleton tracking can only tested with the real Kinect since the skeleton processing is performed on the camera itself. Therefore, an alternative skeleton tracker could be used which is compatible with the VREP software or uses the raw point cloud data to perform its calculations. The performance and reliability of these algorithms are also topics that could be further elaborated.

- Since the skeleton tracking is already used to track certain joints of the skeleton, this data could be stored in order to predict the movement of the human worker and use this as an input for the existing algorithms. In a side project the possibility of movement prediction was already investigated although there are still a lot of improvement that can be done. In Figure 7.1 the result of the movement prediction project is illustrated.

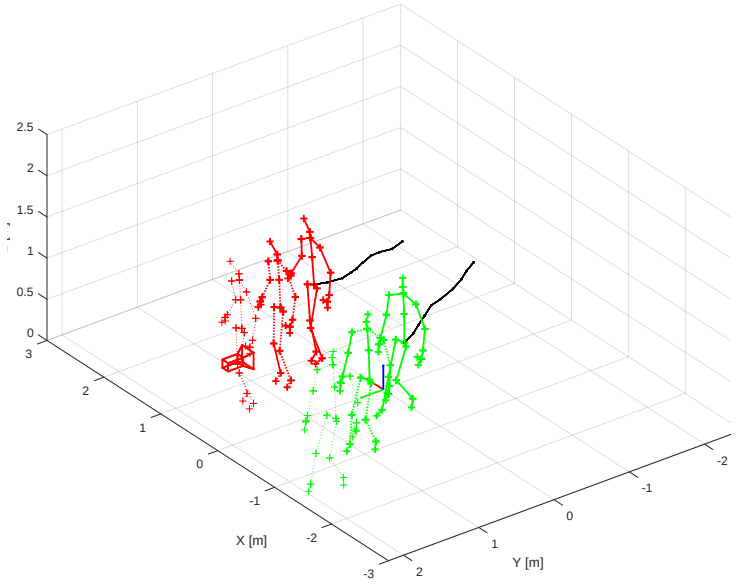


Figure 7.1: Result of a skeleton prediction based and logged joint data for multiple people. The location of every joint is extrapolated in time in order to predict future movements. The skeletons in the figure are the predicted positions with a 1 second interval between them.

- Another possibility is to create a more dynamic environment by implementing pose and gesture recognition on the skeleton data. A basic stop sign recognition was already implemented although many other features can be investigated. A set-up where the robot follows the mimics hand movements is also created although more research needs to be performed.

- This project served as a proof-of-concept and is not yet ready to implement in a real industrial environment. Therefore, more industrial equipment needs to be installed which has better performance and has passed reliability tests to serve as safety equipment. There were already some tests executed with the UR10 controlled by an industrial PLC through ProfiNET although this was not yet compatible with the Kinect camera.

- In order to reduce the cycle time and lag of the algorithms, a more low-level programming language like C++ could be used in order to decrease the cycle time and increase the performance. Both the camera and robot can be controlled in C++ which is one of the reasons why the ROS middleware was used for communication.

Bibliography

[1] W. Robotics, "Executive Summary World Robotics 2017 Industrial Robots," *World Robotic Report*, pp. 15–24, 2017. [Online]. Available: https://ifr.org/downloads/press/Executive{_}Summary{_}WR{_}2017{_}Industrial{_}Robots.pdf

[2] J. Krüger, T. K. Lien, and A. Verl, "Cooperation of human and machines in assembly lines," *CIRP Annals - Manufacturing Technology*, vol. 58, no. 2, pp. 628–646, 2009.

[3] E. H. Østergaard, "White Paper the Role of Cobots," 2017. [Online]. Available: https://info.universal-robots.com/the-role-of-cobots-in-industry4.0

[4] Universal Robots, "Home Page," 2018. [Online]. Available: https://www.universal-robots.com

[5] University of Antwerp, "Op3Mech," 2018. [Online]. Available: https://www.uantwerpen.be/en/research-groups/op3mech/

[6] Microsoft, "Kinect for Xbox One," 2017. [Online]. Available: https://www.xbox.com/en-US/xbox-one/accessories/kinect

[7] K. Khoshelham and S. O. Elberink, "Accuracy and resolution of kinect depth data for indoor mapping applications," *Sensors*, vol. 12, no. 2, pp. 1437–1454, 2012.

[8] O. Wasenmüller and D. Stricker, "Comparison of kinect v1 and v2 depth images in terms of accuracy and precision," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10117 LNCS, pp. 34–45, 2017.

[9] K. Litomisky, "Consumer RGB-D Cameras and their Applications," University of California, Riverside, Tech. Rep., 2012. [Online]. Available: http://alumni.cs.ucr.edu/{~}klitomis/files/RGBD-intro.pdf

[10] Microsoft, "Coordinate mapping," 2014. [Online]. Available: https://docs.microsoft.com/en-us/previous-versions/windows/kinect/dn785530(v=ieb.10)

[11] E. Lachat, H. Macher, T. Landes, and P. Grussenmeyer, "Assessment and calibration of a RGB-D camera (Kinect v2 Sensor) towards a potential use for close-range 3D modeling," *Remote Sensing*, vol. 7, no. 10, pp. 13 070–13 097, 2015.

[12] A. Corti, S. Giancola, G. Mainetti, and R. Sala, "A metrological characterization of the Kinect V2 time-of-flight camera," *Robotics and Autonomous Systems*, vol. 75, pp. 584–594, 2016. [Online]. Available: http://dx.doi.org/10.1016/j.robot.2015.09.024

[13] L. Li, "Time-of-Flight Camera An Introduction," *Texas Instruments - Technical White Paper*, no. January, p. 10, 2014.

[14] M. Hansard, S. Lee, O. Choi, R. Horaud, M. Hansard, S. Lee, O. Choi, R. Horaud, and F. Cameras, *Time of Flight Cameras : Principles , Methods , and Applications.* Seoul: HAL, 2012.

[15] D. Pagliari and L. Pinto, "Calibration of Kinect for Xbox One and comparison between the two generations of microsoft sensors," *Sensors (Switzerland)*, vol. 15, no. 11, pp. 27 569–27 589, 2015.

[16] L. Valgma, "3D Reconstruction Using Kinect V2 Sensor," p. 41, 2016.

[17] MathWorks, "Skeleton Viewer for Kinect V2 Skeletal Data," 2018. [Online]. Available: https://nl.mathworks.com/help/supportpkg/kinectforwindowsruntime/examples/plot-skeletons-with-the-kinect-v2.html

[18] K.-Y. Yeung, T.-H. Kwok, and C. C. L. Wang, "Improved Skeleton Tracking by Duplex Kinects: A Practical Approach for Real-Time Applications," *Journal of Computing and Information Science in Engineering*, vol. 13, no. 4, p. 041007, 2013. [Online]. Available: http://computingengineering.asmedigitalcollection.asme.org/article.aspx?doi=10.1115/1.4025404

[19] C. Lenz, M. Grimm, T. Röder, and A. Knoll, "Fusing multiple Kinects to survey shared Human-Robot-Workspaces," *Technische Universität München, Munich, Germany, Tech. Rep. TUM-I1214*, 2012.

[20] M. R. de Gier, "Control of a robotic arm - Application to on-surface 3D-printing," *Master@Delft University of Technology*, vol. 13, pp. 1–13, 2009.

[21] Universal Robots, "User Manual UR10," Odense, p. 189, 2015.

[22] T. Timm, "Optimizing the Universal Robots ROS driver," Technical University of Denmark, Department of Electrical Engineering., Kongens Lyngby, Tech. Rep., 2015.

[23] N. Axel, T. Timm, N. A. Andersen, and T. T. Andersen, "UR10 Performance Analysis," Technical University of Denmark, Department of Electrical Engineering., Kongens Lyngby, Tech. Rep., 2014.

[24] ISO, "Safety of machinery - Safety-related parts of control systems," *ISO 138491-1:2015*, pp. 1–102, 2015.

[25] F. Reuleaux, *The Kinematics of Machinery.* New York: Dover, 1876.

[26] J. Hartenberg and R. Denavit, *Kinematic Synthesis of Linkages.* New York: McGraw-Hill, 1964.

[27] M. W. Spong, S. Hutchinson, and M. Vidyasagar, "FORWARD KINEMATICS: THE DENAVIT-HARTENBERG CONVENTION BT - Robot Dynamics and Control," *Robot Dynamics and Control*, pp. 1–32, 2003. [Online]. Available: papers2://publication/uuid/E7AC27CB-8AAC-4755-88F2-002FB8929672

[28] S. Kucuk and Z. Bingul, *Industrial Robotics: Theory, Modelling and Control, Sam Cubero (Ed.)*, 2006, no. December. [Online]. Available: http://www.intechopen.com/source/pdfs/379/InTech-Robot{_}kinematics{_}forward{_}and{_}inverse{_}kinematics.pdf{%}5Cnhttp://www.intechopen.com/books/industrial{_}robotics{_}theory{_}modelling{_}and{_}control

[29] J. J. Craig, "Introduction to Robotics: Mechanics and Control 3rd," *Prentice Hall*, vol. 1, no. 3, p. 408, 2004.

[30] R. N. Jazar, *Theory of Applied Robotics*, 2010, vol. 1. [Online]. Available: http://link.springer.com/10.1007/978-1-4419-1750-8

[31] I. Bonev, "Universal Robots Inverse kinematics 8 solutions," 2014. [Online]. Available: https://twitter.com/universal{_}robot/status/516682281155829760

[32] K. P. Hawkins, "Analytic Inverse Kinematics for the Universal Robots," pp. 1–5, 2013. [Online]. Available: http://hdl.handle.net/1853/50782

[33] J. Kay, "Introduction to Homogeneous Transformations & Robot Kinematics," *Rowan University Computer Science Department*, no. January, pp. 1–25, 2005. [Online]. Available: http://scholar.google.com/scholar?hl=en{&}btnG=Search{&}q=intitle:Introduction+to+Homogeneous+Transformations+{&}+Robot+Kinematics{#}0

[34] E. Smet, "Industriële robots," in *Industriële robots: Capita Selecta deel 1/2*. Schoten: Uitgeverij Universitas, 2017, ch. 1, pp. 41–53.

[35] T. Bajd, M. Mihelj, and J. Lenarcic, "Homogeneous Transformation Matrices," in *Robotics*. Springer Netherlands, 2010, ch. 2, pp. 9–22.

[36] F. Flacco, T. Kroeger, A. De Luca, and O. Khatib, "A Depth Space Approach for Evaluating Distance to Objects: with Application to Human-Robot Collision Avoidance," *Journal of Intelligent and Robotic Systems: Theory and Applications*, vol. 80, pp. 7–22, 2015.

[37] T. KEREZOVIĆ, G. SZIEBIG, B. SOLVANG, and T. LATINOVIC, "Human Safety in Robot ApplicationsReview of Safety Trends," *Acta Technica Corviniensis*, vol. 7, no. August 2014, pp. 113–118, 2013. [Online]. Available: http://acta.fih.upt.ro/pdf/2013-4/ACTA-2013-4-20.pdf

[38] N. Pedrocchi, F. Vicentini, M. Malosio, and L. M. Tosatti, "Safe human-robot cooperation in an industrial environment," *International Journal of Advanced Robotic Systems*, vol. 10, 2013.

[39] I. Asimov, *I , Robot*. New York City: Gnome Press, 1970.

[40] Sick, "Safe Robotics Transfer: Productivity Restarts Automatically," dec 2016.

[41] ——, "Guide for Safe Machinery: SIX STEPS TO A SAFE MACHINE," 2015. [Online]. Available: https://www.sick.com/media/docs/8/78/678/Special{_}information{_}Guide{_}for{_}Safe{_}Machinery{_}en{_}IM0014678.PDF

[42] ISO, "Robots and robotic devices - Safety requirements for industrial robots - Part 1: Robots," *ISO 10218-1:2011*, pp. 1–56, 2011.

[43] ——, "Robots and robotic devices - Safety requirements for industrial robots - Part 2: Robot systems and integration," *ISO 10218-2:2011*, pp. 1–86, 2011.

[44] R. N. Shea, "Collaborative Robot Technical Specification ISO/TS 15066 Update," *International Collaborative Robots Workshop*, 2016. [Online]. Available: http://me.umn.edu/courses/me5286/robotlab/Resources/12-TR15066Overview-SafetyforCollaborativeApplications-RobertaNelsonShea.pdf

[45] M. Bjorn and ABB Corporate Research, "Industrial Safety Requirements for Collaborative Robots and Applications," *ERF 2014 Workshop: Workspace Safety in Industrial Robotics: trends, integration and standards*, 2014.

[46] N. A. M. E and K. Veladri, "Modeling and Simulation of 7-dof Robotic Manipulator," no. August, 2016.

[47] N. V. Oosterwyck, "Repository Master Project," 2018. [Online]. Available: https://github.com/NickVanOosterwyck/Master-Project

[48] Universal Robots, "The URScript Programming Language," p. 30, 2015. [Online]. Available: http://www.wmv-robotics.de/home{_}htm{_}files/scriptmanual{_}en{_}1.5.pdf

Nomenclature

2D	Two Dimensional
3D	Three Dimensional
AI	Artificial Intelligence
Cobot	Collaborative Robot
CW	Continuous Wave
DH	Denavit-Hartenberg
DOF	Degree Of Freedom
GUI	Graphical User Interface
IoT	Internet of Things
IR	Infrared
ISO	International Organization for Standardization
PI	Proportional Integral
RGB-D	Red Green Blue - Depth
SDK	Software Development Kit
ToF	Time-Of-Flight
VREP	Virtual Robot Experimentation Platform
WCS	World Coordinate System

List of Figures

List of Tables

Appendices

Appendix A

Supplementary plots robot

This chapter provides the supplementary plots about speed control, stopping and accuracy. Results of both position and TCP speed are presented for joint two to six. Every joint is controlled individually to eliminate the influence of side-effects. All tests are performed on the set-up in the robotics lab and simulator as shown in Figure A.1 and plots are created with MATLAB. The robot movement is started from the home-positions as shown in Figure A.1 as well.

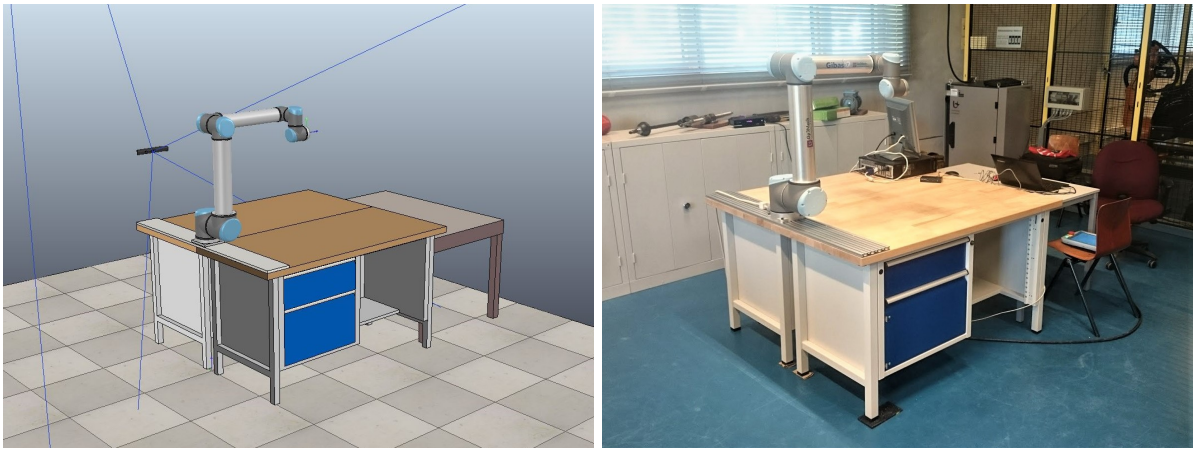


Figure A.1: Illustration of the set-up in VREP (left) and the robotics lab at campus Groenenborger (right). Robot is displayed in his home-position.

A.1 Speed Control

The robot is commanded to turn along one axis after one second and its speed is decreased with 10% every second. The speed slider offers a smoother path and causes less disruptions in the speed pattern.

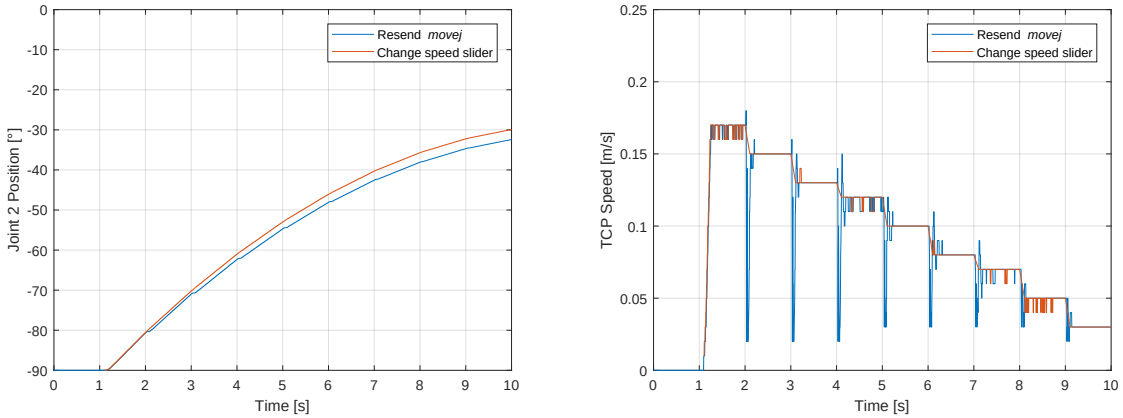


Figure A.2: Result of a speed changes when the robot is executing a path by resending *movej* command or using the speed slider on joint 2.

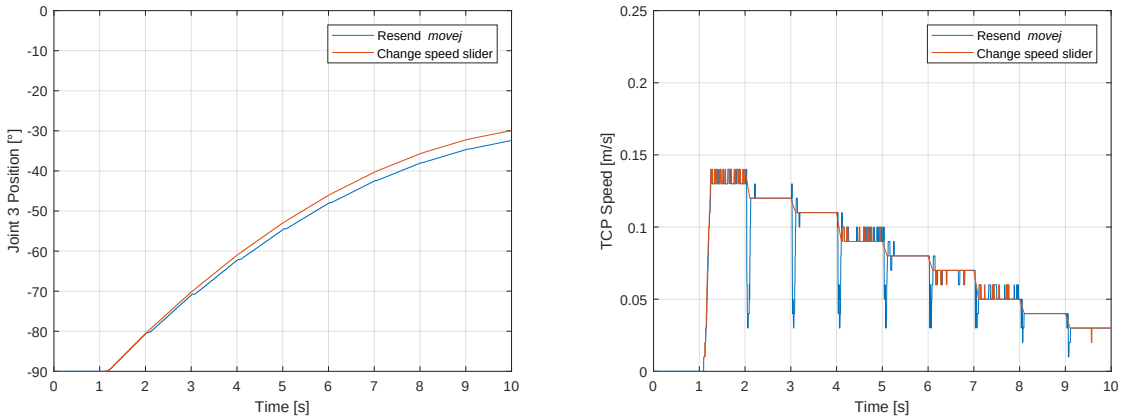


Figure A.3: Result of a speed changes when the robot is executing a path by resending *movej* command or using the speed slider on joint 3

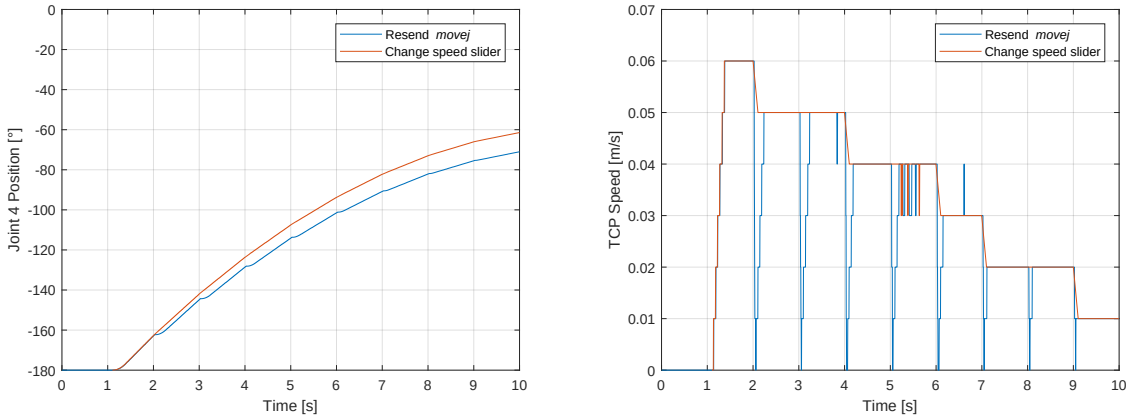


Figure A.4: Result of a speed changes when the robot is executing a path by resending *movej* command or using the speed slider on joint 4.

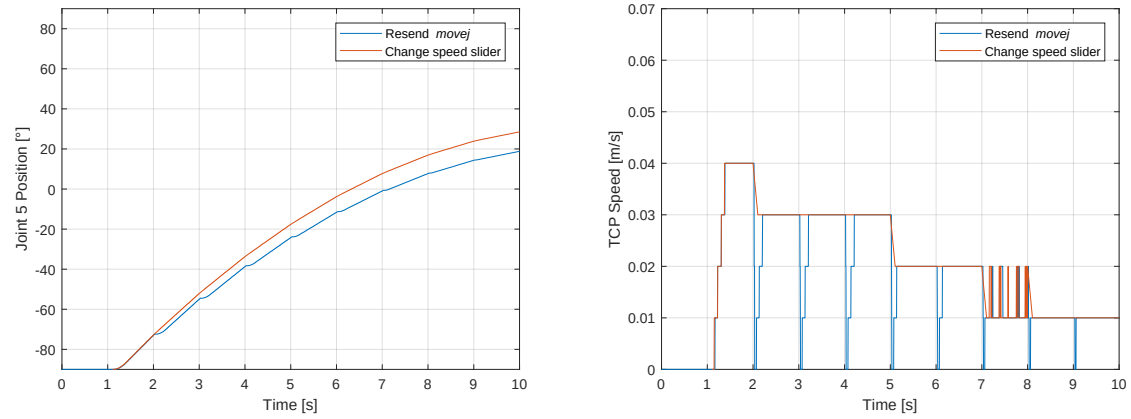


Figure A.5: Result of a speed changes when the robot is executing a path by resending *movej* command or using the speed slider on joint 5.

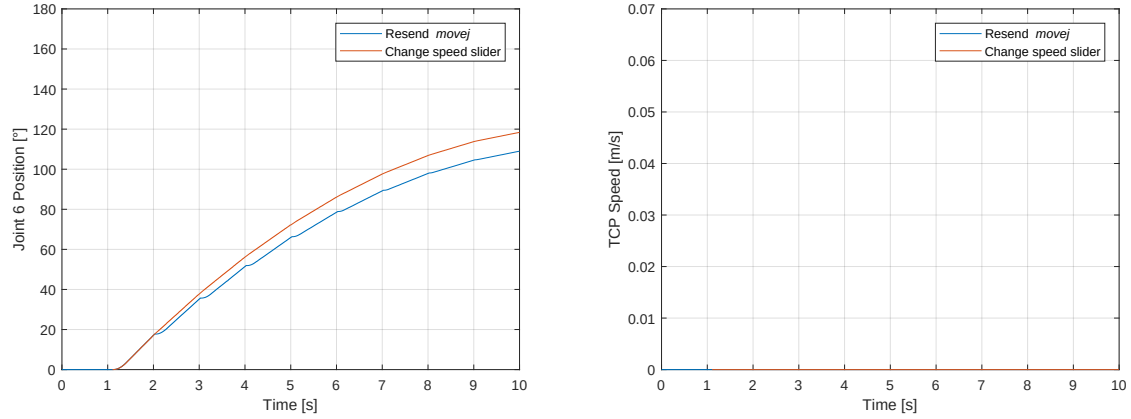


Figure A.6: Result of a speed changes when the robot is executing a path by resending *movej* command or using the speed slider on joint 6. The TCP is zero since a rotation around the sixth axis does not cause a movement of the TCP.

A.2 Stopping

The robot is commanded to turn along one axis after one second and is commanded to stop after 3 seconds. The robot comes faster to a standstill more with the *stopj* command.

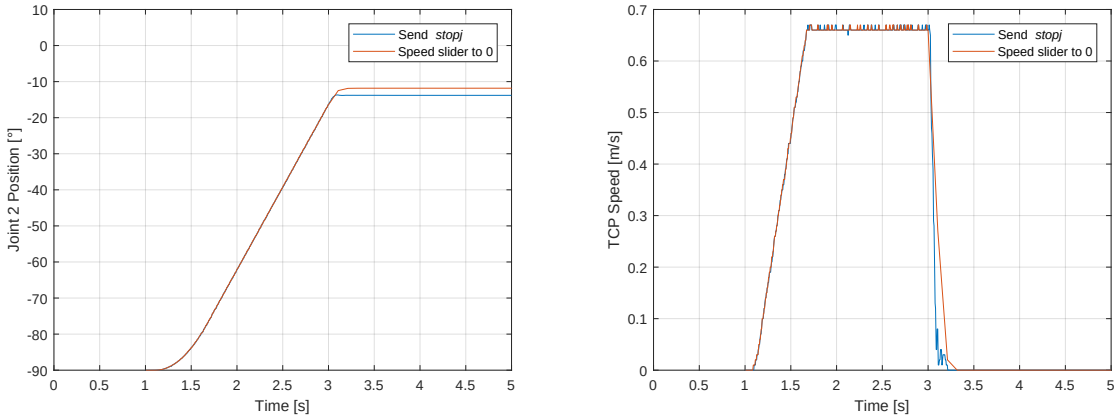


Figure A.7: Result of a stop instruction with the speed slider and *stopj* command when the robot is executing a path on joint 2.

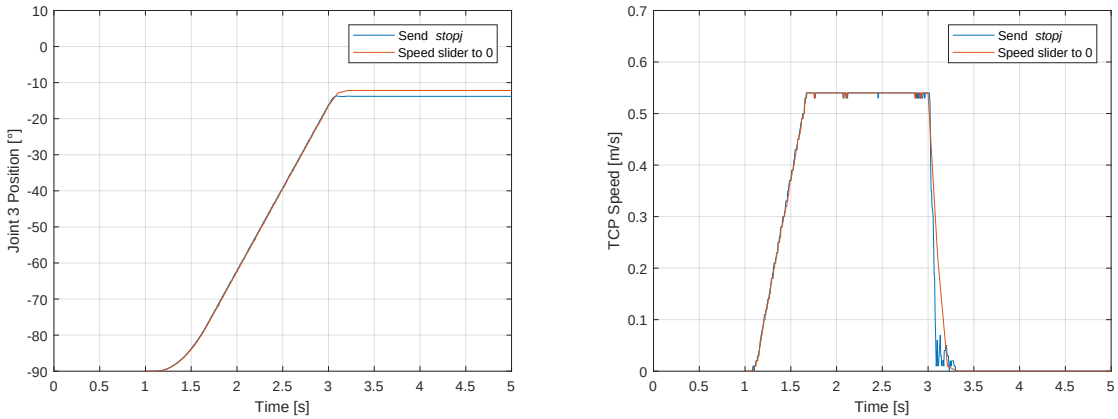


Figure A.8: Result of a stop instruction with the speed slider and *stopj* command when the robot is executing a path on joint 3.

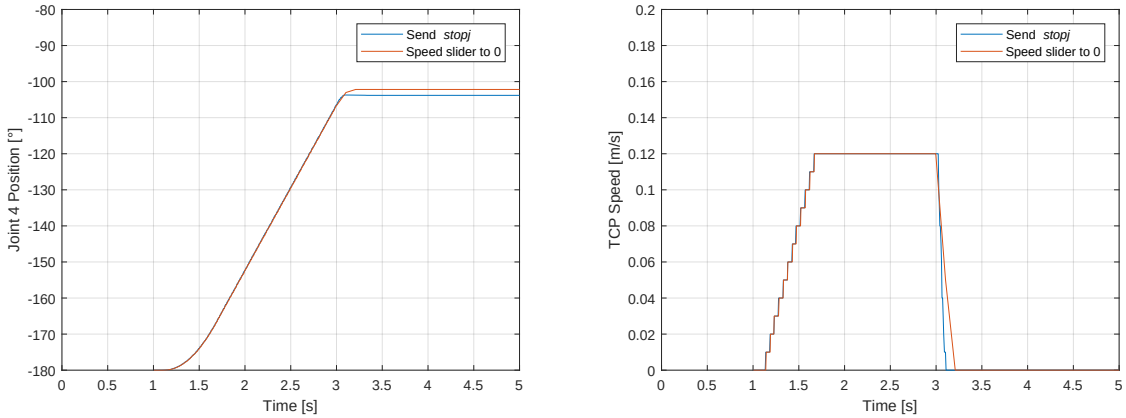


Figure A.9: Result of a stop instruction with the speed slider and *stopj* command when the robot is executing a path on joint 4.

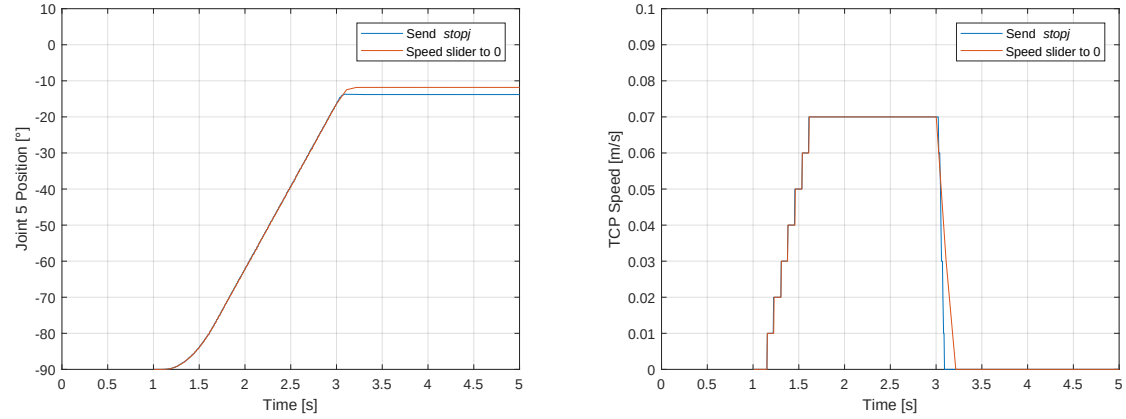


Figure A.10: Result of a stop instruction with the speed slider and *stopj* command when the robot is executing a path on joint 5.

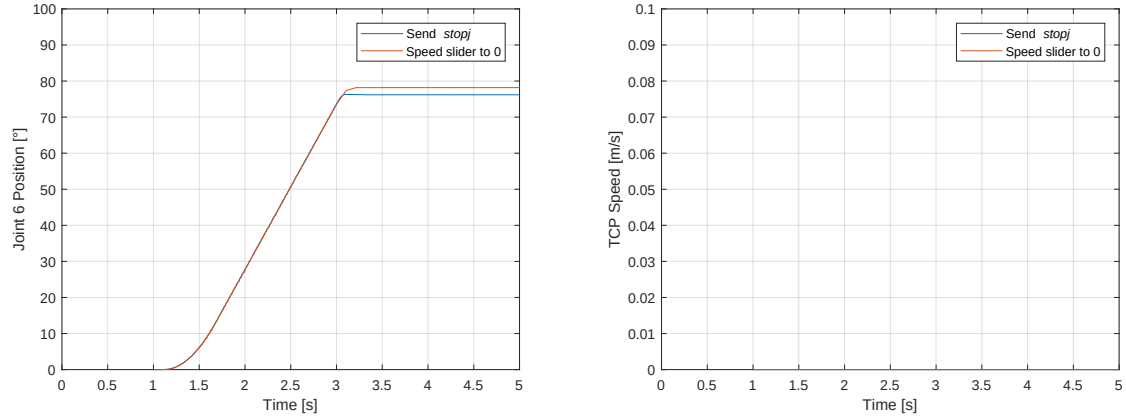


Figure A.11: Result of a stop instruction with the speed slider and *stopj* command when the robot is executing a path on joint 6. The TCP is zero since a rotation around the sixth axis does not cause a movement of the TCP.

A.3 Simulator

To verify the accuracy of the speed control, a *movej* command is send to both the simulator and real robot. The VREP scene is executed in real-time mode in order to mimic a real environment as good as possible. It is clear that the simulated robot moves very similar to the real robot and is therefore suited to verify the various algorithms.

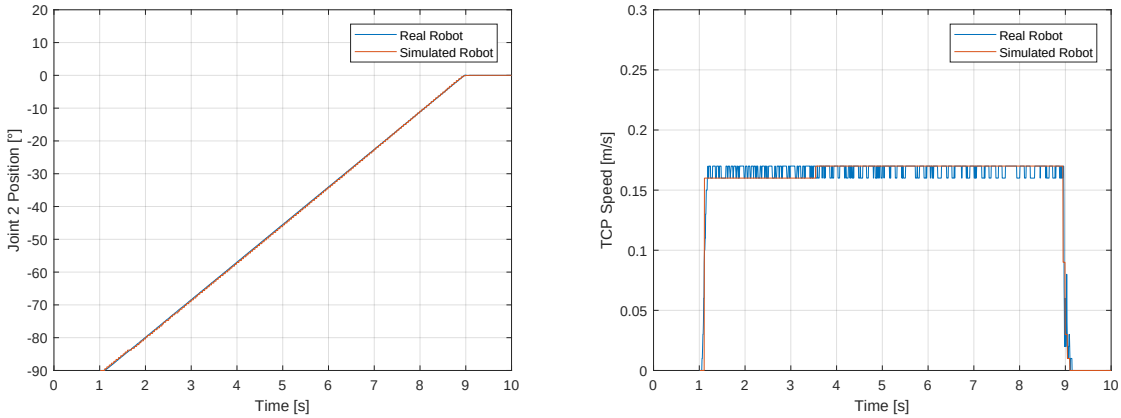


Figure A.12: Comparison of the angular position of joint 2 between the real and simulated robot.

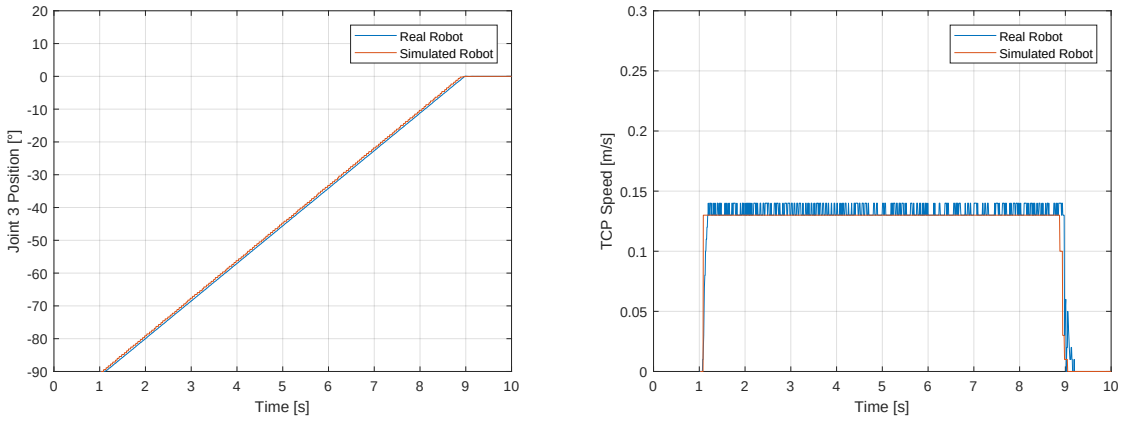


Figure A.13: Comparison of the angular position of joint 3 between the real and simulated robot.

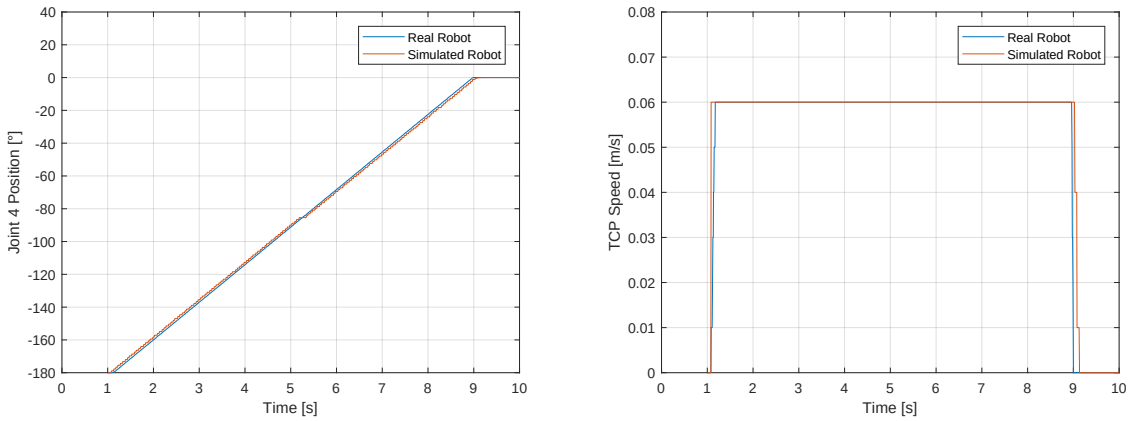


Figure A.14: Comparison of the angular position of joint 4 between the real and simulated robot.

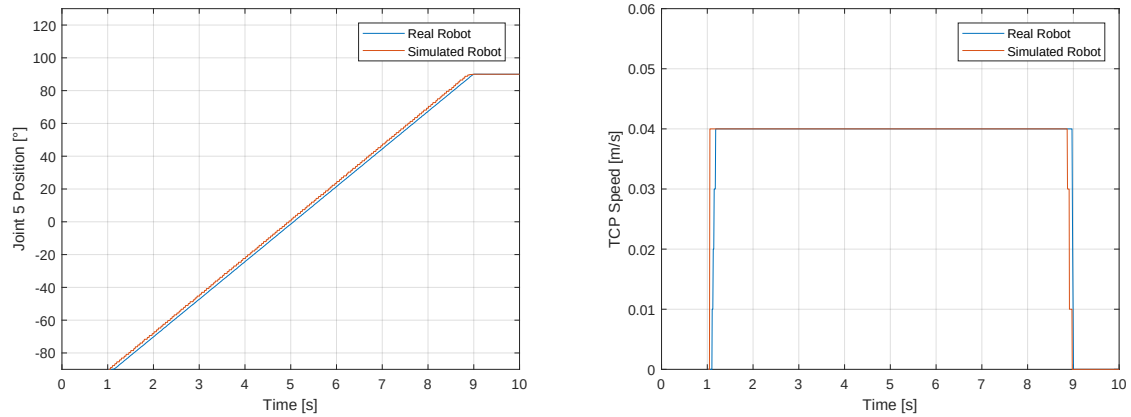


Figure A.15: Comparison of the angular position of joint 5 between the real and simulated robot.

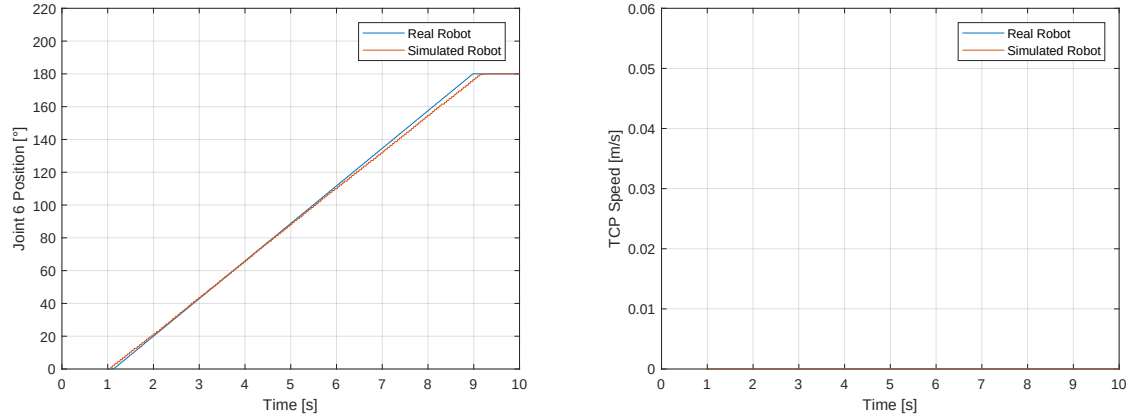


Figure A.16: Comparison of the angular position of joint 6 between the real and simulated robot. The TCP is zero since a rotation around the sixth axis does not cause a movement of the TCP.

Appendix B

URscript commands

Function description of the applied URscript commands in this project. Explanation is originating from the URScript manual [48].

stopj(*a*)

Stop (linear in joint space)

Decelerate joint speeds to zero

Parameters

`a`: joint acceleration (rad/s^2) (of leading axis)

Example command: `stopj(2)`

- Example Parameters:
  - a = 2 rad/s^2 $\rightarrow$ rate of deceleration of the leading axis.

Figure B.1: Function description of the URscript stopj command.

movej(*q*, *a*=`1.4`, *v*=`1.05`, *t*=`0`, *r*=`0`)

Move to position (linear in joint-space)

When using this command, the robot must be at a standstill or come from a movej or movel with a blend. The speed and acceleration parameters control the trapezoid speed profile of the move. Alternatively, the `t` parameter can be used to set the time for this move. Time setting has priority over speed and acceleration settings.

Parameters

`q`: joint positions (q can also be specified as a pose, then inverse kinematics is used to calculate the corresponding joint positions)

`a`: joint acceleration of leading axis (rad/s^2)

`v`: joint speed of leading axis (rad/s)

`t`: time (S)

`r`: blend radius (m)

If a blend radius is set, the robot arm trajectory will be modified to avoid the robot stopping at the point.

However, if the blend region of this move overlaps with the blend radius of previous or following waypoints, this move will be skipped, and an 'Overlapping Blends' warning message will be generated.

Example command: `movej([0,1.57,-1.57,3.14,-1.57,1.57], a=1.4, v=1.05, t=0, r=0)`

- Example Parameters:
  - q = (0,1.57,-1.57,3.14,-1.57,1.57) → base is at 0 deg rotation, shoulder is at 90 deg rotation, elbow is at -90 deg rotation, wrist 1 is at 180 deg rotation, wrist 2 is at -90 deg rotation, wrist 3 is at 90 deg rotation. Note: joint positions (q can also be specified as a pose, then inverse kinematics is used to calculate the corresponding joint positions)
  - a = 1.4 → acceleration is 1.4 rad/s/s
  - v = 1.05 → velocity is 1.05 rad/s
  - t = 0 → the time (seconds) to make move is not specified. If it were specified the command would ignore the a and v values.
  - r = 0 → the blend radius is zero meters.

Figure B.2: Function description of the URscript movej command.