UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Industrial

Estudio y propuesta de mejoras del control de fuerza con el robot colaborativo UR3

Trabajo Fin de Máster

Máster Universitario en Ingeniería Industrial

AUTOR/A: de Smedt , Jarne

Tutor/a: Zotovic Stanisic, Ranko

CURSO ACADÉMICO: 2021/2022

Table of Contents

Table of Figures

Table of Tables

Table of Abbreviations

1. Abstract

Robots and automation have proven themselves to be valuable investments that have a good ROI. When investigating the value of these tools it is important to know that apart from the ROI, they are useful because they increase the flexibility of processes, it is possible to work autonomously, and the repeatability is high. Robots and most of automation technology have the advantage to be usable in different applications, but at the same time this imposes a challenge for them to be as effective as specialized equipment. This is where research enters the picture.

This research on robots and more specifically collaborative robots is done within the scope of a bigger project of professor and coordinator (Ranko Zotovic Stanisic). This big project serves as a contribution to the field of robotics. For several years research on force control with robots has been done, while adding new elements to it, with robots that are not specifically designed for this task and to investigate how and if it is possible to compensate their shortcomings and use them accurately.

The main goal of this Master Thesis is to examine the force control of a collaborative robot. This is very broad and allows to execute multiple different experiments with different goals to gather information, even if the current experiment is not a continuation of the previous one.

The first big goal is to distinguish the difference between the UR3 and UR3e collaborative robot, using force control tasks. The motion commands are programmed on the UR Teach Pendant module and information is sent to a computer using a Python program. This is first done by communicating with the robot and later directly with the force sensor.
In another experiment the two robots were to create a multi-robot communication program, where the robots are able to communicate with each other and synchronize their movements to pick up a box with force control.

The second goal is to do research and conduct experiments on inner/outer loop control of the robot in force control applications. Since it is possible to establish a direct UDP connection between the robot and the force sensor, it is possible to apply different controllers and filters and create an inner loop motion control with an accompanying outer loop force control.

The third goal is on machining with robots, because this is part of the professor's big project a few students are working on this. It is my task to do research on robot machining and milling and, at the end, compare my results and use my knowledge to evaluate these students' outcomes.

A try on remote control of the robot was done but is yet to be solved.

Finally, an experimental setup is created to perform and evaluate the UR3's impedance control, where the interaction control of the robot is tested.

2. Planos

2.1. Project roadmap:

This master thesis was a complete project for me, I had learned about automation and servomotors, so I am familiar with control loops, block diagrams and some of the principles, but I started off with no knowledge of robotics. For this reason, an extensive literature study on robotics has been done, as well as following two courses on the subject; 'Robotics and mechanisms' from my home university in Belgium and 'Mobile Robotics' here at UPV.
Apart from robotics in general, knowledge about coding was needed. I had experience with Java and a basic understanding of MATLAB, but Python was new to me.

This is why the first experiments are not significant but mentionable in this master thesis, they provide the base for further experiments and illustrate the progress made throughout this semester.

2.2. Methodology:

Do literature study and go to the lab. At first learn to use the robot, write in Python, and read sensor values.

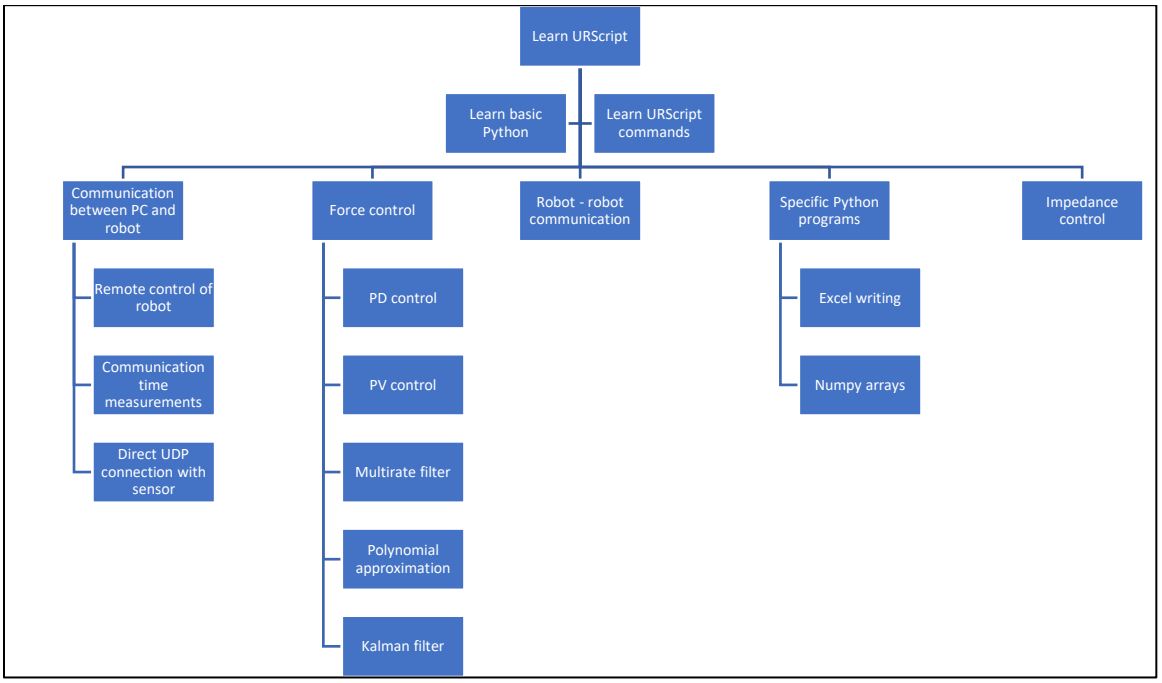


Figure 1: Overview of conducted experiments

Using the outcome of these experiments, the research from the literature study can serve as comparison.
While progress was made in the laboratory, it was possible to create more advanced experiments and the literature study gave inspiration for new experiments.

At the start, in the laboratory, I worked together with Raúl Alfonso Safont (Master in automation and industrial informatics [DISA]) and Ricardo Ruiz Monsalve (Master in mechatronics engineering [ETSID]) to advance more quickly. Especially when getting to know the robot, because this stage is important for later but not valuable for the thesis or research.
During the project we all worked on our own research field and when we were in the lab together it was possible for me to add or improve their code and vice versa. It was an individually challenging

assignment, with each pursuing their own objectives, but it was possible to refine programs together or set up a foundation for more complicated tasks.

3. State of the art

State of the art can be defined as the current way of working, what is generally accepted as good practice. It is worth mentioning what is possible right now and is known to be efficient with the techniques and technologies used in this thesis. With this knowledge in mind, it is possible to start exploring alternative ways or combining current methods to find a solution to the obstacles when using force control with a collaborative robot.

Robots are needed in today's industrial environment, for companies to be competitive they need to innovate in order to maximize their efficiency. Today's market can be identified by quickly changing demands, needs, more customized products and a higher variety in general. This and because a lot of companies are engaged in various industries and applications, adaptivity and flexibility are strived for. If equipment such as robots is non-specialized it can easily be reconfigured and used in other applications, if necessary, it even allows for planning and tool-switching, which increases the flexibility all the more.
Robots are already used for low-impact tasks such as pick-and-place tasks, welding, packaging, painting, … because of their flexibility and the low complexity in environmental interaction, but the study of general robots in milling applications is new. This is because milling requires force control, to compensate the external forces that occur while machining and requires position control, because the tool needs to follow a desired trajectory. Specialized equipment is more often used for these applications because of their increased stiffness and adaptive approach. Robot parameters are improved through hardware modifications, resulting in a higher performance for the specific application.

This is only a subdivision of the bigger scope in this research, force control is the main subject tackled. Due to the complexity in interaction between workpiece and robot, the quickly changing dynamics of the robots and low accuracy of the pre-designed force control mode, a margin of error exists, and it is possible to explore solutions for this. Because a collaborative robot is investigated, the human-robot interaction plays a role as well in the overall performance and leads to even more complicated interactions with the environment.

The rising application of collaborative robots in the industry is a good sign, their advantages such as flexibility, ease of use and allowance for intelligence are often outweighing their disadvantages, being lower stiffness, affecting the accuracy, limited range and load. With research alternative techniques and theories are explored in order to improve the robot's functionalities and by doing so expanding its application range. Research can help reach a point where companies can buy general, modular equipment and use this for different tasks, specialized modules can be attached to keep the production costs at a minimum, while maintaining and hopefully even improving the output and increasing the efficiency.

3.1. Collaborative robot

Nowadays, in Industry 4.0, technology is used to increase company efficiency and productivity and has led to a more competitive environment **[33]**. The use of automation and robots played a big part in this new shift.

Collaborative robots (cobots) are the perfect example of this, they are robots that can work safely together with people, though previously it was necessary to separate the robot and the worker to provide a safe work environment.

Definition 1 (industrial robot)
Automatically controlled, reprogrammable multipurpose manipulator, programmable in three or more axes, which can be either fixed in place or mobile for use in industrial applications.
(ISO 8373)

Definition 2 (collaborative robot)
Robot designed for direct interaction with a human within a defined collaborative workspace (Definition 3).
(ISO 10218–2:2011)

Definition 3 (collaborative workspace)
Workspace within the safeguarded space where the robot and a human can perform tasks simultaneously during production operation.
(ISO 10218–2:2011)

It is well known that automation and robots relieve workers from tedious, labour-intensive and repetitive tasks and allows them to do more knowledge-based work such as creative, strategic and changing tasks, because robots lack the imagination and ability to invent new, better practices.

But cobots should not only be used for repetitive tasks because their application range goes beyond this. They have built in safety functions, which allows them to work side by side with human workers and support them with physical work (increase strength, provide protection or support) as well as cognitive (store a lot of information, monitor the worker's condition to determine when work breaks are necessary to increase productivity and recognize patterns)**[5]**.
Cobots sense the proximity of a worker and adapt their speed and/or force accordingly.

This provides a more ergonomic workspace for the workers: their risk on injuries and overall stress due to work with heavy loads is decreased. Humans are still a valid asset in processes because even though cobots can provide unbiased decision-making, we are better at adapting, innovating and it is up to us to make the final decisions with the available information. **[33]**
This leaves more room for personal development, business-oriented thinking and augments their work experience.

When comparing a collaborative robot with an industrial robot it becomes clear that:
robots are used for high volume tasks with high repeatability and not a lot of variety while cobots are flexible, compact, are better for low volumes with a lot of room for variety. They can be used for a variety of tasks because they are easy to install and program thanks to their user-friendly software and built-in teach-in method, where the developer can manually move the robot and store this information. **[50]**

Both UR3 and UR3e mentioned in this paper are equipped with what is called a "kinesthetic programming" tool. On the robot controller this function is called "Hand guide" and allows the end-

user to program the robot by replicating a shown movement. Figure 2: Universal Robot Hand Guide function shows how the robot can be manipulated by hand, with a minimum amount of force.

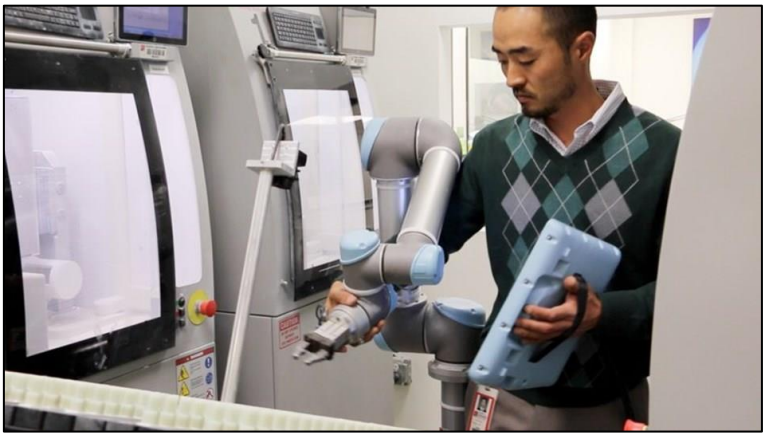


Figure 2: Universal Robot Hand Guide function

This way a real collaboration between human and robot is possible, the user doesn't have to do complex programming and his or her creativity can enhance the process.
When the hand-guide function is enabled it's possible to restrict movements and rotations in certain directions, because the robot will generate a trajectory depending on what is shown by the user. The only thing left to do is guide the robot trough the desired motion by leading the end-effector of the robot.
This translates into the industry as an easy way to adapt according to a high variety in products, without a high necessity for accuracy. Instead of reprogramming the robot multiple times per day or week, the worker can easily demonstrate the movement or adjust the previous one without disrupting the workflow for a long time.

3.2. Force control (in robotics)

There are three big types of robot handlings and the robot control is adapted accordingly.
First it is possible to have negligible interaction forces, the robot executes a movement with no or nearly no external resistance. Here motion control suffices, when the position, velocity and acceleration are controlled the task is executed properly.

In 1977 Donald T. Greenwood **[14]** constructed an equation [Equation 1: General robot control formula] for robot control with gravity compensation g, acceleration-dependent term I(q) which is the mass inertia matrix and velocity-dependent term $f(q, \dot{q})$ which is the moment equation. τ is the actuator input.

$$I(q)\ddot{q} + f(q, \dot{q}) + g(q) = \tau$$

Equation 1: General robot control formula

In case a certain force value should be applied the equation is as follows:

$$I(q)\ddot{q} + f(q, \dot{q}) + g(q) + J^T(q)F = \tau$$

Equation 2: Robot force control formula

The second type is where the interaction forces are not negligible, the robot encounters external resistance of a higher magnitude, but the interaction between manipulator and the environment is static. Here force control suffices, the robot is submitted to a normal force and executes a movement as well. Similar to motion control, in force control it is possible to plan the trajectory of the force in the application.
The difficult part about this control action is that a position control is necessary because the robot needs to perform a motion, and at the same time the force against the workpiece needs to be controlled. During the motion, the configuration of the robot changes and the stiffness of the robot as well, resulting in difficult to predict motion errors.
An example is in application where an object needs to be picked up with a gripper, here the gripper force should be kept between strict boundaries to prevent workpiece or robot tool damage while handling the object.

As last type dynamic interaction occurs, this means that work is done on the environment. The best-known applications of this are machining operations such as drilling, grinding, milling, … .
This is part of robot force control is further explained in **[3.5 Machining with robots]**.
Another example is when assisting someone with an exoskeleton **[54]**, force control is used to enhance or diminish the occurring force on a joint. When lifting a heavy item, the force from the person on the box needs to be enhanced and the counteracting force from the item on the person needs to be diminished.

When a process requires an interaction from the manipulator with the environment, force control is necessary and the feedback complexity increases. Applying force control causes added stress to the robot in the form of internal reaction forces, which can lead to a reduction of its life span if not well designed. Other consequences are or can be: offset in trajectory, unexpected variations in the magnitude due to both tool wear and plastic deformation of workpiece and in the case of machining applications, external lateral forces can occur.
All of previously mentioned by-products of force control lead to a difficult or inaccurate feedback loop, leading to a reduction in accuracy and this is why research on this subject is necessary and will

help further advance automation technology and expands its applicability.

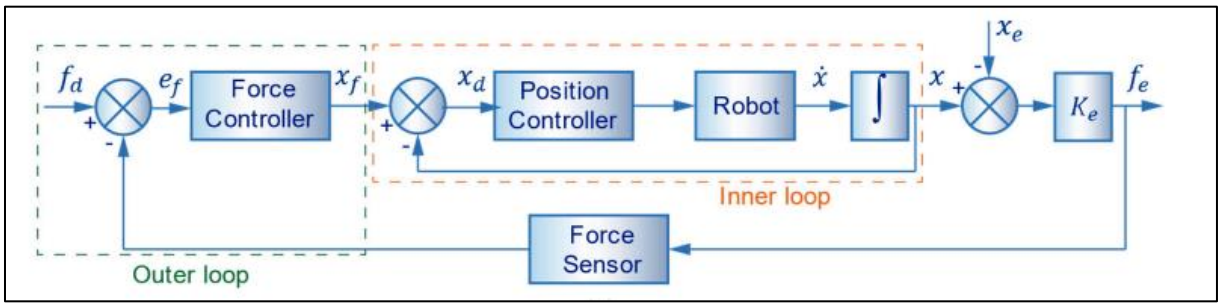


Figure 3: Force control block diagram

[35]

The difficult part about force control is that a position control is necessary because the robot needs to perform a motion, and at the same time the force against the workpiece needs to be controlled. This is part of robot force control is further explained in **[3.5 Machining with robots]**.

Force control makes it possible to determine the execution force during an application. Similar to motion planning, force planning is possible. The applied force has different possible configurations, depending on the application it's possible to apply normal or lateral forces and change their magnitude according to the time, step in the action or other variables.
When the application is known, the different movements are known as well. For example, when a robot needs to perform horizontal movement in the XY-plane, all the other degrees of freedom (Z and the three degrees of rotations) can be used in force control and vice versa. The robot applies the joint forces and torques in order to reach and maintain the desired cartesian force and/or torque values at the end-effector.

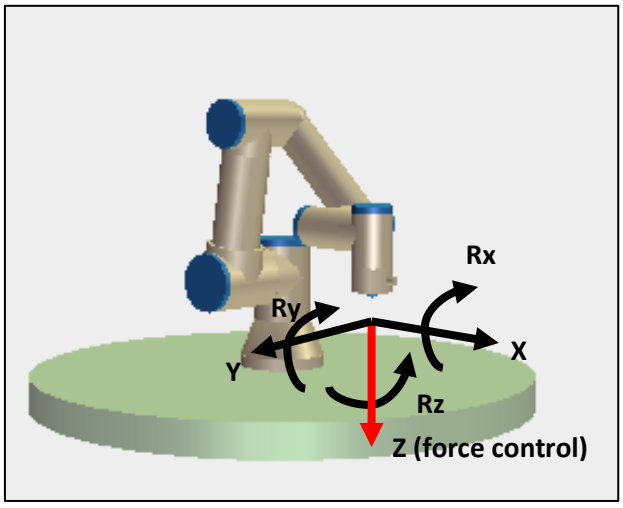


Figure 4: UR robot in force control in z, other DOF are free

Another difficulty in robot force control is the big variety in methods. There is a general and safe force control mode programmed on the cobot, which works but is not focused on performance. A PID controller is used, with certain unknown internal parameters. Many applications are built-in and perform the desired task, but when the importance of a certain task is high, the default control method might not suffice. For this reason, different self-written programs with well-known or more advanced control methods are explored and tested.

In **[8]** a DC motor is used to simulate a 1-DOF robot in force control, displaying different types and degrees in complexity of force control. In **[55]** and in **[53]** the most common ways of applying force

control are mentioned and are further explained below [**3.3 Impedance control** and **3.7 Inner/outer loop control**].

The two big groups in force control methods are fundamental and advanced, where the first one applies a relationship between position/velocity and force and the second one extends this with adaptive control, robust control and/or learning methods. In this research paper the fundamental force control methods are investigated.

- Explicit force control, where there is direct force feedback.
  The force sensor measurements are used to calculate a force error and this is compensated accordingly. The desired value is a constant programmed setpoint and the force control is executed as a PI controller. As can be seen in [Equation 3: Force control with force feedback formula]

$$g(q) + J^T(q)(F + K_p F_e + K_i \int F_e(t)dt) = \tau$$

Equation 3: Force control with force feedback formula

  PI allows the elimination of steady-state wrench error if there is a constant disturbance, which can occur when for example the gravity compensation has an error. A derivative term is not often used, because the lack of dynamics between the joint forces and torques and the wrench at the end-effector does not support the use of a derivate term and force/torque sensors are noise prone. Taking the derivative of noisy measurements only amplifies the noise.
- Hybrid position/force control, where an inner/outer loop scheme is used. [**3.7Inner/outer loop control**]
  Information from the sensor and from the position are evaluated and controlled separately. This way both the position and force are tracked simultaneously and there is one control action for the position errors and one for the force errors. “Normally, the position control law in Figure 3 consists of a PD action, and the force control law consists of a PI action. This is because for the position control a faster response is more desirable, and for the force control a smaller error is more preferable” **[55]**
- Impedance control, where the relation between velocity and force is applied and the stiffness is controlled thanks to position feedback. [**3.3Impedance control**]
  Impedance control is more of an interaction control, with as main parameter the stiffness.

3.3. Impedance control

To improve the robot control, whilst maintaining the safety level, more advanced interaction methodologies are used. In proportional or proportional derivative control high gains are used to help reach the desired value quickly and maintain it in a strict way, this would lead to high impact forces (as can be seen from experiment [**5.4.1 Experiment one: Sampling period in force control [Appendix IX – Sampling period force control]]**), which is negative for both the robot and the environment.

Impedance control can help overcome position errors and avoid large impact forces by programming robots to modulate their motion according to force perceptions, because it generates a relationship between the force and contact point instead of controlling the force directly **[13]**.
As described by Hogan **[16]**: Manipulation requires mechanical interaction with an object or the environment. It is not sufficient to only control the robot's motion when it dynamically interacts with its environment. Position, force and dynamic behaviour is controlled using impedance control.

When comparing impedance control to position control, the big difference is that the goal of position control is to reach a reference point by following a path, while the goal of impedance is to control its dynamic behaviour, the relationship between position (or velocity) and force. This means that in position control the robot will execute the programmed motion, disregarding any external disturbances since it does not take forces or torques into account. This will result in high impact forces. In impedance control external forces are allowed, the robot acts as a spring, where the movement is disrupted but once the external force is released it returns to complete the desired movement. This is why impedance control is a form of interaction control.

Controlling the robot impedance is a way of controlling how the robot behaves when interacting with the environment by defining its stiffness and damping. The control method is equivalent to controlling a mass-spring-damper system. The three parameters are the mass, stiffness and damping, with mass being the most complicated to implement and thus the least used. The stiffness is the most important for safety and the damping is usually used to avoid oscillations.
A balance needs to be found for the stiffness, if the stiffness value is low, the resistance against external forces is low, increasing the safety, but the mass will be more prone to oscillations, but a damper can gradually reduce these to return to the initial position. This is why in robotics often a high stiffness is desired; it assures a greater positional and force maintenance and accuracy. For collaborative robots this also means high impact forces when encountering an obstacle, which can be dangerous because these robots operate in unknown and fast-changing environments.

"Robots that physically interact with their surroundings, in order to accomplish some tasks or assist humans in their activities, require to exploit contact forces in a safe and proficient manner. Impedance control is considered as a prominent approach in robotics to avoid large impact forces while operating in unstructured environments." **[1]**
This applies to collaborative robots, these are designed to work together and interact with humans, so safety is of a big importance **[25 - 34]**. The robot control forms a challenge because the robots need to operate and interact within an unstructured, varying and, in case humans are near, sensitive environment.

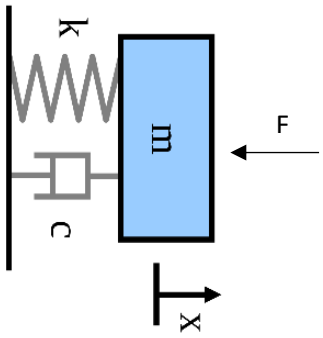


Figure 5: Impedance control mass-spring-damper system

Newton's third law of motion:

$$F(t) = m\ddot{x} + c\dot{x} + kx$$

Equation 4: Newton's third law of motion

$$F(s) = m[s^2X(s) - sx(0) - \dot{x}(0)] + c[sX(s) - x(0)] + kX(s)$$
$$F(s) = (ms^2 + cs + k)X(s) + (-ms - c)x(0) - m\dot{x}(0)$$

Equation 5: Laplace transform of Newton's third law of motion

Laplace transform gives
and we assume $x(0) = \dot{x}(0) = 0$

$$F(s) = (ms^2 + cs + k)X(s)$$
$$\frac{F}{X} = ms^2 + cs + k$$
$$Impedance\ Z = \frac{F}{V} = ms + c + \frac{k}{s}$$

Equation 6: Impedance equation mass-spring-damper

k is equivalent to Kp, the proportional constant for the position control and indirect stiffness control. b is equivalent to Kd, the derivational constant reacting to the difference in velocity and damping the system.

This impedance, as in Equation 6: Impedance equation mass-spring-damper is called the mechanical impedance and represents the ratio of force output to motion input. It is controlled in experiment [**5.5.2Impedance control** **[Appendix XIV – Impedance control]**] by adjusting the active stiffness values of the robot joints.

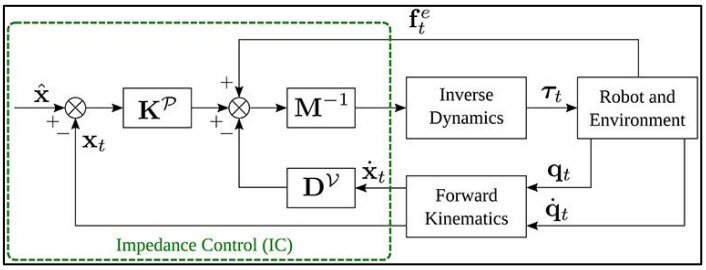


Figure 6: Impedance control block diagram

The robot stiffness (K in Figure 6: Impedance control block diagram) consists of active and passive stiffness, being programable and joint stiffness of the harmonic drives respectively.

The joint stiffness matrix is $K_\theta = \begin{bmatrix} K_1 & 0 & 0 & 0 & 0 & 0 \\ 0 & K_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & K_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & K_4 & 0 & 0 \\ 0 & 0 & 0 & 0 & K_5 & 0 \\ 0 & 0 & 0 & 0 & 0 & K_6 \end{bmatrix}$ and $K_x = J^{-T} K_\theta J^{-}$

This joint stiffness is constant during motion tasks, but because of the transformation using the Jacobian, the cartesian stiffness changes depending on the robot configuration.
The total robot stiffness is the cartesian or joint stiffness, plus the active stiffness introduced by the controller.

The speedL command exists of $\begin{bmatrix} v_x \\ v_y \\ v_z \\ v_\alpha \\ v_\beta \\ v_\gamma \end{bmatrix}$ with $v_x = K_{px}(x_{ref} - x) + K_{dx}(\dot{x}_{ref} - \dot{x})$ and similar for the other velocities. The proportional gain of the control action is equivalent to the active stiffness and the derivative gain to the damping.

The following study **[21]** about adaptive impedance control is a perfect example of what impedance controlled is designed for. Powered exoskeletons to enhance or restore human's muscular force and endurance, in other words they cooperate with the human by assisting or supplementing its motion. The robot system's impedance model is designed by using an impedance algorithm. Because the exoskeleton needs to match the operator's kinematic and dynamic behaviour.

In **[2]** impedance is used for indirect force control and in parallel a direct force control method is used for counteracting contact forces when interacting with a surface. It is confirmed that impedance control finds its use in assisting humans and attain contact forces safely, as mentioned before. This limits the needed safety measurements when employing a collaborative robot in the work environment in a way that the robot can work in close proximity of humans and other robots, extending the robot's work range and using the most of its capabilities. EMG signals are used to measure muscle response or electrical activity in response to a nerve's stimulation of the muscle, allowing a stiffness estimation of the human's arm when cooperating.
In **[32]** impedance control is used to measure interaction forces and compensate the robot deformation and an adaptive material removal formula is set up for milling, this was shown to be successful. A sensor-less force control method was proposed in 2014 **[11]** and worked well enough to be applied in practical applications.

3.4. Distributed control

"A distributed control system is a digital automated industrial control system that uses geographically distributed control loops throughout a factory, machine or control area. Unlike a centralized control system that operates all machines, a distributed control system allows each section of a machine to have its own dedicated controller that runs the operation. It has several local controllers located throughout the area that are connected by a high-speed communication network. While each controller works autonomously, there is a central supervisory control run by an operator." **[39]**

The word distributed shows that in this control method, the network topology is not in a line, where one component can only communicate with one other above or below in hierarchy but is distributed amongst different controllers on different levels and the communication is of a more diverse and complex pattern.

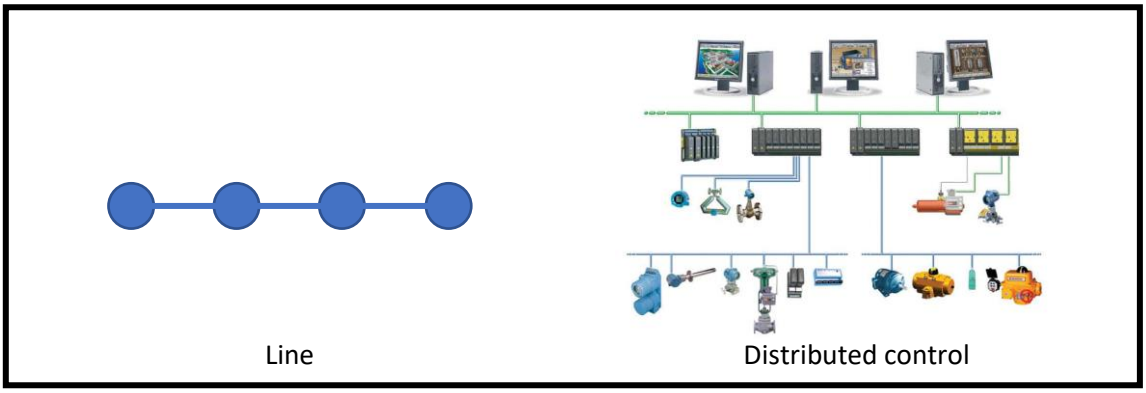


Figure 7: Line network vs distributed control network topology

Distributed control is increasing in importance **[6]** because the system fulfils the wish of flexibility by allowing modularity; functionalities can be added, subtracted and substituted according to the demand **[22 - 37]**. This is a big improvement over ordinary control systems because the innovation rate in today's industry is high, meaning that systems or at least the majority of their components should be reusable. This gives a cost-effective solution to industries where processes change or have to adapt at a fast rate. The ease of scalability this solution offers is a second advantage, because it's possible to integrate external controllers into the distributed system it is possible to expand or reduce the system.

The industrial collaborative robot UR3 and the laboratory setup in general, used for this research utilize distributed control because different machines are unified to share information and resources, while still being able to work autonomously. The UR3 communicates with its force sensor, the PC and the teach pendant while all of these components are controlled separately, thus maintaining the control complexity at a minimum.

Making multiple task-specific controllers work together and share information is more reliable than using one centralized controller for all the different modules. This also allows to work on different levels, in this application the lower-level system can be the force value that's controlled and one level above that the motion controller verifies that the force control doesn't cause trajectory errors. In applications with multiple force controllers, it is possible to install a balance controller on the next level, to even out all the force values and essentially monitor the lower-level controllers.

3.5. Machining with robots

The study of machining with robots is a new facet to the field of robotics. Most of the research written on this is done with specialized equipment and does not go into detail on mathematical equations to estimate force values, which is one of the main problems to be tackled; when machining, a lot of forces occur and these need to be compensated to maintain the accuracy, but without damaging the robot joints or tool due to the increased stress.

Specialized tools and robots are designed because machining operations require high stiffness to assure the accuracy. A balance must be found between flexibility and robustness because the machining tasks involve large reaction forces that can damage the tool, the robot or the environment. Even if this does not damage the tool, it reduces the lifespan significantly. If the robot robustness is increased, the flexibility decreases and this takes away one of the main advantages of robot usage.

Machining with collaborative robots is a subject that is not yet strongly supported by research, because these robots are designed to be flexible and machining with normal robots is yet to be optimized for application. The occurring forces also highly depend on the used feed rate, removal depth, material, … **[24]**

Research on the use of collaborative robots in machining should be pursued because of their big application range and their flexibility.
But the research found on this topic shows how the rather simple task of maintaining a constant force and compensating the tool deviation while machining imposes a lot of other difficulties.

Force compensation

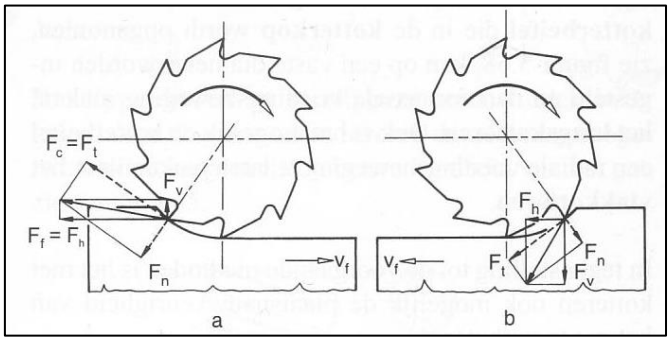


Figure 8: Visual representation of milling cutting forces

$$h_{gem} = f_z\sqrt{\frac{a}{d}}$$

Equation 7: Average cut thickness milling

$$cos\varphi = \frac{\frac{d}{2} - a}{\frac{d}{2}}$$

Equation 8: Angle of impact milling

$$z_i = \frac{\varphi}{360°} * z$$

Equation 9: Amount of cutting teeth milling

$$F_t = k_{c1,1} * h_{gem}^{(1-\epsilon)} * b * z_i$$

Equation 10: Average tangential force in milling

The average tangential force [Equation 10: Average tangential force in milling] can be derived from [Equation 7: Average cut thickness milling, Equation 8: Angle of impact milling and Equation 9: Amount of cutting teeth milling]. In this final equation $k_{c1,1}$ is the specific cutting force with a surface area of 1x1mm², the next symbol is the average cutting thickness, which depends on the feed per tooth, depth of cut and diameter of mill (resp. f_z, a and d). The tangential force is directly proportional to the width of the cut b and by multiplying this with the number of teeth that are cutting z_i, this gives the full force. The number of teeth cutting depends on φ, which is the angle of impact.

When only few teeth cut at the same time, so z_i and ϕ are small, the variations in forces are big. This means that when the tool enters the material, at first the force fluctuations will be high.

These complex calculations show that real time data processing is necessary to have an accurate force control when machining. These forces can be measured immediately but there will always be a delay on the feedback and thus the compensation.

Tool path compensation

'Linear interpolation of the workpiece coordinates is important for machining with industrial robots. A tool motion linearly interpolated in joint coordinates becomes a fluctuating trajectory in the workpiece coordinates' **[38]**, so we need to do the linear interpolation of the tool motion in the workpiece coordinates.

The robot joint stiffness depends on the used configuration and because during a movement the configuration changes, the stiffness values change as well. This causes more stable and more instable robot poses and makes the path compensation more complicated.

This small literature study on machining with robots is done but is very limited in such a way that after doing the literature study, it became clear that to properly implement force control for automated machining, a more profound study should be done.

Analysis of material, implementation of simulations using different software packages to predict the behaviour etc. CAD models, material models, virtual machining technologies for even more realistic simulations and modelling. Virtual machining has been proven **[10]** to work for assembly.
Machine learning and AI could be implemented into the research to make the robot work in the most efficient way imaginable, but this goes beyond this research because of the time and resource limitation. By implementing this, some complex problems can be avoided or reduced in complexity.

3.6. Kalman filter

"The Kalman filter estimates a process by using a form of feedback control: the filter estimates the process state at some time and then obtains feedback in the form of (noisy) measurements. As such, the equations for the Kalman filter fall into two groups: time update equations and measurement update equations." **[49]**

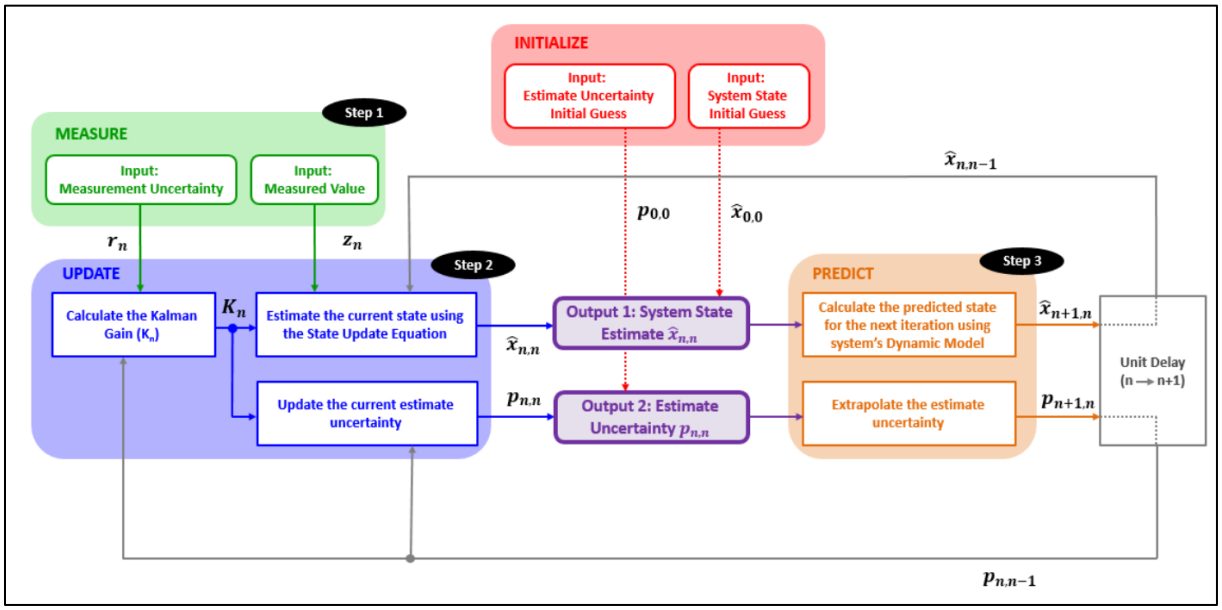


Figure 9: Kalman filter block diagram

These two groups of equations form two phases in the process: the prediction phase and the correction (or update in Figure 9: Kalman filter block diagram) phase.

Prediction phase:

$$\hat{x}_{k+1}^{-} = A_k \hat{x}_k$$

Equation 11: State prediction Kalman Filter

$$P_{k+1}^{-} = A_k P_k A_k^T + Q_k$$

Equation 12: Covariance prediction Kalman Filter

The predictor estimates the next state $\hat{x}_{k+1}^{-}$, using the state matrix A_k and previous state $\hat{x}_k$, as well as the covariance P_{k+1}^{-}, using the state matrix, previous covariance, transposed of the state matrix and adding the process noise covariance Q_k. This process noise covariance matrix reflects the exactness of the model and the probability of disturbances.

Correction phase:

$$K_k = P_k^{-} H_k^T (H_k P_k^{-} H_k^T + R_k)^{-1}$$

Equation 13: Kalman gain Kalman Filter

$$\hat{x}_k = \hat{x}_k^{-} + K(z_k - H_k \hat{x}_k^{-})$$

Equation 14: State correction Kalman Filter

$$P_k = (I - K_k H_k) P_k^{-}$$

Equation 15: Covariance correction Kalman Filter

The corrector calculates the Kalman gain K_k, used as gain factor to determine the weigh on the measurement error $(z_k - H_k\hat{x}_k^-)$. The predicted covariance is used for this, in combination with the measurement function H_k and the sensor noise covariance R_k.
Next, the process is measured z_k and used to compute the updated state estimate.
The last step is to calculate the updated error covariance estimate, out of the identity matrix I, the calculated Kalman gain, the measurement function and the previous covariance.

In the way a Kalman filter is used later, in force control, x represents the state of the robot = $\begin{bmatrix} F \\ \dot{F} \end{bmatrix}$. The force and force derivative. The equations mentioned below are used in this experiment and first require some manipulation, so it's easier to code. The initialized parameters and further calculations are done below in [**5.5.1Force filtering and control**].

3.7. Inner/outer loop control

In most force control applications, a combination of position and force control is needed, a force is applied, and its magnitude needs to be controlled, while executing a motion. The probability of a motion error rises with a rising force value because the latter hinders the former.

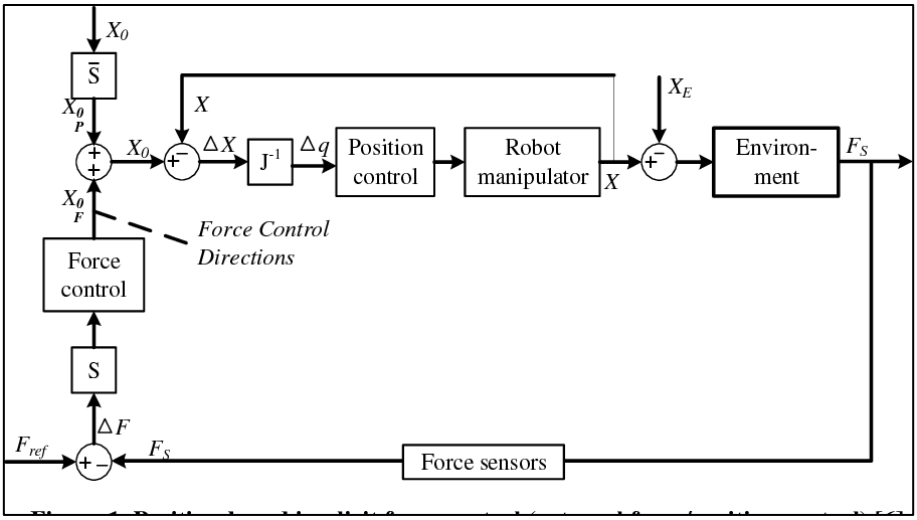


Figure 10: Inner/outer loop control block diagram
[30]

Inner/outer loop control as presented in Figure 10: Inner/outer loop control block diagram acts as a cascade control system where the force controller generates control references for the position controller and this way both controllers work together to achieve the desired output (force).

The inner loop functions as a traditional feedback control system with a setpoint, a process variable, and a controller (in this case a position controller) acting on a process (the robot) by means of an actuator. The inner loop controls the variable directly and thus gives a quick response to errors, before its magnitude expands and transmits throughout the system, resulting in a poor performance.

The outer loop does the same except that it uses the entire inner loop as its actuator, plus an extra controller, being both the force controller and the position controller. It regulates the force variable in an indirect fashion, which is why it operates at a lower frequency and needs sensor data processing for the feedback. The outer loop's indirect nature possibly affects the process in case of a continuously rising error but has the advantage to not oscillate excessively when the error fluctuates at a high rate.

The inner loop disturbances are less severe than the outer loop disturbances. Otherwise, the position controller will be constantly correcting for disturbances to the robot motion and unable to apply consistent corrective efforts to the force control.
This way of controlling shows its applications in tracking applications **[9 - 26 - 52]**, because the outer loop either tracks a position or a force value. In 2016 **[30]** used PI control to evaluate an inner/outer loop force control and good results came from this.

4. Theoretical Analysis

4.1. Robot breakdown: clarifying robot jargon

The used robot in this Master Thesis is a 6-DOF, anthropomorphic, collaborative robot.

The robot exists of different mechanical parts, with different functionalities and properties. The movements are possible thanks to joints in blue, the base, shoulder, elbow and the three wrists in this case. Each of these joints move a link (grey bar/body/arm in between joints) and all higher located joints.

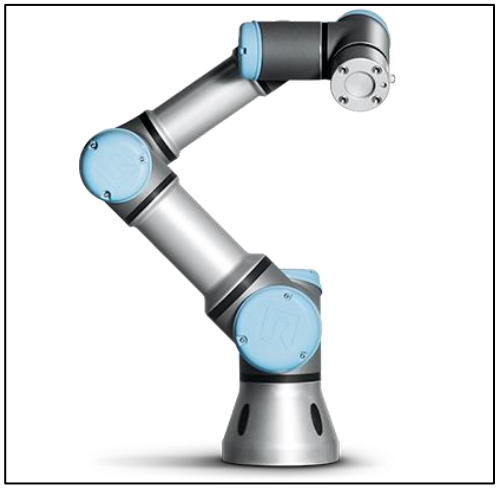


Figure 11: UR3 collaborative robot

Joints can have 1 DOF, thus allowing one movement, shown in Figure 1: the Revolute, Prismatic and Helical joints, or they have multiple DOF: Cylindrical, Universal or Spherical joints.

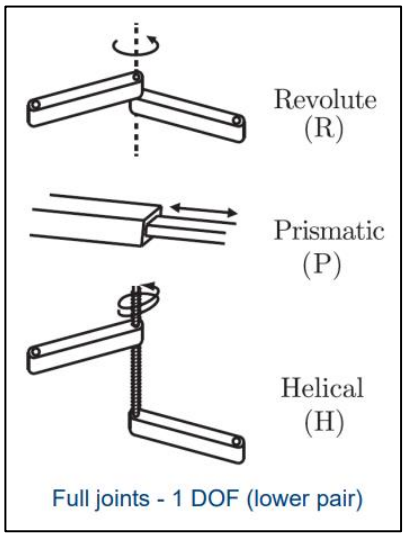


Figure 12: 1 DOF joints

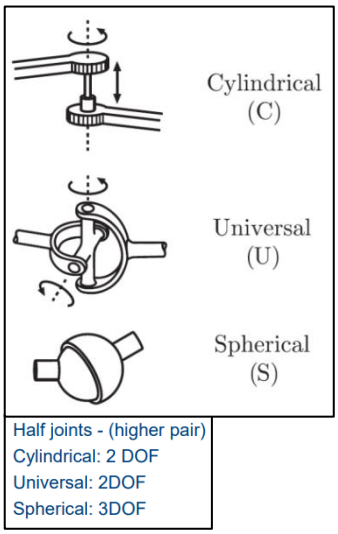


Figure 13: Higher DOF joints

The robots made by Universal Robots are made of 6 revolute joints. They all have a fixed rotation axis and do not allow translations of any kind.

A combination of the six different joint angles describes the robot configuration and if the robot sweeps all possible configurations, it creates a volume called robot workspace.

Two calculations used by the robot are the forward and inverse kinematics of the robot, these serve as methods to calculate the robot end-effector pose relative to reference frame from given joint angles and the joint angles when the end-effector pose is given respectively.

The Denavit-Hartenberg formulas are used for this and follow out of a set of matrix calculations with angles for each joint. In other words, the Denavit-Hartenberg representation divides the motion of the kinematic chain of the robot in the relative motion between different links. As shown in the picture below.

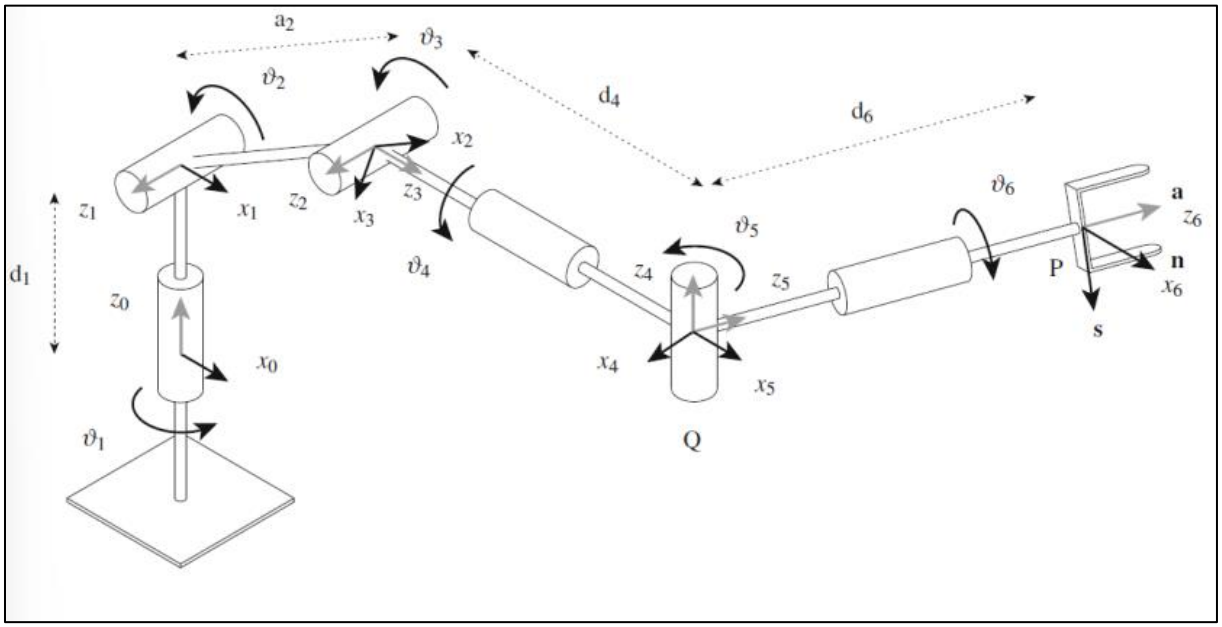


Figure 14: 6-DOF robot analysis per joint

Each relative displacement is described in the form of a matrix A, when multiplying consecutive matrices from 0 to 3, we get the arm model and from 3 to 6 we get the wrist model.
First three values in each row are for orientation and the last value is for the position.

Arm model: ${}^0A_3=\begin{bmatrix} cos\theta_1 cos(\theta_2+\theta_3) & -sin\theta_1 & -cos\theta_1 sin(\theta_2+\theta_3) & a_2 cos\theta_1 cos\theta_2 \\ sin\theta_1 cos(\theta_2+\theta_3) & cos\theta_1 & -sin\theta_1 sin(\theta_2+\theta_3) & a_2 sin\theta_1 cos\theta_2 \\ sin(\theta_2+\theta_3) & 0 & cos(\theta_2+\theta_3) & d_1+a_2 sin\theta_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Wrist model: ${}^3A_6=\begin{bmatrix} c4c5c6-s4s6 & -c4c5s6-s4c6 & -c4s5 & -d6c4s5 \\ s4c5c6+c4s6 & -s4c5s6+c4c6 & -s4s5 & -d6s4s5 \\ s5c6 & -s5s6 & c5 & d_4+d_6c5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

with c1=cosθ_1 and s1=sinθ_1

Full robot model = ${}^0A_3 \times {}^3A_6 = {}^0A_6$

Joint, tool and workpiece coordinates

The robot environment can be described using different coordinate systems, depending on the origin and its unit vectors. They are called joint, tool and workpiece coordinates and illustrated in [Figure 15 - Figure 16 - Figure 17].

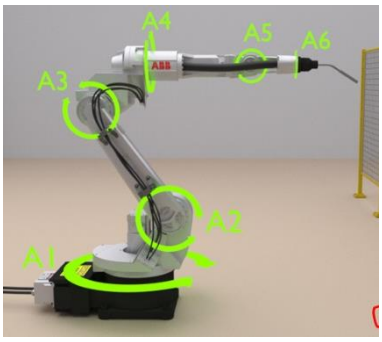


Figure 15: Robot joint coordinates

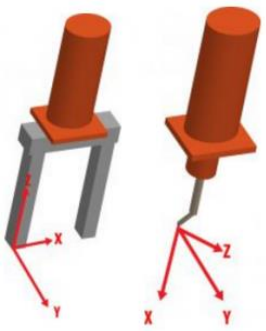


Figure 16: Robot tool coordinates

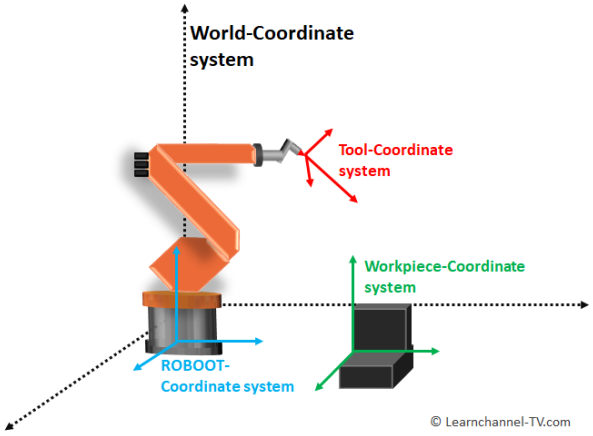


Figure 17: Robot coordinate systems

Joint coordinates

'The angle-position and length of each axis of an articulate robot axes describes the orientation of the TCP exactly. With the joint coordinate-system each robot axis can be moved particularly in positive or negative sense rotation.' **[19]**
The joint coordinates are a set of joint angles and are useful because no transformations are needed, and the robot position and axes positions are easy to visualize.

Tool coordinates

When using tool-coordinates, we use the TCP, so we calibrate our TCP to the desired position and from then on, we know exactly where our tool is located.

'The following tasks are easier to program using the tool coordinate system:

- Turning the tool around the TCP (Tool Center Point)
- To maintain the speed at the TCP even with complex paths
- To push the tool in a certain direction' **[20]**

Workpiece coordinates

'The object coordinate system specifies how a workpiece is positioned in a fixture or workpiece manipulator.' **[17]**

If the workpiece is fixed during the whole process, or we know the movement of the fixture, this coordinate system might be useful.

4.2. Stiffness matrix and Jacobian of robot

Jacobian **[7]**

"The matrix which relates changes in joint parameter velocities to Cartesian velocities is called the Jacobian Matrix. This is a time-varying, position dependent linear transform."
The Jacobian consists of linear and rotational elements, defining the occurrence of these two movements and using the rotation matrix in order to do so. Rotation matrices define the relative rotation of coordinate frames.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = \boldsymbol{J} \cdot \dot{\boldsymbol{q}}$$

J
#colum
#rows=

Time derivatives of joint parameters (d_i, θ_i)

$$\boldsymbol{J} = \begin{bmatrix} J_{v1} & J_{v2} & \cdots & J_{vn} \\ J_{\omega 1} & J_{\omega 2} & & J_{\omega n} \end{bmatrix} \cdot \dot{\boldsymbol{q}}$$

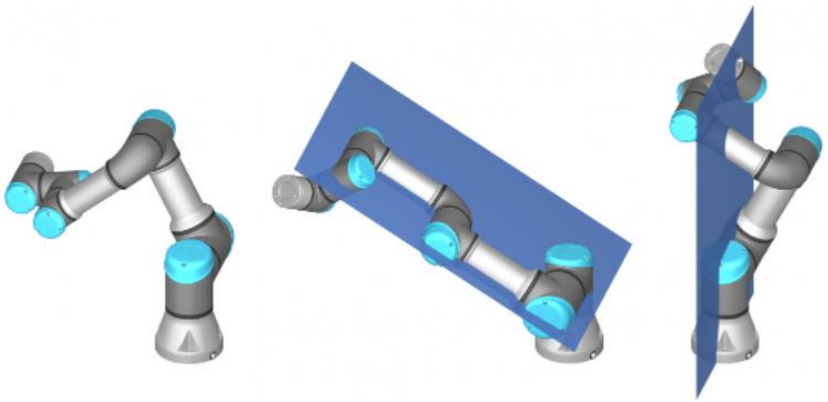
	Prismatic joint	Revolute joint
Linear J_{vi}	*linear along z-axis* $R_{i-1}^0 \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$	$\vec{v} = \vec{\omega} \times \vec{r}$ $R_{i-1}^0 \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \times \left(d_n^0 - d_{i-1}^0 \right)$
Rotational $J_{\omega i}$	*no rotation* $\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$	*rotation around z-axis* $R_{i-1}^0 \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$

Table 1: Jacobian matrix analysis

This is an important matrix because it is used in different control calculations, for example in the Kalman filter and in controlling the end effector velocity by controlling the angular joint velocities. The matrix is set up to calculate singularities in the robot. Where the determinant of the Jacobian is equal to zero, a singularity occurs.

A singularity is when the robot is in a certain configuration and from then wants to move with a linear velocity in cartesian space, but because of the configuration the joint velocities become infinite. The robot then loses one or more degrees of freedom.

"A robot singularity is a configuration in which the robot end-effector becomes blocked in certain directions." **[4]**



Types of singularity in the typical collaborative robot arm: wrist (left), elbow (center) and shoulder (right) singularities

Figure 18: UR singularities

Wrist singularity: axes of joints 4 and 6 become parallel. θ5 = 0°, θ5 = ±180° or θ5 = ±360°. Because the shoulder (1), elbow (2) and first wrist (3) move in the same plane, a singularity occurs when we move the second wrist joint (4) to 0° or 180°. See picture below

Figure 19: UR elbow singularity

[47]

Elbow singularity: stretched out arm. Axes of joints 2,3 and 4 are in the same plane. θ3 = 0°
Shoulder singularity: intersection point of the axes of joints 5 and 6 lies in the plane passing through the axes of joints 1 and 2.

Stiffness matrix

This is the matrix notation of the resistance against external forces. In this 6-DOF robot the stiffness matrix is a 6x6 matrix. The robot joint stiffness shows the resistance of the joints against forces (both internal and external) because a force in one direction does not only initiate a displacement in this same direction, but a matrix notation is also desired for the robot stiffness. The values also differ in different directions, which gives a multi-dimensional set of values.

$$[K]^e\{U\}^e = \{F\}^e \qquad (1.14)$$

Where $[K]^e$ is the element stiffness matrix $\{U\}^e$ is the vector of nodal displacements for the nodes contained in the element and $\{F\}^e$ is the nodal load vector, which specifies which forces are applied to the nodes contained in the element.

The stiffness matrix is not given by the manufacturer but can experimentally be approximated **[40]**. This is beyond the scope of this research, but it can be noted that the stiffness values are dynamic and change depending on the robot configuration.

Stiffness control can either be passive or active **[18]**. In passive stiffness control the robot arm acts as a spring, where the stiffness is pre-set and constant. This control is equivalent to feedback control because first an action needs to take place before a control action is introduced. In active stiffness control the spring constant is programmable trough force feedback. This is equivalent to feedforward control because it adapts according to a possible following action, for example when following a trajectory, the active stiffness depends on the next reference point.

To control the stiffness of the robot, the active stiffness values can be changed. The following formulas demonstrate the relationship between the Jacobian and the stiffness control.
With τ being the joint torque matrix, K being the passive stiffness matrix of the robot and Δq the change in robot pose.

$$\tau = K_\theta \Delta q$$

Equation 16: Stiffness - robot joint torque in function of stiffness and pose difference

$$\Delta x = J(q)\Delta q \Rightarrow \Delta q = J^{-1}\Delta x$$

Equation 17: Stiffness - cartesian position change in function of pose change

$$\tau = J^T F \Leftrightarrow J^T F = K_\theta J^{-1}\Delta x \Leftrightarrow F = J^{-T} K_\theta J^{-1}\Delta x$$

Equation 18: Stiffness - torque in function of force

With $$K_x = J^{-T} K_\theta J^{-1} \Rightarrow F = K_x \Delta x$$

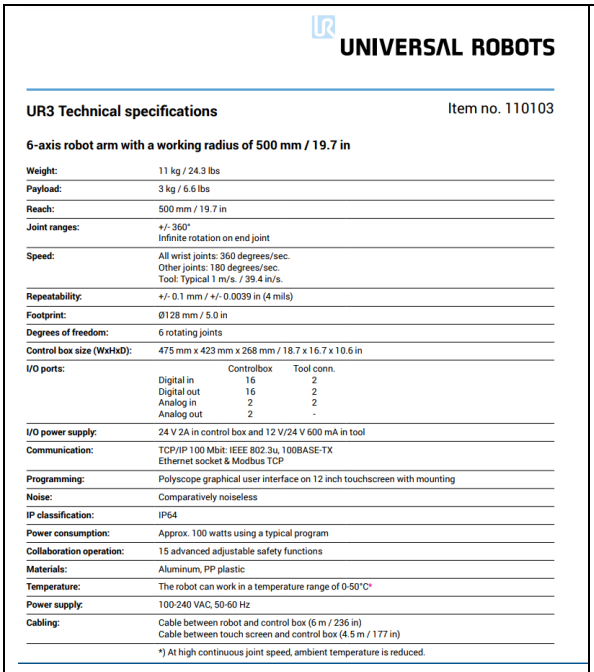
Equation 19: Stiffness - force in function of cartesian position change

Equation 16: Stiffness - robot joint torque in function of stiffness and pose difference can be used for the active stiffness as well, where the active stiffness matrix is a diagonal matrix with elements in the range of 300-2000.
The total stiffness of the robot is the active plus the passive.

4.3. Specifications of both UR3s

Universal Robots is a company founded in Denmark and is now world's largest manufacturer of collaborative robots **[50]**. Their values of adaptability and reliability show in the cobot functions: 'flexible to deploy, easy to program, fast to set up, budget-friendly and safe'. **[42]**
They have two product series: the normal UR (CB-series) and a newer UR e-Series. The number behind the 'UR' stands for the payload capacity. The e-serie robots have the same payloads, except for a new UR16e.

Experiments are a helpful tool to gather information about the differences between these robots, as a verification tool datasheet are published by the manufacturers and summarized below.



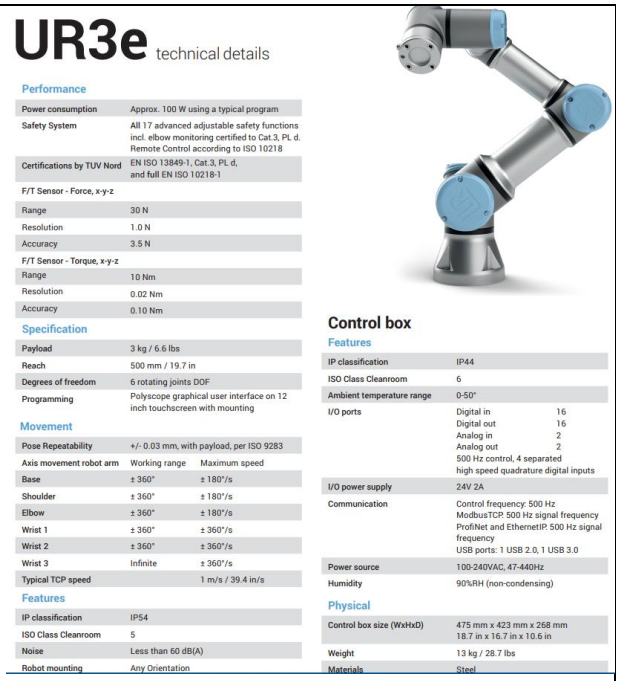
UNIVERSAL ROBOTS

UR3 Technical specifications Item no. 110103

6-axis robot arm with a working radius of 500 mm / 19.7 in

Weight:	11 kg / 24.3 lbs
Payload:	3 kg / 6.6 lbs
Reach:	500 mm / 19.7 in
Joint ranges:	+/- 360° Infinite rotation on end joint
Speed:	All wrist joints: 360 degrees/sec. Other joints: 180 degrees/sec. Tool: Typical 1 m/s. / 39.4 in/s.
Repeatability:	+/- 0.1 mm / +/- 0.0039 in (4 mils)
Footprint:	Ø128 mm / 5.0 in
Degrees of freedom:	6 rotating joints
Control box size (WxHxD):	475 mm x 423 mm x 268 mm / 18.7 x 16.7 x 10.6 in
I/O ports:	Controlbox / Tool conn. Digital in 16 2 Digital out 16 2 Analog in 2 2 Analog out 2 -
I/O power supply:	24 V 2A in control box and 12 V/24 V 600 mA in tool
Communication:	TCP/IP 100 Mbit: IEEE 802.3u, 100BASE-TX Ethernet socket & Modbus TCP
Programming:	Polyscope graphical user interface on 12 inch touchscreen with mounting
Noise:	Comparatively noiseless
IP classification:	IP64
Power consumption:	Approx. 100 watts using a typical program
Collaboration operation:	15 advanced adjustable safety functions
Materials:	Aluminum, PP plastic
Temperature:	The robot can work in a temperature range of 0-50°C*
Power supply:	100-240 VAC, 50-60 Hz
Cabling:	Cable between robot and control box (6 m / 236 in) Cable between touch screen and control box (4.5 m / 177 in)
	*) At high continuous joint speed, ambient temperature is reduced.

Figure 20: UR3 datasheet
[41]



UR3e technical details

Performance

Power consumption	Approx. 100 W using a typical program
Safety System	All 17 advanced adjustable safety functions incl. elbow monitoring certified to Cat.3, PL d. Remote Control according to ISO 10218
Certifications by TUV Nord	EN ISO 13849-1, Cat.3, PL d, and full EN ISO 10218-1
F/T Sensor - Force, x-y-z	
Range	30 N
Resolution	1.0 N
Accuracy	3.5 N
F/T Sensor - Torque, x-y-z	
Range	10 Nm
Resolution	0.02 Nm
Accuracy	0.10 Nm

Specification

Payload	3 kg / 6.6 lbs
Reach	500 mm / 19.7 in
Degrees of freedom	6 rotating joints DOF
Programming	Polyscope graphical user interface on 12 inch touchscreen with mounting

Movement

Pose Repeatability	+/- 0.03 mm, with payload, per ISO 9283	
Axis movement robot arm	Working range	Maximum speed
Base	± 360°	± 180°/s
Shoulder	± 360°	± 180°/s
Elbow	± 360°	± 180°/s
Wrist 1	± 360°	± 360°/s
Wrist 2	± 360°	± 360°/s
Wrist 3	Infinite	± 360°/s
Typical TCP speed		1 m/s / 39.4 in/s

Features

IP classification	IP54
ISO Class Cleanroom	5
Noise	Less than 60 dB(A)
Robot mounting	Any Orientation

Control box

Features

IP classification	IP44
ISO Class Cleanroom	6
Ambient temperature range	0-50°
I/O ports	Digital in 16 Digital out 16 Analog in 2 Analog out 2 500 Hz control, 4 separated high speed quadrature digital inputs
I/O power supply	24V 2A
Communication	Control frequency: 500 Hz ModbusTCP: 500 Hz signal frequency ProfiNet and EthernetIP: 500 Hz signal frequency USB ports: 1 USB 2.0, 1 USB 3.0
Power source	100-240VAC, 47-440Hz
Humidity	90%RH (non-condensing)

Physical

Control box size (WxHxD)	475 mm x 423 mm x 268 mm 18.7 in x 16.7 in x 10.6 in
Weight	13 kg / 28.7 lbs
Materials	Steel

Figure 21: UR3e datasheet
[43]

Repeatability is +/- 0,1mm.	Repeatability is 0,03mm.
No built-in force/torque sensor	Has a built-in force/torque sensor
Size specifications	
Footprint Ø128mm	Footprint Ø128mm
Reach 500mm	Reach 500mm
Weight 11kg	Weight 11,2kg

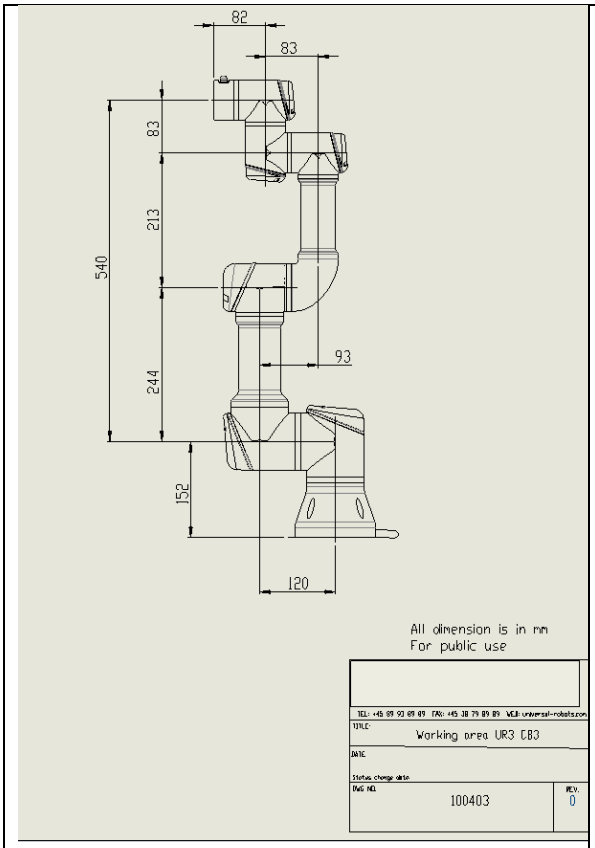


Figure 22: UR3 workspace

[45]

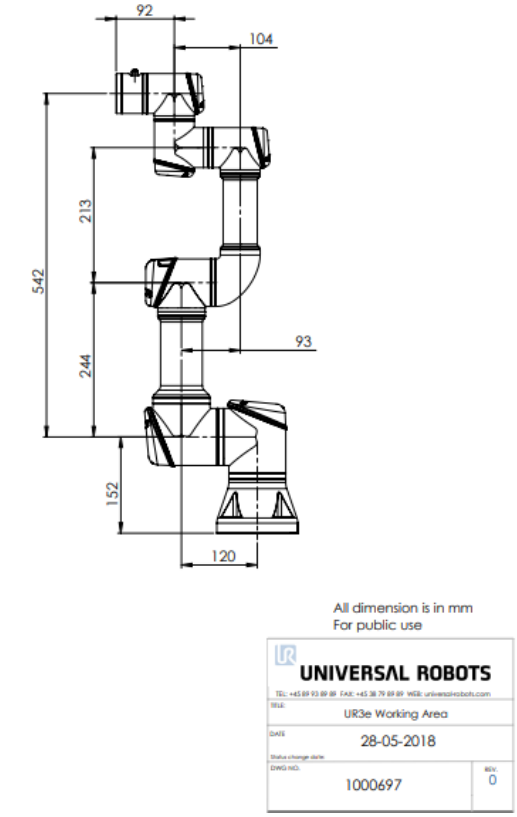


Figure 23: UR3e workspace

[46]

F/T specifications: for UR3 HEX-E/H QC Sensor	
Range 200N	Range 30N
Resolution 0,2 for X-Y & 0,8 for Z	Resolution 1N
Accuracy <2% so <4N	Accuracy 3,5N

DATASHEET

1. Datasheet

1.1. HEX-E QC

General Properties	6-Axis Force/Torque Sensor				Unit
	Fxy	Fz	Txy	Tz	
Nominal Capacity (N.C)	200	200	10	6.5	[N] [Nm]
Single axis deformation at N.C (typical)	± 1.7 ± 0.067	± 0.3 ± 0.011	± 2.5 ± 2.5	± 5 ± 5	[mm] [°] [inch] [°]
Single axis overload	500	500	500	500	[%]
Signal noise* (typical)	0.035	0.15	0.002	0.001	[N] [Nm]
Noise-free resolution (typical)	0.2	0.8	0.01	0.002	[N] [Nm]
Full scale nonlinearity	< 2	< 2	< 2	< 2	[%]
Hysteresis (measured on Fz axis , typical)	< 2	< 2	< 2	< 2	[%]
Crosstalk (typical)	< 5	< 5	< 5	< 5	[%]
IP Classification	67				
Dimensions (H x W x L)	50 x 71 x 93 1.97 x 2.79 x 3.66				[mm] [inch]
Weight (with built-in adapter plates)	0.347 0.76				[kg] [lb]

[44]

Figure 24: HEX sensor datasheet

Table 2: Specifications of both UR3s

Following the information from above, it can theoretically be concluded that:

- The UR3e is more precise, with a repeatability of 0,03mm instead of UR3's 0,1mm
- Both robots have the same size, except for the three wrists. These have different proportions resulting in a 2mm total height difference, but no difference in workspace reach
- The robot joint speeds are equal
- Because the UR3 has an external F/T sensor, this accuracy and resolution is better than the UR3e's

5. Practical experiments

Previous mentioned information is important to conduct experiments with both cobots. First the most important information about the used components is given, then the first experiments that were conducted to test functions and get to know the commands.
Finally, more advanced experiments are described, these are the ones that give results for the research and combine multiple basic functions.

5.1. Laboratory setup

In the laboratory the following equipment is used in the experiments: a desktop computer with Windows 10 (6), a Universal Robots UR3 CB-series robot (3), with corresponding Control box (2) and with this Control box there comes a teach pendant (1). Because the CB-series do not contain an internal f/t-sensor, a HEX-E QC sensor (4) with corresponding compute box (5) is attached.

This figure is for the UR3, for the UR3e the setup is similar except that there is no external sensor or compute box.

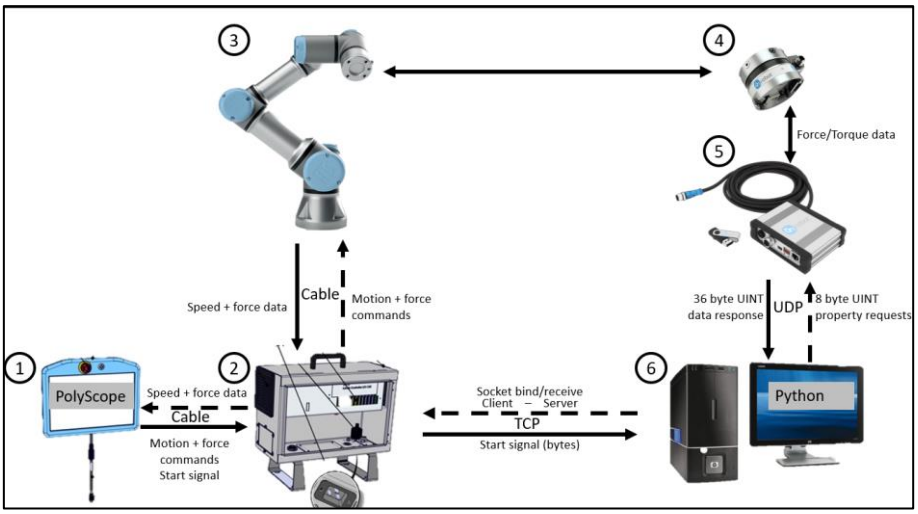


Figure 25: Laboratory setup communication flow

Figure 25: Laboratory setup communication flow shows the hardware + software used in the laboratory for the experiments. The data flow is displayed as well.

The compute box, control box and computer are connected to the same Local Area Network with ethernet cables and communicate this way. For the communication between the PC and the robot control box a TCP IP communication is set up because this is a connection-oriented protocol and between the PC and the sensor compute box a UDP communication, which prioritizes speed over connectivity.

The computer is equipped with Python, MATLAB and Excel. This component in the communication is used to send requests and receive and process data, using Python. Excel files are created to store the received data with any computed data, which can later be used in MATLAB for quick computations. In the same Python program, the PC first sets up the UDP socket connection and is then able to send requests to the compute box in order to set the data transmission parameters (read-out speed, biasing, filtering) as well as the start signal to send sensor output data. In other words, another responsibility of the PC is setting the communication parameters.

The robot controller box contains both digital and analog input and output sockets which can be used for interfacing other components or system components itself. It receives data from both the Teach

Pendant in the form of motion and force commands and the robot in the form of position, speed and force data. The control box transfers information to or reads information from the computer when stated by the Teach Pendant.
The attached teach pendant is used to write the motion and force commands of the program to be executed. The software installed is called PolyScope and allows the end user to write programs using built-in UR Caps and visual representations of robot positions. The teach pendant allows a USB connection to load and run programs.

"Compute box: A unit provided by OnRobot along with the sensor. It performs the calculations needed to use the commands and applications implemented by OnRobot. It needs to be connected to the sensor and the robot controller." **[31]**

5.2. UR3 setup

Before being able to start conducting experiments and in order to work efficient and achieve good results, the hardware setup of the robot is important. The basic and most important steps are explained in this paragraph because this provides extra information on how difficulties or problems during the laboratory sessions are tackled.

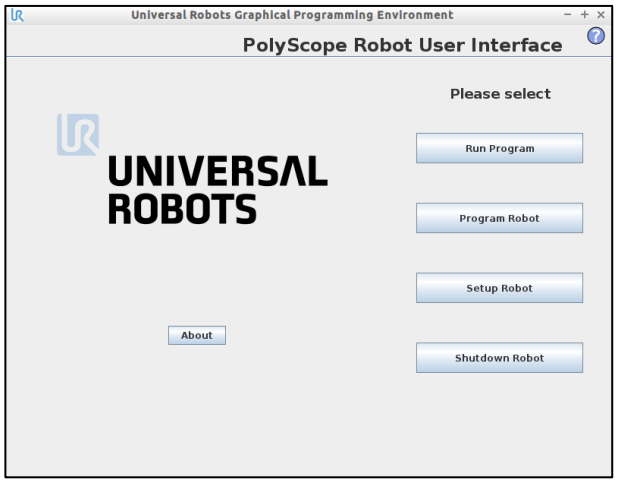


Figure 26: UR3 start screen

Figure 27: UR3 'About' tab

Start screen of UR3, when turning on the robot this screen appears and allows different actions.
'Run Program' is used to move the robot around and run a built program, when the program does not require further changes, this gives a better overview of the variables.
'Program Robot' is used to create and browse through created programs and load these to the robot, with the goal to change the code.
'Setup Robot' is the settings screen of the robot, internet, language, and even time settings are changed here.
'About' gives more information about the robot.

The first step to be taken is 'Setup Robot' and then 'Initialize Robot', to configure the mounting of the robot to the real one. When programming the robot, the mounting will be set as the default view and an accompanying coordinate system is created accordingly.
It is possible to move the robot manually using digital arrow keys, so a logical mounting should be

chosen depending on the relative position of the manipulator to the robot.

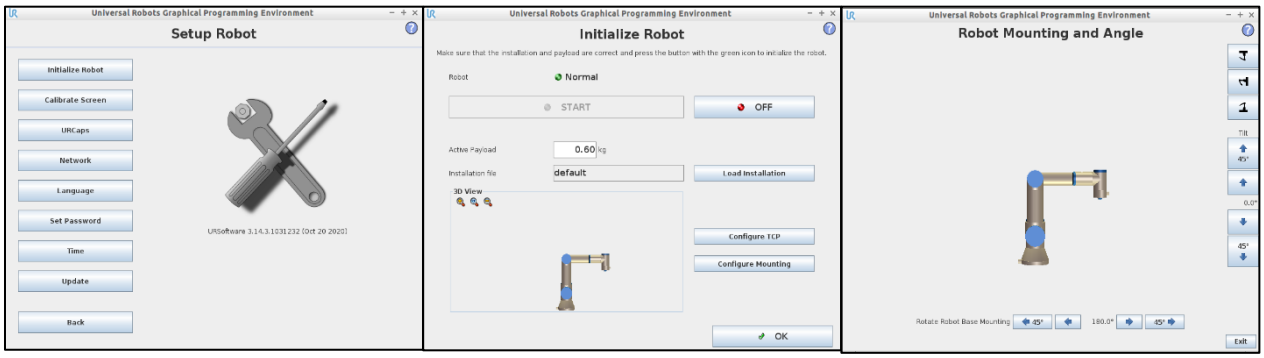


Figure 28: Setup Robot - Mounting initialization

The next thing to do in 'Setup Robot' is connect the robot to the Ethernet network by assigning it an IP address. DHCP automatically provides an IP address, while the static option gives the possibility to choose yourself.

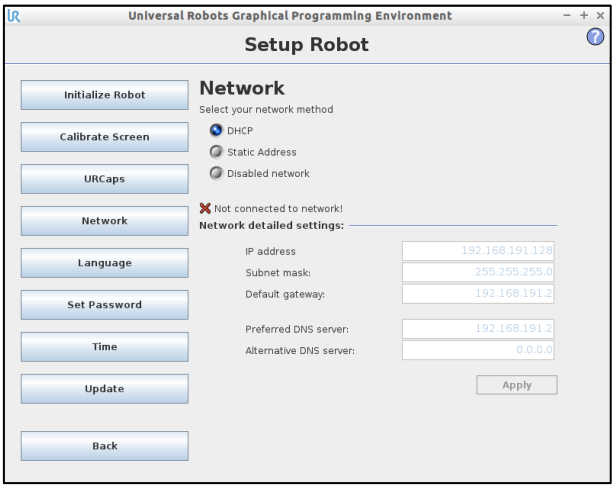


Figure 29: Setup Robot - Network settings

Now the robot is ready to be programmed.

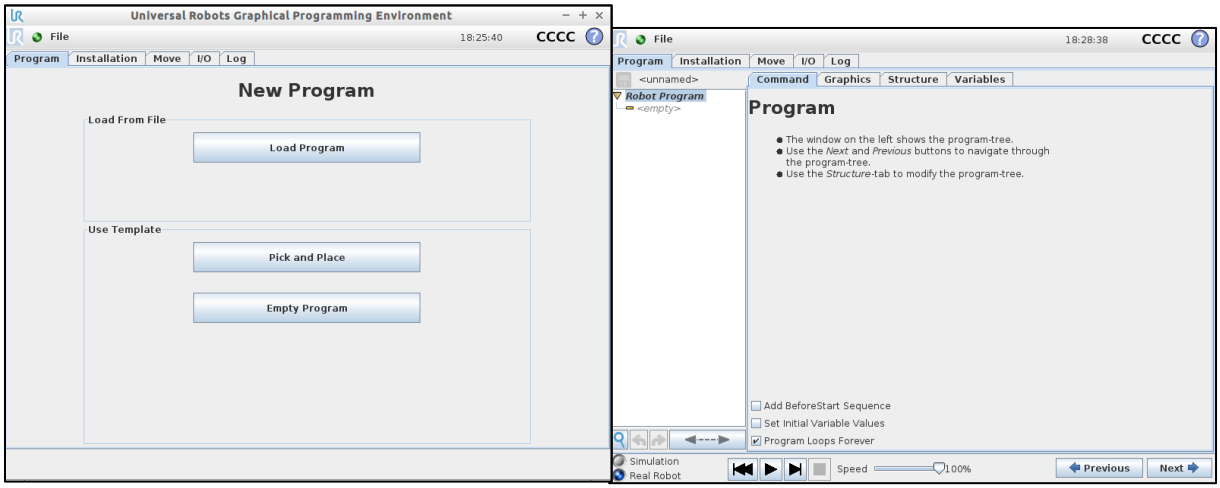


Figure 30: UR options in creating or loading a new program

There is the opportunity to load a program, from a USB or from the internal storage of the teach pendant or create a new program.
Programs are written in a node structure, to make the steps visualizable and when running the code, the active step is highlighted. New code is added in the 'Structure' tab on the top right of the screen

and the parameters of the code can be adjusted in the current visible tab 'Command'.
Universal Robot provides prebuilt commands in the form of UR Caps, that can be downloaded and only require variables to complete them.
"UR Capabilities or URCaps are hardware and/or software extensions for the Universal Robot system. The purpose of URCaps is to seamlessly extend any Universal Robot with customized functionality. Using the URCap Software Platform, a URCap developer can define customized installation screens and program nodes for the end user. These can, for example, encapsulate complex new robot programming concepts, or provide friendly hardware configuration interfaces" **[29]**. This finds it purpose when extending the cobot's functionalities with a camera or a force torque sensor.

During the program it is useful to store data, this is done using variables. Different data types such as lists, strings and numbers can be stored this way and are available for calculations further down the program tree and monitoring in the following screen [Figure 31: UR demonstration 'Variables' tab]. The UR program also allows to set waypoints and save these under a name, to be reused later.

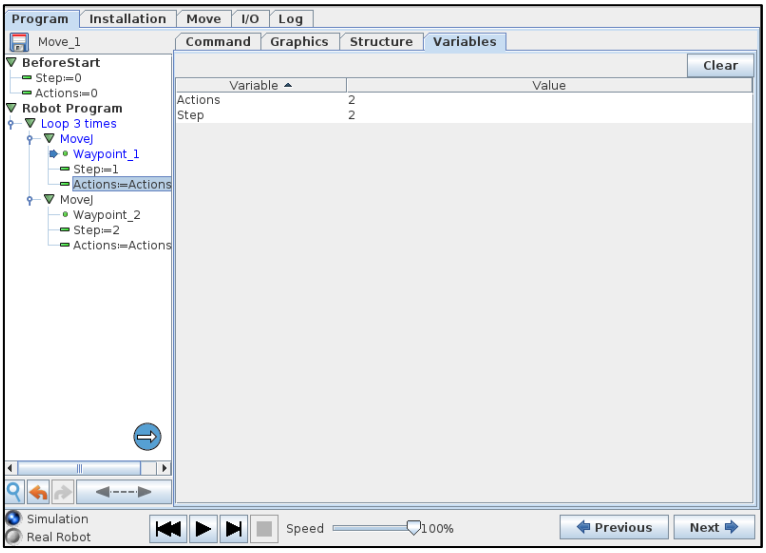


Figure 31: UR demonstration 'Variables' tab

Before any program with movements is run, the robot moves to the first set position. When re-running the program to test its functions, it can be time consuming to move the robot back and forth. A given solution is to not use the real robot and simulate the experiment using the button in the bottom left corner. The movements of the robot can be tracked in the 'Graphics' tab of the program.

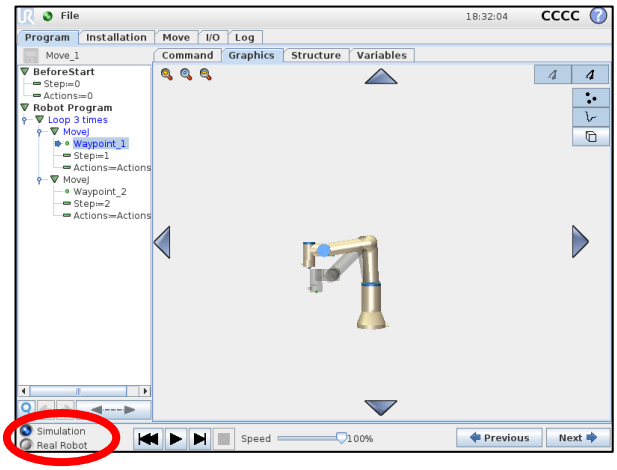


Figure 32: UR demonstration of simulation and 'Graphics' tab

Because both UR3 and UR3e use the same software, the programs written on the UR3 can be read and executed by the UR3e and vice versa, except when using the URCap F/T 'Control', this one is stored under the name of 'Force mode' on the UR3e. This is because the UR3e has an internal force torque sensor and the UR3 has an external one, compatible with the following URCaps.

F/T Center	F/T Fix and rotate
F/T Control	F/T Guard
F/T Insert box	F/T Move
F/T Route	F/T Insert part
F/T Search	F/T Stacking
F/T Waypoint	F/T Zero
F/T Set load	

Table 3: URCaps accompanying OnRobot HEX F/T sensor

Important specifications of UR and sensor

	UR3	UR3e
Update frequency (control period)	125Hz (8ms)	500Hz (2ms)
Force resolution in Z-direction	0,8N	1N
Repeatability	0,1mm	0,03mm

Table 4: Important specifications of UR robots

The accuracy of the sensor compute box is limited to one decimal due to internal filtering when operating at a frequency up to 500Hz. Higher decimals are possible when receiving data at a lower frequency. **[31]**

UR Script commands

moveJ(q, a, v, t, r)	linear movement in joint-space to position q, with time as priority
moveL(pose, a, v, t, r)	linear movement in joint-space to pose, with time as priority
moveC(pose_via, pose_to, a, v, r, mode)	circular movement in tool-space.
moveP(pose, a, v, r)	blend circular in tool-space and move linear in tool space to pose
pose_trans()	transform the current pose to set a new origin
socket_open(address, port)	set up TCP/IP ethernet communication socket with host IP address and port number.
socket_read_ascii_float(number)	reads a number of ascii formatted floats from the socket
get_actual_tcp_pose()	returns current tool pose (X, Y, Z, Rx, Ry, Rz)
get_actual_tcp_speed()	returns current TCP speed (X, Y, Z, Rx, Ry, Rz)
speedL(speed_vector, a, t, a_rot)	linear movement in cartesian space with acceleration a and then constant speed vector
force_mode(taskframe, selectionvector, wrench, type, limits)	taskframe is the current tool pose, selectionvector and type combine for the force control direction ([0,0,1,0,0,0] and 2 for Fz). Wrench sets the magnitudes and limits are the speed values.

Table 5: Most important used UR Script commands

Gathered knowledge/study of topics

- Use of UR Caps and other commands on teach pendant
- MODBUS communication on Teach Pendant
- Python socket communication
- Python Excel writing
- Python web scraping
- Python NumPy
- Python UDP communication with compute box
- Python data transformations
- Kalman filter
- Impedance control

5.3. Preliminary experiments

Getting to know the basic robot and Python commands.

	UR3 (CB)	UR3e	Compute box	PC
IP addresses until 11/04:	158.42.206.10	158.42.206.89	158.42.206.1	158.42.206.7
IP addresses after 11/04:	192.168.1.132	192.168.1.133	192.168.1.98	192.168.1.103
Personal laptop IP	192.168.1.102			

Table 6: IP addresses of robots, compute box and computers

It is important to state the way different commands and functions work, in every application it is important to test the basic functions separately before trying complex tasks. This way it is possible to verify the accuracy and even feasibility of the application, possible system faults, shortcomings and calibration errors are identified this way and considered when using these same commands in next experiments.
Especially when working in a real environment and when you are new to the subject it is recommended to test the hardware and software functionality. As research the UR Script manual served its purpose **[48]** to know the different commands and their required parameters.

5.3.1. UR – PC position communication [Appendix I – UR – PC communication]

Goal

In the pursuit of robot remote control using the computer, this experiment is set up with the goal of connecting the robot to the computer and transferring data between the components.
In this particular experiment the goal is for the robot to send its tool center point position to the computer and for the computer to send the next position coordinates to the robot. This is read out and the robot executes the movement.

Theory

TCP IP socket communication.
TCP stands for Transmission Control Protocol and is an IP protocol where first a connection is set up between the two components using the host's IP address and a specific port number. The connection is set up with a three-way-handshaking and only after this is executed, data transfer is possible.

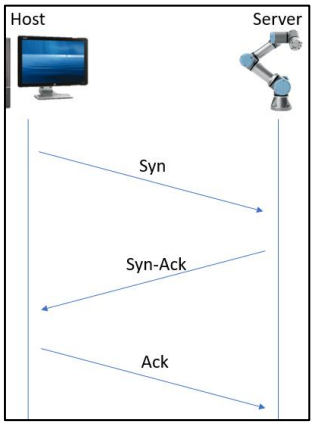


Figure 33: TCP IP: Synchronization, synchronization acknowledgement and acknowledgement

Since TCP is connection oriented, it provides a reliable connection, and the data delivery is in-order. If by mistake data is sent to another port, this gives an error because the connection is refused.

In Python a TCP socket connection is classified as a stream socket and coded: socket.SOCK_STREAM

Setup

This experiment is executed on the UR3, a socket TCP IP communication is set up with the PC and a Python program is written. The robot stores its start position in a variable and sends this to the computer. The computer is used to send a waypoint in cartesian coordinates, the Teach Pendant reads this, stores it in a variable and moves to this waypoint. Next, the robot measures its position and moves 250mm in the z-direction, after which it measures its position again and moves 250mm in the negative z-direction. To finish it returns to the initial position.

Results

This eliminates programming every waypoint on the Teach Pendant, because it is easier to change the Python code and have a compact execution code on the Teach Pendant. The downside of this is that knowledge is needed about the used coordinate system, due to the robot workspace limitations.

After doing more experiments this method can be seen as inefficient, because the teach pendant allows for more than relative movements. It can be made possible to repeat the first steps, where the computer sends coordinates to the robot and the robot moves towards this point, to do the other movements as well and thus eliminating coordinate calculations.
Another simplification would be to save each received value as a different value, making it easier to move back to a certain point.

5.3.2. Basic force control [Appendix II – Basic force control]

Goal

To test out the force control function of the robot, this experiment is set up. This is not state-of-the art but vital to check all functions. The goal is to move the robot to the surface and for it to maintain a certain value while executing a horizontal movement.

Theory

With the UR3's external sensor it is possible to load URCaps onto the Teach Pendant and expand the prebuilt commands for easy integration of the sensor module.
F/T Control allows different force control configurations.

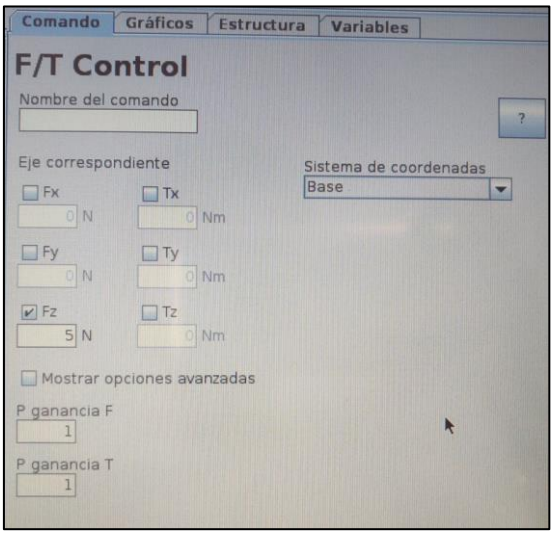


Figure 34: UR3 F/T Control parameter options

Setup

The robot moves its tool with moveJ to a safe distance from the table, making this motion harmless so speed and trajectory do not have to be controlled. From this position, MoveL is used to have a controlled vertical trajectory until a vertical force of 2N is measured, meaning the robot touches the surface. Using F/T Control and F/T move it is possible to control the applied force along a set of waypoints, a direction or a predefined absolute or relative route. The motion back is the same: first MoverL to a safe distance and then MoverJ to the initial position.

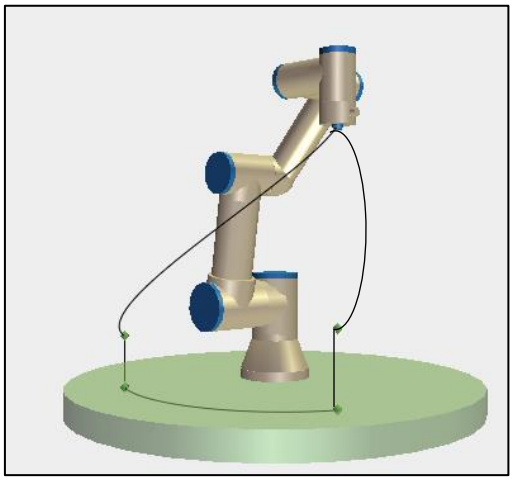


Figure 35: Movements UR3 of basic Force control.

The first and last movement are not in a straight line because these are executed with joint movements, while the other vertical ones are executed in cartesian space. The force control movement path in this experiment is the arc on the green surface.

Results

The robot successfully executes this movement and in the 'variables' tab it is possible to monitor the output values, being Fx, Fy, Fz, Tx, Ty, Tz and F3D.

5.3.3. Remote robot force reading [Appendix III – Remote robot force reading]

Goal

The goal is to connect the computer to the robot, in order to send and receive force data. This allows for displaying results and monitoring the process, because it is possible to keep a data record.

Setup

On the Teach Pendant a port is opened to establish a socket communication, using the following command with the computer IP and a chosen port socket_open("158.42.206.7",30000). The connection is realized by the computer via Python, where a socket.bind command is used with its own IP address and the same port as used by the Teach Pendant.

The robot moves the tool towards the surface and executes a force control movement. In parallel it updates and sends a variable every 8ms, containing the Fx, Fy and Fz values as a String.
The computer receives this, decodes this byte array and then splits it into a list of three Strings. Finally, it prints them with their label and is able to convert them into floats for calculations.

Results

This provides quantitative feedback for the force control application. This program sets up the base for calculations, data storage and further evaluation of processes, because data transmission between two components is possible. This is an example of distributed control.
The output looks like this:

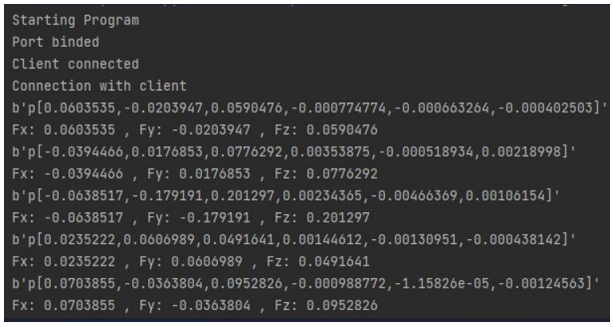


Figure 36: Remote robot force reading Python output

The data "b'p…" is the actual received, encoded data from the robot. This is then decoded and split into the desired Fx, Fy and Fz values, which are displayed in the next line.

5.3.4. Remote control of UR3/UR3e [Appendix IV – Remote control]

Goal

The goal is to set up a connection between computer and robot, to make communication possible without the Teach Pendant.

Setup

By writing full commands in byte form and sending these to the robot, it should be possible to control the robot.

In order to create a TCP socket communication with the robot, the socket needs to be opened and this has to be written on the Teach Pendant.

Results

This was unsuccessful on both UR3 and UR3e. After further research it was noted to remove the Teach Pendant from the control box in order to make this work. This was not tested, because there was still no guarantee.

```
program start
turning on digital output port2
command sent
command sent
Getdata
b'HTTP/1.1 408 Request Timeout\r\n'
finish
WARNING:ursecmon:tried 11 times to find a packet in data, advertised packet size: -2, type: 3
WARNING:ursecmon:Data length: 44
WARNING:ursecmon:tried 11 times to find a packet in data, advertised packet size: -2, type: 3
WARNING:ursecmon:Data length: 68
Traceback (most recent call last):
  File "E:/Experiments/d.RemoteControl/RemoteControlUR3e_Fail.py", line 40, in <module>
    robot = urx.Robot("192.168.1.133")
  File "C:\RoboDK\Python37\lib\site-packages\urx\robot.py", line 27, in __init__
    URRobot.__init__(self, host, use_rt)
  File "C:\RoboDK\Python37\lib\site-packages\urx\urrobot.py", line 38, in __init__
    self.secmon = ursecmon.SecondaryMonitor(self.host)  # data from robot at 10Hz
  File "C:\RoboDK\Python37\lib\site-packages\urx\ursecmon.py", line 252, in __init__
    self.wait()  # make sure we got some data before someone calls us
  File "C:\RoboDK\Python37\lib\site-packages\urx\ursecmon.py", line 344, in wait
    raise TimeoutException("Did not receive a valid data packet from robot in {}".format(timeout))
urx.ursecmon.TimeoutException: Did not receive a valid data packet from robot in 0.5
WARNING:ursecmon:tried 11 times to find a packet in data, advertised packet size: -2, type: 3
WARNING:ursecmon:Data length: 1092
WARNING:ursecmon:tried 11 times to find a packet in data, advertised packet size: -2, type: 3
WARNING:ursecmon:Data length: 1454
INFO:ursecmon:Remove 79 bytes of garbage at begining of packet
```

Figure 37: UR remote control output error display

5.3.5. Attempts direct sensor reading [Appendix V – Attempt direct sensor reading]

Goal

To improve the force control of the robot it would be useful to gather information at a higher rate than the robot can, this gives a buffer for calculations and the opportunity to diminish noise and data irregularities. Different experiments were set up to create a direct connection with the sensor, without the use of the Teach Pendant.

The first one is reading the webpage of the sensor, where the data is available for monitoring. When typing the IP address of the compute box, a website is visible with the live values. The goal is to extract data from this webpage and display it on the computer.

The second one is by creating a connection with the compute box. A TCP connection is set up and data requests are sent for the compute box to react to.

Theory

Web scraping is the process of extracting content and data from a website. The HTML code is extracted, and this content can be used elsewhere. It is used by search engines as Google, price comparison sites and more.

First the webpage is visited, and its URL address is used, then by inspecting the page the HTML structure becomes visible and allows to locate where in the structure the data to be extracted is located. When all of this is known, a Python program can be written to move through the structure and extract and store the HTML data.

Finally, another attempt was by manually assigning the compute box as the listener in the connection. In the code a TCP client is assigned and a server, so it should be possible to set the compute box of the sensor as the server.

Setup

Web scraping:
The first step is to find the website and expect its HTML code to find the data to be extracted.
For this experiment this is the sensor compute box IP address, but the old one: 158.42.206.

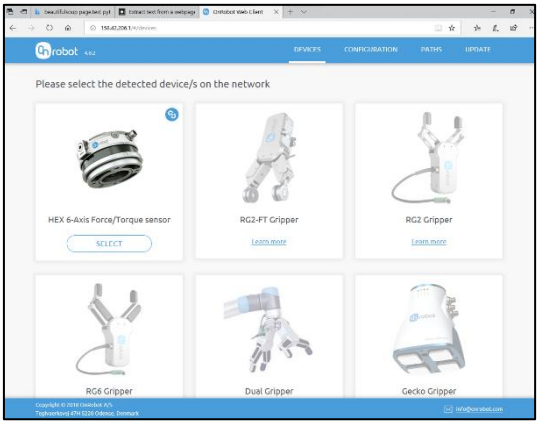


Figure 38: OnRobot HEX F/T Sensor web display

After selecting the sensor, the force data is visible, and it is possible to see where the data is stored in the HTML code.

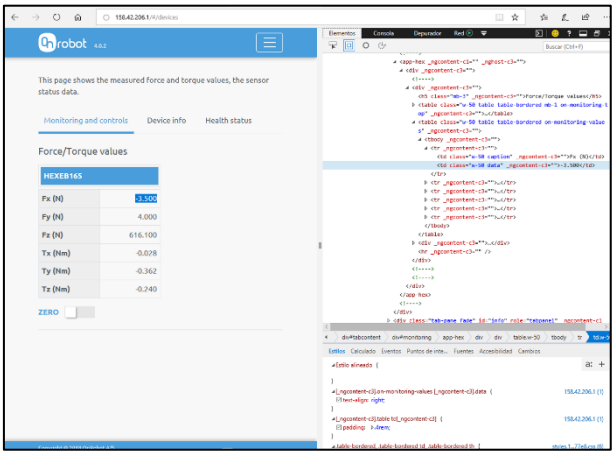
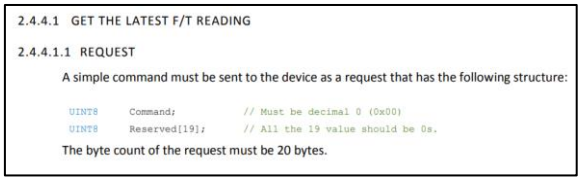


Figure 39: OnRobot F/T Sensor web display HTML code for sensor reading

The next and last step is to write the Python code using the BeautifulSoup and requests package and run it to display and store the extracted data.

Compute box connection:
In its manual [Figure below] it is said to be possible to read data via a TCP connection on port 49151. Requests should be sent to the robot in the form of bytes.



2.4.4.1 GET THE LATEST F/T READING

2.4.4.1.1 REQUEST

A simple command must be sent to the device as a request that has the following structure:

```
UINT8    Command;          // Must be decimal 0 (0x00)
UINT8    Reserved[19];     // All the 19 value should be 0s.
```

The byte count of the request must be 20 bytes.

Figure 40: OnRobot F/T Sensor compute box request data format

Results

Web scraping:
The experiment was successful in the way that HTML structure is extracted and available, but the results were unsuccessful because the force data is not shared in the HTML code of the website. A private, secured connection is set up and thus unavailable for the Python program to access this information.

Compute box connection:
The failure of this experiment is possibly due to a poor data byte conversion or socket setup.

```
Traceback (most recent call last):
  File "E:/Experiments/d.RemoteControl/TestReadWebpage.py", line 15, in <module>
    for i in table1.find_all('td'):
AttributeError: 'NoneType' object has no attribute 'find_all'
<Response [200]>
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title></title>
  <base href="/">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="/assets/images/misc/favicon.png">
<link rel="stylesheet" href="styles.12b29e5c3735a91777e8.css"></head>
<body>
  <app-root></app-root>
<script type="text/javascript" src="runtime.a66f828dca56eeb90e02.js"></script><script
type="text/javascript" src="polyfills.dec546e0e001fb45c7d2.js"></script><script type="
text/javascript" src="scripts.8cf4f555c24f67d7419e.js"></script><script type="text/
javascript" src="main.72b2c6f707bdb4815631.js"></script></body>
</html>

content:
b'<!doctype html>\n<html lang="en">\n<head>\n  <meta charset="utf-8">\n  <title></title>\
n  <base href="/">\n  <meta name="viewport" content="width=device-width, initial-scale=1
">\n  <link rel="icon" type="image/x-icon" href="/assets/images/misc/favicon.png">\n<link
 rel="stylesheet" href="styles.12b29e5c3735a91777e8.css"></head>\n<body>\n  <app-root></
app-root>\n<script type="text/javascript" src="runtime.a66f828dca56eeb90e02.js"></script
><script type="text/javascript" src="polyfills.dec546e0e001fb45c7d2.js"></script><script
type="text/javascript" src="scripts.8cf4f555c24f67d7419e.js"></script><script type="text/
javascript" src="main.72b2c6f707bdb4815631.js"></script></body>\n</html>\n'
<!DOCTYPE html>

<html lang="en">
<head>
<meta charset="utf-8"/>
<title></title>
<base href="/"/>
<meta content="width=device-width, initial-scale=1" name="viewport"/>
<link href="/assets/images/misc/favicon.png" rel="icon" type="image/x-icon"/>
<link href="styles.12b29e5c3735a91777e8.css" rel="stylesheet"/></head>
<body>
<app-root></app-root>
<script src="runtime.a66f828dca56eeb90e02.js" type="text/javascript"></script><script src
="polyfills.dec546e0e001fb45c7d2.js" type="text/javascript"></script><script src="scripts
.8cf4f555c24f67d7419e.js" type="text/javascript"></script><script src="main.
72b2c6f707bdb4815631.js" type="text/javascript"></script></body>
</html>

None

Process finished with exit code 1
```

Figure 41: Direct sensor reading - webscraping1 with Beautifulsoup

```
Traceback (most recent call last):
  File "E:/Experiments/e.AttemptsDirectSensorReading/TestReadWebpage2.py", line 9, in <
module>
    req = urllib.request.urlopen('http://192.168.1.98/#/devices')
NameError: name 'urllib' is not defined
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title></title>
  <base href="/">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="/assets/images/misc/favicon.png">
<link rel="stylesheet" href="styles.12b29e5c3735a91777e8.css"></head>
<body>
  <app-root></app-root>
<script type="text/javascript" src="runtime.a66f828dca56eeb90e02.js"></script><script
type="text/javascript" src="polyfills.dec546e0e001fb45c7d2.js"></script><script type="
text/javascript" src="scripts.8cf4f555c24f67d7419e.js"></script><script type="text/
javascript" src="main.72b2c6f707bdb4815631.js"></script></body>
</html>


Process finished with exit code 1
```

Figure 42: Direct sensor reading - webscraping2 with requests

```
Port binded
Client connected
Connection with client
Traceback (most recent call last):
  File "C:/Users/jdesme/Desktop/ConnectionSensorFT_24_03.py", line 21, in <module>
    client.connect(('158.42.206.1', 40000))
ConnectionRefusedError: [WinError 10061] No se puede establecer una conexión ya que el equipo de destino denegó expresamente dicha conexión

Process finished with exit code 1
```

Figure 43: Direct sensor reading - compute box direct communication error

5.3.6. Force control with excel writing [Appendix VI – Force control with excel writing]

Goal

Gathering and displaying force data on the computer is great for verifying the output and checking it on program errors. If the extracted data needs in-depth analyzation or further calculations, it is more useful to store it in an excel file. The goal here is to create a Python program that creates an excel file and writes data into it.

Setup

First communication is opened, the robot moves to the table with MoveJ, then force mode is started, with a vertical force of 5N. MoveL is used to make a horizontal movement on the table.
In the subtask of the UR Script the speed values are retrieved; an array with the first three values being the cartesian speeds, then the other three are angular velocities. (Universal Robots).

The movement on the table is in the negative Y-direction, so when the norm of the y-speed value is bigger than 20mm/s (0.02) the force control movement has started and thus the vertical force has been applied to the table. This is the ideal condition to start sampling force data.

A trigger signal “var_start_send” with value 1 is send at this point in time. This is read by the computer and the robot. For the robot this is the start signal to send force values and for the computer to read messages received from the robot, print them, and store them in an Excel file as well.

In the Python program, an excel file is created using ‘XlsxWriter’, a workbook is opened and given a name, which will be the document name, within this workbook a worksheet is created, where the data will be written onto. worksheet.write(a,b,c) allows to write data c to row a and column b, this way the first row in the file contains the data titles, being time, Fx, Fy and Fz.
Every 8ms the computer receives the force values and calculates the elapsed time, and this is repeated during 2 seconds. Depending on the application it is possible to choose for how long samples are read and stored.
This application is used in further applications to store and evaluate data efficiently.

Results

This base code can be used to story any kind of information from the Python program to an Excel file. Information can be saved, transferred, and manipulated to create graphs and compare different experiments.
The output of this program is as expected, an excel file is created under a given name, with on each row the total elapsed time, force in X, force in Y and force in Z.

	A	B	C	D
1		Fx	Fy	Fz
2	0.015994	0.5	-1.8	-5.50079
3	0.023991	0.5	-1.8	-5.50079
4	0.031989	0.5	-1.8	-5.60083
5	0.039986	0.4	-1.6	-5.80078
6	0.047984	0.4	-1.4	-5.90082
7	0.055983	0.4	-1.5	-6.00079
8	0.063979	0.4	-1.5	-6.00079
9	0.072007	0.4	-1.5	-6.10083
10	0.080004	0.4	-1.5	-6.20081
11	0.087979	0.4	-1.5	-6.10083
12	0.095979	0.4	-1.6	-5.80078
13	0.103966	0.4	-1.6	-5.80078
14	0.111981	0.4	-1.5	-5.70081
15	0.119975	0.4	-1.5	-5.60083
16	0.127977	0.4	-1.5	-5.40082
17	0.135959	0.4	-1.5	-5.40082
18	0.143956	0.4	-1.5	-5.40082
19	0.151973	0.4	-1.5	-5.20081
20	0.159978	0.4	-1.5	-5.20081
21	0.167965	0.4	-1.5	-5.10083

Figure 44: Force control with excel writing Excel output of force values

5.3.7. Adaptive force control [Appendix VII – Adaptive force control]

Goal

The robot force control action contains a lot of noise, and this causes oscillations in the force control action. By applying direct sensor reading, noise is filtered out trough averaging or a second order filter and from this value a new calculated value is send to the robot as reference force value.
Any constant offset due to disturbances will be filtered out, as well as periods where the force value differs from the reference, especially if the force varies at a slow rate.

Theory

The idea behind this experiment is described with examples in the table below.

Step	F_{ref}	$F_{average}$	e_f: $F_{ref1} - F_{average}$	Compensated: $F_{ref} + e_f$
1	5	4	1	6
2	6	5.2	-0.2	5.8
3	5.8	5.05	-0.05	5.75
…				
Step	F_{ref}	$F_{average}$	e_f: $F_{ref1} - F_{average}$	Compensated: $F_{ref} + e_f$
1	5	6	-1	4
2	4	4.8	0.2	4.2
3	4.2	5.05	-0.05	4.15
…				

Table 7: Illustrative examples adaptive force control experiment

Every 8ms the robot updates its force control value to F_{ref} and $F_{ref1}$is the true desired force value (5N in this case).

Setup

The computer is connected to the Teach Pendant with a TCP IP connection and to the sensor compute box with a UDP IP connection, initialized in the Python code.

The robot moves towards the surface and once contact is made, force control is applied, and the tool is moved in one direction over a fixed distance. In parallel of this process, every update period of 8ms, the teach pendant reads and stores the received value from the socket connection as a variable ‘f_correction’. Force mode is activated and chooses the type of force control and its magnitude. For the magnitude of the force ‘f_correction’ is used, updating the force setpoint every 8ms.

The computer reads sensor force data every 2ms and every 8ms computations are done and a compensated value is sent to the teach pendant.

Results

Output of the program can be found in [Figure 45: Output graphs adaptive force control].
The multirate filter with averaging and without compensation has the highest peak force and reaches a steady state just after 4seconds, while with compensation this is reduced to just under 3 seconds.

The second order multirate filter is overall better, the compensation gets the force control to steady state in under 3 seconds.

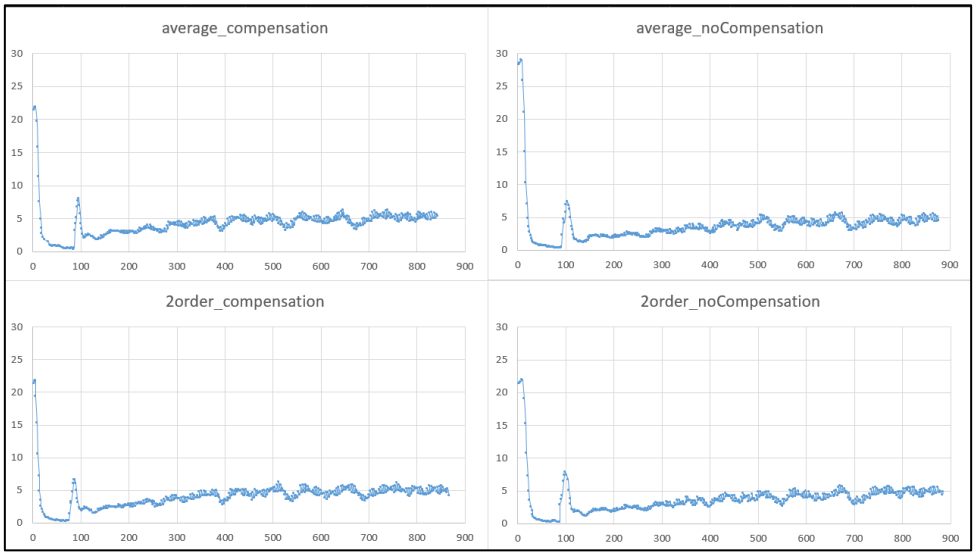


Figure 45: Output graphs adaptive force control

Overall, this method works poorly. The desired value is set to 5N, and this is only reached after 4 seconds. It is better to use a proportional control, that reacts to the force error with an added gain, instead of this experiment which is similar to proportional control but with a gain of 1.
Once the desired value of 5N is reached, the output value is stable when compensation is active.

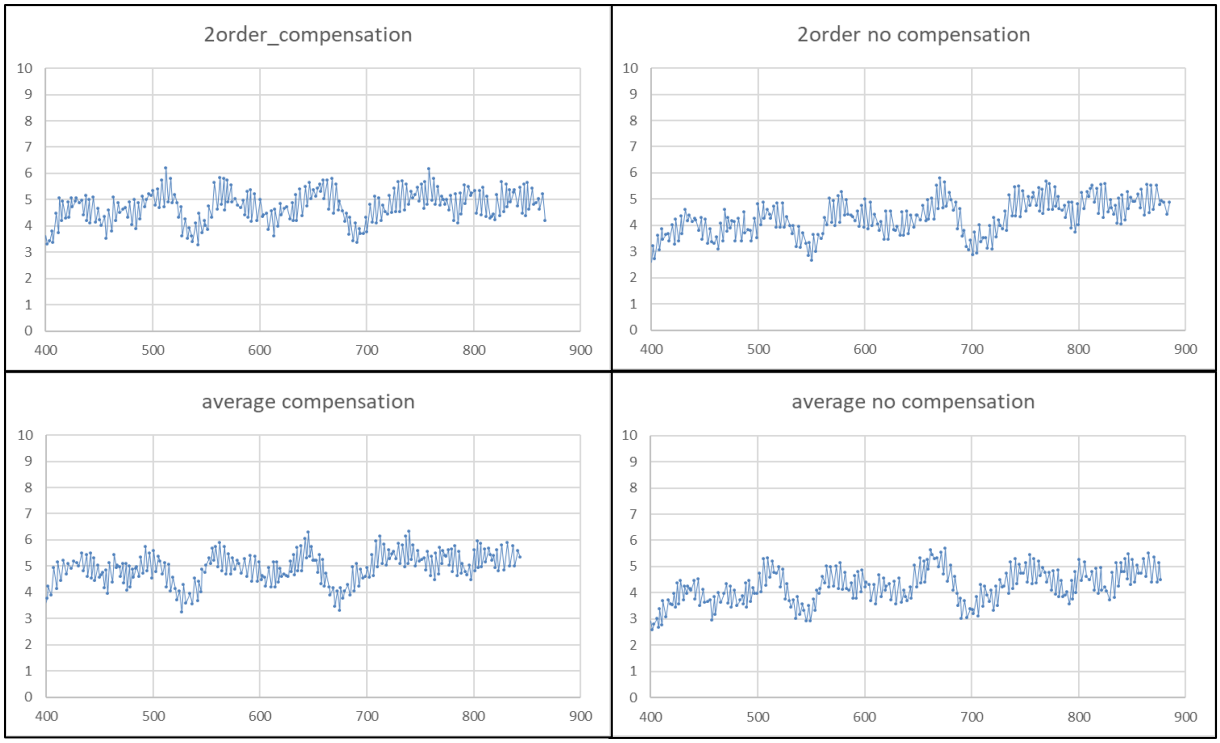


Figure 46: Output graphs adaptive force control - stable

5.3.8. Trapezoidal Velocity Profile PD force control [Appendix VIII – Trapezoidal speed trajectory]

Goal

For the inner loop control of the force control and filtering experiments, motion control is demanded. Therefore, a trapezoidal velocity profile generator is used because it is one of the most common ones and because the limited acceleration and velocity gives it a more stable output than the built-in trajectory generators.

Outer loop control is also applied in this experiment, in the form of a PD controller, reacting every 8ms on the vertical force measured by the sensor. But the main goal of this experiment is making the trapezoidal velocity profile, proportional derivative control is handled in [**5.5 Experimental setup for advanced control**]

Theory

A trapezoidal speed trajectory generator works in three phases, first there is a phase with constant acceleration, then constant velocity and the final phase is constant deceleration.
When all parameters are known (max acceleration, velocity, and distance to travel), it is possible to calculate the formulas in [Table 8: Trapezoidal profile formulas]. These formulas follow from basic physics formulas for motion.

$$acceleration\ a = \frac{dv}{dt}$$

Equation 20: Acceleration in terms of velocity and time

$$velocity\ v = \frac{ds}{dt}$$

Equation 21: Velocity in terms of distance and time

Out of equation 1 and 2, the inverse formulas can be calculated:

$$v = \int a\,dt = at$$

Equation 22: Velocity in terms of acceleration and time

$$s = \int v\,dt = \int a * t\,dt = \frac{a * t^2}{2}$$

Equation 23: Distance in terms of acceleration and time

Rewriting equation 3 gives

$$t = \frac{v}{a}$$

Equation 24: Time in terms of velocity and acceleration

and by combining equation 4 and 5, this gives

$$s = \frac{v^2}{2a}$$

Equation 25: Distance covered during the acceleration phase

The integral in equation 4 is equivalent to saying that the distance is proportional to the surface under the velocity profile and because the profile is symmetrical, the distance covered during the deceleration phase is equal to the one during the acceleration phase.

When the velocity is constant, the distance covered during this time period can be calculated with following formula

$$s = \int_{t1}^{t2} v dt = v(t_2 - t_1)$$

Equation 26: Distance covered during constant velocity phase

which makes it able to calculate the time with constant velocity $t_2 = \frac{s}{v} + t_1$ and t_1 follows out of equation 5.

$$t_2 = \frac{s}{v} + \frac{v}{a}$$

Equation 27: Equation for time at constant velocity

The total time can be calculated and is equal to $t_{tot} = 2 * t_1 + t_2$.

If the covered distance during the acceleration is bigger than or equal to half of the total, there is insufficient time for the constant velocity phase. Instead, a triangular profile is generated, only containing an acceleration and a deceleration phase.

$$s_{tot} = 2 * s_1 = a * t_1^2$$

Equation 28: Equation for total covered distance

Which gives the acceleration time, which is half of the total time. $t_{tot} = 2 * t_1$

$$t_1 = \sqrt{\frac{s}{a}}$$

Equation 29: Equation for acceleration time

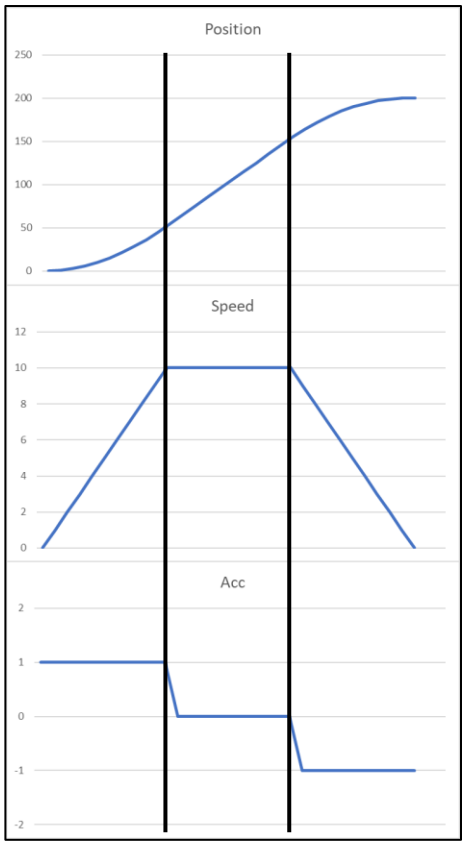


Figure 47: Trapezoidal velocity profile

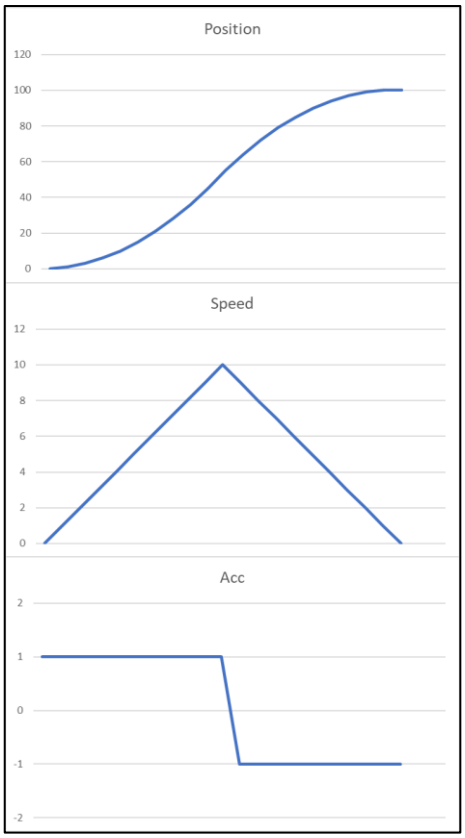


Figure 48: Triangular speed profile

[23]

Trapezoidal	Constant acceleration	Constant velocity	Triangular
s	$\frac{a * {t_1}^2}{2}$ or $\frac{v^2}{2a}$	$v * (t_2 - t_1)$	$\frac{a * {t_1}^2}{2}$
v	$a * t_1$	v	$a * t_1$
a	a	0	a
time	$t_1 = \frac{v}{a}$	$t_2 = \frac{s}{v} + \frac{v}{a}$	$t_1 = \sqrt{\frac{s}{a}}$

Table 8: Trapezoidal profile formulas

Setup

The Teach Pendant's UR Script is used to program the speed trajectory generator and the motion commands, while the computer provides visual feedback on the process. Output data is sent to the computer, which stores the data in excel and allows me to plot graphs and make calculations in order to evaluate the results.

The velocity and acceleration are chosen as constants and the distance is calculated from the coordinates of the waypoint. The formulas from table [8] are implemented to calculate the acceleration and constant velocity time.

The robot descends to the surface, once contact is made the trapezoidal is activated by implementing a constant acceleration with a given maximum velocity. Command 'speedL' allows to give the desired speed values, to be reached with a certain acceleration and when the speed is reached it maintains this value.

To program the deceleration phase, the same command is used but with speed values set to zero and a negative acceleration.

During this whole process proportional derivative controller is implemented with a fixed reference value of 5N and the robot updating the proportional and derivative action every 8ms, depending on the value measured by the sensor.

Results

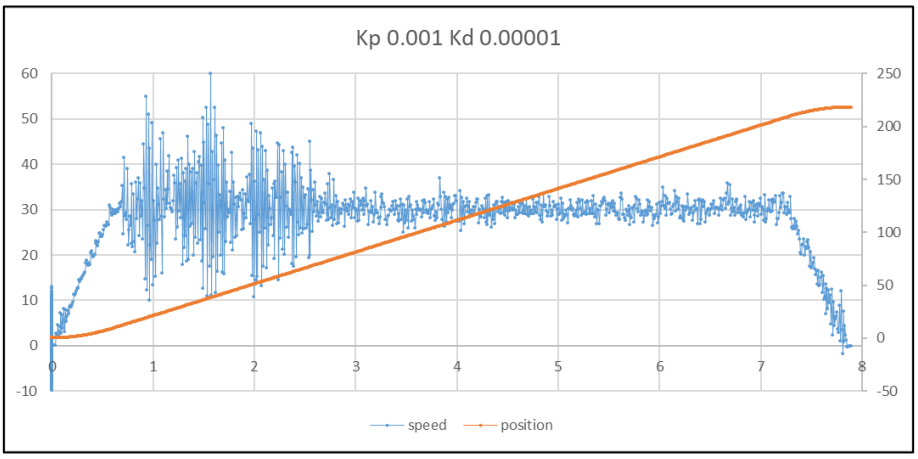


Figure 49: Trapezoidal velocity profile output graph

There is a clear acceleration and deceleration phase. The speed in this case is set to 30mm/s and the acceleration to 50mm/s, giving an acceleration time of 0.6 seconds. This checks out with the output.

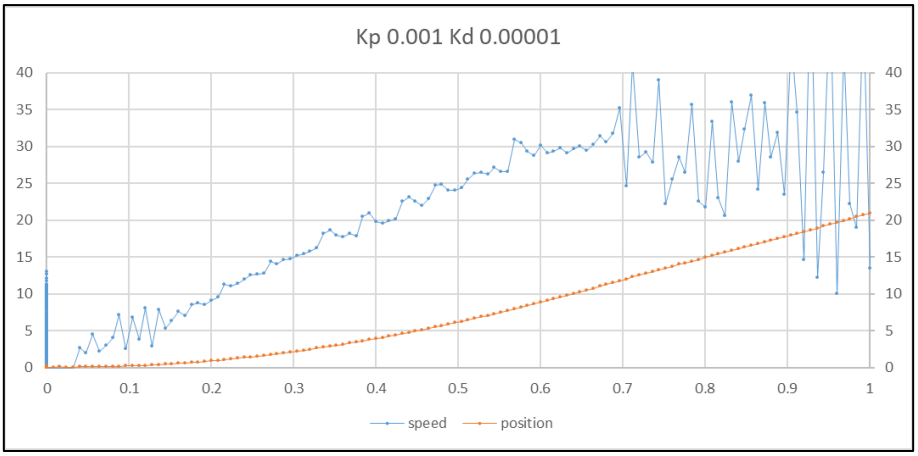


Figure 50: Trapezoidal velocity profile output graph - zoomed in - noise

It is also visible that during the acceleration the position behaves parabolic and during the deceleration hyperbolic, due to the double integration.
Noise exists on the control action due to process noise and the combination in vertical (force control) and horizontal (position control) movement.

5.4. Experimental setup for comparison UR3 and UR3e

The experiments mentioned above are limited in complexity and application range but are necessary to get used to the individual functions before combining them.

To further define the differences between the UR3 and UR3e collaborative robots, more advanced experiments are set up. Table 4: Important specifications of UR robots gives reference values to evaluate the success of the experiments or the difference between theory and practice.

Both robots force controls are compared on impact force and on force maintenance using the built-in force control command, if these robots are used for these tasks, it is importance to evaluate to what degree the accuracy is and what the magnitude of the oscillations on the variable is. If the actual reached value deviates strongly from the setpoint or if the oscillations are significant, this would mean other force control methods should be applied or the robot is not suited for delicate operations, where the force value should be followed in a strict manner.

Because both robots are similar in size, it should be possible to make them collaborate. When handling big objects or performing difficult tasks it can be useful for efficiency purposes or even feasibility of the process to let two or more robots work together. To avoid collisions and the need to control both robots separately, they communicate via a MODBUS protocol. If the experiment gives odd results, this could mean there are (internal or external) differences in the two robots that weren't considered or documented.

Differences in communication time of both robots are evaluated, as a verification of the datasheet values (8ms for UR3 and 2ms for UR3e) and in another experiment, a solution has been found to avoid the slow communication time of the UR3 and receive values from the sensor every 2ms.

5.4.1. Experiment one: Sampling period in force control [Appendix IX – Sampling period force control]

Goal

The goal of this experiment is to evaluate the performance of the UR3 and UR3e's force control functions. This is done by letting them execute the same force control movement and gathering their force values along the way. The goal is to maintain a vertical force of 5N, while performing a horizontal linear movement.

The accuracy of both should be similar, with the UR3 being more accurate with a resolution of 0.8N, while the UR3e has a resolution of 1N. The deviation and the time until the force value is constant are evaluated as well.

Theory

Since the UR3e has an update period of 2ms and the UR3 one of 8ms, it is fairer to evaluate both on the same sample rate. If more values are measured, more values will be closer to the desired one thus bringing the average value closer to this and manipulating the variance.

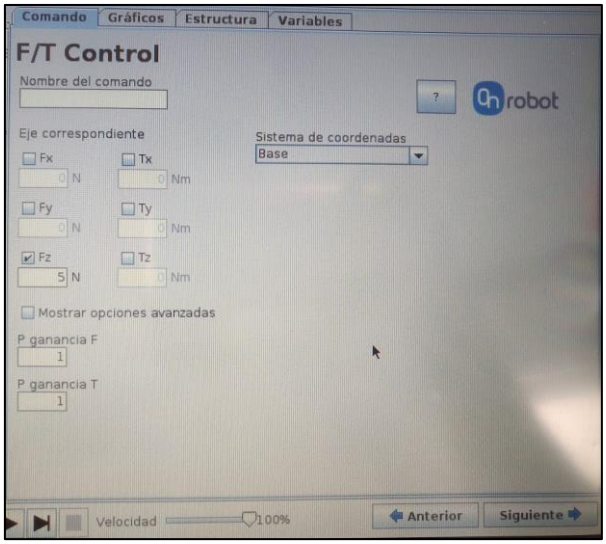


Figure 51: URCap F/T Control configuration options

There are different kinds of 'Force mode', but for this application Simple or Frame are the most convenient.

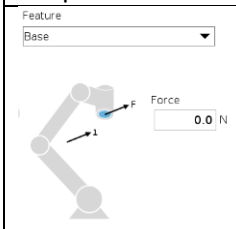
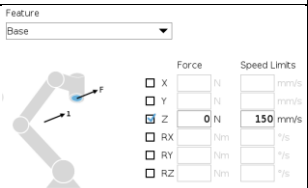
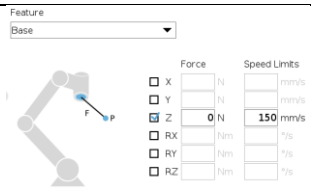
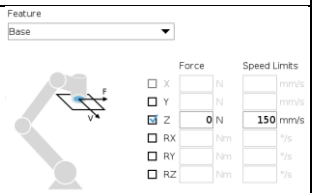
Simple	Frame	Point	Motion
			
Along z-axis of the selected feature	All six degrees of freedom available	y-axis points from robot TCP to the origin of the selected feature, so x- and z-axis change as the position of the robot TCP changes	Task frame (for force control) changes with the direction of the TCP movement

Figure 52: UR3e Force mode configuration options

Setup

The computer is connected to the robot (either UR3 or UR3e, but one at a time) with a TCP IP socket connection. Motion and force control commands are written on the Teach Pendant, while the computer receives, stores, and does calculations with data from the robot.

The robot start pose has the base at 0°, the other joints are set to start the tool at X=-450mm, then the tool is moved vertically towards the table until it touches and after that, the robot executes a unidirectional force control movement of 150mm in the positive X-direction.
The cartesian velocity is set to 25mm/s.

For the UR3, the force control is executed using 'F/T Control' and then the motion command, where a waypoint is set. The UR3e does not support this script, here 'Force mode' has to be introduced. Force mode Frame is used in this experiment, this allows to set the force value in one of the six degrees of freedom and the z-axis in this mode is static.
The Fz force on both robots is set at -5N.

The UR3e needs to update the force values 'manually', by coding. In parallel of the main force control program a subprogram is created and executed in loops, where a variable named 'force_torque' is initialized, holding the get_tcp_force() return values, being all the current forces and torques: Fx, Fy, Fz, Tx, Ty, Tz.

The rest of the program differs depending on the sample type: force or touch.

Force:
This sample is one of 3 seconds and starts whenever the velocity in the x-direction is greater than 20mm/s, because this is meant to be the stable part of the force control. In parallel of the main program the TCP speed is measured and once the value of the speed in the x-direction exceeds 0,02m/s or 20mm/s the robot sends a start signal to the PC and starts sending output variables. Because the control cycle time of the UR3e robot is 2ms, while the one of the UR3 is 8ms, the 'Wait' command in the Loop of the parallel program is once set to 0.002 and once to 0.008 in the UR3e UR program.
This is to verify if, even with the same sampling frequency, the UR3e is more accurate than the UR3.

Touch:

This sample starts when the robot tool descends and continuous to measure the whole force control movement. In order to check how quickly the force control stabilizes, the start signal is immediately sent to the computer. This gives a sample where the impact force and the force control are measured together.

In the Python program, an excel file is created and the different force values are vertically printed next to each other. As soon as the start signal from the robot is received, a timer to end the data gathering and Excel writing starts and for each data exchange the communication time is measured and returned to the Excel file. For sample 'force' the timer is set to 3seconds, while in 'touch' this is set to 10seconds.

Results

	UR3	UR3e	
Force	**Slow (8ms)**	**Slow (8ms)**	**Fast (2ms)**
Average [N]	5.095	4.996	5.005
Deviation [N]	0.633	0.301	0.263
Touch			
Average [N]	4.977	4.994	4.998
Deviation [N]	0.513	0.217	0.221
Peak [N]	10.537	11.093	11.399
Settling time [ms]	399.87	255.91	247.92

Table 9: Sampling period in force control output summary

Settling time is the time difference between the start of the first force peak and the steady-state (+/- 1N).

It can be concluded that the UR3e approaches the required force value more than the UR3. As expected, the higher rate gives better result, but the standard deviation doesn't improve significantly. When the UR3 and UR3e send data at the same rate, the UR3e has a deviation half the size of the UR's.
The UR3 needs more time to get into a steady state, even though its force peak is lower.
The only downsides of the UR3e are the oscillations at a higher frequency than with the UR3 and the slightly higher force peak when touching the surface.

Results are visualized in [Figure 53 - Figure 54] and when evaluating the results from Table 9: Sampling period in force control output summary, it can be seen that the average value alone might not give the best representation of the force control, since symmetrical oscillations are ignored. The deviation on the other hand does not evaluate around what value the oscillations occur.

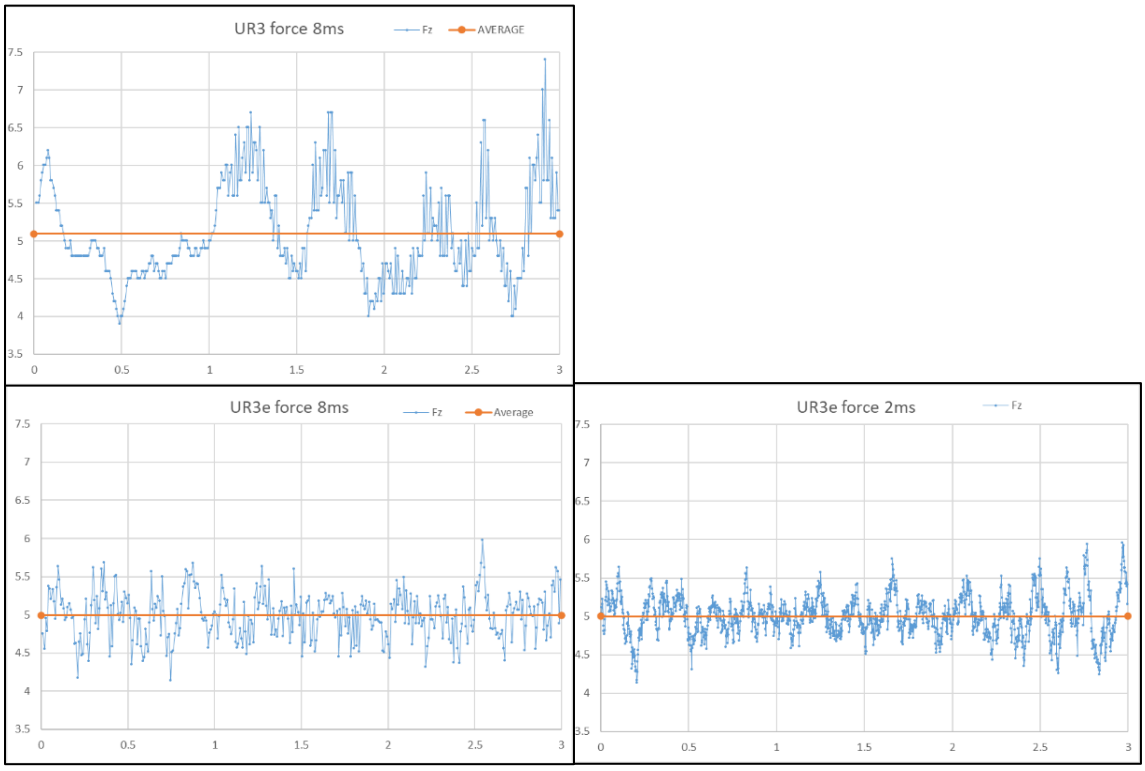


Figure 53: Force samples UR3 vs UR3e

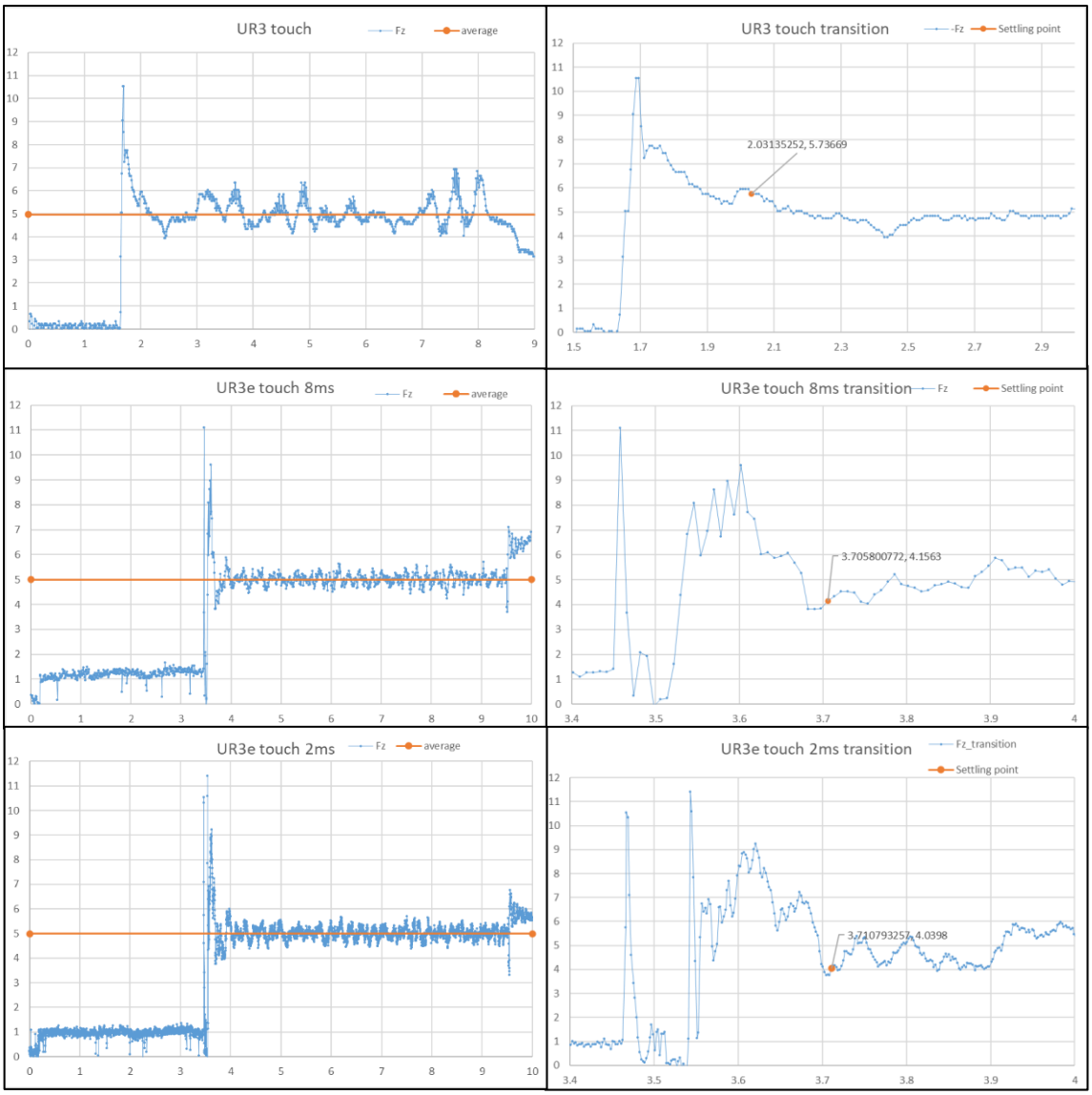


Figure 54: Touch samples UR3 vs UR3e

5.4.2. Experiment two: Multi-robot communication [Appendix X – Multi-robot communication]

Goal

Collaborative robots are widely used in areas where humans can benefit from assistance, to make their work-environment safer and tackle more varying tasks.
In line with distributed control [**3.4**], an experiment is set up to further explore this topic. Different machines and controllers are connected within the same network for data-transfer, and it is possible to set up a client-server connection between the computer and the robots, in this experiment is an attempt to set up a client-server connection between the two robots.

The goal of this specific experiment is to simulate how two collaborative robots (UR3 and UR3e) can cooperate in order to pick up a box in between the two of them, using force control. It is a simulation, so there is no physical object present and the two robots communicate via a MODBUS TCP connection between the two of them. In this communication mode, the computer is eliminated.

Theory

Theory of MODBUS communication:
"Modbus is a serial communication protocol developed by Modicon published by Modicon® in 1979 for use with its programmable logic controllers (PLCs). In simple terms, it is a method used for transmitting information over serial lines between electronic devices. The device requesting the information is called the Modbus Master and the devices supplying information are Modbus Slaves" **[28]**

On the Teach Pendant it is possible to use the IP address of a device and set is as a MODBUS client. Signals are created, which can be setup as register input/output or digital input/output, contains addresses and a name can be assigned to them. This is the data the client can access and monitor. Some register addresses contain internal data such as robot positions, joint angles or self-written data, but all of these variables can from then on be read-out. **[3]**

Setup

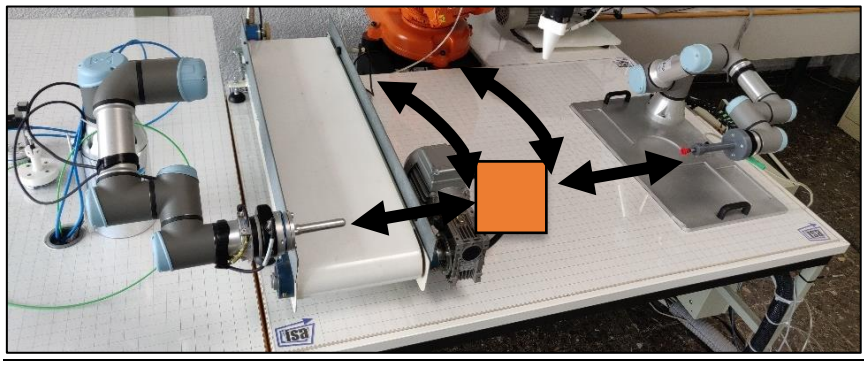


Figure 55: Multi-robot communication - setup with movements and box

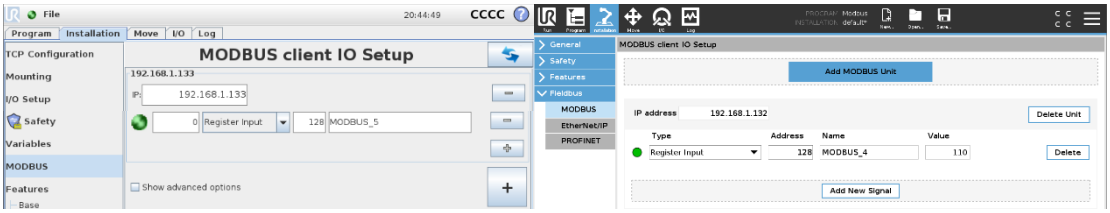


Figure 56: Multi-robot communication - UR3 and UR3e MODBUS settings

Modbus allows to setup registers which are data pathways between the two devices. When a register on one device is set to a certain value, it can be read by the other device.
Each of the robots sets the other as client using its IP address, this way a two-way communication is possible. The master sends the waypoint data and the slave receives it.

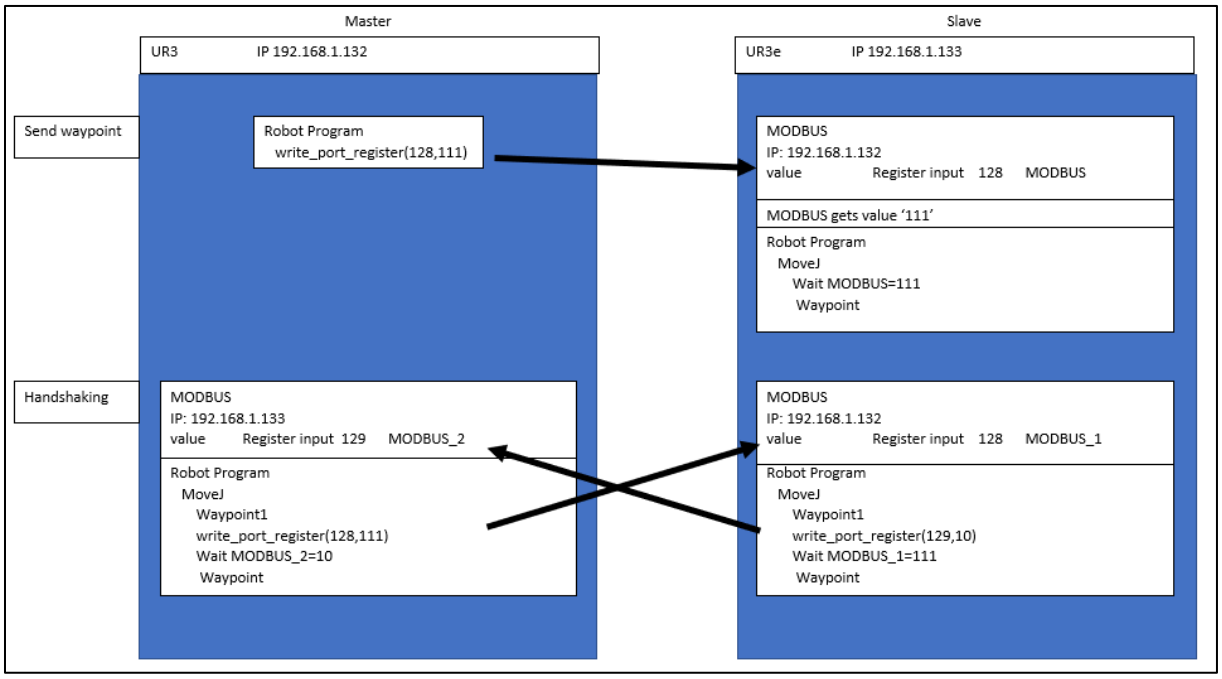


Figure 57: Multi-robot communication - communication logic

Movements are synchronized by executing a handshake, containing the next waypoint: the master sends the data to the slave using 'write_port_register('slave_address', 'data'), this is received and read out by the slave and confirmed by sending data back to the master.
The slave can either move to the waypoint coordinates, but this only works if the two robots use the same coordinate system, or, in case of any offsets, it is better if the slave uses the waypoint data as a relative movement from the current position. Once the waypoint is reached, another handshake is executed.

First the two robots start at a save position and then quickly move towards the box with moveJ, but still at a safe distance from the box. For a correct contact the robots then move linear (in tool-space) towards the box until contact is made (measure force normal to tool).
From this position force control is applied, to verify that the contact force is big enough to pick up the box and yet not too big to damage the object.
After the required movement, the box is put down and then first moveL is executed to move the robot to a safe distance before the two robots execute a next task.

Results

The two robots were able to work together and perform a task at a slow speed in order to evaluate the accuracy. Because both robots can be given speed parameters in cartesian coordinates or in joint velocities, assuring that they operate the same way is no problem. Both robots nearly have the same dimensions and even if these were different, using cartesian speeds eliminates this problem.

This experiment has a few pauses, so it is visible when synchronization or mirroring might occur, which was not the case after a few improvements.

This sets the foundation for a setup where multiple robots can work together at a higher speed, on more complicated tasks that require more than one robot. Extending the program with anterior and or posterior tasks makes it possible to simulate a full process as it will be used in the industry. Another advantage of this arises when one robot's performance is improved using control techniques. Instead of having to copy the whole code for the other robot, the output variables can be transmitted using the MODBUS communication and the two robots perform at the same level.

5.4.3. Experiment three: Communication time delay [Appendix XI – Communication time delay]

Goal

When looking for improvement in control it is important to find where improvement is possible. Out of the manufacturer's datasheet different properties are available, but first experimentational validation of these is needed to find their state in current setup.
With this in mind, this experiment is created to validate the communication delay between the robot and the computer when sending data using TCP IP connection. For the UR3 this should be around 8ms and for the UR3e 2ms.

Setup

The same experiment is executed twice, once by creating a connection with the UR3 and once with the UR3e. The Python and Teach Pendant code are the same for both.

A socket communication is established with the computer as the host and as soon as the connection is established the Teach Pendant creates an empty variable and repeats a loop where it reads the communication socket, once this value is different from zero it sends an integer to the computer. The computer creates an empty variable as well, to store the returned data. Going through the cycle 50 times, it sends data to the Teach Pendant, starts a timer, checks for the return value and once this is received the timer is stopped. The time difference is calculated and printed onto the screen.

Results

This experiment was successful, the results were as given by the manufacturers. The time delay for the UR3 was 8ms and for the UR3e 2ms. Validating this, ways can be found to diminish the effect on delay on the robot's overall and especially force control performance.
Output of the Python program can be found in [Figure 58 - Figure 59].

UR3

```
0:00:00.006028
0:00:00.007997
0:00:00.007998
0:00:00.007997
0:00:00.007998
0:00:00.007998
0:00:00.007997
0:00:00.007997
0:00:00.007997
0:00:00.007998
0:00:00.007997
0:00:00.007998
0:00:00.007997
0:00:00.007968
0:00:00.008027
0:00:00.007998
0:00:00.007997
0:00:00.007997
0:00:00.007969
0:00:00.008026
0:00:00.007972
0:00:00.008024
0:00:00.007997
0:00:00.007997
0:00:00.007998
0:00:00.007967
0:00:00.008028
0:00:00.007967
0:00:00.008028
0:00:00.007988
0:00:00.008009
```

Figure 58: Output time delay UR3

UR3e

```
0:00:00.000993
0:00:00.002000
0:00:00.002000
0:00:00.001999
0:00:00.001999
0:00:00.001999
0:00:00.002000
0:00:00.001999
0:00:00.001999
0:00:00.002000
0:00:00.001999
0:00:00.002000
0:00:00.001996
0:00:00.001999
0:00:00.002007
0:00:00.002000
0:00:00.002000
0:00:00.001982
0:00:00.001999
0:00:00.001999
0:00:00.001999
0:00:00.002012
0:00:00.002000
0:00:00.001999
0:00:00.001999
0:00:00.002000
0:00:00.001999
0:00:00.001999
0:00:00.001999
0:00:00.002001
0:00:00.001998
```

Figure 59: Output time delay UR3e

5.4.4. Appendix XII – Force sensor connection]

Goal

After the verification of the communication time, different experiments were conducted in order to receive force values from the robot at a higher frequency than the control cycle of the robot.
The experiment was unsuccessful using previous setup in [**5.3.5Attempts direct sensor reading [Appendix V – Attempt direct sensor reading]**], until a direct connection with the compute box of the force sensor was set up, which is done in this experiment. The goal is to read sensor values quicker than the robot, so at a significantly higher rate than every 8ms, using a UDP connection.

Theory

In Python a UDP socket connection is classified as a datagram type socket and coded: socket.SOCK_DGRAM. No handshaking is required so it a connection-less protocol, meaning it can only send data and not receive data, which is a big difference with the TCP protocol and thus limits its applicability. In this case, where a sensor is read out this forms no limitation.
UDP has the advantage to be fast, since it's connection-less, it does not require package acknowledgement and can establish a continuous packet stream.

Setup

A UDP connection is set up between the computer and the compute box of the UR HEX sensor. UDP is used because of its higher speed (up to 500Hz) **[31]** in comparison to TCP.

At first this setup gave either 0 as time value or 15ms, because the used 'time.time()' command has a poor resolution of 15ms.
[https://www.webucator.com/article/python-clocks-explained/]
Using time.time_ns(), the time is given in nanoseconds and thus at a higher resolution. Then it still did not work on my computer, but it did on the PC in the laboratory and on other people's computer. After some research I found out the reason is the Windows version. In Windows 11 (my PC) the timer resolution is 15,6ms **[12]**, while in Windows 10 the default timer resolution is 1ms.
This issue was tested using following code:

```
import time

while True:
    start_time=time.time_ns()/1000000
    time.sleep(0.0001)
    end_time=time.time_ns()/1000000
    print(end_time - start_time)
```

Figure 60: Python code computer clockspeed test

The sleep time of 0,1ms is too short to be measured by both computers, so it gives the smallest measurable time. As can be seen in the output:

for my PC

```
16.027099609375
15.2275390625
15.62451171875
15.627197265625
15.60791015625
15.6259765625
15.638427734375
15.859375
15.38232421875
15.6416015625
15.61083984375
15.638427734375
15.9326171875
15.3134765625
15.61865234375
15.640380859375
```

Figure 61: Clock speed Windows 11

for the lab PC

```
1.999755859375
1.9990234375
1.999267578125
1.99951171875
1.999755859375
1.998779296875
1.99951171875
1.99951171875
1.999267578125
1.999267578125
1.0
1.9990234375
1.99951171875
1.999267578125
1.9990234375
```

Figure 62: Clock speed Windows 10

In this setup only the computer is used and has an IP address of 192.168.1.103. The sensor compute box has an IP address of 192.168.1.98 and UDP requests are sent to port 49152, in the data form of unsigned integers (UINT). UINT32 is an unsigned integer with a size of 32bits and the same for UINT16. **[31]**

Three requests are sent to the compute box before being able to read data. The first one is 'request_biasing', the second one 'request_speed' and the third and last one is 'request_send'. Their structure and action are described below.

Variable	UINT16 Header	UINT16 Command	UINT32 Data
request_speed	must be 0x1234	0x0082 set read-out speed	0x0002 time in milliseconds
encoded:	18 52 0 130 0 0 0 2		
request_send	must be 0x1234	0x0002 start sending output	0x0001 amount of samples
encoded:	18 52 0 2 0 0 0 1		
request_biasing	must be 0x1234	0x0042 set software bias	0x0255 set or 0x0000 reset
encoded:	18 52 0 66 0 0 2 85		

Table 10: Force sensor connection - variables compute box connection

These values are sent by converting the original input into a hexadecimal number format and then appending them into one NumPy array. NumPy has the advantage to be quick and use less data than normal Python arrays.

During the infinite while-loop the elapsed time is measured between the request sending and message receiving.

The received data package has a structure of 36 bytes:

```
b'\x00\x00\x00\x01\x00\x00\xc4\xa8\x00\x00\x00\x00\xff\xff{0\x00\x00\xaf\xc8\x00_Z(\xff\xff\xfe\x0c\xff\xffox\xff\xff\
xa44'  encoded received data
```

Figure 63: Force sensor connection - encoded received data package

and needs to be split into nine UINT32 data with the following values: HS_sequence (sequence number of current UDP record) – FT_sequence (Compute Box internal sample counter) – Status (needs to be 0) – Fx – Fy – Fz – Tx – Ty – Tz with Fx, Fy and Fz in 10000N and Tx, Ty and Tz in 100000N/m.
The data splitting and converting is done using Python's struct module:
Recibido=struct.unpack('IIIIIIIII', receivePacket), 'IIIIIIIII' which is nine times 'I', returning nine unsigned integers with a size of 4 and thus splitting the 36 bytes perfectly into the nine required variables.
These variables are stored in a Python variable and can be split into the desired status values, force

values or torque values.

```
(16777216, 2831417344, 0, 813432831, 3366912000, 677011200, 218038271, 2020605951, 883228671)  unpacked received data
[      1   50344       0  -34000   45000 6249000    -500  -37000  -23500]  decoded received data
[ -3.4   4.5 624.9]  Force values
[-0.005 -0.37  -0.235] Torque values
```

Figure 64: Force sensor connection - decoded and transformed data package

The received values are not zero even though there are no forces applied to the sensor. This can easily be solved by setting a software bias with command '0x0042' and data '255', this way the current values are set to be zero and accurate measurements can be made.

In order to stop the UDP package sending, another request can be sent, or the socket connection can be closed using s.close(). In this experiment the data sending only stops when the Python program is stopped.

Results

This is special because this makes it possible to read sensor values quicker (every 2ms) than the control cycle of the robot (8ms).

```
request_biasing sent
request_read-out speed sent
[-0.1  0.   0.4]  Force values
1.994140625  millseconds
[-0.1  0.   0.3]  Force values
1.999755859375  millseconds
[0.  0.  0.1]  Force values
1.999755859375  millseconds
[0.  0.  0.1]  Force values
1.99609375  millseconds
[0. 0. 0.]  Force values
2.00390625  millseconds
[0.  0.  0.1]  Force values
1.994384765625  millseconds
[0. 0. 0.]  Force values
1.989990234375  millseconds
[ 0.  -0.1 -0.1]  Force values
2.00830078125  millseconds
[ 0.  -0.1 -0.1]  Force values
```

Figure 65: Output sensor compute box connection for force measurements

5.5. Experimental setup for advanced control systems

5.5.1. Force filtering and control

After the successful experiment for reading the sensor every 2ms in [**0 Experiment four: Force sensor connection**] it's possible to exploit the possibilities of this.
The read-out rate of the PC is 4x faster than the robot's update frequency of 125Hz, which gives the opportunity to get four measurements within one robot control period and do calculations. For this practice six different options are available listed below:

Theoretical explanation

To apply force control with the industrial robot, sensing is needed. It is known that in all applications uncertainties exist on the process, due to a lack in resolution, suboptimal process conditions or uncalculated parameters and furthermore, the inaccuracy increases due to sensor noise. Sensor noise is random variations of output, unrelated to variations in input and occurs because of the wiring or external noise in the sensor's environment.

To further support the theory of the multi rate filters and polynomial approximation, they are first applied on sensor data without any form of control action or feedback to the robot, only to see what the adjusted output could look like and in the part 'practical' experiments they are tested out with a control action.
All the control and filter formulas are written in Python, the motion and force control actions are written on the Teach Pendant.

The real practical experiments where the filters and controllers are applied are summarized in [**Practical experiments [Table 37: Force sensor connection Python code**

Appendix XIII – Force filtering and control]], the transferred data is subject to different calculations and the output (hopefully) gives a result with less oscillations and thus higher accuracy.

a) PD control

Proportional Derivative control is a combination of feedback and feedforward control because it uses the current error and the derivative of the error. If the error in steady state does not have to be compensated, PD can be used, because it lacks the integral term of the PID controller.

The PD controller looks at the sensor data and compares this to a reference value. The proportional action reaction is directly proportional to the difference between measured and reference value, meaning it increases with increasing error and the other way around, while the derivative reaction is proportional to the difference in force value, if the force error is constant, the derivate action is equal to zero.
The derivative action responds poorly to noise because it varies quickly.

It's the most basic controller, with simple inputs and a single output.

$$u = K_p\left(F_{ref} - F\right) + K_d(\dot{F}_{ref} - \dot{F})$$

Equation 30: PD control

with

$$\dot{F} = \frac{F - F_z^{-1}}{8ms}$$

Equation 31: Force derivative

Because the force change rate can be high in force control applications, the derivative output will be high as well and in this case oscillations and peaks will occur in the control action, significantly decreasing the control performance and potentially damaging the workpiece and/or robot.

An improvement is made from this controller that compensates last-mentioned problem and is called the Proportional Velocity controller.

b) PV control

In Proportional Velocity control, the control system is equivalent to proportional control with velocity feedback. A pure proportional control action does not eliminate the error in steady state and therefore requires to be accompanied by a velocity feedback term. A second function of the velocity term is to damp the system on peaks and oscillations.

$$u = K_p\left(F_{ref} - F\right) - K_v\dot{x}$$

Equation 32: PV control

Out of Equation 32: PV control can be derived that when the velocity is increased the control action is reduced, which means that a speed increase tolerates a bigger force error because the proportional control action is counteracted by the velocity control action. The reason behind this is that it is harder to control the force in a fast-changing process, what results into oscillations and drastic control actions when using a proportional or proportional derivative control system.

c) Multi rate PD with averaging

This filter is multi rate because instead of measuring every 8ms, the measurements are done approximately every 2ms and out of these measurements, the average is calculated every 8ms, as the estimated force value. This way the original data is decimated and is less prone to oscillations and noise. Multi rate filtering is proven to be useful in control applications **[15 - 27 - 51]**.

In this experiment the elapsed time period is measured, to verify the possibility of measuring four times and if not three values are used for the force estimation.

This technique filters out a big part of the sensor noise, because these occur in the form of oscillations, but process noise in the form of surface irregularities along the motion are not accurately filtered out.

$$\hat{F} = \frac{1}{4}(F_1 + F_2 + F_3 + F_4)$$

Equation 33: Multi rate PD average

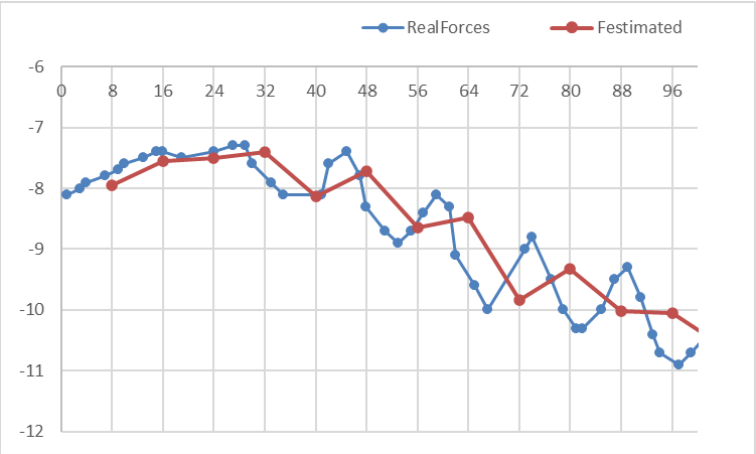


Figure 66: Multi rate PD with averaging filter – theoretical working in graph

d) Multi rate PD 2nd order filter

An averaging method gives every measured value the same weight, meaning most recent values are as significant as the least ones. For control purposes it makes more sense to give more importance to the latest measurement and less to the oldest, because approximately 6ms passed in between those two measurements.
The proposed solution is a second order filter, that gives a fixed coefficient to every force value.

$$\hat{F} = 0.1F_1 + 0.2F_2 + 0.3F_3 + 0.4F_4$$

Equation 34: Multi rate PD 2nd order filter

For example.
Sum of the coefficients has to be one because it's a variant of the averaging method and the result would otherwise be that the estimated value is too big (sum bigger than 1) or too small (sum smaller than 1).
It can be seen in Figure 67 that this method follows the real data very well, while ignoring local peaks and fluctuating at a lower frequency. Because of the chosen coefficients it gives more importance to the most recent measurement.

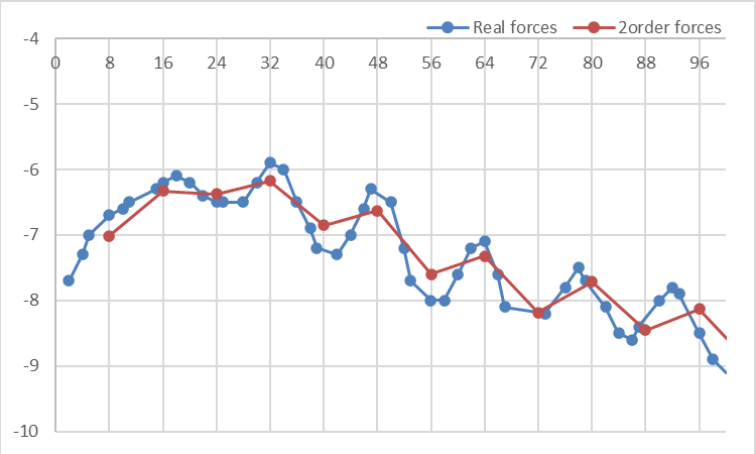


Figure 67: Multi rate PD 2nd order filter – theoretical working in graph

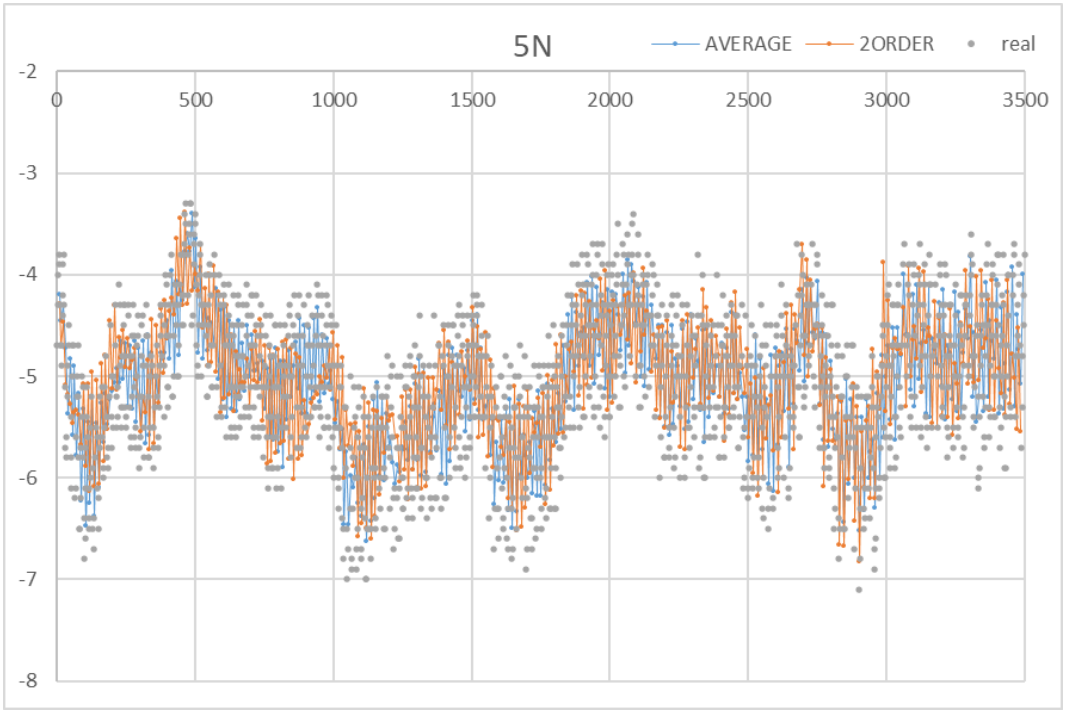


Figure 68: Multi rate PD averaging vs 2nd order filter – theoretical in graph

Both averaging and second order filter ignore the force peaks, with the averaging method giving the best result on this aspect of the force control. This is because the importance here is equal for every measurement, while the 2nd order will still try to follow the last given value more. If this last value is a peak, the resulting force value will be one as well.

e) Polynomial approximation

Polynomial interpolation is a way of representing a data set as a polynomial function. A curve is fitted in the given data set, and this results in a continuous function with smooth transitions between values. Python can calculate the coefficients itself using numpy.polyfit().

$$F(t) = a_0 t^3 + a_1 t^2 + a_2 t + a_3$$

Equation 35: Polynomial approximation

The blue curve gives the real and red the fitted data.

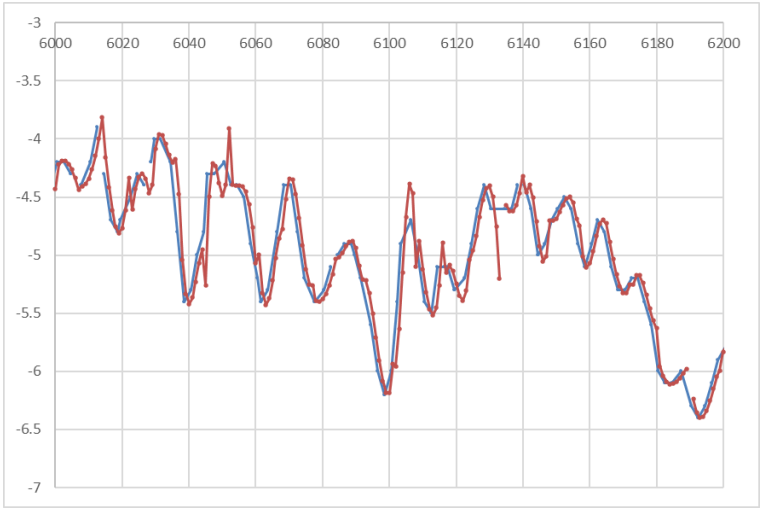


Figure 69: Polynomial approximation - theoretical in graph

f) Kalman filter

The Kalman filter is used in an experiment to measure the sensor noise, in a static setup and later in a force control application. In both of these the setpoint is static, allowing mathematical simplifications to the model because the desired derivative term becomes zero.

As described in [**3.6 Kalman filter**], this filtering method exists of a prediction and a correction phase. Starting with the matrices from, the steps taken to find the final formulas are mentioned in this paragraph.

$$H_k = [1 \quad 0]$$
$$F = [1 \quad 0]\begin{bmatrix} F \\ \dot{F} \end{bmatrix}$$
$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$A_k = \begin{bmatrix} 1 & T_m \\ 0 & 0 \end{bmatrix}$ with Tm = 0.002, representing the sample period of 2 milliseconds.

Prediction phase:

$$\hat{x}_{k+1}^- = A_k \hat{x}_k = \begin{bmatrix} 1 & T_m \\ 0 & 1 \end{bmatrix} * \begin{bmatrix} F \\ \dot{F} \end{bmatrix} = \begin{bmatrix} 1 & 0.002 \\ 0 & 1 \end{bmatrix} * \begin{bmatrix} \hat{x}_1 \\ \hat{x}_2 \end{bmatrix} = \begin{bmatrix} \hat{x}_1 + 0.002\hat{x}_2 \\ \hat{x}_2 \end{bmatrix}$$

Equation 36: Kalman filter - prediction of robot state

$$\begin{aligned} P_{k+1}^- &= A_k P_k A_k^T + Q_k \\ &= \begin{bmatrix} 1 & T_m \\ 0 & 1 \end{bmatrix} * \begin{bmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{bmatrix} * \begin{bmatrix} 1 & 0 \\ T_m & 1 \end{bmatrix} + Q_k \\ &= \begin{bmatrix} p_{11} + p_{21} * T_m & p_{12} + p_{22} * T_m \\ p_{21} & p_{22} \end{bmatrix} * \begin{bmatrix} 1 & 0 \\ T_m & 1 \end{bmatrix} + Q_k \\ &= \begin{bmatrix} p_{11} + p_{21} * T_m + T_m * (p_{12} + p_{22} * T_m) & p_{12} + p_{22} * T_m \\ p_{21} + p_{22} * T_m & p_{22} \end{bmatrix} + Q_k \\ &= \begin{bmatrix} p_{11} + p_{21} * T_m + T_m * (p_{12} + p_{22} * T_m) & p_{12} + p_{22} * T_m \\ p_{21} + p_{22} * T_m & p_{22} + Q_k \end{bmatrix} \end{aligned}$$

Equation 37: Kalman filter - prediction of covariance matrix

Correction phase:

$$\begin{aligned} K_k &= P_k^- H_k^T \left(H_k P_k^- H_k^T + R_k\right)^{-1} \\ &= \begin{bmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{bmatrix} * \begin{bmatrix} 1 \\ 0 \end{bmatrix} * \left([1 \quad 0] * \begin{bmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{bmatrix} * \begin{bmatrix} 1 \\ 0 \end{bmatrix} + R_k\right)^{-1} \\ &= \begin{bmatrix} p_{11} \\ p_{21} \end{bmatrix} * ([p_{11}] + R_k)^{-1} \\ &= \begin{bmatrix} p_{11}(p_{11} + R_k) \\ p_{21}(p_{11} + R_k) \end{bmatrix} \end{aligned}$$

Equation 38: Kalman filter - correction Kalman gain

$$\begin{aligned} \hat{x}_k &= \hat{x}_k^- + K(z_k - H_k \hat{x}_k^-) \\ &= \begin{bmatrix} \hat{x}_1 \\ \hat{x}_2 \end{bmatrix} + \begin{bmatrix} k_1 \\ k_2 \end{bmatrix} * \left(F - [1 \quad 0] * \begin{bmatrix} \hat{x}_1 \\ \hat{x}_2 \end{bmatrix}\right) \\ &= \begin{bmatrix} \hat{x}_1 + k_1(F - \hat{x}_1) \\ \hat{x}_2 + k_2(F - \hat{x}_1) \end{bmatrix} \end{aligned}$$

Equation 39: Kalman filter - correction of robot state

$$
\begin{aligned}
P_k &= (I - K_k H_k) P_k^- \\
&= \left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \begin{bmatrix} k_1 \\ k_2 \end{bmatrix} * [1 \quad 0] \right) * \begin{bmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{bmatrix} \\
&= \left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \begin{bmatrix} k_1 & 0 \\ k_2 & 0 \end{bmatrix} \right) * \begin{bmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{bmatrix} \\
&= \begin{bmatrix} p_{11}(1 - k_1) & p_{12}(1 - k_1) \\ p_{11}(-k_2) + p_{21} & p_{12}(-k_2) + p_{22} \end{bmatrix}
\end{aligned}
$$

Equation 40: Kalman filter - correction of covariance matrix

The initial value of $P_k = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

Resulting in the final formulas.

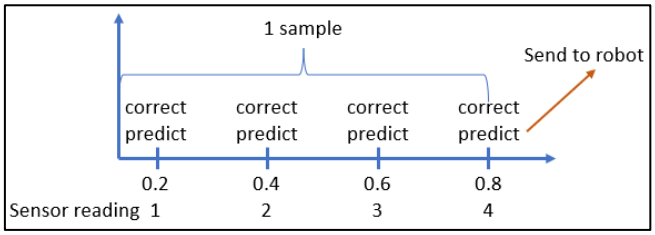


Figure 70: Kalman filter experiment working principle

It would also be possible to create a continuous Kalman filter, instead of a discrete one. Then a continuous covariance matrix of the process is needed. Over a big time period, the covariance should be higher than over a small period. This can be seen in the following formulas received from Professor Antonio Sala.

$$M = \exp\left(\begin{bmatrix} -A & WW^T \\ 0 & A^T \end{bmatrix} T_k \right) = \begin{bmatrix} H_1(T_k) & H_2(T_k) \\ 0 & H_3(T_k) \end{bmatrix}$$

$$Wd = H_3^T(T_k) * H_2(T_k)$$

Practical experiments [Table 37: Force sensor connection Python code

Appendix XIII – Force filtering and control]

Each experiment is conducted multiple times to eliminate the factor of unexpected events and they are conducted using 5N, given the maximum workload of 30N. All the movements are executed with inner/outer loop control and the tool follows a trapezoidal trajectory.
The inner loop control is provided by the trapezoidal velocity profile generator, the motion is executed with a limited acceleration and velocity, as an attempt to give more stable output. If the maximum velocity is too high for the acceleration or the distance too short, a triangular velocity profile is generated instead.
The outer loop control is provided by the different filters and controllers mentioned in [Table 11: Force filtering and control - outer loop control equations].

Experiment one to five is dynamic, during the force control a movement is executed in a fixed direction. The tool is mounted on the robot and the gravity (tool weight) is compensated with a calculated offset. From a given position the robot descends, until it touches the surface and from then force mode is activated where the robot attempts to maintain a constant force value normal to the surface, whilst executing a unidirectional movement with constant speed.
Experiment six, the one of the Kalman filter, is static and does not require force control.

The stable part of the experiment is taken from 1second to 7seconds because this is also the stable part of the motion control.

$$\dot{F} = \frac{F - F_z^{-1}}{8ms}$$

Equation 41: Force filtering and control - force derivative formula

	Python	Teach pendant
P	/	$u = K_p(F_{ref} - F)$
PD	/	$u = K_p(F_{ref} - F) + K_d(\dot{F}_{ref} - \dot{F})$
PV	/	$u = K_p(F_{ref} - F) - K_v\dot{x}$
PD with averaging	$\hat{F} = \frac{1}{4}(F_1 + F_2 + F_3 + F_4)$	$u = K_p(F_{ref} - \hat{F}) + K_d(\dot{F}_{ref} - \dot{\hat{F}})$
PD with filtering	$\hat{F} = 0.1F_1 + 0.2F_2 + 0.3F_3 + 0.4F_4$	$u = K_p(F_{ref} - \hat{F}) + K_d(\dot{F}_{ref} - \dot{\hat{F}})$
Polynomial approximation	$\hat{F}(t) = a_0t^3 + a_1t^2 + a_2t + a_3$	$u = K_p(F_{ref} - F) + K_d(\dot{F}_{ref} - \dot{F})$
Kalman filter	Predictor & corrector	$u = K_p(F_{ref} - F) + K_d(\dot{F}_{ref} - \dot{F})$

Table 11: Force filtering and control - outer loop control equations

a) Proportional control

K_p	Mean	Standard deviation	Max	Min	#peaks <4N	#peaks >6N
0.001	5.01603	0.573354	6.50391	3.20391	29	38
0.0005	5.03466	0.516616	6.75411	3.25411	13	29
0.002	7.18277	9.126594	48.0672	0.167187	271	237

Table 12: Force filtering - proportional control summary

Before using PD, a proportional controller is implemented to find a Kp value that fits the application. The force error is estimated to be in the order of 1N, so the Kp should be in the order of 0.001 to transform it into a speed value. A higher value generates high force peaks, followed by surface contact loss, while a smaller value a lower frequency in peaks has but, because of its lower response,

the magnitude of it remains the same.
To give a reference to the amount of peaks, a total number of 750 data points are sampled.

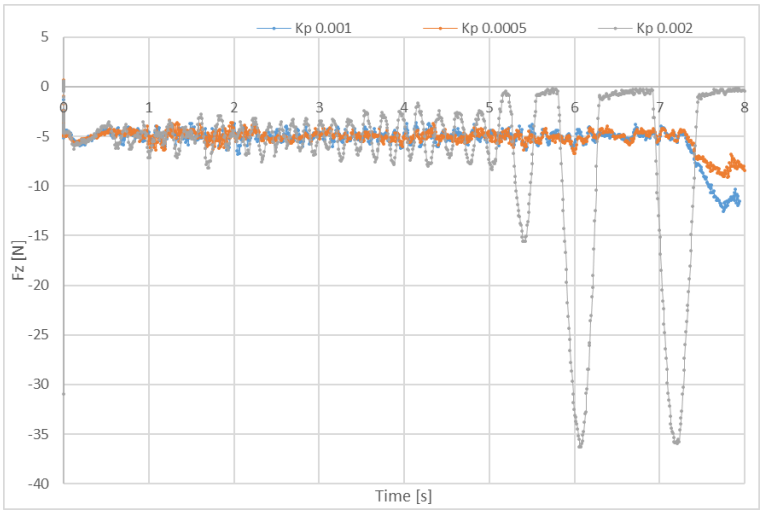


Figure 71: Force filtering - proportional control graph

As expected, a high proportional gain gives more and higher peaks but it reaches the desired value more closely (mean value closer to 5N).

b) PD control

K_p	K_d	Mean	Standard deviation	Max	Min	#peaks <4N	#peaks >6N
0.001	0.00003	5.84112	7.406285	23.4514	0.048634	465	260
0.001	0.00001	5.006	0.743706	7.5791	2.8791	72	60
0.001	0.000005	4.99939	0.650689	7.08008	2.98008	53	44

Table 13: Force filtering - proportional derivative control summary

In regular control systems, the derivative term provides a big improvement in terms of damping the system, but as expected, when used on noisy data this results in a stability loss.
A K_p value of 0.001 is chosen because this still gave oscillations and should have enough room for improvement.

A big derivative gain creates instability in the process, but a small value (0.000005) leads to an improvement in reaching the mean value. An increase in standard deviation occurs (0.65 instead of 0.573 in proportional control), leading to a significant number of extra peaks out of the +/- 1N range.

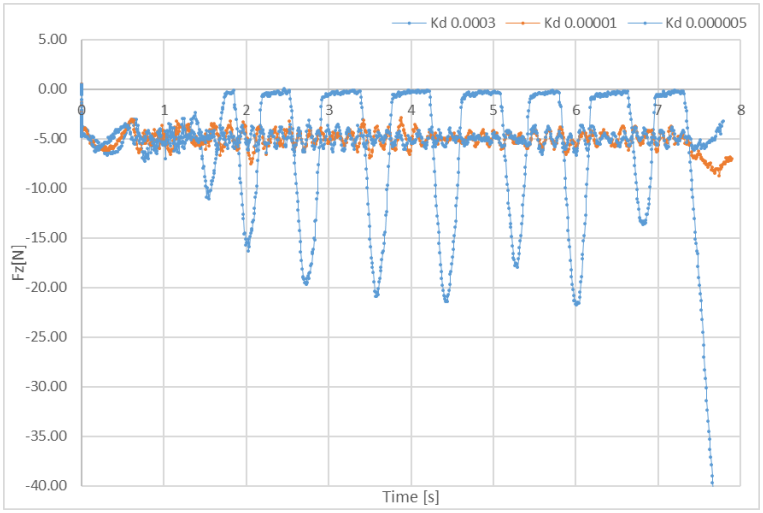


Figure 72: Force filtering - proportional derivative graph

c) PV control

K_p	K_v	Mean	Standard deviation	Max	Min	#peaks <4N	#peaks >6N
0.001	0.03	4.98685	1.078344	7.86035	2.06035	153	143
0.001	0.06	4.98049	1.005806	8.12695	2.32695	127	128
0.001	0.015	4.97951	1.026188	8.11133	1.81133	114	126

Table 14: proportional velocity control summary

Since the derivative control action of a noisy system augments the oscillations, an alternative way to damp the system is by using the velocity used to control the force.

Even though the derivative action is chosen to be small, it has a big negative impact on the system's performance in force control.

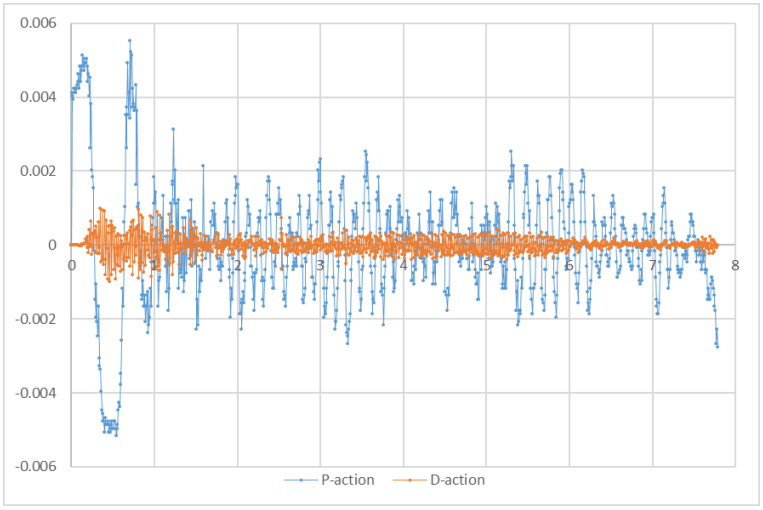


Figure 73: Force filtering - PV control actions graph

d) PD with averaging

K_p	K_d	Mean	Standard deviation	Max	Min	#peaks <4N	#peaks >6N
0.001	0.00001	5.05705	0.746747	7.43652	2.83652	61	77
0.001	0.000005	5.1734	0.745065	8.08008	3.18008	48	88
0.001	0.00003	6.8276	9.379864	28.8176	0.017578	484	247

Table 15: Force filtering - PD control with averaging summary

This technique filters out a big part of the sensor noise, because these occur in the form of repetitive oscillations, but process noise in the form of surface irregularities along the motion for example are not accurately filtered out.

In theory it is a safe option to choose the average measured value as the true value, but sensor and process noise does not behave regularly, so this assumption approaches the real value but still needs verification. The accuracy is hard to evaluate but it is certain that the output is more stable because of the averaging.

The estimated force follows the real one, but high-frequency fluctuations in the measured force are ignored thus it maintains the desired value in a more constant manner. The filtered-out force is still subject to peaks because of the dynamics of the executed movement, friction, surface irregularities and internal feedback delay.

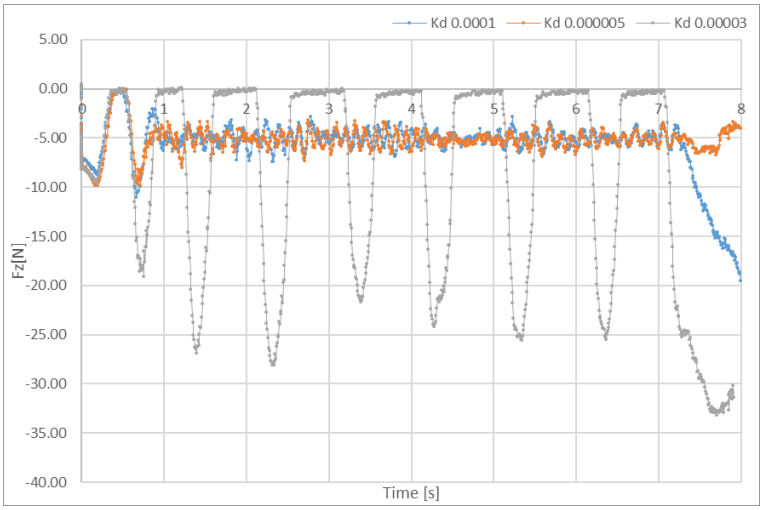


Figure 74: Force filtering - PD with averaging graph

e) PD with 2order filter

K_p	K_d	Mean	Standard deviation	Max	Min	#peaks <4N	#peaks >6N
0.001	0.00001	5.03271	1.395388	8.56934	1.66934	176	179
0.001	0.000005	5.02308	0.928597	8.00801	2.50801	96	127
0.001	0.00003	7.0823	8.836956	25.8896	0.010352	457	277

Table 16: Force filtering - PD control with 2nd order filter summary

The second order filter works worse than the classic PD controller, because out of a small sample size, it gives great importance to one value. Because the sample size is small, this one value will vary a lot throughout the process, resulting in a poor control performance.

A small derivative gain gives a better control because the proportional and the derivative control action are added together to form the total control action.

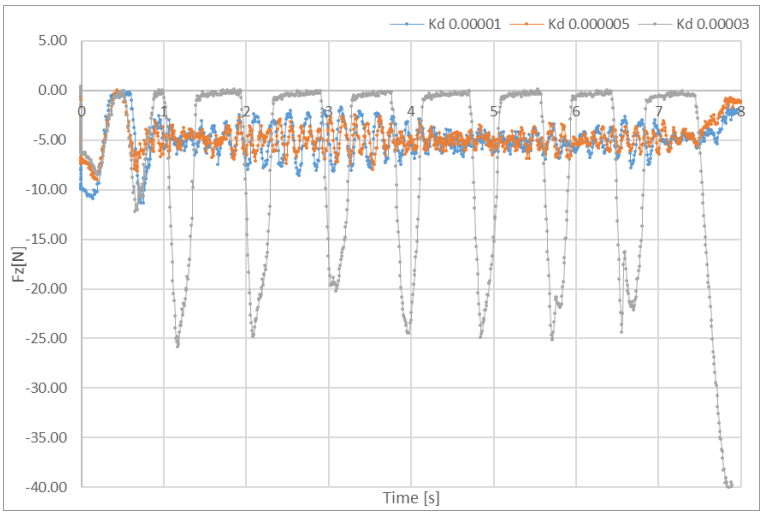


Figure 75: Force filtering - PD with 2nd order filter graph

f) Polynomial approximation

K_p	K_d	Mean	Standard deviation	Max	Min	#peaks <4N	#peaks >6N	Sample size
0.001	0.00001	5.02808	1.161284	9.80949	2.012444	1161	1300	5979
0.001	0.000005	4.82006	1.301085	9.16959	1.197505	1513	1009	5969
0.001	0.00003	4.57307	1.836676	10.54228	0.006849	2412	1246	5942

Table 17: Force filtering - polynomial approximation summary

The polynomial approximation fits a curve out of a data sample. This data sample in this case is four force values. The standard deviations are higher than in the experiment with the ordinary PD control. As can be seen in the graph below, the polynomial fits a curve within the sample and uses this for extrapolations if necessary. These extrapolations cause the extra deviations from the desired value.

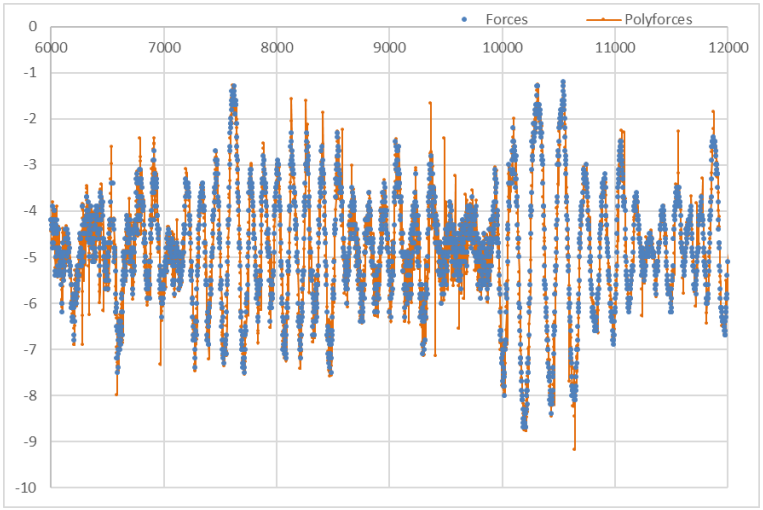
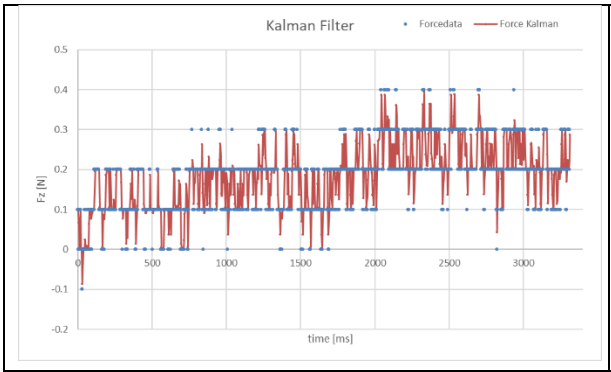


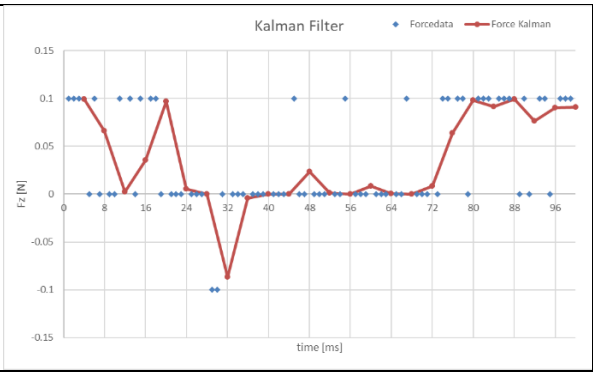
Figure 76: Force filtering - polynomial approximation graph

g) Kalman filter

As been stated in [**3.6Kalman filter**], the Kalman filter method exists of two phases. Both are implemented in a Python program, where the computer sets up a connection with the compute box. For every sample, measurements are read out and for each value the Kalman filter corrects and predicts the robot state x, which is a 1x2 matrix: $\begin{bmatrix} F \\ \dot{F} \end{bmatrix}$.

In order to use the Kalman filter, knowledge about the sensor and process noise covariance is needed. This first one is easily calculated in MATLAB and the second one is approximated trough trial-and-error. The output of this experiment is the updated/predicted robot state matrix x (force and force derivative in a 2x1 matrix) and the state covariance matrix P (where the diagonal elements are the variances and the off-diagonal elements the correlations represent in a 2x2 matrix).





Experiment	Sensor		Kalman output					
	Mean	Variance	x	$\dot{x}$	p11	p12	p21	p22
1	0.0494	0.0061	0.2323	-0.0129	0.29186	0	-0.1994	0.5
2	0.2133	0.0090	0.023938	-0.00957	0.290899	0	-0.19988	0.5
3	-0.2538	0.0073	-0.1679	-0.01281	0.291462	0	-0.19959	0.5
4	0.0465	0.0090	-0.09502	-0.00199	0.290899	0	-0.19988	0.5
5	0.0443	0.0095	0.007971	-0.00319	0.290734	0	-0.19996	0.5

Table 18: Kalman filter - sensor noise covariance summary

The robot state (x and $\dot{x}$) differ because these are based on the on-line measurements from the sensor. It is visible that the force and force derivate are both close to zero, as desired. This is significant because the data resolution is only one decimal, allowing the data to jump from 0 to 0.1 and because the force derivative is calculated using a time period of the order milliseconds.
The covariance matrices are almost identical and contain small values in all the experiments.

To find out the correct process noise covariance, the same experiment is conducted with a constant sensor noise covariance R value and for the process noise Q the following values are tested:
0.005 0.05 0.5 1
The results of each of the tests are summarized in this table

Experiment		Sensor		Kalman output					
#	Q-value	Mean	Variance	x	$\dot{x}$	p11	p12	p21	p22
1	0.005	0.0443	0.0095	-0.12306	0.000109	0.162889	0	-0.0000044	0.005
2	0.05	0.0443	0.0095	-0.04113	-0.0011	0.315261	0	-0.00094	0.05
3	0.5	0.0443	0.0095	0.00497	-0.00199	0.290734	0	-0.19996	0.5
4	1	0.0443	0.0095	-0.10433	0.005455	-0.71386	0	-1.25839	1

Figure 77: Kalman filter - process noise covariance summary

When either the process (low Q-value) or the sensor data (high Q-value) is given a high accuracy, so Q=0.005 and Q=1, the result of the robot state deviate a lot from the expected 0 Newton. Q=0.5 is a

safe option, where the sensor measurements and the process data are given an equal amount of importance.

For practical reasons it is possible to eliminate variance +1 as an option because in these cases the data is inaccurate, this means that (given the gaussian distribution of the Kalman Filter) the probability of a value being between -0.5 and +0.5 is less than 40%.

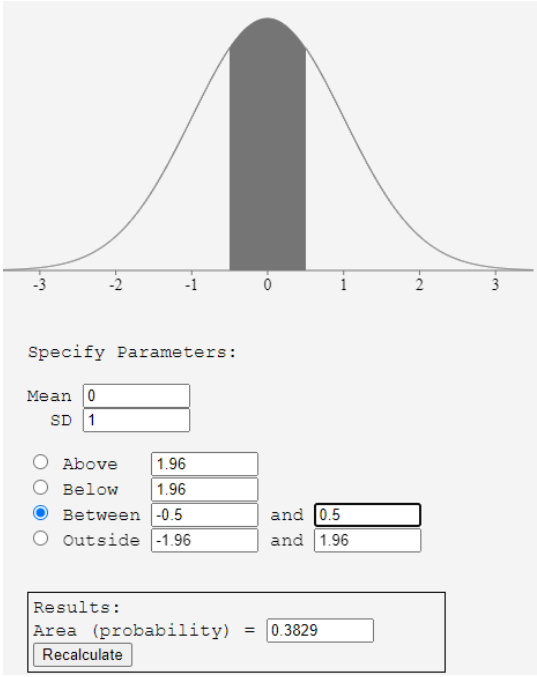


Figure 78: Kalman filter - gaussian distribution example deviation of 1

The real experiment with on-line force control with Kalman filter was not able to be conducted because the robot sensor broke down during this experiment.

Results

A general technique is proposed than can be implemented in different processes and on different variables, that leads to a performance improvement.

a) Proportional control

A small control action works on the robot, this gives the most stable output in these force control experiments.

b) PD control

The proportional control gives a small control action and thus a low standard deviation, if a derivative action is introduced, the control action becomes bigger and gives a higher standard deviation. The predictive ability of the derivative component reduces the stability of the process because of the high frequent oscillations due to noise.

c) PV control

The PV control gives almost constant output, where the average is reached with an approvable accuracy, but the deviation of 1N can be high if the desired value is as small as 5N. The PD control was more accurate than this one.

d) PD with average

For Kp 0.001 and Kd 0.00001 this controller has the same accuracy as the normal PD controller, for all the other values the performance is worse. A reason for this could be that the calculations are done on the computer and then sent to the robot, meaning there is a communication delay, whereas in the ordinary PD control all the calculations were done on the Teach Pendant.

This means that the advantage of reading force values 4x quicker than the robot does not have an effect when using a PD control with averaging, because the gained efficiency is at the expense of another time delay.

e) PD with 2order filter

The standard deviation here is even higher than with averaging, this because the last value of the four data samples gets a significant importance and thus is followed better. Because the data is noisy, this PD filter follows the noise and as a result oscillates more.

f) Polynomial approximation

The expected result seemed promising but because of the small sample size of 4 measurements, the fitted curve follows the real values with high accuracy. In addition to this, a lot of computation time is required for this method, where the Python program is occupied and does not store new received force values. This leads to big overshoots in this dead time, because the estimation after receiving four high-frequent values gives an even worse approximation.

If this method was used with a bigger sample size, as in 16 or more values, I am convinced, and theory **[36]** supports that noise would be eliminated.

g) Kalman filter

During this last set of experiments the sensor of the robot broke down. Even though this experiment did not give positive results, the concept and ability to predict values became clear. Out of every measurement a prediction is made and corrected, as well as the uncertainty of the system. This is a powerful tool because after a few oscillations, the Kalman filter will recognize this and filter it out.

If the parameters are tuned well, this method looks promising to deliver good results because of its estimation ability and iterations with corrections on this value create a higher belief on the true value.

5.5.2. Impedance control [Appendix XIV – Impedance control]

Goal

In this experiment a PD controller is used to control the position with a trapezoidal velocity profile. Using impedance control the robot executes a linear movement and then a circular one.

The impedance control acts in all possible directions, except the vertical. This last one is preserved for force control actions. The linear movement is unidirectional so has a reference value to compare the real one to, while the other directions are measured in the initial position and kept constant throughout the movement with the impedance control.

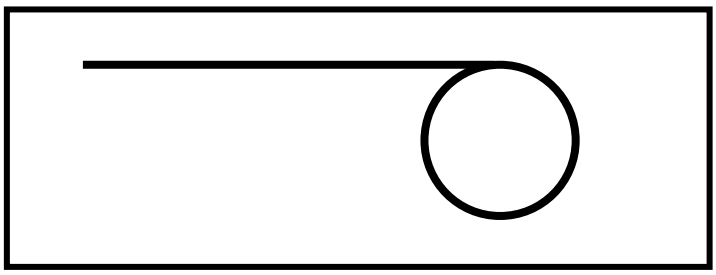


Figure 79: Impedance control - reference trajectory

Theory mentioned in [**3.3Impedance control**]

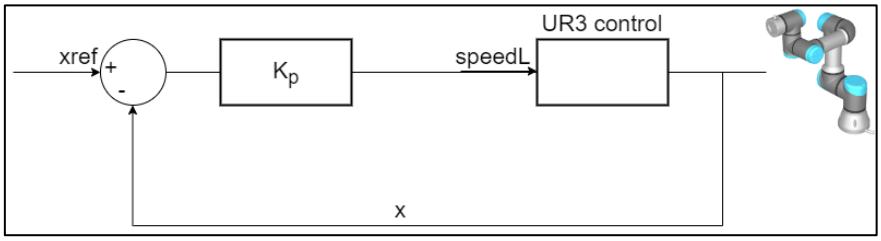


Figure 80: Impedance control - block diagram

The speedL command exists of $\begin{bmatrix} v_x \\ v_y \\ v_z \\ v_\alpha \\ v_\beta \\ v_\gamma \end{bmatrix}$ with $v_x = K_{px}(x_{ref} - x) + K_{dx}(\dot{x}_{ref} - \dot{x})$ and similar for the other velocities. The proportional gain of the control action is equivalent to the active stiffness and the derivative gain to the damping.

x_{ref} is calculated and updated every 8ms from the trapezoidal speed trajectory generator. The x in Figure 80 contains all six position values: x, y, z, and the three rotational values.

Setup

A TCP IP connection is set up between the robot and the computer, allowing communication from the Teach Pendant to the Python program. The motion and impedance control are programmed on the Teach Pendant and the Python program is used to store received data.

Results

For the line a proportional Kp and a derivative gain Kd is used.

		Average error [mm]		Standard deviation [mm]	
Kp	Kd	y	x	y	x
0.5	0	-0.17162	-0.0427	0.439227	0.090531
0.5	0.001	-0.19858	-0.10675	0.995345	0.130152
0.5	0.002	-0.26733	-0.05303	2.027663	0.093442
0.2	0	-0.17065	-0.0867	0.289876	0.117252
0.2	0.001	-0.25314	-0.07659	0.448141	0.112696
0.2	0.002	-0.35966	-0.04791	0.686037	0.133144
0.1	0	-0.17512	-0.12824	0.237925	0.097512
0.1	0.001	-0.32063	-0.09709	0.542904	0.092139
0.1	0.002	-0.52561	-0.08549	1.060119	0.124055
0.01	0	-0.16565	-0.12214	0.256214	0.097437
0.01	0.001	-1.60709	-0.11624	2.822332	0.109392
0.01	0.002	-2.78258	-0.06718	6.373058	0.140276
0.0001	0	-0.07013	-0.14021	0.18127	0.113878

Table 19: Impedance control - linear motion summary

When the derivative gain is set to zero, a small Kp value leads to a worse average value and deviation in the x-direction. This is because the stiffness is set to a small value. This results in a better control of the y-direction, which is the movement direction because the control action is not too big. For a big Kp value the control in a position error is big, leading to oscillations on the system.

Introducing a derivative action does not improve the position control of the robot. In all cases the standard deviation rises and the average value as well.

This all is graphically displayed in the figure below [Figure 81]. Since the stiffness is the inverse of the elasticity, it can be seen that for a low stiffness the elasticity is high. This results in a constant position error when Kp is equal to 0.01, until the derivative action is 0.02, then the average value in the x-direction is maintained better. For the high stiffness, the impact of the derivative action is low and the process is visible more unstable than when the stiffness is low. The oscillations occur at the same frequency as when the stiffness is low, but the magnitude of it is higher.

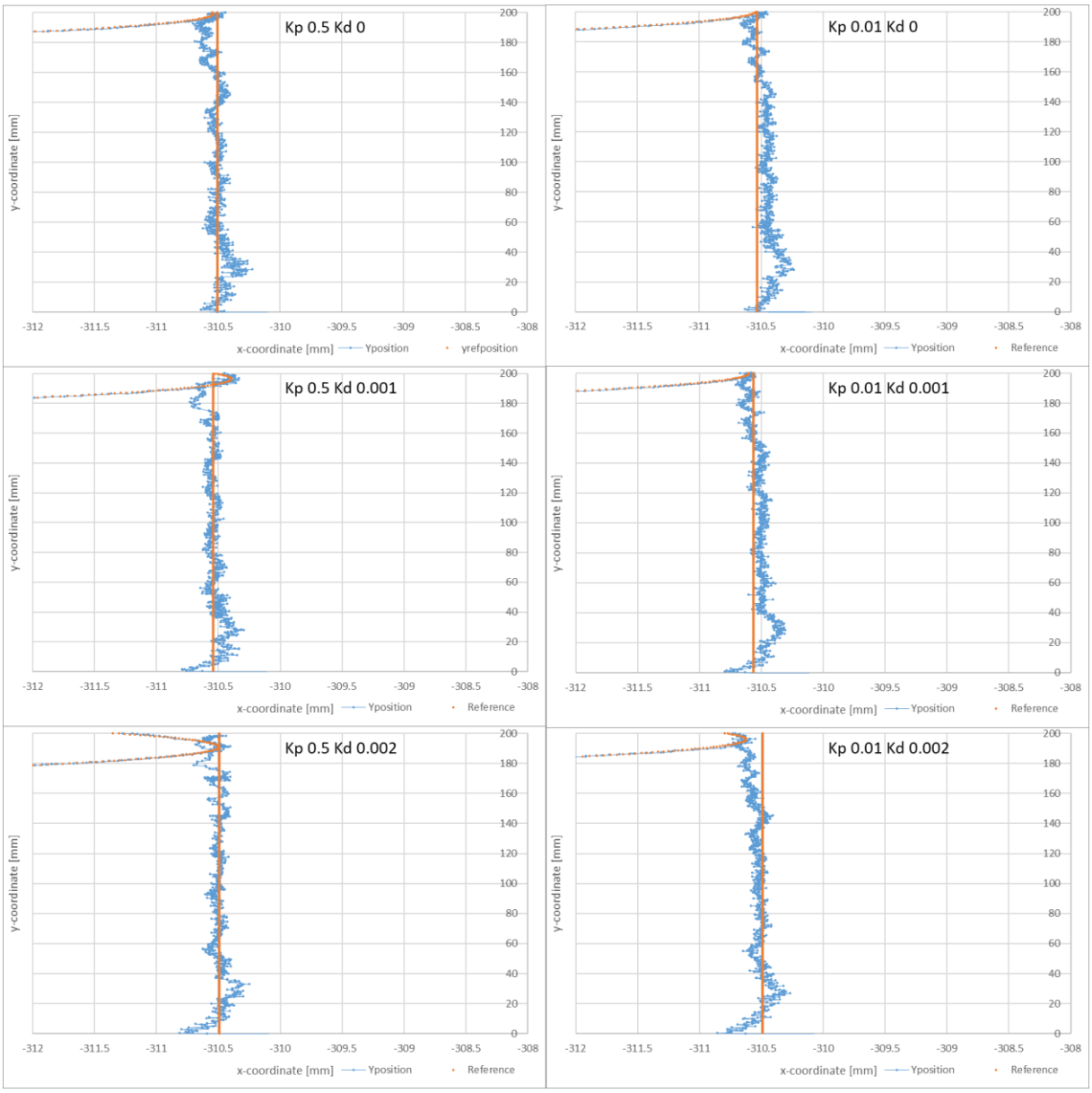


Figure 81: Impedance control - difference in active stiffness

To evaluate the y-direction accuracy, it is necessary to plot the y-direction to the time, because on the (x,y)-graph it is not visible if the y-coordinate is ahead or behind the reference value at the required timestamp. As displayed in [Figure 82], introducing a derivative action causes a control action that is too big, resulting in big oscillations. The robot lags behind, then compensates this but this causes the end-effector to be ahead of the reference value. This way the oscillations keep occurring. A low active stiffness is desirable in the movement direction, when there is no derivative action, the trajectory is followed with high accuracy. This means that the passive stiffness is sufficient for motion control applications.

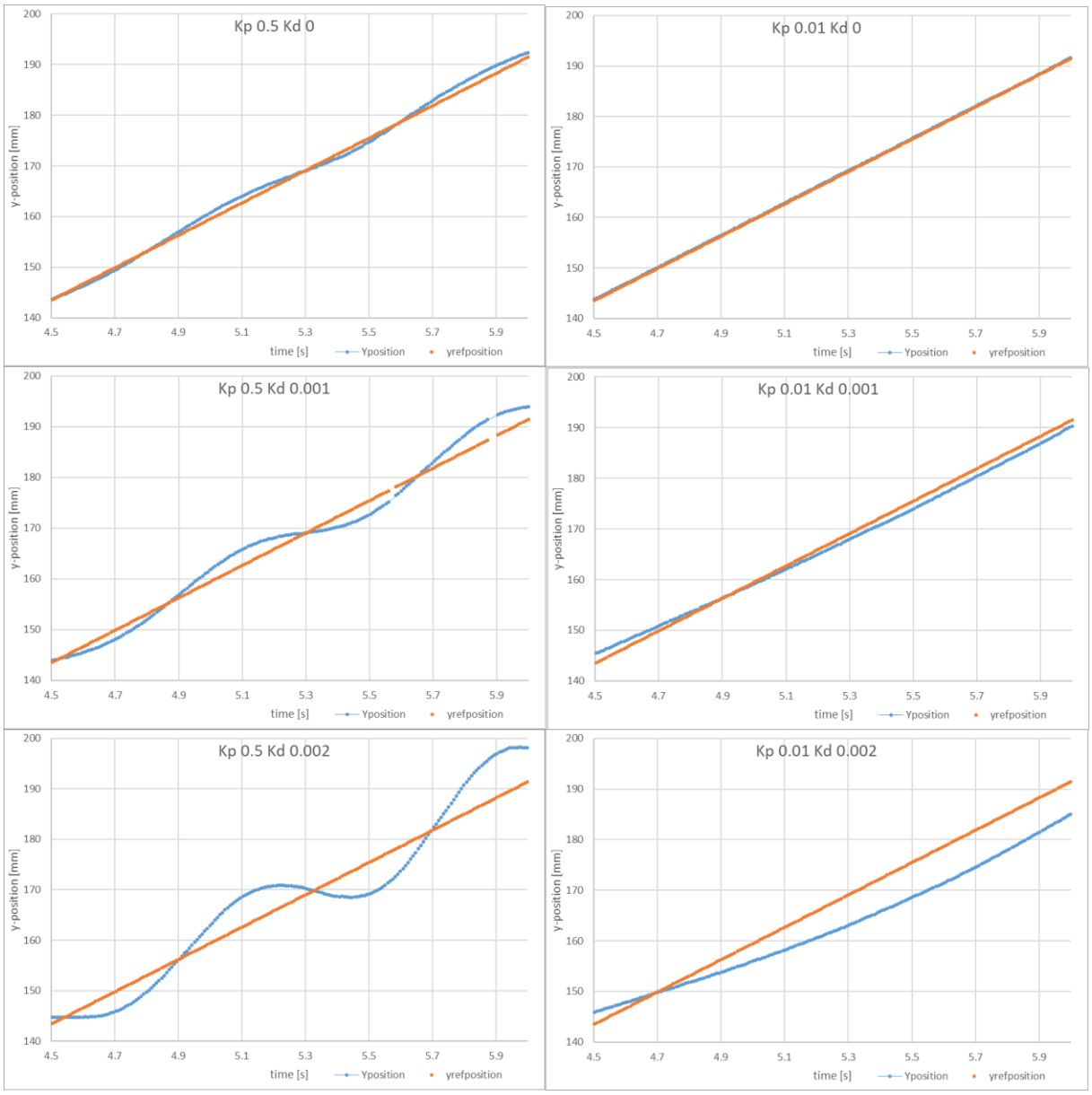


Figure 82: Impedance control - y coordinate plotted over time

For the circle a gain KJ is used

	Average error [mm]		Standard deviation [mm]	
KJ	y	x	y	x
100	-0.0237	-0.0198	0.88625	0.72782
500	-0.0202	-0.00999	0.724407	0.697575
1000	-0.01236	-0.01038	0.714813	0.696257
2000	-0.0882	-0.01458	1.269412	0.69123

Table 20: Impedance control - circular motion summary

Up to a certain point a higher stiffness leads to an improvement in trajectory tracking, when the KJ is too high, the deviation rises again because the control action is too strong. For small values the stiffness is too low, causing the robot to lag behind relative to the reference position.

The big radius and relatively low speed make it easier for the robot to follow the circular trajectory, the influence of the stiffness or impedance is more visible when either the radius is lower or the speed is higher. For a big radius the simplification of dividing the circular trajectory into multiple

linear segments is accurate, because every 8ms the angle towards the goal only changes slightly. The smaller the circle and higher the velocity, the more the update period of 8ms impacts the performance.

The figure below shows the motion of half of the circle, it can be seen that the low stiffness oscillates at the start but later maintains the correct value. For the highest KJ, there is a small but constant position error, which is visible in the ascending part from second 9.2 to 11.2 and this shape is repeated twice. Overall, this method worked well, but as mentioned before this is due to the low velocity and big radius.

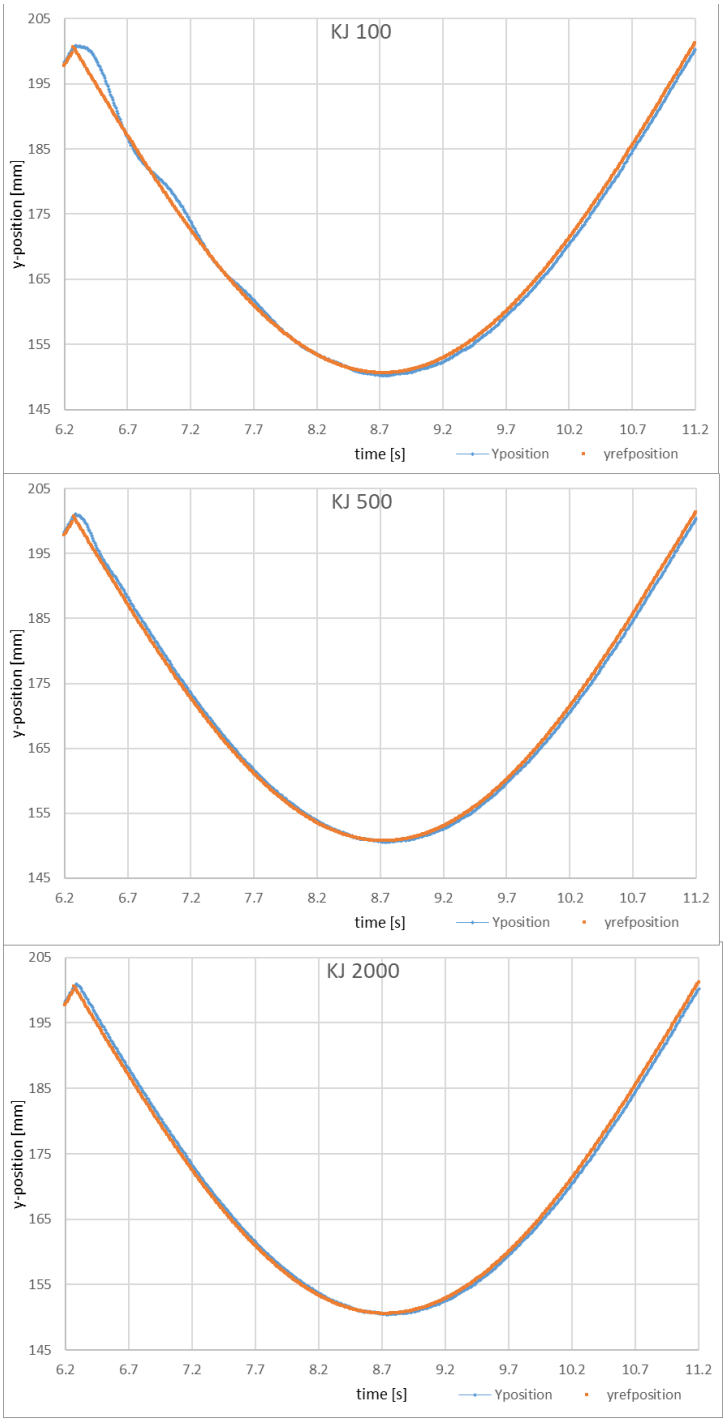


Figure 83: Impedance control - KJ of circular part

6. Results and conclusions

Difference UR3 and UR3e

It has been proven in [**Experiment three: Communication time delay [Appendix XI – Communication time delay]**], as given by the manufacturers that the communication time of the UR3 is 8ms and the one of the UR3e is 2ms. This gives the opportunity for the UR3e to react 4x as fast to errors in the control and thus generate a higher accuracy.
The dimensions are almost identical and as was seen in [**Experiment two: Multi-robot communication[Appendix X – Multi-robot communication]**], the commands written on the Teach Pendant allow for synchronization in terms of joint positions/velocities and cartesian positions/velocities. Meaning that any difference in dimensions can be avoided and eliminated by configurating the robot's coordinate system and code in the right way.
In terms of force control [**Experiment one: Sampling period in force control [Appendix IX – Sampling period force control]**], the force deviation is the most significant indicator to evaluate the noisiness and ability to maintain a constant value. It is visible that the repeatability of the UR3e leads to a better maintenance of a vertical force and indirectly position. When the UR3e operates at its highest frequency of 500Hz, this improves the force control even more.

	UR3	**UR3e**
Force	**Slow (8ms)**	**Slow (8ms)**
Deviation [N]	0.633	0.301
Touch		
Deviation [N]	0.513	0.217

Table 21: Conclusion - difference UR3 and UR3e force control

Filters

Filter	K_p	K_d/K_v	Mean	Standard deviation	% of data within +/- 1N
P	0.001	0	5.01603	0.573354	91.07
PD	0.001	0.000005	4.99939	0.650689	87.07
PV	0.001	0.06	4.98049	1.005806	66.00
PD average	0.001	0.00001	5.05705	0.746747	81.60
PD 2order	0.001	0.000005	5.02308	0.928597	70.27
Polynomial	0.001	0.00001	5.02808	1.161284	58.84
Kalman	/	/	/	/	/

Table 22: Conclusion - force filtering and control of UR3

The derivative action is usually responsible for damping the system, because of its predictive behaviour. When trying to reach a setpoint, the derivative gives an indication of how far this setpoint is from the current value and the reaction speed. If the setpoint is far away, the derivative is high and control action as well. Oscillations cause the derivative action to think the control action should be higher than it actually is.

PD with averaging and second order filter have a worse performance than the ordinary PD control, this is because there is a communication time delay of 8ms, which means the robot reacts to old data. During 8ms information is gathered and 8ms later the robot receives the new value, so there is a delay of 16ms on the system. During this time the force is able to deviate a lot from its previous value.

The polynomial approximation works, but it does not generate an output with less noise. Perhaps if the sample size was bigger, this would have been the case because it is rather easy to plot a polynomial out of four values, but if this needs to be done out of sixteen or more, the fitted curve will not follow the peak values very well. If this small change is made, this method could lead to better results.

The self-programmed force control method comes with the ability to be tuned, during the conducted force control experiments a somewhat random proportional gain is used, so there was room for improvement for further (PD, PV, …) controllers.
Out of the table below it can be seen that if the proportional controller is tuned correctly, it will work better than the built-in F/T Control method because it already works at the same level of accuracy. This means that the other control methods also might give better results when tuned until the best possible result is achieved.

Numbers of [**5.4.1**] – Force sample period			Numbers of [**5.5.1**]– Proportional control		
	Average [N]	**Deviation [N]**	**Kp**	**Average [N]**	**Deviation [N]**
Force	5.095	0.633	0.0005	5.03466	0.5166
Touch	4.977	0.513			

Table 23: Conclusion - difference between F/T Control and self-programmed force control of UR3

Impedance control

A method of implementing impedance control has been introduced, applying a normal force to a surface with motion commands and changing the active stiffness and damping of the collaborative robot.
Especially in the static x-direction, the stiffness value plays a great role in maintaining this value. For small active stiffness values, the deviations and average value in this direction worsen. The gain for the active stiffness is the proportional action in the control and thus for big values more oscillations occur. The derivative action, which is responsible for the damping, does not improve the control. This is probably because a high damping prevents the robot from responding quickly, which is necessary since the motion updates occur every 8ms. The extra friction due to the normal force makes this even worse.
A low active stiffness is safe and when introducing damping, the error will increase significantly. If the active stiffness is high, the derivative action has less of a negative influence, which is as expected. This shows that it is dangerous for a collaborative robot that interacts with the environment to be programmed with high stiffness values, despite its advantages in terms of control accuracy.

		Average [mm]		Standard deviation [mm]	
Kp	Kd	y	x	y	x
0.5	0	-0.17162	-0.0427	0.439227	0.090531
0.0001	0	-0.07013	-0.14021	0.18127	0.113878
0.5	0	-0.17162	-0.0427	0.439227	0.090531
0.5	0.001	-0.19858	-0.10675	0.995345	0.130152
0.01	0	-0.16565	-0.12214	0.256214	0.097437
0.01	0.001	-1.60709	-0.11624	2.822332	0.109392

Table 24: Conclusion - impedance control line

In the active direction (movement), a low stiffness value and no damping gives the best result. For the passive direction a high stiffness gives the best results, since maintaining a pose is easier when the stiffness is increased.

For the circular movement, the effect of the stiffness is not big because the radius is chosen big and the velocity rather slow. It is still visible that when going from 100 to 500 as active stiffness value, the performance improves significantly. When the value rises from 1000 to 2000 it is visible that the stiffness value becomes too high and leads to a control action that is too strong.

	Average [mm]		Standard deviation [mm]	
KJ	y	x	y	x
100	-0.0237	-0.0198	0.88625	0.72782
500	-0.0202	-0.00999	0.724407	0.697575
1000	-0.01236	-0.01038	0.714813	0.696257
2000	-0.0882	-0.01458	1.269412	0.69123

Table 25: Conclusion - impedance control circle

Machining with collaborative robot

Because in milling a combination of normal and lateral forces occur, in theory the force control in this application is more complicated and needs a force compensation and trajectory compensation as well. Literature confirmed that complex calculations are needed or specialized equipment needs to be designed for accurate force control of this kind.

A colleague student Ricardo Ruiz Monsalve was assigned to make his Master Thesis about the application of collaborative robots for milling. Thanks to his approval for me to use the results of his research, the literature study done on this topic is enforced by experiments.

He started with the UR3 and later performed the same experiments using the UR3e. The goal is to mill in a linear motion, while maintaining a certain depth and velocity. First only motion control in the movement direction was used, but this delivered bad results for the UR3, trying to solve this with an extra control parameter in the other direction was unsuccessful. Especially when entering the material, the high elasticity of the collaborative robot influences the performance.

The UR3e created almost perfect lines, but not always with a great surface finishing. When using harder material, it was necessary to mill at a slow feed rate and a shallow depth of cut. The best results came when applying motion control in X, Y and Z, because the line has to be straight and the cutting depth has to be constant. It should be noted that the internal joint stiffnesses of the robots is information we do not get access to, so this is a possible reason why the accuracy of the UR3e is higher than the UR3's.

In conclusion, it is made possible to perform milling tasks with a collaborative robot, despite not finding research papers with this sort of robot. In the used setup (controlling the motor of the mill with an Arduino and controlling the motion of the mill with the robot Teach Pendant) Ricardo achieved positive results, without using complicated force or motion compensation formulas or predicting lateral force values.

7. References

1. Abu-Dakka, F. J., & Saveriano, M. (2020, December). Variable Impedance Control and Learning—A Review. *Frontiers in Robotics and AI, 7*(590681). doi:10.3389/frobt.2020.590681

2. Amersdorfer, M., Kappey, J., & Meurer, T. (2020, October). Real-time freeform surface and path tracking for force controlled robotic tooling applications. *Robotics an Computer-Integrated Manufacturing, 65*. doi:10.1016/j.rcim.2020.101955

3. Axis New England. (n.d.). Universal Robots Multi-Robot Communication. Retrieved from https://us.v-cdn.net/6027406/uploads/editor/yt/p993fs105yxo.pdf

4. Bonev, I. (2019, August 27). *What are Singularities in a Six-Axis Robot Arm*. Retrieved from mecademic.com: https://www.mecademic.com/en/what-are-singularities-in-a-six-axis-robot-arm

5. Bragança, S., Costa, E., Castellucci, I., & Arezes, P. M. (2019). A Brief Overview of the Use of Collaborative Robots in Industry 4.0: Human Role and Safety. *Occupational and Environmental Safety and Health*, 641-650.

6. Brugali, D., & Fayad, M. (2002, September). Distributed computing in robotics and automation. *IEEE Transactions on Robotics and Automation, 18*(4), 409-420. doi:10.1109/TRA.2002.802937

7. Bruggeman, K. (2022). Robotica en mechanismen. *JLI32G*. Catholic University of Leuven, Belgium.

8. Burn, K., Robinson, E., & Dixon, D. (2014). Introducing fundamental concepts of control: A case study in robot force control. *International Journal of Mechanical Engineering Education, 42*(4), 320-339. doi:10.1177/0306419015574223

9. Cao, N., & Lynch, A. F. (2016, September). Inner-outer loop control for quadrotor UAVs with input and state constraints. *IEEE Transactions on Control Systems, 24*(5), 1797-1804. doi:10.1109/TCST.2015.2505642

10. Chang, R.-J., & Jau, J.-C. (2016). Augmented Reality in Peg-in-Hole Microassembly Operations. *International Journal of Automation Technology, 10*(3), 438-446. doi:https://doi.org/10.20965/ijat.2016.p0438

11. Cho, H., Kim, M., Lim, H., & Kim, D. (2014). Cartesian sensor-less force control for industrial robots. *International Conference on Intelligent Robots and Systems*, 4497-4502. doi:https://doi.org/10.1109/IROS.2014.6943199

12. Dawson, B. (2013, July 8). *randomascii.wordpress.com*. Retrieved from Windows Timer Resolution: Megawatts Wasted: https://randomascii.wordpress.com/2013/07/08/windows-timer-resolution-megawatts-wasted/#:~:text=The%20default%20timer%20resolution%20on,consumption%20and%20harm%20battery%20life.

13. Goldenberg, A. A. (1988). Implementation of force and impedance control in robot manipulators. *1988 IEEE International Conference on Robotics and Automation. 3*, pp. 1626-1632. IEEE. doi:10.1109/ROBOT.1988.12299

14. Greenwood, D. T. (1977). *Classical dynamics.* Mineola, New York: Dover Publications, INC.

15. Harris, F. J., & Rice, M. (2001, December). Multirate digital filters for symbol timing synchronization in software defined radios. *IEEE Journal on Selected Areas in Communications, 19*(12), 2346-2357. doi:10.1109/49.974601

16. Hogan, N. (1985, March). Impedance Control: An Approach to Manipulation. *Journal of Dynamic Systems, Measurement, and Control, 107*, 1-24. doi:10.23919/ACC.1984.4788393

17. IPA SA. (n.d.). *Robot Motion*. Retrieved from http://www.ipacv.ro/proiecte/robotstudio/textbooks/file/robot_motion.htm

18. Iqbal, K., & Mughal, A. (2007). Active control vs. passive stiffness in posture and movement coordination. *2007 IEEE International Conference on Systems, Man and Cybernetics* (pp. 3367-3372). IEEE. doi:10.1109/ICSMC.2007.4414105

19. Learnchannel-TV. (n.d.). *Robot JOINT-coordinates*. Retrieved from Learnchannel-TV.com: https://learnchannel-tv.com/robot/robot-coordinate-systems/joint-coordinates/

20. Learnchannel-TV. (n.d.). *Robot tool coordinates*. Retrieved from https://learnchannel-tv.com/robot/robot-coordinate-systems/tool-coordinates/

21. Li, Z., Huang, Z., He, W., & Su, C.-Y. (2017, February). Adaptive Impedance Control for an Upper Limb Robotic Exoskeleton Using Biological Signals. *IEEE Transactions on Industrial Electronics, 64*(2), 1664-1674. doi:10.1109/TIE.2016.2538741

22. Liu, G., Abdul, S., & Goldenberg, A. A. (2007, June 22). Distributed control of modular and reconfigurable robot with torque sensing. *Robotica, 26*, 75-84.

23. Mathworks. (n.d.). *Design Trajectory with Velocity Limits Using Trapezoidal Velocity Profile*. Retrieved from mathworks.com: https://nl.mathworks.com/help/robotics/ug/design-a-trajectory-with-velocity-limits-using-a-trapezoidal-velocity-profile.html

24. Matsubara, A., & Ibaraki, S. (2009). Monitoring and Control of Cutting Forces in Machining Processes: A Review. *International Journal of Automation Technology, 3*(4), 445-456.

25. Matthias, B., Kock, S., Jerregard, H., Kallman, M., Lundberg, I., & Mellander, R. (2011). Safety of collaborative industrial robots: Certification possibilities for a collaborative assembly robot concept. *2011 IEEE International Symposium on Assembly and Manufacturing (ISAM)* (pp. 1-6). Tampere, Finland: IEEE. doi:10.1109/ISAM.2011.5942307

26. Maurya, P., Aguiar, A. P., & Pascoal, A. (2009). Marine Vehicle Path Following Using Inner-Outer Loop Control. *IFAC International Conference on Manoeuvring and Control of Marine Craft*, *42*, pp. 38-43. Guarujá (SP), Brazil. doi:10.3182/20090916-3-BR-3001.0071

27. Mizuochi, M., Tsuji, T., & Ohnishi, K. (2005). Force sensing and force control using multirate sampling method. *31st Annual Conference of IEEE Industrial Electronics Society, 2005. IECON 2005.* (pp. 1919-1924). IEEE. doi:10.1109/IECON.2005.1569198

28. Modbus. (2012, April 26). *MODBUS protocol.* Retrieved from modbus.org: https://modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf

29. Naumann, M., & Alexopoulos, K. (n.d.). *UR Capabilities*. Retrieved from portal.effra.eu: https://portal.effra.eu/result/show/1177

30. Neranon, P., & Bicker, R. (2016). Force/position control of robot manipulator for human-robot interaction. *Thermal Science*, 537-548. doi:10.2298/TSCI151005036N

31. OnRobot. (2018, September). DESCRIPTION Compute Box. *(Compute Box Version 4.0.0), Edition E10*.

32. Pan, Z., & Zhang, H. (2008). Robotic machining from programming to process control: a complete solution by force control. *Industrial Robot: An International Journal, 35*(5), 400-409. doi: 10.1108/01439910810893572

33. Pauliková, A., Babel'ová, Z. G., & Ubárová, M. (2021). Analysis of the Impact of Human–Cobot Collaborative Manufacturing Implementation on the Occupational Health and Safety and the Quality Requirements. *International Journal of Environmental Research and Public Health*. doi:https://doi.org/10.3390/ijerph18041927

34. Raiola, G., Cárdenas, C., Tadele, T. S., Vries, T. d., & Stramigioli, S. (2018). Development of a Safety- and Energy-Aware Impedance Controller for Collaborative Robots. *IEEE Robotics and Automation Letters*, 1237-1244. doi:10.1109/LRA.2018.2795639

35. Rodrigo, P.-U., & Ranko, Z.-S. (2020). Force Control Improvement in Collaborative Robots through Theory Analysis and Experimental Endorsement. *applied sciences*.

36. Selesnick, I. W., Arnold, S., & Dantham, V. R. (2012, December). Polynomial Smoothing of Time Series With Additive Step Discontinuities. *IEEE Transactions on Signal Processing, 60*(12), 6305-6318. doi:10.1109/TSP.2012.2214219

37. Strasser, T., Rooker, M., & Ebenhofer, G. (2008). Distributed Control Concept for a 6-DOF Reconfigurable Robot Arm. *PROFACTOR GmbH*.

38. Tajima, S., Iwamoto, S., & Yoshioka, H. (2021). Kinematic Tool-Path Smoothing for 6-Axis Industrial Machining Robots. *IJAT, 15*(5), 621-630. Retrieved from https://doi.org/10.20965/ijat.2021.p0621

39. Tech Target Contributor. (2017, september). *distributed control system (DCS)*. Retrieved from WhatIs.com: https://whatis.techtarget.com/definition/distributed-control-system

40. Testa, G. (2017, June). Experimental stiffness identification in the joints of a lightweight robot. Barcelona, Spain.

41. Universal Robots. (2015). *UR3 Technical specifications.* Retrieved from Retrieved from: https://www.universal-robots.com/media/240787/ur3_us.pdf

42. Universal Robots. (2018). *e-series*. Retrieved from Universal-Robots.com: https://www.universal-robots.com/e-series/

43. Universal Robots. (2018). *UR3e Technical details.* Retrieved from https://www.universal-robots.com/media/1802780/ur3e-32528_ur_technical_details_.pdf

44. Universal Robots. (2019). Datasheet HEX-E/H QC. Retrieved from https://onrobot.com/sites/default/files/documents/Datasheet_HEX_QC_v1.1_EN%20%281%29.pdf

45. Universal Robots. (2021, May 28). *ROBOT WORKING AREA DXF-UR3-CB-SERIES.* Retrieved from universal-robots.com: https://www.universal-robots.com/download/mechanical-cb-series/ur3/robot-working-area-dxf-ur3-cb-series/

46. Universal Robots. (2021, May 28). *ROBOT WORKING AREA PDF - UR3e - E-SERIES.* Retrieved from universal-robots.com: https://www.universal-robots.com/download/mechanical-e-series/ur3e/robot-working-area-pdf-ur3e-e-series/

47. Universal Robots. (2022, January 19). What is a singularity? Retrieved from https://www.universal-robots.com/articles/ur/application-installation/what-is-a-singularity/

48. Universal Robots. (n.d.). The URScript Programming Language.

49. Welch, G., & Bishop, G. (1995). *An introduction to the Kalman filter.*

50. WiredWorkers. (n.d.). *Cobots*. Retrieved from Wiredworkers.io: https://wiredworkers.io/cobot/

51. Xie, C., Zhao, X., Savaghebi, M., Meng, L., & Guerrero, J. M. (2017, June). Multirate Fractional-Order Repetitive Control of Shunt Active Power Filter Suitable for Microgrid Applications. *IEEE Journal of Emerging and Selected Topics in Power Electronics, 5*(2), 809-819. doi:10.1109/JESTPE.2016.2639552

52. Xie, Y., Sun, D., Liu, C., Tse, H., & Cheng, S. (2010, August). A Force Control Approach to a Robot-assisted Cell Microinjection System. *The International Journal of Robotics Research, 29*(9), 1222-1232. doi:10.1177/0278364909354325

53. Yoshikawa, T. (2000). Force control of robot manipulators. *2000 IEEE International Conference on Robotics & Automation*, (pp. 220-226). San Francisco, CA. doi:10.1109/ROBOT.2000.844062

54. Yu, W., & Rosen, J. (2013, April). Neural PID Control of Robot Manipulators With Application to an Upper Limb Exoskeleton. *IEEE Transactions on Cybernetics, 43*(2), 673-684. doi:10.1109/TSMCB.2012.2214381

55. Zeng, G., & Hemami, A. (1997). An overview of robot force control. *Robotica, 15*, 473-482. doi:10.1017/S026357479700057X

8. Appendix

Appendix I – UR – PC communication

```python
import socket
import time

HOST = "158.42.236.117"# The remote host
PORT = 30000 # The same port as used by the server
print("Starting Program")
i = 0
while i<1:
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    s.bind((HOST, PORT)) # Bind to the port
    print("Port binded")
    s.listen(5) # Now wait for client connection.
    print("Client connected")
    c, addr = s.accept() # Establish connection with client.
    print("Connection with client")
    try:
        msg = c.recv(1024)
        print(msg)
        time.sleep(2)
        i = i + 1
        pos = "(390,200,180)"
        c.send(pos.encode())
        print("Data sent")
        time.sleep(0.5)
        print("")
        time.sleep(0.5)
        pos2 = msg.decode()
        pos2=pos2[2:len(pos2)]
        b=pos2.replace("[","").replace("]","")
        lst=[x for x in b.split(',')]
        print(lst)
        p1 = str(float(lst[0])*1000)
        p2 = str(float(lst[1])*1000)
        p3 = str(float(lst[2])*1000)
        pos2 = "("+p1+","+p2+","+p3+")"
        print(pos2)
    except socket.error as socketerror:
        print("Error")
c.close()
s.close()
print("Program finish")
```

```
Programa
  AntesDeIniciar
    var_1:=socket_open("158.42.206.7",30000)
    MoverJ
      Punto_de_paso_1
  Programa de robot
    Bucle var_1≟ False
      var_1:=socket_open("158.42.206.7",30000)
      Esperar: 0.5
    var_2:=get_actual_tcp_pose()
    socket_send_string(var_2)
    Esperar: 0.5
    var_3:=[0,0,0,0]
    Bucle var_3[0]≟0
      var_3:=socket_read_ascii_float(3)
    Esperar: 0.5
    var_4:=p[var_3[1]/1000,var_3[2]/1000,(var_3[3]/1000)-0.05,0,0,0]
    var_5:=pose_trans(var_2,var_4)
    var_7:=p[0,0,0.25,0,0,0]
    MoverJ
      var_5
      Esperar: 1.0
    var_8:=get_actual_tcp_pose()
    var_6:=pose_trans(var_8,var_7)
    MoverL
      var_6
      Esperar: 1.0
    var_10:=get_actual_tcp_pose()
    var_11:=p[0,0,-0.25,0,0,0]
    var_9:=pose_trans(var_10,var_11)
    MoverL
      var_9
      Esperar: 1.0
    var_13:=get_actual_tcp_pose()
    var_11:=p[-var_3[1]/1000,-var_3[2]/1000,-(var_3[3]/1000)+0.05,0,0,0]
    var_12:=pose_trans(var_13,var_11)
    MoverJ
      var_12
      Esperar: 1.0
```

Table 26: UR - PC communication Python and UR code

Appendix II – Basic force control

```
Programa
  AntesDeIniciar
    F/T Cero
  Programa de robot
    MoverJ
      Punto_de_paso_2
    Esperar: 0.2
    F/T Cero
    Bucle norm(Fz)<5
      MoverL
        Dirección: Base Z-
    F/T Mover
      F/T Ruta: route_01
```

Table 27: Basic force control UR code

Appendix III – Remote robot force reading

```python
import socket
import time

HOST = "158.42.206.7"# The remote host
PORT = 30000 # The same port as used by the server
print("Starting Program")
i = 0

while (1):
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    s.bind((HOST, PORT)) # Bind to the port
    print("Port binded")
    s.listen(5) # Now wait for client connection.
    print("Client connected")
    c, addr = s.accept() # Establish connection with client.
    print("Connection with client")
    try:
        while (1):
            msg = c.recv(1024)
            print(msg)
            msg2 = msg.decode()
            msg2 = msg2[1:len(msg2)]
            b=msg2.replace("[","").replace("]","")
            lst=[x for x in b.split(',')]
            print("Fx: {0} , Fy: {1} , Fz: {2} ".format(lst[0],lst[1],lst[2]))
            Fx = float(lst[0])
    except socket.error as socketerror:
        print("Error")

c.close()
s.close()
print("Program finish")
```

```
Programa
  AntesDeIniciar
    socket_open("158.42.206.7",30000)
    F/T Cero
  Programa de robot
    MoverJ
      Punto_de_paso_2
    F/T Cero
    Esperar: 1.0
    Bucle norm(Fz)<5
      MoverL
        Dirección: Base Z-
          Until (expression)
    F/T Cero
    F/T Control
      F/T Mover
        F/T Ruta
    Esperar: 1.0
  SubTarea_1
    Esperar: 0.008
    force1:=[Fx,Fy,Fz]
    socket_send_string(force1)
```

Table 28: Remote robot force reading Python and UR code

Appendix IV – Remote control

```python
import urx
import logging
import time
import socket
HOST = "192.168.1.133"
PORT = 40000
print("program start")
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((HOST,PORT))
time.sleep(0.5)
print("turning on digital output port2")
stringstr = "set_digital_out(2,true)"
bitbit = stringstr.encode()
s.send(bitbit)
time.sleep(0.1)
command = "movej(p[0.2, 0.3, 0.5, 0, 0, 3.14], a=0.5, v=0.5)"
s.send(command.encode())
print("command sent ")
time.sleep(1)
command = "movej(p[1.21, -1.9, -1.4, -2.89, -0.85, -1.12], a=0.5, v=0.5)"
s.send(command.encode())
print("command sent ")
time.sleep(1)
print("Getdata")
data = s.recv(1024)
s.close()
print(repr(data))
print("finish")
if __name__ == "__main__":
    logging.basicConfig(level=logging.INFO)
    robot = urx.Robot("192.168.1.133")
    robot.set_tcp((0, 0, 0, 0, 0, 0))
    robot.set_payload(0.5, (0,0,0))
    try:
        v = 0.5
        a = 0.3
        r = 0.01
        init_pos = robot.getj()
        print("Initial joint position is ", init_pos)
        transf = robot.get_pose()
        print("Transformation from base to tcp is: ", transf)
        joints_pos1 = (1.21, -1.90, -1.40, -2.89, -0.85, -1.12)
        joints_pos2 = (2.26, -1.71, -1.27, -1.84, -2.28, -1.18)
        print("Move to position 1")
        robot.movej(joints_pos1, acc=a, vel=v)
        time.sleep(5)
        print("Move to position 2")
        robot.movej(joints_pos2, acc=a, vel=v)
    finally:
        robot.close()
    print("Finished")
```

```python
import socket
import time

HOST = "158.42.206.7"# The remote host
PORT = 49151 # The same port as used by the server

print("Starting Program")
i = 0

while (1):
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    s.bind((HOST, PORT)) # Bind to the port
    print("Port binded")
    s.listen(5) # Now wait for client connection.
    print("Client connected")
    #c, addr = s.accept() # Establish connection with client.
    #print("Connection with client")
    try:
        while (1):
            print("start")
            datarequest = '1000000000000000000'
            print(datarequest.encode())

            array = [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
            byte_array = bytearray(array)
            print(byte_array)
            c.send(byte_array)
            print("message sent")

            msg = s.recv(1024)
            #print(msg)
            msg2 = msg.decode()
            print(msg2)

    except socket.error as socketerror:
        print("Error")

c.close()
s.close()
print("Program finish")
```

Table 29: Attempts remote control Python codes

Appendix V – Attempt direct sensor reading

Web scraping	
1	2

`import requests`
`from bs4 import BeautifulSoup`
 	
`url = 'http://192.168.1.98/#/devices'`
`page = requests.get(url)`
`print(page) #check connection`
`print(page.text)`
`print('content: ')`
`print(page.content)`
`soup = BeautifulSoup(page.text, 'html.parser')`
`print(soup)`
`table1 = soup.find('div', id = 'monitoring')`
`print(table1)`
`forces=[]`
`for i in table1.find_all('td'):`
`  valores = i.text`
`  forces.append(valores)`
`print(forces)`
	`import requests`
	
	`resp = requests.get('http://192.168.1.98/#/devices')`
	`print(resp.text)`
	`req = urllib.request.urlopen('http://192.168.1.98/#/devices')`
	`print(str(req.getcode() ))`
	`data = req.read()`
	`print(data)`

Compute box connection a	

```python
import socket
import time

HOST = "158.42.206.7"# The remote host
PORT = 49151 # The same port as used by the server
CLIENT = "158.42.206.1"
PORT_S = 40000
print("Starting Program")
i = 0

while (1):
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    s.bind((HOST, PORT)) # Bind to the port
    print("Port binded")
    s.listen(5) # Now wait for client connection.
    print("Client connected")
    c, addr = s.accept() # Establish connection with client.
    print("Connection with client")
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client.connect(("158.42.206.1", 40000))
    try:
        while (1):
            msg = c.recv(1024)
            print(msg)
            msg2 = msg.decode()
            msg2 = msg2[1:len(msg2)]
            b=msg2.replace("[","").replace("]","")
            lst=[x for x in b.split(',')]
            print("Fx: {0} , Fy: {1} , Fz: {2} ".format(lst[0],lst[1],lst[2]))
            #valor_recibido = int.from_bytes(msg,"big", signed = "True")
            Fx = float(lst[0])
            #print(Fx)
            print("Client test value")
            from_server = client.recv(4096)
            print(from_server)
    except socket.error as socketerror:
        print("Error")
    client.close ()
c.close()
s.close()
print("Program finish")
```

Table 30: Attempts direct ssensor reading Python codes

Appendix VI – Force control with excel writing

```python
import socket
import time
import xlsxwriter
workbook = xlsxwriter.Workbook('UR3_29.04_InitiateSample.xlsx')
worksheet = workbook.add_worksheet()
print("excel created")
row = 1
column = 1
worksheet.write('B1', 'Fx')
worksheet.write('C1', 'Fy')
worksheet.write('D1', 'Fz')

HOST = "192.168.1.102"# PC IP address
PORT = 40000 # The same port as used by the server
print("Starting Program")
i = 0
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
s.bind((HOST, PORT)) # Bind to the port
print("Port binded")
s.listen(5) # Now wait for client connection.
print("Client connected")
c, addr = s.accept() # Establish connection with client.
print("Connection with client")
startSignal = c.recv(1024)
print(startSignal) #the signal is var_start_send, with value 1
start_time = time.time()
print("Start writing ")

while startSignal == b'1' : #byte notation
    msg = c.recv(1024)
    current_time = time.time()
    elapsed_time = current_time - start_time
    if elapsed_time > 3:    #sample time of 2sec
        print("time elapsed")
        break
    else:
        msg2 = msg.decode()
        msg2 = msg2[1:len(msg2)]
        b=msg2.replace("[","").replace("]","")
        lst=[x for x in b.split(',')]
        #print("Fx: {0} , Fy: {1} , Fz: {2} ".format(lst[0],lst[1],lst[2]) )
        column = 1
        for item in lst:
            worksheet.write(row,0,elapsed_time)
            worksheet.write(row, column, item)
            column += 1
        row += 1
workbook.close()
print("Excel is finished")
c.close()
s.close()
print("Program finish")
```

```
Program
  BeforeStart
    var_1:=socket_open("158.42.206.7",40000)
  Robot Program
    var_start_send:=0
    Loop var_1≟ False
      var_1:=socket_open("158.42.206.7",40000)
    MoveJ
      Punto_de_paso_2
    Wait: 0.01
    MoveJ
      Waypoint_1
    Force
      MoveP
        Waypoint_2
    Wait: 1.0
  SubTarea_1
    var_speed:=get_target_tcp_speed()
    If var_speed[1]<-0.02
      Wait: 1.0
      var_start_send:=1
      socket_send_string(var_start_send)
    Loop var_start_send≟1
      Wait: 0.01
      Forces_var:=get_tcp_force()
      socket_send_string(Forces_var)
```

Table 31: Force control with excel writing Python and UR code

Appendix VII – Adaptive force control

```python
from socket import*
import struct
import zlib
import numpy as np
import time
import xlsxwriter
workbook = xlsxwriter.Workbook('multirate_average_comp.xlsx')
worksheet = workbook.add_worksheet()
print("excel created")
row = 1
worksheet.write('A1', 'time')
worksheet.write('B1', 'Fz')
worksheet.write('C1', 'F1')
worksheet.write('D1', 'F2')
worksheet.write('E1', 'F3')
worksheet.write('F1', 'F4')

s = socket(AF_INET,SOCK_DGRAM) #UDP compute box
s.bind(('192.168.1.103', 30001))
IPAddress = '192.168.1.98'
port = 49152
v = socket(AF_INET, SOCK_STREAM) #computer socket
v.bind(('192.168.1.103' , 40000))
v.listen(5)
print("connection request sent")
client,addr = v.accept()
print("connection with Robot")
def transform(command, data):
    command = int(command, 16)
    command = np.array([command], dtype=np.uint16)
    command.byteswap(inplace=True)
    data = np.array([data], dtype=np.uint32)
    data.byteswap(inplace=True)
    x = np.array([a, command], dtype=np.uint16)
    request_header_command = x.view(np.uint8)
    request_data = data.view(np.uint8)
    request_header_command = request_header_command.flatten().tolist()
    request_data = request_data.flatten().tolist()
    request = np.append(request_header_command, request_data)
    request = np.array([request], dtype=np.uint8)
    return request
header = '1234'
a = int(header,16)
a = np.array([a], dtype=np.uint16)
a.byteswap(inplace=True)
command_1 = '2'     #command to start sending output
data_1 = '1'        #number of sent samples
command_2 = '0082'  #command to set read-out speed in ms
data_2 = '2'        #2ms
command_3 = '0042'  #command to set biasing
data_3 = '255'      #set bias, 0 is reset bias
request_send = transform(command_1,data_1)
request_speed = transform(command_2, data_2)
request_biasing = transform(command_3,data_3)
s.sendto(request_biasing,(IPAddress,port))
print('request_biasing sent')
s.sendto(request_speed,(IPAddress,port))
print('request_read-out speed sent')
startSignal=client.recv(8)
print(startSignal)

programtime= time.time_ns()/1000000
listvalues = []
i=0
Fest=0
averagePeriod = time.time_ns()/1000000
Fgoal=5
```

```
Programa
  AntesDeIniciar
    var_1:=socket_open("192.168.1.103",40000)
  Programa de robot
    var_start_send:=0
    Bucle var_1≟ False
      var_1:=socket_open("192.168.1.103",40000)
    MoverJ
      Punto_de_paso_2
    Esperar: 0.01
    MoverL
      Dirección: Base Z-
        Until (expression)
    Esperar var_start_send≟1
    F/T Mover
      F/T Punto de ref.
    var_start_send:=0
    socket_send_string(var_start_send)
    Esperar: 0.5
  SubTarea_1
    var_data:=socket_read_ascii_float(1)
    f_correction:=5
    If norm(Fz)>2
      var_start_send:=1
      socket_send_string(var_start_send)
    Bucle var_start_send≟1
      var_data:=socket_read_ascii_float(1)
      f_correction:=var_data[1]
      force_mode(tool_pose(), [0,0,1,0,0,0],
[0,0,f_correction,0,0,0],2,[0.1,0.1,0.15,0.17,0.17,0.17])
    end_force_mode()
    Esperar: 1.0
```

```python
while startSignal==b'1':
    i+=1
    start_time = time.time_ns()/1000000
    s.sendto(request_send, (IPAddress, port))
    receivePacket, addr = s.recvfrom(36)
    end_time = time.time_ns() / 1000000
    delta_time = end_time - start_time
    recibido = struct.unpack('IIIIIIIII', receivePacket)
    recibido = np.array(recibido, dtype=np.uint32)
    recibido = recibido.astype('int32')
    recibido.byteswap(inplace=True)
    valuesAll = np.array_split(recibido,3)
    valuesF = valuesAll[1]/10000 #Force
    Fz = valuesF[2]
    listvalues.append(Fz)
    if (i % 4 == 0):
        end_period = time.time_ns()/1000000
        period = end_period - averagePeriod
        if (period>8):
            Fest = 1/3*(listvalues[0]+listvalues[1]+listvalues[2])
            del listvalues[-1:]
            period = period - delta_time
        else:
            Fest = 1/4*(listvalues[0]+listvalues[1]+listvalues[2]+listvalues[3])
        print("LIST RESET, AVERAGE IS: ", average)
        Fcorrection = 2*Fgoal-abs(Fest)
        print("CORRECTION IS: ", Fcorrection)
        f_d = "(" + str(Fcorrection) + ")"
        f_d = f_d.encode()
        client.send(f_d)
        elapsed_time=end_period-start_time
        worksheet.write(row,0,elapsed_time)
        worksheet.write(row, 1, Fest)
        worksheet.write(row, 2, listvalues[0])
        worksheet.write(row, 3, listvalues[1])
        worksheet.write(row, 4, listvalues[2])
        worksheet.write(row, 5, listvalues[3])
        row += 1
        listvalues = []
        averagePeriod = time.time_ns()/1000000
    elif (averagePeriod-programtime>3500):
        workbook.close()
        print("Excel is finished")
        break
client.close()
v.close()
print("Program finished")
```

```python
while startSignal==b'1':
    i+=1
    start_time = time.time_ns()/1000000
    s.sendto(request_send, (IPAddress, port))
    receivePacket, addr = s.recvfrom(36)
    end_time = time.time_ns() / 1000000
    delta_time = end_time - start_time
    recibido = struct.unpack('IIIIIIIII', receivePacket)
    recibido = np.array(recibido, dtype=np.uint32)
    recibido = recibido.astype('int32')
    recibido.byteswap(inplace=True)
    valuesAll = np.array_split(recibido,3)
    valuesF = valuesAll[1]/10000 #Force
    Fz = valuesF[2]
    listvalues.append(Fz)
    if (i % 4 == 0):
        end_period = time.time_ns()/1000000
        period = end_period - averagePeriod
        if (period>8):
            Fest = 0.2*listvalues[0] + 0.2*listvalues[1] + 0.6*listvalues[2]
            del listvalues[-1:]
            period = period - delta_time
        else:
            Fest = 0.1 * listvalues[0] + 0.2 * listvalues[1] + 0.3 * listvalues[2] + 0.4
 * listvalues[3]
        print("LIST RESET, AVERAGE IS: ", average)
        Fcorrection = 2*Fgoal-abs(Fest)
        print("CORRECTION IS: ", Fcorrection)
        f_d = "(" + str(Fcorrection) + ")"
        f_d = f_d.encode()
        client.send(f_d)
        elapsed_time=end_period-start_time
        worksheet.write(row,0,elapsed_time)
        worksheet.write(row, 1, Fest)
        worksheet.write(row, 2, listvalues[0])
        worksheet.write(row, 3, listvalues[1])
        worksheet.write(row, 4, listvalues[2])
        worksheet.write(row, 5, listvalues[3])
        row += 1
        listvalues = []
        averagePeriod = time.time_ns()/1000000
    elif (averagePeriod-programtime>3500):
        workbook.close()
        print("Excel is finished")
        break
client.close()
v.close()
print("Program finished")
```

Table 32: Adaptive force control Python and UR code

Appendix VIII – Trapezoidal speed trajectory

```python
import socket
import time
import xlsxwriter

workbook = xlsxwriter.Workbook('Trapezoidal.xlsx')
worksheet = workbook.add_worksheet()
worksheet.write('A1', 'time')
worksheet.write('B1','position')
worksheet.write('C1','speed')
row=1

HOST = "192.168.1.103"# The remote host
PORT = 40000 # The same port as used by the server
print("Starting Program")
i = 0

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
s.bind((HOST, PORT)) # Bind to the port
print("Port binded")
s.listen(5) # Now wait for client connection.
print("Client connected")
c, addr = s.accept() # Establish connection with client.
print("Connection with client")
startSignal = c.recv(1024)
print(startSignal) #the signal is var_start_send, with value 1
start_time = time.time()
print("Start writing ")
while startSignal == b'1' : #byte notation
    msg = c.recv(1024)
    msg2 = msg.decode()
    msg2 = msg2[1:len(msg2)]
    b=msg2.replace("[","").replace("]","")
    lst=[x for x in b.split(',')]
    print("FORCE: ",lst)
    worksheet.write_row(row,0,lst)
    row+=1
    if (msg == b'0'):
        c.close()
        break
workbook.close()
s.close()
print("Program finished")
```

```
Programa
  AntesDeIniciar
    F/T Cero
    Fz_ant=0
    Fref=-5
    ei=0
    ed=0
    tm=0.008
    Vyref=0.03
    t0=0
    start=0
    var_1=socket_open("192.168.1.103",40000)
    Control SpeedL
      P_act=0
      D_act=0
      Kp=0.001
      Kd=0.000005
      Kv=0.0
  Programa de robot
    Bucle var_1≠ False
      var_1=socket_open("192.168.1.103",40000)
    Bucle
      Control SpeedL Z
        MoverJ
          Punto_de_paso_1
        start=1
        socket_send_string(start)
        Esperar: 0.5
        Bucle norm(Fz)<1
          speedl([0,0,-0.005,0,0,0],1,0)
        Esperar: 0.5
        Vref=Vyref
        Aref=0.05
        Trapezium
          P0=get_actual_tcp_pose()
          Pf=[0,0.2,0,0,0,0]
          s=Pf[1]-P0[1]
          If s<0
            sign=-1
          Else
            sign=1
          If norm(s)>Vref*Vref/Aref
            t1=Vref/Aref
            t2=norm(s)/Vref
            tfin=2*t1+t2
          Else
            t1=sqrt(norm(s)/Aref)
            t2=0
            tfin=2*t1
          Vt=0
          Yt=0
          t0=0
          tcp_pos0=get_actual_tcp_pose()
        Bucle
          Force=Fz
          ef=Fref-Force
          ed=(Force-Fz_ant)/0.008
          speed=get_actual_tcp_speed()
          P_act=Kp*ef
          D_act=Kd*ed
          Vz_ref=P_act+D_act
          If t0≤t1+t2
            speedl([0,Vyref,vz_ref,0,0,0],Aref,0)
          Else
            speedl([0,0,0,0,0,0],-Aref,0)
            If norm(x1)<0.1
              start=0
              socket_send_string(start)
              MoverJ
                Punto_de_paso_1
              Detener
          Fz_ant=Force
          t0=t0+tm
  Subproceso_1
    Esperar: 0.008
    tiempo: Iniciar
    targe_joint_pos=get_target_joint_positions()
    targ_tcp_speed=get_target_tcp_speed()
    tcp_speed=get_actual_tcp_speed()
    tcp_pose=get_actual_tcp_pose()
    target_tcp_pose=get_target_tcp_pose()
      x1=tcp_speed[1]*1000
      y1=tcp_pose[1]
    datos=[t0,x1,y1,Fz,P_act,D_act]

    If start≠1
      socket_send_string(datos)
```

Table 33: Trapezoidal speed trajectory Python and UR code

Appendix IX – Sampling period force control

```python
import socket
import time
import xlsxwriter
workbook = xlsxwriter.Workbook('UR3e_sampleForce3s_05_05_2ms.xlsx')
worksheet = workbook.add_worksheet()
print("excel created")
row = 1
column = 1
worksheet.write('B1', 'Fx')
worksheet.write('C1', 'Fy')
worksheet.write('D1', 'Fz')

HOST = "192.168.1.103"# The remote host
PORT = 40000 # The same port as used by the server
print("Starting Program")
i = 0
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
s.bind((HOST, PORT)) # Bind to the port
print("Port binded")
s.listen(5) # Now wait for client connection.
print("Client connected")
c, addr = s.accept() # Establish connection with client.
print("Connection with client")
startSignal = c.recv(1024)
print(startSignal) #the signal is var_start_send, with value 1
start_time = time.time()
print("Start writing ")

while startSignal == b'1' : #byte notation
    msg = c.recv(1024)
    current_time = time.time()
    elapsed_time = current_time - start_time
    if elapsed_time > 3:    #sample time of Xsec
        print("time elapsed")
        break
    else:
        msg2 = msg.decode()
        msg2 = msg2[1:len(msg2)]
        b=msg2.replace("[","").replace("]","")
        lst=[x for x in b.split(',')]
        print("Fx: {0} , Fy: {1} , Fz: {2} ".format(lst[0],lst[1],lst[2]))
        column = 1
        for item in lst:
            worksheet.write(row,0,elapsed_time)
            worksheet.write(row, column, item)
            column += 1
        row += 1
workbook.close()
print("Excel is finished")
c.close()
s.close()
print("Program finish")
```

UR3 Force	UR3 Touch
Programa AntesDeIniciar var_1:=socket_open("192.168.1.103",40000) Programa de robot var_start_send:=0 Bucle var_1≟ False var_1:=socket_open("192.168.1.103",40000) MoverJ Punto_de_paso_2 Esperar: 0.01 MoverL Dirección: Base Z- Until (expression) F/T Control F/T Mover F/T Punto de ref. Esperar: 1.0 SubTarea_1 var_speed:=get_target_tcp_speed() If var_speed[0]>0.02 var_start_send:=1 socket_send_string(var_start_send) Bucle var_start_send≟1 Esperar: 0.008 Forces_var:=[Fx, Fy, Fz] socket_send_string(Forces_var)	Programa AntesDeIniciar var_1:=socket_open("192.168.1.103",40000) Programa de robot var_start_send:=0 Bucle var_1≟ False var_1:=socket_open("192.168.1.103",40000) MoverJ Punto_de_paso_2 Esperar: 0.01 MoverL var_start_send:=1 socket_send_string(var_start_send) Dirección: Base Z- Until (expression) F/T Control F/T Mover F/T Punto de ref. Esperar: 1.0 SubTarea_1 Bucle var_start_send≟1 Esperar: 0.008 Forces_var:=[Fx, Fy, Fz] socket_send_string(Forces_var)
UR3e Force	UR3e Touch

```
Program
  BeforeStart
    var_1:=socket_open("192.168.1.103",40000)
  Robot Program
    var_start_send:=0
    force_torque:=get_tcp_force()
    Loop var_1≟ False
      var_1:=socket_open("192.168.1.103",40000)
    MoveJ
      Punto_de_paso_2
    Wait: 0.1
    MoveL
      Direction: Base Z-
        Until (tool_contact_detection)
    Force
      MoveL
        Direction: Base X+
          Until (distance)
    Wait: 1.0
  SubTarea_1
    var_speed:=get_target_tcp_speed()
    If var_speed[0]>0.02
      Wait: 1.0
      var_start_send:=1
      socket_send_string(var_start_send)
    Loop var_start_send≟1
      Wait: 0.002     //2 for fast and 8 for slow
      force_torque:=get_tcp_force()
      socket_send_string(force_torque)
```

```
Program
  BeforeStart
    var_1:=socket_open("192.168.1.103",40000)
  Robot Program
    var_start_send:=0
    force_torque:=get_tcp_force()
    Loop var_1≟ False
      var_1:=socket_open("192.168.1.103",40000)
    MoveJ
      Punto_de_paso_2
    Wait: 0.1
    MoveL
      var_start_send:=1
      socket_send_string(var_start_send)
      Direction: Base Z-
        Until (tool_contact_detection)
    Force
      MoveL
        Direction: Base X+
          Until (distance)
    Wait: 1.0
  SubTarea_1
    Loop var_start_send≟1
      Wait: 0.002     //2 for fast and 8 for slow
      force_torque:=get_tcp_force()
      socket_send_string(force_torque)
```

Table 34: Sampling period force control Python and UR codes

Appendix X – Multi-robot communication

UR3

```
Programa
  AntesDeIniciar
    write_port_register(128,0)
    MoverJ
      StartPose
  Programa de robot
    MoverJ
      Esperar MODBUS_5≟0
      Punto_de_paso_2
    MoverL
      Dirección: Base X+
        Until (expression)
    write_port_register(128,10)
    F/T Control
      F/T Mover
        Esperar MODBUS_5≟50
        F/T Punto de ref.: LiftBox
        Esperar: 0.5
        F/T Punto de ref.
    Esperar: 1.0
    write_port_register(128,20)
    Esperar MODBUS_5≟60
    MoverL
      Dirección: Base X-
        Until (distance)
    write_port_register(128,0)
    Esperar: 1.0
```

UR3e

```
Program
  BeforeStart
    write_port_register(128,0)
    MoveJ
      StartPose
  Robot Program
    MoveJ
      Wait MODBUS_4≟0
      Waypoint_3
    MoveL
      Direction: Base Y-
        Until (expression)
    write_port_register(128,50)
    Force
      MoveL
        Direction: Base Z+
          Until (distance)
        Wait: 0.5
        Direction: Base Z-
          Until (distance)
    Wait: 1.0
    write_port_register(128,60)
    Wait MODBUS_4≟20
    MoveL
      Direction: Base Y+
        Until (distance)
    write_port_register(128,0)
    Wait: 1.0
```

Table 35: Multi-robot communication UR codes

Appendix XI – Communication time delay

```python
import datetime
import socket
import time

HOST = "192.168.1.103"# The remote host
PORT = 30000 # The same port as used by the server
print("Starting Program")
i = 0

while (1):
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    s.bind((HOST, PORT)) # Bind to the port
    print("Port binded")
    s.listen(5) # Now wait for client connection.
    print("Client connected")
    c, addr = s.accept() # Establish connection with client.
    print("Connection with client")
    try:
        for x in range(0, 50):
                returnUnit = 0
                testunit = "(1,1,1)"
                c.send(testunit.encode())
                time_start = datetime.datetime.now()
                returnUni= c.recv(1024)
                returnUnit = returnUni.decode()
                if returnUnit != 0:
                    time_end = datetime.datetime.now()
                    deltaTime = time_end - time_start
                    print(deltaTime)

    except socket.error as socketerror:
        print("Error")

c.close()
s.close()
print("Program finish")
```

```
Programa
  AntesDeIniciar
    var_1≔socket_open("192.168.1.103",30000)
    MoverJ
      Punto_de_paso_1
  Programa de robot
    Bucle var_1≟ False
      var_1≔socket_open("192.168.1.103",30000)
    var_3≔[0,0,0,0]
    Bucle var_3[0]≟0
      var_3≔socket_read_ascii_float(3)
      socket_send_int(1)
```

```
Program
  BeforeStart
    var_1≔socket_open("192.168.1.103",30000)
    MoveJ
      Punto_de_paso_1
  Robot Program
    Loop var_1≟ False
      var_1≔socket_open("192.168.1.103",30000)
    var_3≔[0,0,0,0]
    Loop var_3[0]≟0
      var_3≔socket_read_ascii_float(3)
      socket_send_int(1)
```

Table 36: Communication time delay Python and UR codes

Appendix XII – Force sensor connection

```python
from socket import*
import struct
import numpy as np
import time
s = socket(AF_INET,SOCK_DGRAM)
s.bind(('192.168.1.103', 30001))
IPAddress = '192.168.1.98'
port = 49152
def transform(command, data):
    command = int(command, 16)
    command = np.array([command], dtype=np.uint16)
    command.byteswap(inplace=True)
    data = np.array([data], dtype=np.uint32)
    data.byteswap(inplace=True)
    x = np.array([a, command], dtype=np.uint16)
    request_header_command = x.view(np.uint8)
    request_data = data.view(np.uint8)
    request_header_command = request_header_command.flatten().tolist()
    request_data = request_data.flatten().tolist()
    request = np.append(request_header_command, request_data)
    request = np.array([request], dtype=np.uint8)
    return request
header = '1234'
a = int(header,16)
a = np.array([a], dtype=np.uint16)
a.byteswap(inplace=True)
command_1 = '2'     #command to start sending output
data_1 = '1'        #number of sent samples
command_2 = '0082'  #command to set read-out speed in ms
data_2 = '2'        #2ms
command_3 = '0042'  #command to set biasing
data_3 = '255'      #set bias, 0 is reset bias
request_send = transform(command_1,data_1)
request_speed = transform(command_2, data_2)
request_biasing = transform(command_3,data_3)
s.sendto(request_biasing,(IPAddress,port))
print('request_biasing sent')
s.sendto(request_speed,(IPAddress,port))
print('request_read-out speed sent')
while True:
    start_time = time.time_ns()/1000000
    s.sendto(request_send, (IPAddress, port))
    receivePacket, addr = s.recvfrom(36)
    end_time = time.time_ns() / 1000000
    delta_time = end_time - start_time
    recibido = struct.unpack('IIIIIIIII', receivePacket)
    recibido = np.array(recibido, dtype=np.uint32)
    recibido = recibido.astype('int32')
    recibido.byteswap(inplace=True)
    valuesAll = np.array_split(recibido,3)
    valuesS = valuesAll[0] #sensor status
    valuesF = valuesAll[1]/10000 #Force
    valuesT = valuesAll[2]/100000 #Torque
    print(valuesF , " Force values")
    print(delta_time , " millseconds")
```

Table 37: Force sensor connection Python code

Appendix XIII – Force filtering and control

Proportional – PD – PV

```
import socket
import time
import xlsxwriter

workbook = xlsxwriter.Workbook('PV_trap_(0001).xlsx')
worksheet = workbook.add_worksheet('0001')
Kp=0.001
Kd=0.0015
print("excel created")
row = 1
column = 1
worksheet.write('A1', 'time')
worksheet.write('B1', 'speed')
worksheet.write('C1', 'position')
worksheet.write('D1', 'Force')
worksheet.write('E1', 'P-action')
worksheet.write('F1', 'D-action')

HOST = "192.168.1.103"# The remote host
PORT = 40000 # The same port as used by the server
print("Starting Program")
i = 0

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
s.bind((HOST, PORT)) # Bind to the port
print("Port binded")
s.listen(5) # Now wait for client connection.
print("Client connected")
c, addr = s.accept() # Establish connection with client.
print("Connection with client")

startSignal = c.recv(1024)
print(startSignal) #the signal is var_start_send, with value 1
start_time = time.time()
print("Start writing ")


while startSignal == b'1' : #byte notation
    msg = c.recv(1024)
    msg2 = msg.decode()
    msg2 = msg2[1:len(msg2)]
    b=msg2.replace("[","").replace("]","")
    lst=[x for x in b.split(',')]
    print ("force values ", lst)
    worksheet.write_row(row, 0, lst)
    row += 1
    if (msg == b'0'):
        worksheet.write(0, 7, 'Kp')
        worksheet.write(0, 8, 'Kv')
        worksheet.write(1, 7, Kp)
        worksheet.write(1, 8, Kd)
        c.close()
        break
workbook.close()
print("Excel is finished")
c.close()
s.close()
print("Program finish")
```

```
Programa
  AntesDeIniciar
    F/T Cero
    Fz_ant=0
    Fref=-5
    ei=0
    ed=0
    tm=0.008
    Vz=0
    Vyref=0.03
    t0=0
    start=0
    var_1=socket_open("192.168.1.103",40000)
    Control SpeedL
      P_act=0
      D_act=0
      Kp=0.001
      Kd=0
      Kv=0.015
  Programa de robot
    Bucle var_1≠ False
      var_1=socket_open("192.168.1.103",40000)
    Bucle
      Control SpeedL Z
        MoverJ
          Punto_de_paso_1
        start=1
        socket_send_string(start)
        Esperar: 0.5
        Bucle norm(Fz)<1
          speedl([0,0,-0.005,0,0,0],1,0)
        Esperar: 0.5
        Vref=Vyref
        Aref=0.05
        Trapezium
          P0=get_actual_tcp_pose()
          Pf=[0,0.2,0,0,0,0]
          s=Pf[1]-P0[1]
          If s<0
            sign=-1
          Else
            sign=1
          If norm(s)>Vref*Vref/Aref
            t1=Vref/Aref
            t2=norm(s)/Vref
            tfin=2*t1+t2
          Else
            t1=sqrt(norm(s)/Aref)
            t2=0
            tfin=2*t1
          Vt=0
          Yt=0
          t0=0
          tcp_pos0=get_actual_tcp_pose()
        Bucle
          Force=Fz
          ef=Fref-Force
          'ed=(Force-Fz_ant)/0.008'
          P_act=Kp*ef
          D_act=Kv*tcp_speed[2]
          Vz_ref=P_act-D_act
          If t0≤t1+t2
            speedl([0,Vyref,Vz_ref,0,0,0],Aref,0)
          Else
            speedl([0,0,0,0,0,0],-Aref,0)
            If norm(x1)<0.1
              start=0
              socket_send_string(start)
              MoverJ
                Punto_de_paso_1
              Detener
          Fz_ant=Force
          t0=t0+tm
  Subproceso_1
    Esperar: 0.008
    targ_tcp_speed=get_target_tcp_speed()
    tcp_speed=get_actual_tcp_speed()
    tcp_pose=get_actual_tcp_pose()
    dat_target_tcp_speed
      w6=targ_tcp_speed[0]*1000
      w7=targ_tcp_speed[1]*1000
      w8=targ_tcp_speed[2]*1000
    dat_actual_tcp_speed
      x0=tcp_speed[0]*1000
      x1=tcp_speed[1]*1000
      x2=tcp_speed[2]*1000
    dat_tcp_pose

  y0=tcp_pose[0]
  y1=tcp_pose[1]*1000
  y2=tcp_pose[2]
datos=[t0,x1,y1,Fz,P_act,D_act]
If start≠1
  socket_send_string(datos)
```

PD with averaging filter

```
from socket import*
import struct
import zlib
import numpy as np
import time
import xlsxwriter
workbook = xlsxwriter.Workbook('PD_trap_average_(0003).xlsx')
worksheet = workbook.add_worksheet('av_0003')
Kp = 0.001
Kd = 0.00003
print("excel created")
worksheet.write('A1', 'time')            #0
worksheet.write('B1', 'Forces_measured') #1
worksheet.write('D1', 'time')            #3
worksheet.write('E1', 'velocity')        #4
worksheet.write('F1', 'position')        #5
worksheet.write('G1', 'force')           #6
worksheet.write('H1', 'P action')        #7
worksheet.write('I1', 'D action')        #8
worksheet.write('J1','tiempo')           #9
s = socket(AF_INET,SOCK_DGRAM) #UDP compute box
s.bind(('192.168.1.103', 30001))
IPAddress = '192.168.1.98'
port = 49152
v = socket(AF_INET, SOCK_STREAM) #computer socket
v.bind(('192.168.1.103' , 40000))
v.listen(5)
print("connection request sent")
client,addr = v.accept()
print("connection with Robot")
def transform(command, data):
    command = int(command, 16)
    command = np.array([command], dtype=np.uint16)
    command.byteswap(inplace=True)
    data = np.array([data], dtype=np.uint32)
    data.byteswap(inplace=True)
    x = np.array([a, command], dtype=np.uint16)
    request_header_command = x.view(np.uint8)
    request_data = data.view(np.uint8)
    request_header_command = request_header_command.flatten().tolist()
    request_data = request_data.flatten().tolist()
    request = np.append(request_header_command, request_data)
    request = np.array([request], dtype=np.uint8)
    return request
header = '1234'     #header of every request
a = int(header,16)
a = np.array([a], dtype=np.uint16)
a.byteswap(inplace=True)
command_1 = '2'     #command to start sending output
data_1 = '1'        #number of sent samples
command_2 = '0082'  #command to set read-out speed in ms
data_2 = '2'        #2ms
command_3 = '0042'  #command to set biasing
data_3 = '255'      #set bias, 0 is reset bias
request_send = transform(command_1,data_1)
request_speed = transform(command_2, data_2)
request_biasing = transform(command_3,data_3)
s.sendto(request_biasing,(IPAddress,port))
print('request_biasing sent')
s.sendto(request_speed,(IPAddress,port))
print('request_read-out speed sent')
startSignal=client.recv(8)
print(startSignal)
programtime= time.time_ns()/1000000
listvalues = []
listperiods = []
row = 1
i=0
average=0
Fp=5 #Fprevious, assume 5N
Fv=0 #Fvelocity
averagePeriod = time.time_ns()/1000000
while startSignal==b'1':
    i+=1
    #msg = client.recv(1024)
    start_time = time.time_ns()/1000000
    s.sendto(request_send, (IPAddress, port))
    receivePacket, addr = s.recvfrom(36)
    end_time = time.time_ns() / 1000000
    delta_time = end_time - start_time
    elapsed_t = end_time - programtime
    recibido = struct.unpack('IIIIIIIII', receivePacket)
    recibido = np.array(recibido, dtype=np.uint32)
    recibido = recibido.astype('int32')
    recibido.byteswap(inplace=True)
    valuesAll = np.array_split(recibido,3)
    valuesF = valuesAll[1]/10000 #Force
    Fz = valuesF[2]
    listvalues.append(Fz)
    listperiods.append(elapsed_t)
    if (i==4):
        end_period = time.time_ns()/1000000
        period = end_period-averagePeriod
        if(period>8):
            average = 1/3*(listvalues[0]+listvalues[1]+listvalues[2])
            del listvalues[-2:] #delete last two values in list
            period = period - delta_time
        else:
            average = 1/4*(listvalues[0]+listvalues[1]+listvalues[2]+listvalues[3])
        Fv=(average-Fp)/0.008
        Fp=average
        sent = "(" + str(average) + ")"
        msg = client.recv(1024)
        msg2 = msg.decode()
        msg2 = msg2[1:len(msg2)]
        b = msg2.replace("[", "").replace("]", "")
        lst = [x for x in b.split(',')]
        worksheet.write_row(row, 3, lst)
        if (msg==b'0'):
            worksheet.write(0, 10, 'Kp')
            worksheet.write(0, 11, 'Kd')
            worksheet.write(1, 10, Kp)
            worksheet.write(1, 11, Kd)
            workbook.close()
            client.close()
            print("EXCEL FINISHED")
            break
        sent = sent.encode()
        client.send(sent)
        worksheet.write_column(row,1, listvalues)
        worksheet.write_column(row,0,listperiods)
        row += 4
        listvalues = []
        listperiods = []
        averagePeriod = time.time_ns() / 1000000
        i=0
v.close()
print("Program finished")
```

2 order filter

```
    if (i==4):
        end_period = time.time_ns()/1000000
        period = end_period-averagePeriod
        if(period>8):
            average = 0.2*listvalues[0]+0.2*listvalues[1]+0.6*listvalues[2]
            del listvalues[-2:] #delete last two values in list
            period = period - delta_time
        else:
            average = 0.1*listvalues[0]+0.2*listvalues[1]+0.3*listvalues[2]+0.4*
listvalues[3]
```

```
Programa
  AntesDeIniciar
    F/T Cero
    Fz_ant=0
    Fref=-5
    ei=0
    ed=0
    tm=0.008
    Vz=0
    Vyref=0.03
    t0=0
    start=0
    var_1=socket_open("192.168.1.103",40000)
    Control SpeedL
      P_act=0
      D_act=0
      Kp=0.001
      Kd=0.00003
  Programa de robot
    Bucle var_1≠ False
      var_1=socket_open("192.168.1.103",40000)
    Bucle
      Control SpeedL Z
        MoverJ
          Punto_de_paso_1
        start=1
        socket_send_string(start)
        Esperar: 0.5
        Bucle norm(Fz)<1
          speedl([0,0,-0.005,0,0,0],1,0)
        Esperar: 0.5
        Vref=Vyref
        Aref=0.05
        Trapezium
          P0=get_actual_tcp_pose()
          Pf=[0,0.2,0,0,0,0]
          s=Pf[1]-P0[1]
          If s<0
            sign=-1
          Else
            sign=1
          If norm(s)>Vref*Vref/Aref
            t1=Vref/Aref
            t2=norm(s)/Vref
            tfin=2*t1+t2
          Else
            t1=sqrt(norm(s)/Aref)
            t2=0
            tfin=2*t1
          Vt=0
          Yt=0
          t0=0
          tcp_pos0=get_actual_tcp_pose()
        Bucle
          Force=f_average
          ef=Fref-Force
          ed=(Force-Fz_ant)/0.008
          speed=get_actual_tcp_speed()
          Vz=speed[2]
          P_act=Kp*ef
          D_act=Kd*ed
          Vz_ref=P_act+D_act
          If t0≤t1+t2
            speedl([0,vyref,Vz_ref,0,0,0],Aref,0)
          Else
            speedl([0,0,0,0,0,0],-Aref,0)
            If norm(x1)<0.1
              start=0
              socket_send_string(start)
              MoverJ
                Punto_de_paso_1
              Detener
          Fz_ant=Force
          t0=t0+tm
  Subproceso_1
    Esperar: 0.008
    fdata=socket_read_ascii_float(1)
    f_average=fdata[1]
    targ_tcp_speed=get_target_tcp_speed()
    tcp_speed=get_actual_tcp_speed()
    tcp_pose=get_actual_tcp_pose()
    dat_target_tcp_speed
      w6=targ_tcp_speed[0]*1000
      w7=targ_tcp_speed[1]*1000
      w8=targ_tcp_speed[2]*1000
    dat_actual_tcp_speed
      x0=tcp_speed[0]*1000
      x1=tcp_speed[1]*1000
      x2=tcp_speed[2]*1000
    dat_tcp_pose
      y0=tcp_pose[0]
      y1=tcp_pose[1]*1000
      y2=tcp_pose[2]
    datos=[t0,x1,y1,Fz,P_act,D_act]
    If start≠1
      socket_send_string(datos)
```

Polynomial approximation	

```python
from socket import*
import struct
import zlib
import numpy as np
import time
import xlsxwriter
workbook = xlsxwriter.Workbook('polynomial_trap_0003.xlsx')
worksheet = workbook.add_worksheet('poly_0003')
print("excel created")
Kp=0.001
Kd=0.00003
worksheet.write('A1', 'time period') #0
worksheet.write('B1', 'x3')          #1
worksheet.write('C1', 'x2')          #2
worksheet.write('D1', 'x1')          #3
worksheet.write('E1', 'C')           #4
worksheet.write('G1', 'time')     #6
worksheet.write('H1', 'Forces')       #7
worksheet.write('J1', 'Polytime')          #9
worksheet.write('K1', 'Polyforces')          #10
worksheet.write('J1', 'Kp')          #11
worksheet.write('K1', 'Kd')          #12
worksheet.write('J2', Kp)
worksheet.write('K2', Kd)
s = socket(AF_INET,SOCK_DGRAM) #UDP compute box
s.bind(('192.168.1.103', 30001))
IPAddress = '192.168.1.98'
port = 49152
v = socket(AF_INET, SOCK_STREAM) #computer socket
v.bind(('192.168.1.103' , 40000))
v.listen(5)
print("connection request sent")
client,addr = v.accept()
print("connection with Robot")
def transform(command, data):
    command = int(command, 16)
    command = np.array([command], dtype=np.uint16)
    command.byteswap(inplace=True)
    data = np.array([data], dtype=np.uint32)
    data.byteswap(inplace=True)
    x = np.array([a, command], dtype=np.uint16)
    request_header_command = x.view(np.uint8)
    request_data = data.view(np.uint8)
    request_header_command = request_header_command.flatten().tolist()
    request_data = request_data.flatten().tolist()
    request = np.append(request_header_command, request_data)
    request = np.array([request], dtype=np.uint8)
    return request
def polynom(t,x0,x1,x2,x3):
    value = x0*pow(t,3) + x1*pow(t,2) + x2 * t + x3
    return  value
header = '1234'     #header of every request
a = int(header,16)
a = np.array([a], dtype=np.uint16)
a.byteswap(inplace=True)
command_1 = '2'     #command to start sending output
data_1 = '1'        #number of sent samples
command_2 = '0082'  #command to set read-out speed in ms
data_2 = '2'        #2ms
command_3 = '0042'  #command to set biasing
data_3 = '255'      #set bias, 0 is reset bias
request_send = transform(command_1,data_1)
request_speed = transform(command_2, data_2)
request_biasing = transform(command_3,data_3)
s.sendto(request_biasing,(IPAddress,port))
print('request_biasing sent')
s.sendto(request_speed,(IPAddress,port))
print('request_read-out speed sent')
startSignal=client.recv(8)
print(startSignal)
programtime= time.time_ns()/1000000
listvalues = []
listperiods = []
row = 1
n=1
i=0
Fest=0
averagePeriod = time.time_ns()/1000000
Fgoal=5
while startSignal==b'1':
    i+=1
    start_time = time.time_ns()/1000000
    s.sendto(request_send, (IPAddress, port))
    receivePacket, addr = s.recvfrom(36)
    end_time = time.time_ns() / 1000000
    delta_time = end_time - start_time
    elapsed_t = end_time - programtime
    recibido = struct.unpack('IIIIIIIII', receivePacket)
    recibido = np.array(recibido, dtype=np.uint32)
    recibido = recibido.astype('int32')
    recibido.byteswap(inplace=True)
    valuesAll = np.array_split(recibido,3)
    valuesF = valuesAll[1]/10000 #Force
    Fz = valuesF[2]
    listvalues.append(Fz)
    listperiods.append(elapsed_t)
    if (i==4):
        end_period = time.time_ns()/1000000
        period = end_period-averagePeriod
        if(period>8):
            del listvalues[-1]
            del listperiods[-1]
            period=period-delta_time
            programtime+=delta_time
            z = np.polyfit(listperiods, listvalues, 2)
            z = np.insert(z, 0, 0)
        else:
            z = np.polyfit(listperiods, listvalues, 3)
        worksheet.write(row, 0, period)
        worksheet.write_row(row, 1, z)
        worksheet.write_column(row, 7, listvalues)
        worksheet.write_column(row, 6, listperiods)
        msg = client.recv(1024)
        msg2 = msg.decode()
        msg2 = msg2[1:len(msg2)]
        b = msg2.replace("[", "").replace("]", "")
        lst = [x for x in b.split(',')]
        if (msg==b'0'):
            workbook.close()
            client.close()
            print("EXCEL FINISHED")
            break
        n=round(listperiods[0]-0.5)
        for k in range(9):
            worksheet.write(n,9,n)
            polyF = polynom(n,z[0],z[1],z[2],z[3])
            worksheet.write(n,10,polyF)
            n+=1
        row += 4
        listvalues = []
        listperiods = []
        averagePeriod = time.time_ns() / 1000000
        i=0
client.close()
```

Kalman filter

```python
from socket import*
import struct
import zlib
import numpy as np
import time
import xlsxwriter
        #define commands
def transform(command, data):
    command = int(command, 16)
    command = np.array([command], dtype=np.uint16)
    command.byteswap(inplace=True)
    data = np.array([data], dtype=np.uint32)
    data.byteswap(inplace=True)
    x = np.array([a, command], dtype=np.uint16)
    request_header_command = x.view(np.uint8)
    request_data = data.view(np.uint8)
    request_header_command = request_header_command.flatten().tolist()
    request_data = request_data.flatten().tolist()
    request = np.append(request_header_command, request_data)
    request = np.array([request], dtype=np.uint8)
    return request
        # prepare excel file
workbook = xlsxwriter.Workbook('Kalman_Kp0.0005_(3).xlsx')
worksheet = workbook.add_worksheet('p0.0005_3')
Kp=0.0005
Kd=0.0005
Q= 0.5      #process noise, estimation
worksheet.write('A1', 'Measurement#')  # 0
worksheet.write('B1', 'RealF')  # 1
worksheet.write('D1', 'Kp')  # 3
worksheet.write('E1', 'Kd')  # 4
        # initial parameters
Tm = 0.002  #time of 2ms
Rk = 0.0095   #sensor variance, out of Matlab
x1=1        #state estimation of force

x2=0        #state estimation of force derivative

p11=1       #covariance matrix elements
p12=0
p21=0
p22=1
F=5         #Force value should be five
listperiods = []
listvalues = []
i=0
averagePeriod = time.time_ns()/1000000
row=1
        #define kalman steps:predict and correct
def predictor(x1,x2,p11,p12,p21,p22):
    x1 = x1 + x2*Tm
    x2 = x2
    p11 = p11 + p21*Tm + Tm*(p12+p22*Tm)
    p12 = p12 + p22*Tm
    p21 = p21 + p22*Tm
    p22 = p22 + Q
    return [x1, x2,p11,p12,p21,p22]
def corrector(x1,x2,p11,p12,p21,p22,F):
    k1 = p11*(p11+Rk)
    k2 = p21*(p11+Rk)
    x1 = x1+k1*(F-x1)
    x2 = x2+k2*(F-x1)
    p11 = p11*(1-k1)
    p12 = p12*(1-k1)
    p21 = p11*(-k2)+p21
    p22 = p12*(-k2)+p22
    return [x1,x2,p11,p12,p21,p22]
        #set up connection
s = socket(AF_INET,SOCK_DGRAM) #UDP compute box
s.bind(('192.168.1.103', 30001))
IPAddress = '192.168.1.98'
port = 49152
v = socket(AF_INET, SOCK_STREAM) #robot-computer socket
v.bind(('192.168.1.103' , 40000))
v.listen(5)
print("connection request sent")
client,addr = v.accept()
print("connection with Robot")
        #prepare data requests
header = '1234'     #header of every request
a = int(header,16)
a = np.array([a], dtype=np.uint16)
a.byteswap(inplace=True)
command_1 = '2'     #command to start sending output
data_1 = '1'        #number of sent samples
command_2 = '0082'  #command to set read-out speed in ms
data_2 = '2'        #2ms
command_3 = '0042'  #command to set biasing
data_3 = '255'      #set bias, 0 is reset bias
request_send = transform(command_1,data_1)
request_speed = transform(command_2, data_2)
request_biasing = transform(command_3,data_3)
        #set up connection with compute box
s.sendto(request_biasing,(IPAddress,port))
print('request_biasing sent')
s.sendto(request_speed,(IPAddress,port))
print('request_read-out speed sent')
startSignal=client.recv(8)
print("Start signal received")
programtime= time.time_ns()/1000000

while startSignal==b'1':
    i+=1
    start_time = time.time_ns()/1000000
    s.sendto(request_send, (IPAddress, port))
    receivePacket, addr = s.recvfrom(36)
    end_time = time.time_ns() / 1000000
    delta_time = (end_time - start_time)/1000            #time in (milli)seconds
    elapsed_t = end_time - programtime
    recibido = struct.unpack('IIIIIIIII', receivePacket)
    recibido = np.array(recibido, dtype=np.uint32)
    recibido = recibido.astype('int32')
    recibido.byteswap(inplace=True)
    valuesAll = np.array_split(recibido,3)
    valuesF = valuesAll[1]/10000                        #Force in N
    Fz = valuesF[2]
        #Fz value is extracted, what action to perform?
    listvalues.append(Fz)
    listperiods.append(elapsed_t)
    if (i==4):
        worksheet.write_column(row, 0, listperiods)
        worksheet.write_column(row, 1, listvalues)
        row = row + 4
        for n in range(len(listvalues)):
            x1, x2, p11, p12, p21, p22 = corrector(x1, x2, p11, p12, p21, p22,
listvalues[n])  # measurement update
            #print("corrected", x1, " , ", x2)
            #worksheet.write((row - 1), 6, p11)
            #worksheet.write((row - 1), 3, x1)
            x1, x2, p11, p12, p21, p22 = predictor(x1, x2, p11, p12, p21, p22)
            #worksheet.write((row - 1), 7, p11)
            #worksheet.write((row - 1), 4, x1)
            #print("predicted ", x1, " ",x2)
        send="("+str(x1)+"," + str(x2) +")"
        send=send.encode()
        print(send)
        client.send(send)
        msg = client.recv(48)
        msg2 = msg.decode()
        msg2 = msg2[1:len(msg2)]
        b = msg2.replace("[", "").replace("]", "")
        lst = [x for x in b.split(',')]
        if (msg==b'0'):
            break
        listvalues=[]
        listperiods=[]
        i=0

worksheet.write(0,5,'FINAL')
worksheet.write(0,6,"State vector")
worksheet.write(0,8,"Covariance vector")
worksheet.write(1,6,x1)
worksheet.write(2,6,x2)
worksheet.write(1,8,p11)
worksheet.write(1,9,p12)
worksheet.write(2,8,p21)
worksheet.write(2,9,p22)
worksheet.write(1,3,Kp)
worksheet.write(1,4,Kd)
workbook.close()
client.close()
v.close()
print("Program finished")
```

```
Programa
  AntesDeIniciar
    F/T Cero
    Fz_ant=0
    Fref=-5
    ei=0
    ed=0
    tm=0.008
    Vz=0
    Vyref=0.03
    f_der=0
    t0=0
    start=0
    var_1=socket_open("192.168.1.103",40000)
    Control SpeedL
      Kff=0.000
      P_act=0
      D_act=0
      Kp=0.001
      Ki=0.00
      Kd=0.0005
      Kv=0.0
  Programa de robot
    Bucle var_1≠ False
      var_1=socket_open("192.168.1.103",40000)
    Bucle
      Control SpeedL Z
        MoverJ
          Punto_de_paso_1
        Esperar: 0.5
        start=1
        socket_send_string(start)
        Bucle norm(Fz)<1
          speedl([0,0,-0.005,0,0,0],1,0)
        Esperar: 0.5
        Vref=Vyref
        Aref=0.05
        Trapezium
          P0=get_actual_tcp_pose()
          Pf=[0,0.2,0,0,0,0]
          s=Pf[1]-P0[1]
          If s<0
            sign=-1
          Else
            sign=1
          If norm(s)>Vref*Vref/Aref
            t1=Vref/Aref
            t2=norm(s)/Vref
            tfin=2*t1+t2
          Else
            t1=sqrt(norm(s)/Aref)
            t2=0
            tfin=2*t1
          Vt=0
          Yt=0
          t0=0
          tcp_pos0=get_actual_tcp_pose()
        Bucle
          Force=f_av
          ef=Fref-Force
          ed=f_der
          speed=get_actual_tcp_speed()
          Vz=speed[2]
          P_act=Kp*ef
          D_act=Kd*ed
          Vz_ref=P_act+D_act
          If t0≤t1+t2
            speedl([0,Vyref,Vz_ref,0,0,0],Aref,0)
          Else
            speedl([0,0,0,0,0,0],-Aref,0)
            If norm(x1)<0.1
              start=0
              'socket_send_string(start)'
              MoverJ
                Punto_de_paso_1
              Detener
          Fz_ant=Force
          t0=t0+tm
  Subproceso_1
    Esperar: 0.008
    fdata=socket_read_ascii_float(2)
    f_av=fdata[1]
    f_der=fdata[2]
    socket_send_string(start)
    tcp_speed=get_actual_tcp_speed()
    tcp_pose=get_actual_tcp_pose()
    target_tcp_pose=get_target_tcp_pose()
    dat_actual_tcp_speed
      x1=tcp_speed[1]*1000
      x2=tcp_speed[2]*1000
    dat_tcp_pose
      y1=tcp_pose[1]*1000
```

Table 38: Force filtering and control Python and UR codes

Appendix XIV – Impedance control

```python
import socket
import time
import xlsxwriter
workbook = xlsxwriter.Workbook('ImpedanceControl(5).xlsx')
worksheet = workbook.add_worksheet('5')
Kp=0.0001
Kd=0.0000
KJ=1000
print("excel created")
row = 1
column = 1
worksheet.write('A1', 'time')
worksheet.write('B1', 'speed')
worksheet.write('C1', 'Xposition')
worksheet.write('D1', 'Yposition')
worksheet.write('E1', 'xrefposition')
worksheet.write('F1', 'yrefposition')
worksheet.write('G1', 'P-action')
worksheet.write('H1', 'D-action')
HOST = "192.168.1.103"# The remote host
PORT = 40000 # The same port as used by the server
print("Starting Program")
i = 0
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
s.bind((HOST, PORT)) # Bind to the port
print("Port binded")
s.listen(5) # Now wait for client connection.
print("Client connected")
c, addr = s.accept() # Establish connection with client.
print("Connection with client")
startSignal = c.recv(1024)
print(startSignal) #the signal is var_start_send, with value 1
start_time = time.time()
print("Start writing ")
while startSignal == b'1' : #byte notation
    msg = c.recv(1024)
    msg2 = msg.decode()
    msg2 = msg2[1:len(msg2)]
    b=msg2.replace("[","").replace("]","")
    lst=[x for x in b.split(',')]
    print ("position values ", lst)
    worksheet.write_row(row, 0, lst)
    row += 1
    if (msg == b'0'):
        worksheet.write(0, 8, 'Kp')
        worksheet.write(0, 9, 'Kv')
        worksheet.write(1, 8, Kp)
        worksheet.write(1, 9, Kd)
        worksheet.write(1,10,KJ)
        c.close()
        break
workbook.close()
print("Excel is finished")
c.close()
s.close()
print("Program finish")
```

```
Programa
  AntesDeIniciar
    x_ant=0
    P0=get_actual_tcp_pose()
    y_ant=0
    Fref=5
    ei=0
    ed=0
    tm=0.008
    Vz=0
    zref=0
    yref=0
    xref=0
    y_0=0
    pos_y=0
    Vyref=0.032
    v2=0
    Vxref=0
    vref=0
    t0=0
    start=0
    var_1=socket_open("192.168.1.103",40000)
    R=0.05
    w=2.5
    Control SpeedL
      P_act=0
      D_act=0
      Kp=0.1
      Kd=0
      KJ=2000
  Programa de robot
    Bucle var_1≠ False
      var_1=socket_open("192.168.1.103",40000)

    Bucle
      Control SpeedL Z
        MoverJ
          Punto_de_paso_1
        start=1
        socket_send_string(start)

        MoverL
          Dirección: Base Z-
            Until (distance)
        Esperar: 0.5
        Vref=Vyref
        Aref=4
        Trapezium
          P0=get_actual_tcp_pose()
          xref=P0[0]
          y_0=P0[1]
          zref=P0[2]
          Pf=[0,0.2,0,0,0,0]
          s=Pf[1]-P0[1]
          If s<0
            sign=-1
          Else
            sign=1
          If norm(s)>vref*Vref/Aref
            t1=Vref/Aref
            t2=norm(s)/Vref
            tfin=2*t1+t2
          Else
            t1=sqrt(norm(s)/Aref)
            t2=0
            tfin=2*t1
          Vt=0
          y_1=y_0+Aref*t1*t1/2
          y_2=y1+Vyref* t2
          Yt=0
          t0=0
          tcp_pos0=get_actual_tcp_pose()
        Bucle t0≤tfin
          pos_y=y1/1000
          pos_x=y0/1000
          ef=yref-pos_y
          efx=xref-pos_x
          ei=Fref-v2
          ed=(pos_y-y_ant)/0.008
          edx=(pos_x-x_ant)/0.008
          speed=get_actual_tcp_speed()

          P_act=Kp*ef
          D_act=Kd*ed
          Vz=0.00005*ei
          Vxref= Kp*efx+Kd*edx
          Vref=Vref+P_act+D_act
```

```
            If t0≤t1
              speedl([Vxref,Vref,-Vz,0,0,0],Aref,0)
              yref=y_0+Aref*t0*t0/2
              vref=Aref*t0
            ElseIf t0≤t1+t2 and t0>t1
              speedl([Vxref,Vref,-Vz,0,0,0],Aref,0)
              yref=yref+Vyref*tm
            Else
              speedl([0,0,0,0,0,0],-Aref,0)
              vref=Vyref-Aref*(t0-t2)
              yref=y_0
              If norm(x1)<0.1
                start=1
                socket_send_string(start)
            x_ant=pos_x
            y_ant=pos_y
            t0=t0+tm
          tend=t0
          Bucle 1 veces
            Vref=Vyref
            Trapezium
              P0=get_actual_tcp_pose()
              x_c=P0[0]-0.02
              s=2*3.14*R
              w=Vref/R
              'If s<0'
              If norm(s)>Vref*Vref/Aref
                t1=Vref/Aref
                t2=norm(s)/Vref
                tfin=2*t1+t2
              Else
                t1=sqrt(norm(s)/Aref)
                t2=0
                tfin=2*t1
              Vt=0
              Yt=0
              Xt=0
              t0=tend
              tcp_pos0=get_actual_tcp_pose()
            P0=get_actual_tcp_pose()
            Bucle t0≤tfin+tend
              If t0≤tfin+tend
                Vt=Vyref
                yref=-R*sin((t0-tend)*w)
                xref=-R+R*cos(w*(t0-tend))
              Else
                Aref=0
                Vt=0
                yref=0
                xref=0
              speed=get_actual_tcp_speed()
              Vz=speed[2]
              pos2=get_actual_tcp_pose()

tcp_new=pose_add(p[tcp_pos0[0],tcp_pos0[1],pos2[2],P0[3],P0[4],P0[5]],p[xref,yref,0,0,0,0])
              ikine=get_inverse_kin(tcp_new)
              servoj(ikine,Aref,0.03,0.008,0.03,KJ)
              t0=t0+tm
            start=0
            socket_send_string(start)
            Esperar: 0.008
            Detener
    Subproceso_1
      Esperar: 0.008
      tiempo: Iniciar
      targ_tcp_speed=get_target_tcp_speed()
      tcp_speed=get_actual_tcp_speed()
      tcpforce=get_tcp_force()
      tcp_pose=get_actual_tcp_pose()
      target_tcp_pose=get_target_tcp_pose()
      dat_force ur3
        v2=tcpforce[2]
      dat_actual_tcp_speed
        x0=tcp_speed[0]*1000
        x1=tcp_speed[1]*1000
        x2=tcp_speed[2]*1000
      dat_tcp_pose
        y0=tcp_pose[0]*1000
        y1=tcp_pose[1]*1000
        y2=tcp_pose[2]*1000
      datos=[t0,x1,y0,y1,(xref+P0[0])*1000,(yref+P0[1])*1000,P_act,D_act]
      If start=1
        socket_send_string(datos)
```

Table 39: Impedance control Python and UR code