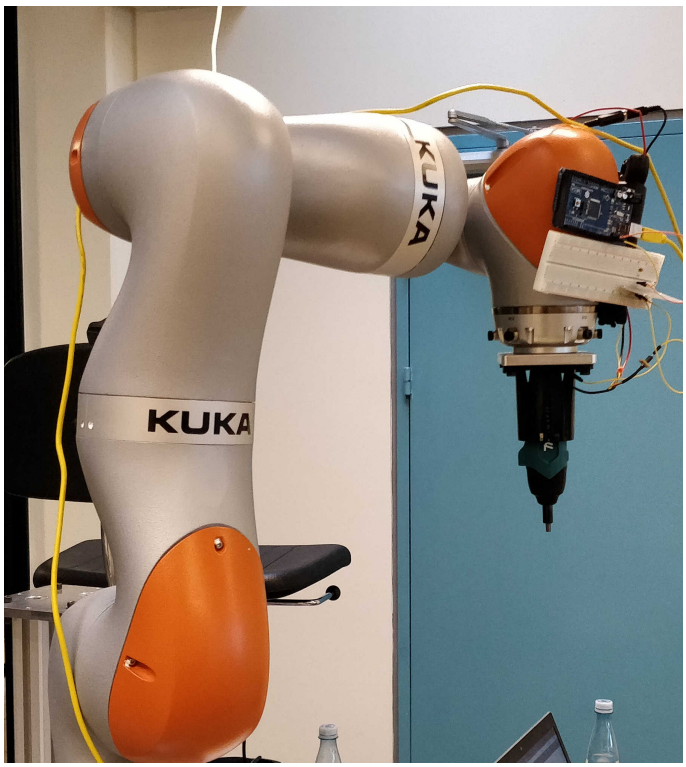
An Impedance Control Driven Robotic Disassembly Platform used For Screw Unfastening

Unfastening of Screws



AALBORG UNIVERSITY
DENMARK

Department of Electronic Systems

Attributions

This report was typeset using LaTeX.

AALBORG UNIVERSITY
DENMARK

Department of Electronic Systems
Department of Materials and Production
M.Sc. Robotics

Fredrik Bajers Vej 7
9220 Aalborg Ø
http://www.es.aau.dk

Title:

An Impedance Control Driven Robotic Disassembly Platform used For Screw Unfastening

Theme:

Master thesis

Project Period:

Spring 2021

Project Group:

Group 1066

Authors:

Jacob Krunderup Sørensen

Supervisors:

Dimitris Chrysostomou
Sebastian Hjorth

Number of Pages: 52
Date of completion: June 2, 2021

Synopsis:

A robotic system using a KUKA LBR iiwa and an Arduino Mega 2560 is presented, which acquires the user/worker input from the integration into the skill-based system, which uses positions taught by the user/-worker to unfasten screws. It does this by utilizing the skill-based system user interface, which is built to make task-level programming possible, enabling an inexperienced worker to program the robot to do complex tasks. The robotic system uses the skill-based system to control and manage the connected devices, which enables the robotic system to use a series of simple commands to the robot resulting in the complex task of unfastening screws. The skill-based system makes this possible through 3 layers of programming, which is utilized to create complex tasks. The robotic system is successful in tests using positions taught by the user/-worker and is able to unfasten screws, despite having screws of different lengths successfully.

Preface

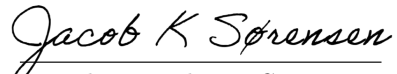
This report is written by Jacob Krunderup Sørensen and contains their thesis for the Robotics Masters programme at Aalborg University.

I thank Dimitris Chrysostomou, my supervisor, for his support during not only the masters but the bachelor as well. He enabled the projects during my time at AAU to always be fun, and sometimes a bit over the top.

I also extend my thanks to my other supervisor Sebastian Hjorth, who helped me to understand the more complex maths used to control the robots.

Furthermore, I thank my colleagues and friends throughout both the bachelor and masters, Daniela, Guilherme, Jan, Jens and Rune for making my time at Aalborg University both fun and academically enriching.

Also, I thank my parents and my brother for helping me get through the tough Covid-19 times, as this has not been the easiest of times. They helped me get through this semester in a big way, which enabled me to finish the study.



Jacob Krunderup Sørensen
<jksa16@student.aau.dk>

Reading Guide

A Git repository for the project is made available on Bitbucket with the following link:

`https://bitbucket.org/Krunderup/kukadisassembly/src/master/`

The list of acronyms ordered alphabetically

Acronym	**Definition**
AAU	Aalborg Universitet
EOL	End of life
SBS	Skill-Based System
HRI	Human-Robot Interaction
GUI	Graphical User Interface
ROS	Robot Operating System
UDP	User Datagram Protocol
IP	Internet Protocol

Table of Contents

List of Figures

List of Tables

1 Introduction

The largest pump manufacturer in the world is Grundfos, which is based in Bjerringbro, Denmark. [3] For this reason, a lot of waste is created once these pumps reach their EOL, or when other problems occur. Because of this, the pumps that are sent to Grundfos to be disassembled come in two different conditions. The first of these is that the pump is almost new, due to it being returned because of either a manufacturing or a delivery problem. The second condition is that the pump can be old, having sustained wear and tear for an extended period of time. The biggest reason for Grundfos investing in the disassembly of these pumps is that it can potentially be made into an economical gain, and it enables Grundfos to use older parts of older pumps, which also helps to protect the environment. [4]

The way that Grundfos is disassembling the pumps currently, is by doing it manually by human workers. Because the disassembly is done manually, it is not cost-effective, compared to what an automatic robotic system could do. [3; 4] For this reason, Grundfos is also looking into design for disassembly for the newer pumps. The current situation at Grundfos is that the returns currently received, is the older models of pumps, meaning that some form of flexibility is needed for the disassembly. This means that while a robotic system would be cheaper in the long run, it is not feasible with the current situation. However, if a robotic system could be created to aid in the disassembly instead, the entire disassembly process could be made more efficient. For this reason, a system using the flex workers currently at Grundfos is explored.

A system using a KUKA LBR iiwa with a camera was used previously to insert a tool into a screw [5]. For this purpose, a tool was created by Technicon. This tool is a pneumatic screwdriver, with a 3D printed shell designed for an Intel D435 camera, and LED lights, which is seen on Figure 1.1. The system takes a picture of the test platform, after which the tool is inserted into the screws one by one, by using the KUKA Sunrise Toolbox to control the robot.

The system managed to prove that as a proof of concept, the system is capable of detecting screws, and inserting the tool into the screws. As mentioned in [5], the KUKA Sunrise Toolbox has some issues in regards to the communication with the robot, as well as the control of the robot. For this reason, a different control strategy is explored for this project. Another thing further explored in this project is the use of impedance control. Another thing mentioned in [5], is that impedance control is used to make the robot compliant, which is useful for human-robot interaction, as the robot can react to a human touching the robot. There was, however, problems with the KUKA Sunrise Toolbox, which made it

Figure 1.1: Tool created by Technicon, with a D435 camera and four LED lights attached

so that impedance control was separate from the other control functions. For this reason, a control strategy is developed with impedance control in mind for this project, which can enable the system to work better with humans.

As this is a continuation of [5], the objective is to create a system that works with flex workers, a simple user interface is needed to guide the workers. For this reason, a system is needed to make the robot easier to use by the flex workers. The system used for this is the Skill-Based System (SBS), presented in chapter 3. SBS is designed to be used by workers having low to no experience working with robots, which is helpful for the flex workers at Grundfos.

The main contributions of this project are:

- UDP connection to the robot
- Full impedance controlled system
- Integration into SBS

The report starts by exploring the related works (chapter 2), where control strategies for assembly and disassembly are explored. Next, the Skill-Based System is explored (chapter 3). After the SBS, the system architecture (chapter 4) is explained, as well as

the connections used to the Arduino and the KUKA LBR iiwa. After the connections are explained, the approach taken to unfasten the screws is explored (chapter 5). Lastly, the report tests and evaluates the system (chapter 6) using the skill execution, finishing with the discussion (chapter 7) and conclusion.

2 Related Works

The task explored in this report concerns the disassembly of Grundfos pumps, or more specifically the unfastening of screws. Because of this, works by other authors regarding disassembly, and unscrewing are explored, as well as how the HRI is handled in assembly and disassembly cases.

2.1 Product Disassembly

As the task studied in this report focuses on the unfastening of screws, other works with similar disassembly tasks are explored.

When dealing with an automatic robotic solution to disassembly and re-manufacturing, different strategies have to be taken into account as failure is likely to happen during the process. There are different ways of dealing with the failures of disassembly e.g. [6] identified three common failures when dealing with the unfastening of screws, these being: tool missing the screw, tool slipping on screw and the screw being too tight to unfasten. They also present a way of dealing with these three issues. The first of these problems is dealt with by having two stages of "peg-in-hole" strategies, the first being a rough search, while the second searching strategy is more thorough. As this is a similar problem to the one presented in this project learnings from this will be taken into account.

In [7], a robotic system is proposed for disassembling electrical vehicles, because of the increasing problem of disposing of them. The work is centered around the disassembly of the steering component where a group of PCBs are extracted by the robot. For this, different operations are used by the robot, such as cutting and manipulation, which is achieved using a tool changer.

DiFilippo proposed a robotic system to unfasten screws from a laptop in [8]. For this purpose, the robot was equipped with a screwdriver and an accelerometer. The purpose was to use a vision system to identify the screw holes, after which the accelerometer would determine whether or not a screw was present to be unfastened.

Another proposed automatic disassembly system proposed in [4], focuses on the insertion of the tool into the screw. To accomplish the task, a particular searching pattern is developed, where the end-effector does a circular movement in the area of the screw. Using this searching pattern the robot will eventually insert the tool into the screw, after which the unfastening of the screw is done using a powered screwdriver.

The use of sampling-based motion planners has also been considered by [9], where an

algorithm is presented to overcome the challenge of the invalid initial state disassembly problem. This is done by considering the information about the flexible parts of the object to be disassembled, which is then used in the disassembly planning.

In [10], a system is proposed to address the uncertainties in the disassembly process automatically. It does this by assessing and selecting the most desirable disassembly actions based on the estimated destructiveness of those actions. The system was tested on several models of screens, where the inaccuracies on the sensors and model were resolved by user intervention.

2.2 HRI

There is a trend when using robotic systems for disassembly, where human workers are brought into the system to allow the system to be more flexible and disassemble more complex products.

To make workers more likely to trust and work well with a robotic system, [11] proposed a framework for testing the trust the worker has in the robot. To make it simple to understand by the workers, a face is created on a screen, as this is something humans understand very well and instinctively. This paper calculates the trust the human has in the robot and uses this to select trajectories and modify the speed of the robot. They use a face that changes dynamically to provide feedback to the worker, which can increase the trust the worker has in the robot. They show that with increased trust and usability of the robot, the worker has a significant drop in perceived workload, while the assembly time remains the same.

In a work presented by Quan Liu [12], a framework is created and combined with a set of other technologies. It uses perception, cognition, decision, execution and evolution to tackle the uncertainties and complexities in disassembly. The work shows that a multi-modal perception system, and sequence planning for a human-robot interaction disassembly task can be done at high feasibility and effectiveness.

In [13], a system is proposed to use gestures from the worker to start and stop the movement of the robot to allow the worker to feel safer and allow him to collaborate with the robot. They also have other gestures to allow the worker to more accurately control the robot. However, the robots are not collaborative by nature and are not compliant with the worker.

Another example of HRI is presented in [14], where the safety of the worker in the hybrid cell is a heavy focus. This means that collision avoidance is one of the focuses of the work, and thus use an implementation of potential fields combined with a constraint least-squares optimization, to make sure that the robot does not move into areas that are blocked. The objects including the worker's arms are tracked in a virtual environment, which is used for collision avoidance. Here the potential fields are used to repel the robot from the objects, while the constraint least-squares restricts the motion to safe sub-spaces of the collision avoidance.

Another work explored how two factors, namely extroversion and negative attitude towards the robot affected the assembly of an object. [15] They found that the more extroverted

people are, the longer they tend to talk to the robot, while people with a negative attitude towards robots, the more they look at the robots hands.

In [16], a human-robot hybrid cell was presented for flexible assembly. The work focused on assembling LEGO blocks into a final product. To do this, the task was divided into multiple subtasks. They found that an optimum number of subtasks is effective to maintain a high level of trust, which in turn led to a satisfactory performance of human-robot interactions and assembly performance.

Another work proposed a human-robot collaborative platform for disassembly in [17]. The work focused on car batteries because they can be partly disassembled by the robot, using just a powered screwdriver, while the human worker sharing the workspace would use different tools. Another way of creating a collaborative platform used for disassembly is proposed by [18]. In this work, the human worker teaches the robot how the disassembly of different parts is done by teaching things like how to cut a cable, unfasten screws and where to discard the different parts.

A skill-based approach can also be taken to teach the robot a complex task as seen in [19]. Here the previous work of the authors where an adaptive impedance controller, together with a defined skill formalism, show that the complexity of manipulation tasks is significantly reduced. One of the tasks tested on the system was the peg-in-hole task, which was achieved with sub-millimeter tolerances.

This chapter explored work done in the disassembly scene, focusing on the disassembly of screws and peg-in-hole. HRI was also explored both in assembly and disassembly.

3 Skill-Based System

The Skill-Based System is a system initially presented by Casper Schou. The system is built to make normal workers without previous robot experience able to program a robot, by using very high-level programming. [20]This combined with collaborative robots gives the opportunity to have fast change-overs, as the system does not need an operator to configure the robotic system to the new robot cell. This can be done by teaching the robot the positions instead. This can be done as SBS is made with workers teaching the robot in mind.

3.1 SBS Architecture

SBS is built on having a central state machine to control the different aspects of the SBS. A visualization of the architecture of SBS can be seen on Figure 3.1.

The central control node is surrounded by the task manager, the skill manager and the device manager. These make up the creation and execution of the tasks created. The device manager takes care of the device primitives, which controls the robot using simple commands, while the skill manager uses sequences of these primitives to achieve more complex robot movements. The task manager enables the user to create sequences of skills, as well as the teaching of these skills.

All of the tools of SBS are made available to the user through the GUI, which is used to supply the user with information on the current state of the robot as well as the creation of tasks. The UI manager takes care of the information coming from the robot through the ROS bridge. This is where the information used for the teaching of skills is gathered

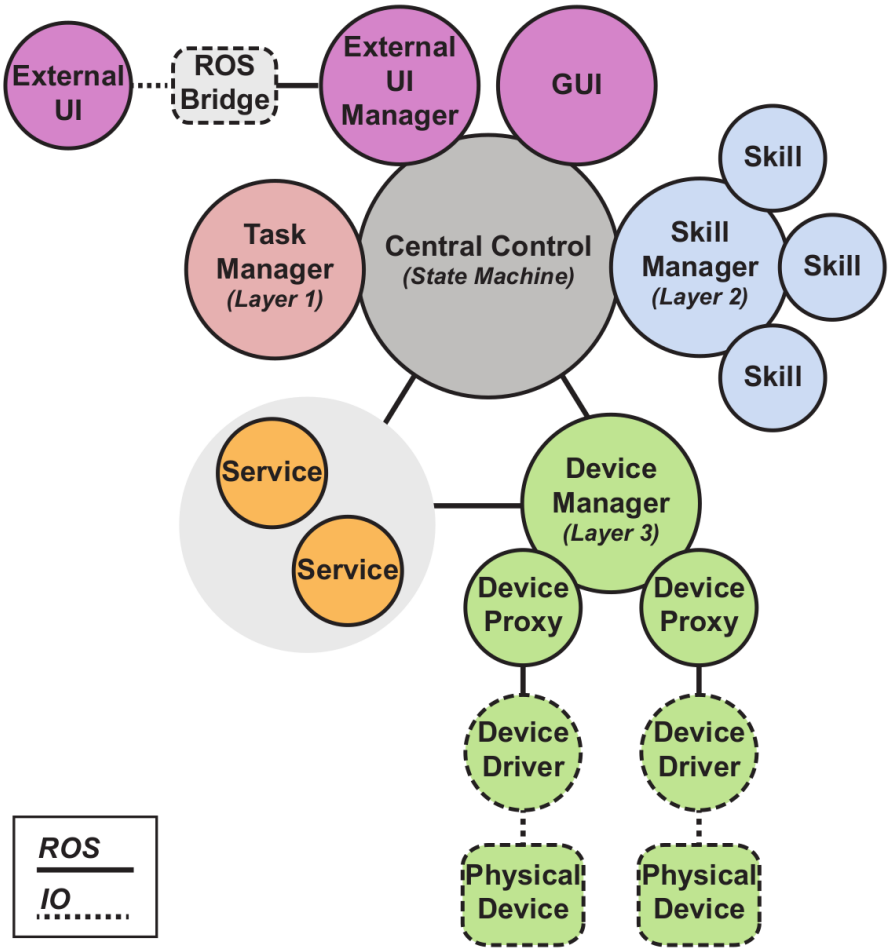


Figure 3.1: The SBS architecture as depicted in [1].

3.2 SBS programming

Programming the robot in SBS is done in a way that complex tasks can be created, and then through teaching, the parameters can be saved. By teaching the robot the positions of objects in the workspace, the setup of the robotic system can be done more easily. This removes the need for an operator to do low-level programming to configure the hardware and software to suit the new task. This is possible due to the SBS using three layers to program the robot; Tasks, skills and device primitives. The three layers can be seen on Figure 3.2.

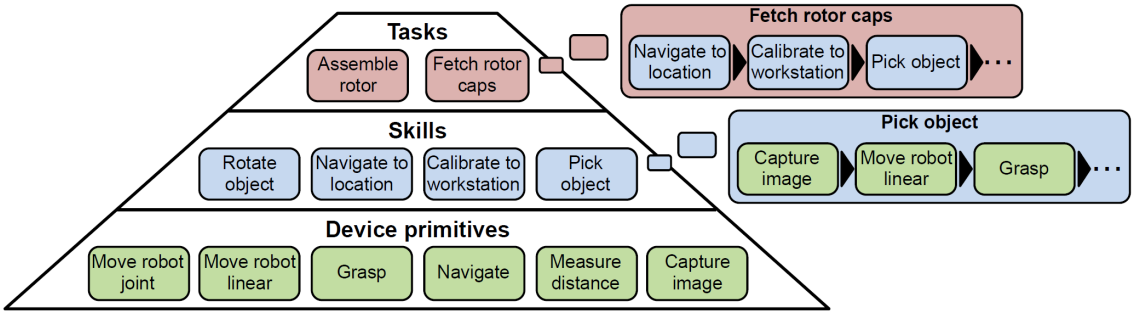


Figure 3.2: The three layers of programming used in the skill-based system, as depicted in [1].

Using this three-layer system, SBS can be set up by an operator with different tasks, skills and device primitives, enabling workers to intuitively pick the tasks that need to be done at the workstation. The skills will however need to be "taught" to the robot, as parts of the parameters needed for the execution of the skill, is not known to the robot e.g. the position of an object which needs to be picked up. Once the needed parameters are taught to the robot, the system can effectively work without a worker there, as the robot has been reprogrammed to the new workspace.

3.2.1 Tasks

The top layer of the SBS is what workers will use to program the robot. This layer consists of tasks, which are made up using a sequence of skills. Here a task could be to fetch an object, which is made up using several skills, such as "move" and "pick up".

The tasks of the SBS after being programmed by the concatenation of skills are then parameterized by utilizing the teaching functions of the skills, which is why these tasks are directly made to solve goals in the factory. As the task layer is made to be used by workers, it is not coupled with the hardware layer, where commands are given to the robot. The tasks created are stored in external files, where the skills and parameters are saved, which is used to be able to create, save, load and run the tasks. [1]

3.2.2 Skills

The middle layer of the SBS is the skill layer. These skills are made up of device primitives, which can be both sensing and manipulation primitives. The skills are created to do singular behaviours such as picking up or placing an object. The skills not only contains the primitives needed to perform the skill, but also the parameters to do so. The parameters are specified in a two-step process, the first being offline where simple parameters are picked. Once the robot is at the workspace, the second step is the online parameters, which are supplied by teaching. This is where end-effector positions can be stored and used for skill execution. [1]

3.2.3 Device Primitives

The lowest layer of the SBS programming is the device primitive layer. These are the different functions that the different connected devices can perform, such as moving the robot to a Cartesian or joint space position. The device primitives are called by the SBS using the device manager, which takes care of the connected devices. The SBS, when a primitive is needed, calls the device manager to find an available device with the given device primitive, after which it is called. The primitives are the lowest level of programming in SBS, where the device is programmed. These contain direct communication to the device, which is used to control it. For the KUKA LBR iiwa, an example of a device primitive is the "move_cart" primitive, which moves the end-effector of the robot to the given goal pose. [1]

3.3 Skill execution

A skill is executed in a way where there is a start state and a goal state. In-between these states, the first thing the skill does is the pre-condition check, which is the robot checking if all conditions are full-filled before the skill is executed. Once the pre-conditions have been checked, the skill is executed using the parameters that have either been taught to the robot or specified to the robot. This is to make programming the robot easier, as teaching the robot can be done by anyone, whereas giving specific parameters can only be done if you really know the robot. Once the skill has been executed, an evaluation is done to check whether the skill was executed in a correct way. This is done using a prediction of the desired goal. If the difference between the prediction and the evaluation is within the error margin, the skill is deemed successful.

The idea is that a task can be created offline, with some of the parameters specified before connecting to the robot, and teaching the last parameters online, as this is more intuitive to the workers that are going to use the robot. This also enables the worker to correct and help the robot, by keeping the "human-in-the-loop". [1]

3.4 Teaching skills

There are two parts to the parameterizing of skills, these being offline and online parameterizing. This is done so that the more specific parameters can be set up beforehand by more experienced operators, while the simpler parameters can be taught to the robot via online teaching by the workers at the workstation.

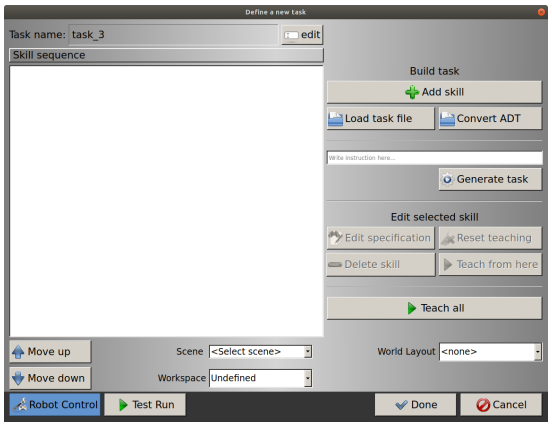
To explain the teaching of skills in the SBS, an example is made from the teaching of the "MoveTo" skill. The different parameters that can be specified or taught are the parameters for the "MoveTo" skill, which can be seen in Table 3.1.

	Name	Type	User Input	Description
x1	MoveFrameJoint	ParamVector	Teach	Target joint positions
x2	MoveFrameCartesian	ParamVector	Teach	Target EE pose
x3	Velocity	Double	Specify	Velocity
x4	FrameType	Char	Specify	Frame Type
x5	MotionType	Char	Specify	Motion Type
x6	OrientationLock	Boolean	Specify	Make tool orientation free
x7	MoveGripper	Char	Specify	Initial gripper position

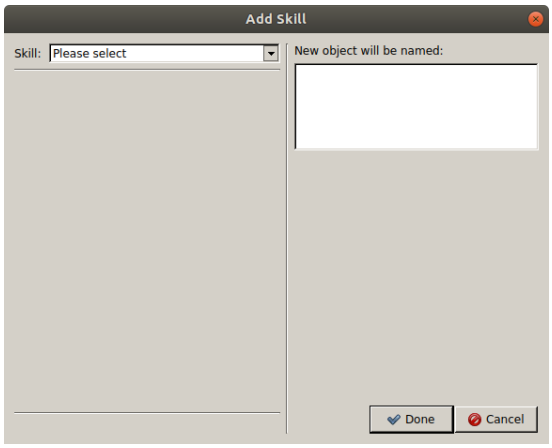
Table 3.1: Input parameters for the execution of a "MoveTo" skill

To use this skill in the SBS, the skill must first be chosen and added in the SBS task creation GUI, as shown on Figure 3.3.

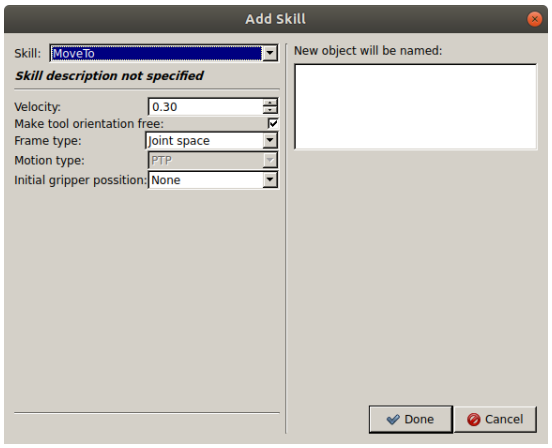
(a) Opening SBS GUI



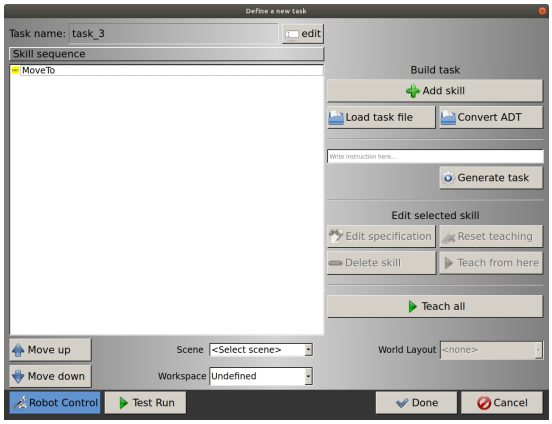
(b) The task creation GUI, from pressing "program" on the opening GUI.



(c) The AddSkill GUI, after pressing the Add skill button.



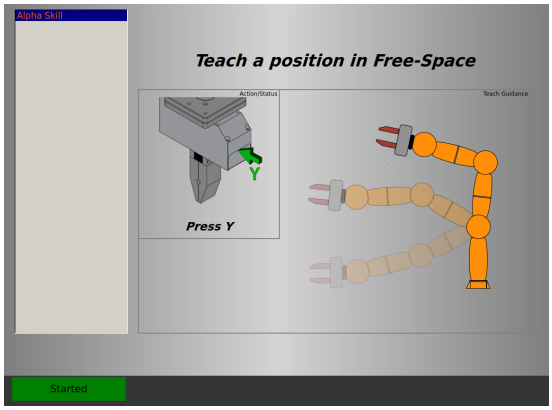
(d) The parameters specified for the MoveTo skill



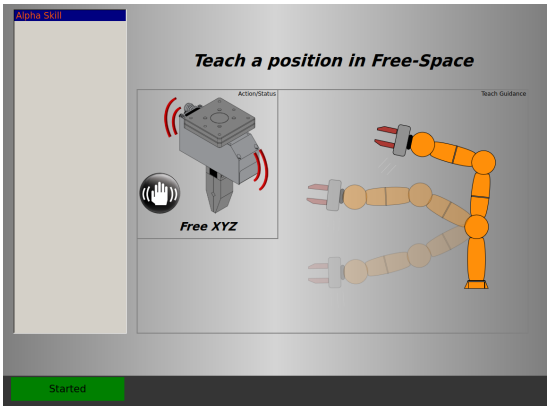
(e) Task creation GUI, showing the added skill in yellow, meaning it has yet to be fully parameterized.

Figure 3.3: Figures showing how to add a skill and begin teaching

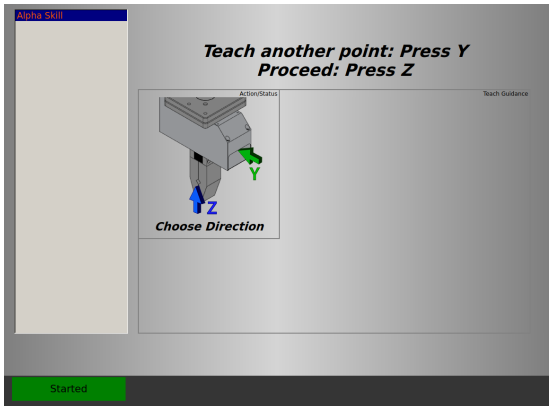
After the skill has been chosen and the offline parameters have been specified, as seen on Figure 3.3c and Figure 3.3d, the skill shows up with a yellow box next to the name of the skill, as seen on Figure 3.3e. This means that the skill has not been fully taught yet. To fully teach the "MoveTo" skill, the robot must be taught the target joint position or Cartesian frame, based on what was specified in Figure 3.3d. This is done by using the teaching interface, which is used to instruct the worker on how to teach the robot. The teaching interface used for the "MoveTo" skill can be seen on Figure 3.4.



(a) Teaching interface instructing the worker to press on the tool to teach the robot a position.



(b) Teaching interface showing the worker that the tool can be moved in free space.



(c) Teaching interface instructing the worker to either teach another point, or to stop the teaching.

Figure 3.4: Teaching interface used for the "MoveTo" skill.

The teaching interface allows the worker to know how to teach the robot, while also giving the worker feedback in some cases. In the case of the "MoveTo" skill example, the teaching interface first tells the worker to press on the robot end-effector to begin teaching the robot, as seen on Figure 3.4a, after which the interface informs the worker that the teaching has begun, as seen on Figure 3.4b. Lastly, once the robot is detected as stationary, the worker is given a choice between teaching more points or stopping, as seen on Figure 3.4c.

Through the teaching interface, instructions can be given to the worker, which enables the robot to be taught using online teaching. This makes it possible to place the robot into a new environment, and have the robot up and running, without having an expert set up the system.

3.5 Device Manager

To control the different devices connected to the system, SBS uses a device manager, which keeps track of what devices are connected, as well as what different device primitives the different devices are capable of using, such as "move_cart".

To keep track of the connected devices the device manager uses an advertiser class, which is built to interface with the manager. This advertiser uses a "heartbeat" ROS-publisher to make sure that the device is connected. In this way the device manager knows if a device has an error because the device's "heartbeat" will stop if the device crashes in any way.

The device manager uses device config files in the form of XML files, as well as device proxy CPP libraries to know which devices are available to be connected, as well as what functions are available from the different devices.

In this file, information can be found on what the device is, its name, what ROS-node is used to manage the device, and lastly what functions are available for the device.

3.5.1 Creating a device

To create a new device for the device manager, the first thing that must be done is to create a config file, together with a proxy library to be used by the device manager.

Config file

To create a new device the first thing that needs to be created is the device config file. In this config file, the name, class and type are required for the device manager to know what type of device it is.

The config file contains four tags that are used by the device manager. These four tags are identification, driver, physical and functions. The identification tag contains the basic information about the device, which is the name, class, type etc. As an example, these could be "KUKA_iiwa", "arm", and "articulated". The driver tag contains information about what ROS node is used to launch the device. The physical tag contains the physical properties of the device, which could be the weight. Lastly, the functions tag contains the device primitives that the device can perform. This is also where the required parameters for the device primitives are specified.

Device proxy

Once the config file is created, the next step is to create a device proxy, which the device manager uses as a library to connect to the device driver. This proxy library is used to create the ROS publishers, subscribers and service clients that control the robot.

The proxy library is used to program the device primitives for the device. This is done by turning the library into a ROS service which is called by the SBS when executing skills or tasks. This means that once a command is sent from the SBS, a request is received containing the device primitive needed, as well as the required parameters.

This chapter explored how the SBS is built, as well as how the tasks and skills are created and taught. How the device manager is used to control the devices connected to the SBS was also explored, as well as how to create new devices to connect to the SBS.

4 Information Flow

To allow the system to accomplish the task of unfastening screws, a robot and a tool are needed, this chapter explores the different devices used for the project, as well as how these devices communicate and are controlled.

4.1 System Architecture

As different forms of communication are needed for the different devices used in the project, these forms of communication are explored. The devices used for the project is the KUKA LBR iiwa, an Arduino Mega 2560, and a powered screwdriver.

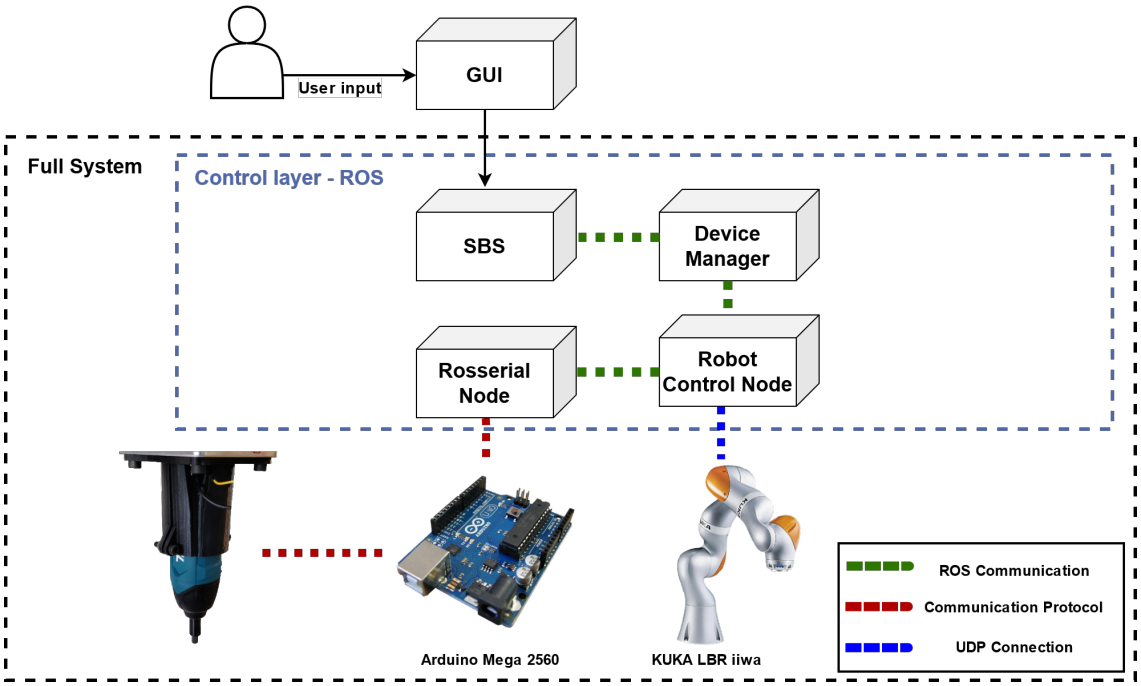


Figure 4.1: A simplified version of the system architecture.

As shown on Figure 4.1, the KUKA LBR iiwa and Arduino Mega 2560 are controlled using separate ROS nodes, which are explained in section 4.3 and section 4.4 respectively.

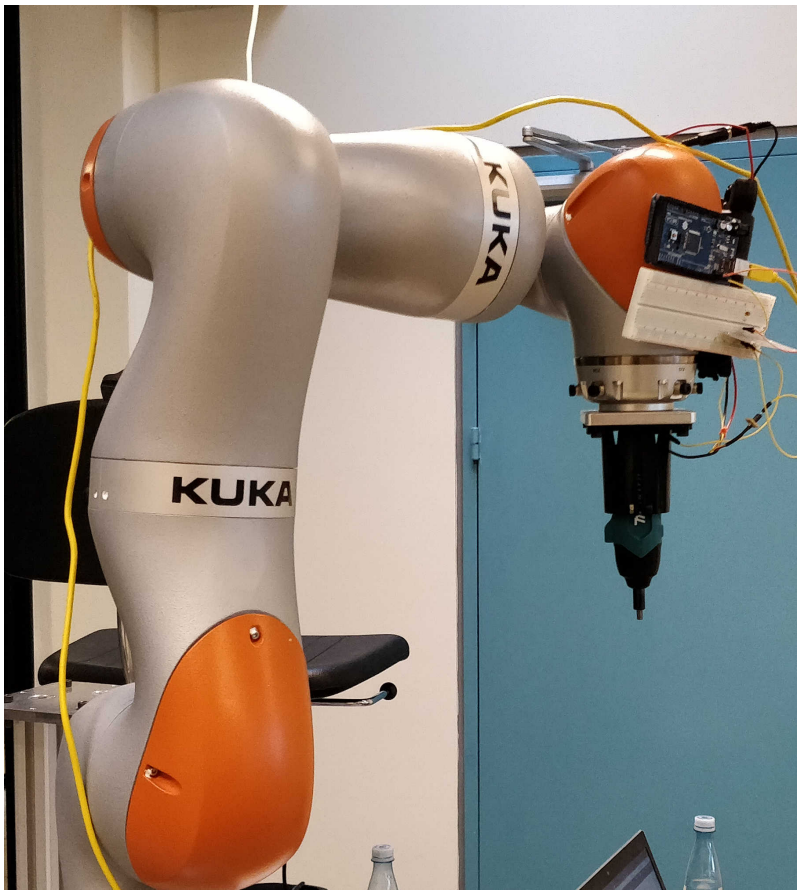


Figure 4.2: Picture showing the robot with the tool attached to the end-effector.

As seen in Figure 4.2, the KUKA LBR iiwa is equipped with a screwdriver, which is controlled using an Arduino Mega 2560. The screwdriver uses a hex bit, which is used for the unscrewing skill. The tool is created using a 3D printed fixture to hold the screwdriver in place, which is explained in section 4.2.

4.2 Screwdriver

For [5] a tool was created by Technicon to be attached to the robot. However, due to issues mentioned in that report, a different tool is used for the unfastening of the screws.

To attach the screwdriver to the robot, a 3D print is used to keep the screwdriver in place. A metal plate connector is utilised to make sure that the 3D print can be attached to the robot. This metal plate has screw holes in the four corners to be used for the 3D print. The dimensions for these screw holes can be seen on Figure 4.3, together with the bottom of the 3D print.

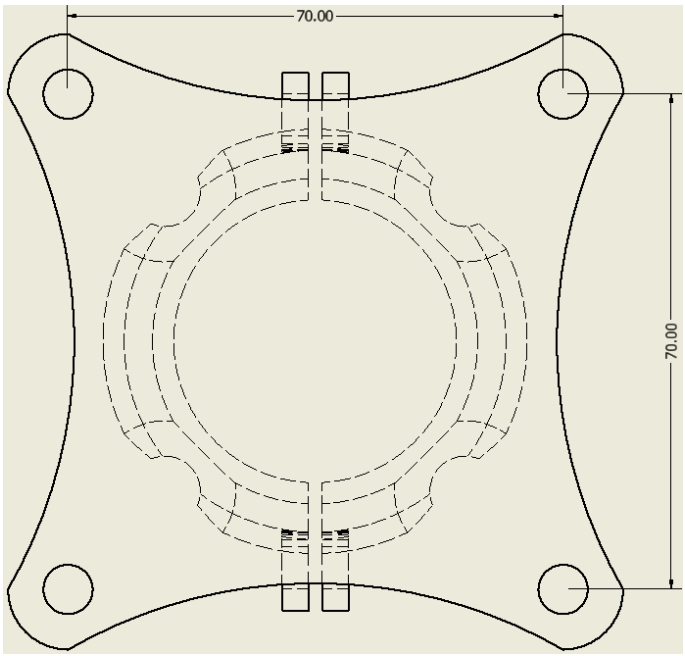
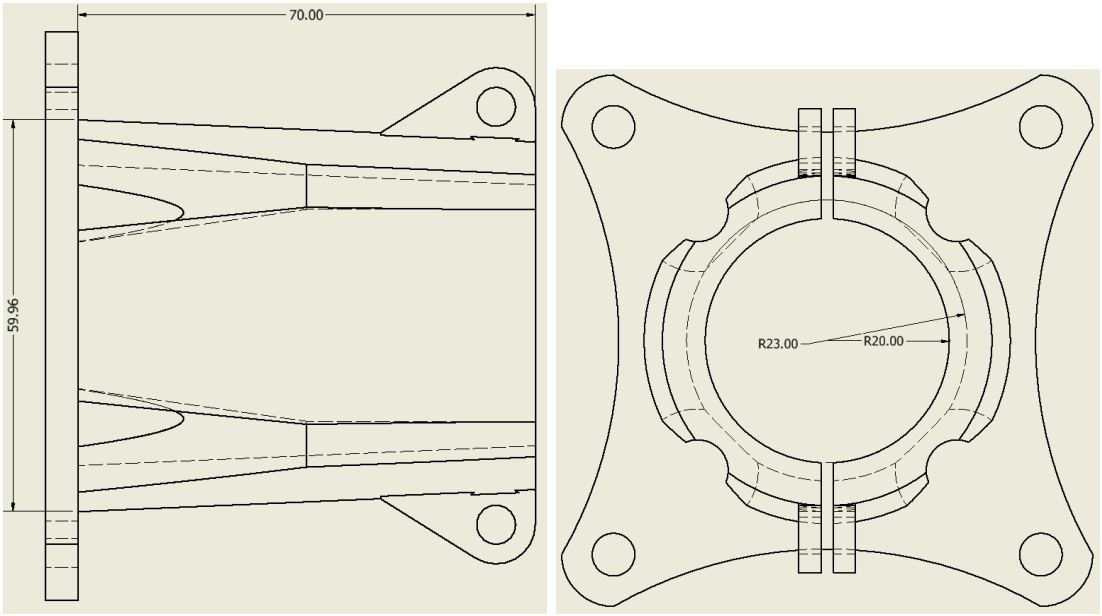


Figure 4.3: Drawing of the CAD model seen from the bottom, showing the dimensions used.

For the screwdriver to be held in place by the 3D print, the dimensions of the screwdriver are taken and used to create the tube for the screwdriver, as seen on Figure 4.4.

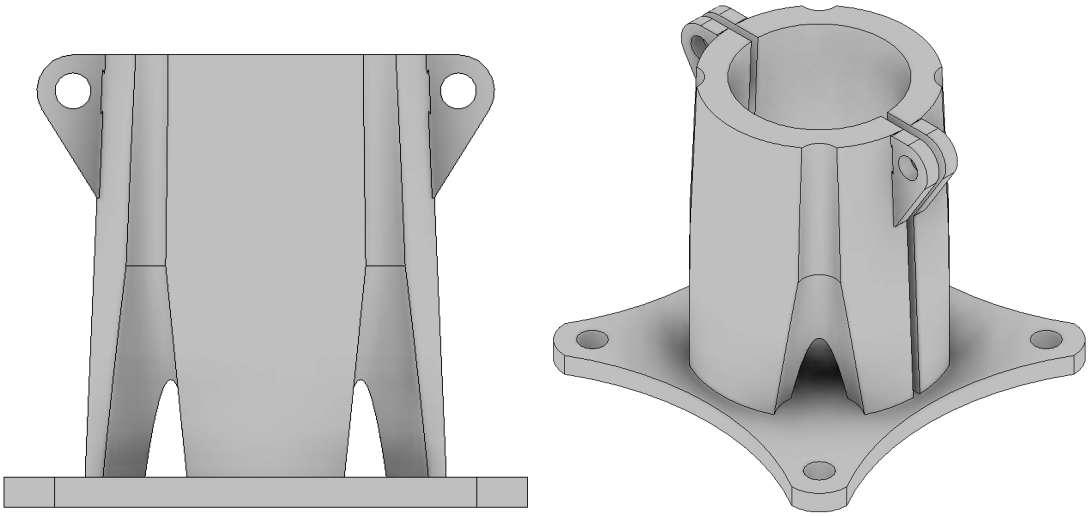


(a) Drawing of the CAD model seen from the side, showing the dimensions used for the tube.

(b) Drawing of the CAD model seen from the top, showing the dimensions used for the tube.

Figure 4.4: CAD drawings showing the dimensions used to attach the screwdriver

After the dimensions for the metal plate connector and the screwdriver are known, the next step is to create the final CAD model for the 3D print. To do this, the bottom and tube seen on Figure 4.3 and Figure 4.4, are put together. However, as the screwdriver needs wires to control it, four holes are added around the tube to allow for the wires to be put through. The final CAD design can be seen on Figure 4.5.



(a) Final CAD model seen from the side. *(b) Final CAD model seen from the top right*

Figure 4.5: Final CAD design seen from multiple angles.

As can be seen on Figure 4.5, extra screw holes are added to the tube to allow the screwdriver to be fastened to the 3D print.

The final CAD design allows the screwdriver to be held in place by the 3D print while allowing the wires needed to control the screwdriver to go through the bottom of the 3D print.

4.3 Arduino Mega 2560

The Arduino Mega 2560 is a microcontroller, built by Arduino for easy and low-cost prototyping with sensors and actuators. The Arduino is controlled using the Arduino programming language, which is a language based on wiring, which utilizes the IO pins on the Arduino Mega 2560 (Seen on Figure 4.6). This section explains how the Arduino is used to control the screwdriver used to unfasten screws.

Figure 4.6: Arduino Mega 2560

Programming the Arduino is done using the Arduino IDE which was made for programming and uploading the software to the Arduino board. To start programming the Arduino, the first thing to do is to set up the pins that are going to be used in the setup function. This sets the different pins to either input or output. While all pins can output a digital signal (5v or 0v), some of the pins on the board are set up to be able to read an analog signal.

In the setup function, a pin is set up for either input or output for runtime. This is also where the baud rate is set up for communication with the board. The board can communicate in two ways, one being the USB connection, and the other being the RX/TX pins, which are built for communication with other devices. This means that the Arduino can receive information from the external PC, and then act accordingly using the output pins. The ROS communication used for the project is also initialized in the setup function, to allow the Arduino to communicate through ROS.

4.3.1 Rosserial-arduino

The rosserial library for Arduino built by Joshua Frank [21], uses the USB connection to send and receive ROS messages to and from the Arduino. It is made as a wrapper for the standard ROS serialized messages, making it possible to integrate the Arduino into a ROS framework.

As shown earlier, the ROS node, subscribers and publishers are initiated in the setup function. This is where the external PC figures out what to send and receive from the Arduino. This is done on the external PC by launching the rosserial-arduino ROS node, with the USB port name as an argument. This creates a node running on the external PC, which converts and sends the related messages to the Arduino. This is useful as the screwdriver used for the unfastening of the screws can be controlled by the Arduino, meaning that a control signal can be sent through ROS. The control signal used for the Arduino is a callback function.

The first thing to creating a callback function on the Arduino is to create a ROS subscriber, by giving the topic name, and the callback function as seen below.

```
ros::Subscriber<std_msgs::String> sub("Activate_tool", messageCb );
```

As can be seen above, the subscriber receives a string, which is used in the callback function. Once the subscriber is created and initiated in the setup function, the callback function is called every time a ROS spin is called, which in this case is called using the "spinOnce" function.

The "spinOnce" function is needed due to how callback functions work in ROS. Callback functions are not automatically called every time a message is received on the topic, instead the message is put into a queue. The "spinOnce" function then calls all the queued callback functions. This means that the callback function for controlling the screwdriver is called if a message has been sent.

If the callback function is called, the data contained in the message is used to control the screwdriver. If the message is an "a", that means that the screwdriver is to be activated,

and the output pin is set to "HIGH". After the screwdriver is activated, the Arduino sends back the pin output. How this is used to control the screwdriver is explained in the next section.

4.3.2 Screwdriver control

The screwdriver works by utilizing a 12v battery, which can be charged using a USB cable. As the screwdriver essentially is just a DC motor, which rotates in the direction of the current, this can be controlled using a MOSFET as seen on Figure 4.7.

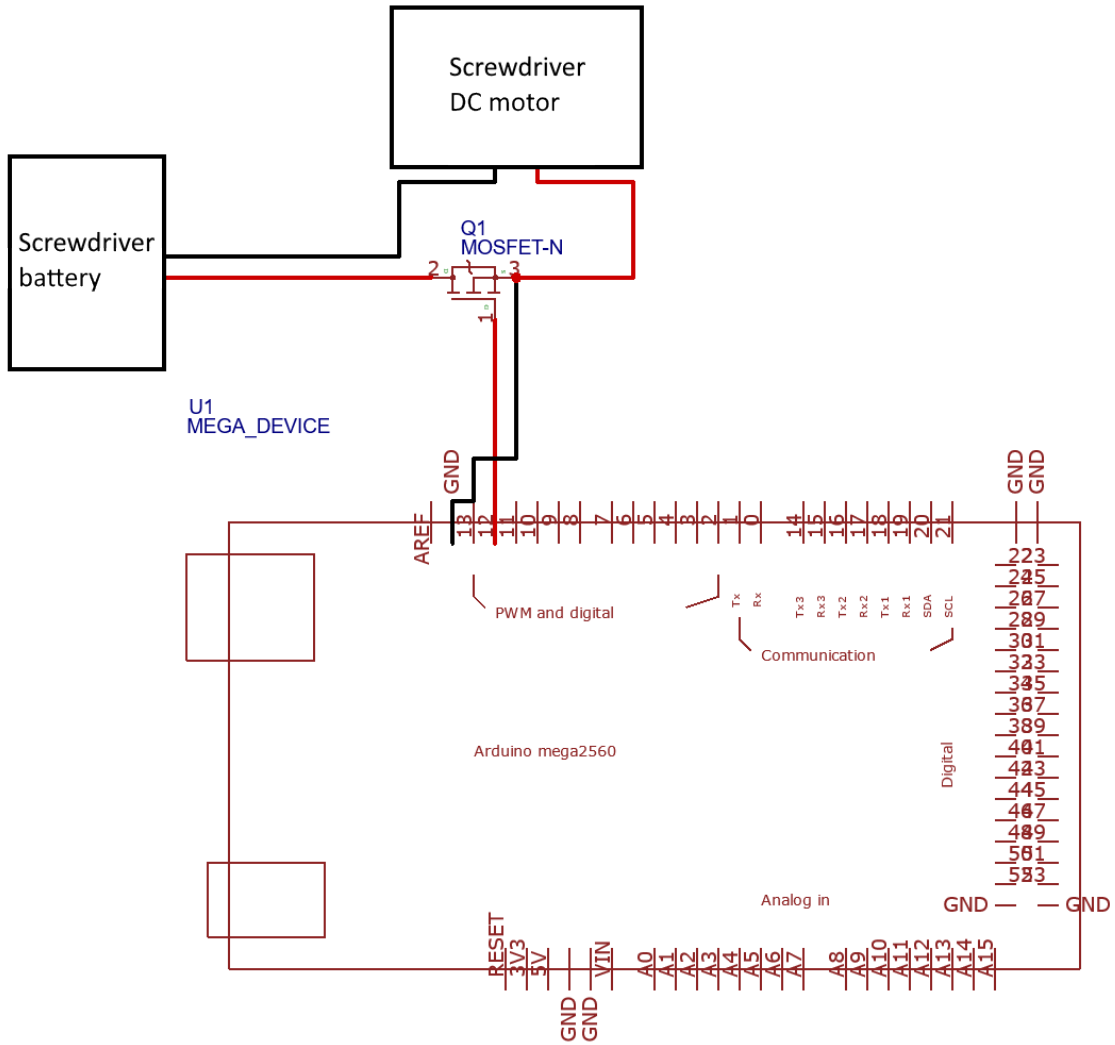


Figure 4.7: Arduino with connections to screwdriver

The figure shows the Arduino using the MOSFET as a switch to control the current from the battery. This means that if a 5v signal is given on pin 12, the current flows through the MOSFET powering the DC motor.

With the MOSFET combined with the rosserial library, the screwdriver can be controlled through ROS. This means that the external PC can easily activate the screwdriver, and thus be used directly by SBS, if the unscrewing device primitive is called (explained in subsection 5.1.3).

4.4 KUKA LBR iiwa

The KUKA LBR iiwa is controlled using the Sunrise cabinet/controller, which uses KUKA's collection of Java packages to control the robot. One package that can be used to control the robot is the FRI (Fast Research Interface) package. Other ways of controlling the robot are through direct connections to the sunrise controller, through either a TCP/IP or a UDP connection.

FRI was initially chosen as the way to control the robot. However, during initial testing, the connection proved to not be able to send anything other than movement commands. This made it impossible to use as a stand-alone connection, as different impedance control stiffnesses is required for the project. Due to this, a different way of controlling the robot is explored, this being the UDP connection.

4.4.1 UDP Connection

The UDP (User Datagram Protocol) connection is a connection between devices, which is often used for time-sensitive applications. For this reason, it is often used for video playback. This is due to the connection not needing a confirmation message from the receiver, which enables the source to send data as fast as possible.

The connection, not needing a handshake, makes it faster than a TCP/IP (Transmission Control Protocol) connection, which was utilized by the KST (KUKA Sunrise Toolbox) used in [5]. As a TCP/IP connection also requires a confirmation message to be sent for every command, this slows down the control frequency. As a faster control frequency is needed for some applications of robotics, the TCP/IP connection is not fast enough, as a test done by the author of the KST show that each message has a delay of 3-4 ms.[22]

This means that the UDP connection is a better choice, as the control frequency can increase. Another advantage of using a UDP connection is that when sending a message, only the receivers address needs to be specified, making sending messages easier. The same is true for the receiving end, as only the address and port are needed to create a socket.

For the KUKA LBR iiwa, the robot is programmed using the KUKA Sunrise IDE, which uses java to control the robot. To create a UDP connection in java, the java.net packages are used. For the UDP connection, the DatagramSocket and DatagramPacket are used to create the socket and send and receive packets from the external PC.

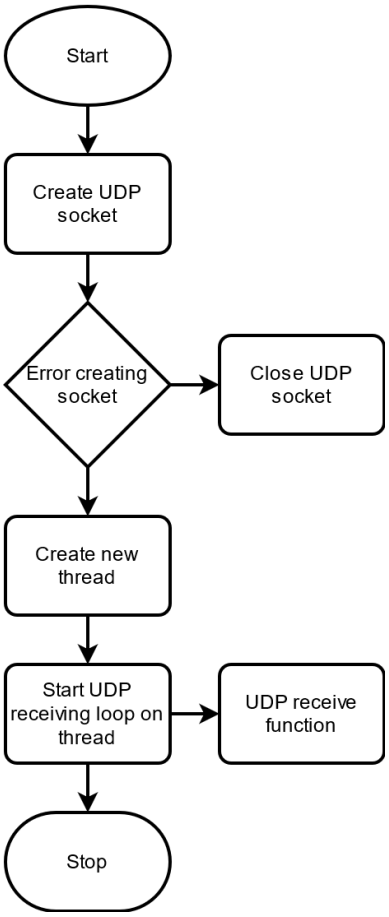


Figure 4.8: Flowchart of the robot UDP connection

As on Figure 4.8, the controller creates a UDP socket, after which a thread is created to manage the received packets.

Once the socket is created, the robot controller waits for a packet to arrive from the external PC. This is done to create a connection from the controller to the external PC. After the packet has been received, the controller sends back a message to let the external PC know that the connection has been established. This also allows the controller to save the IP address of the external PC, which is used to send the current state of the robot to the external PC.

4.4.2 Threading for receiving

To allow the robot to be controlled while receiving commands at the same time, a thread for receiving packets is created. This allows the controller to control the robot while also receiving commands from the external PC. This is done inheriting the Runnable class in java.

Once the thread is created, the thread enters a loop, which checks the received packets for commands and the parameters from the external PC. This enables the robot to run in Cartesian impedance control mode for the entire duration of the robot application, while the receiving thread can change the stiffness parameters or give a new desired goal pose.

4.4.3 Receiving state of the robot

Because the SBS needs the state of the robot for workers to be able to teach the robot skills, the state has to be sent to the external PC.

The state is read from the robot sensors, after which the state is formatted in a specific way before being sent to the external PC.

The message is formatted as a string containing a keyword, followed by the state associated with the keyword separated using a ",". The separation is done so that when the string is decoded, it can be easily split on the receiving end. There is one exception to this, however, which is the last parameter in the message. This parameter tells the external PC whether the robot is currently in impedance control mode.

Once the message is created, it is converted into bytes, after which a packet containing the message, the length of the message, the receiving IP address and the receiving port. After the packet is created, the packet is sent through the UDP socket.

4.4.4 Sending commands to the robot

To control the robot, the controller must be able to receive, decode and check the packets from the external PC.

The command messages are formatted in a similar way as the robot state, in the same way, that the state is formatted, with the command as the first keyword, followed by the parameters for the command.

Once the packet is received from the external PC, the packet is decoded into a string. After the packet is decoded, the string is split, converting the string into a list of strings. In this way, the first element of the list is the command for the robot. An example of a command is the "cartImpLIN" command, which puts the robot into Cartesian impedance control using the SmartServoLIN package. The parameters for the impedance control command is the different stiffness parameters for the robot.

An example of the impedance control command message is seen below.

"cartImpLIN,1000,1000,200,5,150,150,"

How this is used to control the robot is explained further in section 4.5.

4.5 Pipeline from SBS to KUKA

To control the KUKA LBR iiwa using the SBS, a pipeline is created to convert and send the SBS commands to the robot. This is done through the use of a "bridge", which is created in python to interface with the robot using the UDP connection, and with the SBS using ROS.

4.5.1 SBS to python

Commands from the SBS are sent using a ROS service call, with a "FcnCall" as a request. This "FcnCall" contains the command, and a parameter handler, which contains all the

required parameters for the command. The required parameters are specified in the device config file (Explained in subsection 3.5.1).

As explained in chapter 3, commands from the SBS go through the device manager, using the device proxy created for each device. The proxy used for the project converts the SBS commands into ROS messages, which are sent to the python bridge. Once the command is received by the device manager, the proxy "execFcn" function is called as a ROS service. Here, the command is checked using the command string stored in the "FcnCall", after which the appropriate device primitive is called using the given parameters. In the case of a "move_cart" device primitive, the given parameters are the Cartesian position of the goal position in meters and the velocity for the robot. After the primitive is called, the proxy generates a ROS message to be sent to the python bridge. This message contains the command for the robot and the parameters, which are used to call the ROS service created to control the robot.

The ROS service checks the command and calls the device primitive needed to control the robot. For the "move_cart" primitive, the Cartesian position is converted from meters to millimeters due to the robot using millimeters for the goal positions.

4.5.2 Python to KUKA

Once the service request is received from the SBS device manager and the device primitive is called, the first step is to convert the SBS command into the UDP packet sent to the robot.

With the service request received, the SBS command is converted into a string, containing the command followed by the parameters for the robot separated using a ",". After the string is generated containing the command for the robot, the string is converted to bytes after which it is sent to the robot controller using the UDP connection. When the controller receives the message from the python bridge, the message is converted into a string, after which the string is split using the "," separator. After the string is split, the first element in the list, the command, is used to decide which primitive should be used by the robot. After the command is read, the parameters are read by converting the string into a double to be used during the execution of the primitive. In the case of the "move_cart" primitive being called, the parameters are used to generate a "Frame", which is the KUKA LBR iiwa's way of representing Cartesian positions. A Frame consists of the Cartesian position in millimeters, and the orientation of the end-effector in Euler angles. After the Frame is given to the robot controller, the controller uses inverse kinematics to calculate the goal joint positions and create a trajectory for the robot. This is done using the SmartServoLIN class, which is explained in section 5.3.

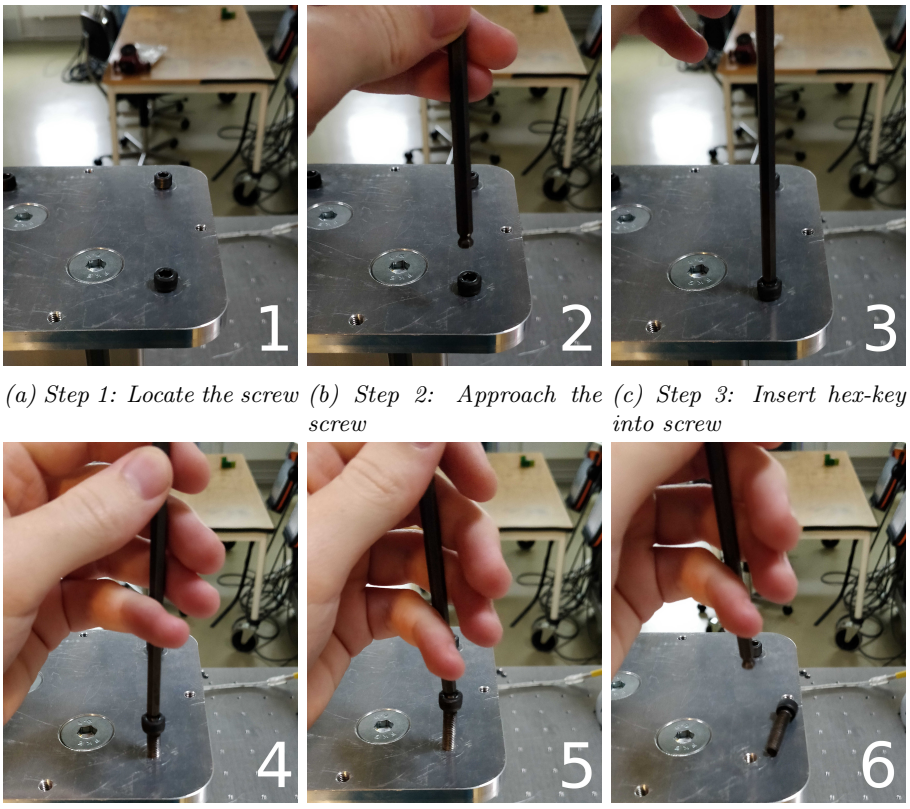
This chapter explored the system architecture, as well as the use of an Arduino for controlling the screwdriver, as well as how the Arduino is controlled using the "rosserial" package. After that, the connection to the KUKA LBR iiwa was explained, as well as why the UDP connection was chosen. Lastly, the connection between SBS and the robot was explored.

5 Unscrewing Task Creation

To make the robot accomplish the task of unfastening screws, the unscrewing task itself has to be created. This chapter explores how the unscrewing task is created and taught the positions of screws. Lastly, the Sunrise servoing package is explored, as this is used for impedance control.

5.1 Unscrewing Task

As an unscrewing task is a complex task, it is split into smaller tasks that a robot can execute. This also enables each part of the task to be parameterized instead of being hardcoded. Because the parameterization of the task is important for the SBS integration and the teaching of the robot, the parameters for each device primitive are explored.



(a) Step 1: Locate the screw *(b) Step 2: Approach the screw* *(c) Step 3: Insert hex-key into screw*

(d) Step 4: Unfasten the screw *(e) Step 5: Detect the unfastening of the screw* *(f) Step 6: Remove the tool from the screw*

Figure 5.1: Pictures showing the steps to unfastening a screw.

For the robot to achieve the unfastening of a screw, the task must be split into smaller and simpler skills or device primitives. For this reason, each part of screwing a screw is explored. As seen on Figure 5.1, the steps to unfastening a screw are to first approach the screw with the tool, then insert the tool. Once the tool is inserted into the screw, the screw is unfastened.

While unfastening a screw is a simple task for humans, there are parts of the task that is a very difficult task for a robot. The parts in question here is step 3 and 5, the first being a re-occurring research topic called the peg-in-hole problem. The second part is the detection of the unfastening, which is explored in subsection 5.1.3. For this project, however, due to knowing the position of the screw beforehand, this task becomes simpler.

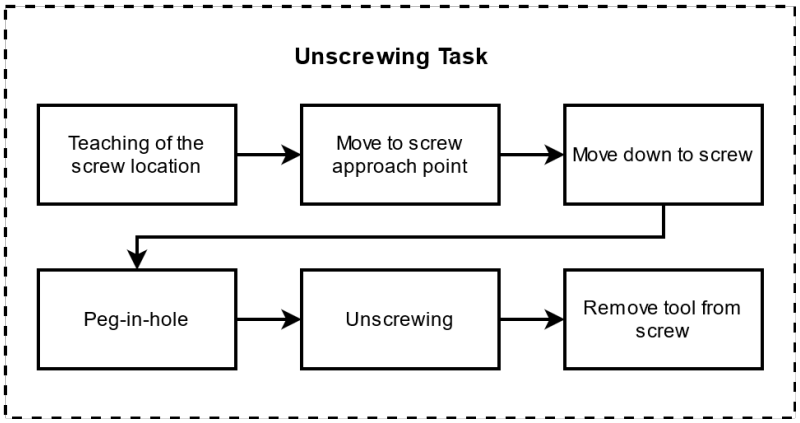


Figure 5.2: Figure showing the approach taken to unfasten screws, starting from the teaching of the screw locations and ending at the removal of the tool from the screw.

For this project, the unscrewing task is split into 5 separate skills, or device primitives, which can be seen on Figure 5.2. The figure shows that the task requires the screw positions to be taught to the robot before the task can be executed. Once the screw positions are saved, the positions are used to move the end-effector of the robot to the approach point for each screw. This approach point is set to a static height directly above the screw (Explained in subsection 5.1.1). Once the end-effector has been moved to the approach point, the robot starts to move the end-effector downwards until a contact force is detected. Once a contact force is detected, the end-effector is expected to be in contact with the screw, which is where the simple peg-in-hole device primitive is used (explained further in subsection 5.1.2). Once the end-effector is inserted into the screw, the unfastening is done by activating the screwdriver (Explained in subsection 5.1.3). After the unfastening has been done, the last step is to remove the end-effector from the screw (Explained in subsection 5.1.4).

5.1.1 Tool contact search

To ensure that the peg-in-hole device primitive is as simple as possible, a few steps have to be done, the first being knowing the Cartesian position of the screw. The next step is the tool contact search device primitive.

The tool contact search makes the robot move the end-effector downwards at a constant pace, while the tools external upwards force is being monitored. A flowchart of the primitive can be seen on Figure 5.3.

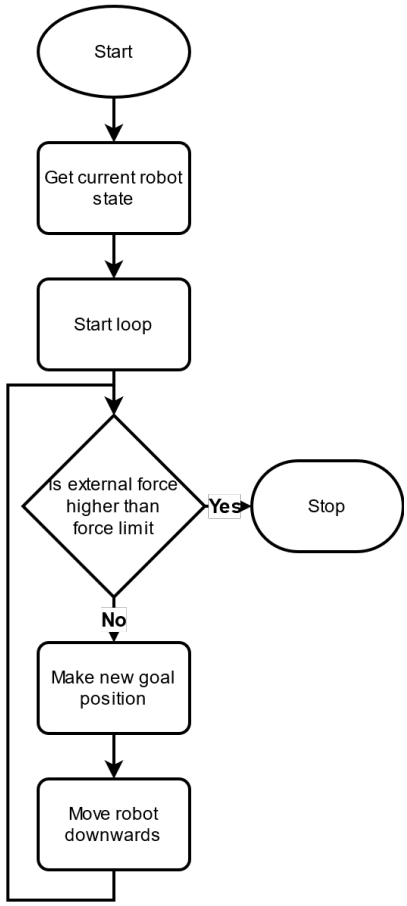


Figure 5.3: Flowchart of the tool contact search device primitive.

As seen in the flowchart, the robot moves downwards from the current robot state, until the external force has exceeded the force limit. To do this, the goal position is used to generate a control message for the robot, after which it is sent to the robot controller. Once the external force larger than the force limit, the robot is expected to be in contact with either the table or the screw.

To make this device primitive more flexible with different situations, the force limit and speed of the robot can be parameterized using the SBS framework.

5.1.2 Peg-in-Hole

The peg-in-hole task itself is done in a simple manner. The first step to performing this task is getting the tool itself into a favorable position. As this is done using the tool contact search, the current step is the tool insertion.

As the task is to insert a tool into a screw, some information about the screw must be known beforehand. In this case, the needed information is the type of screw, which in this case is a hexagonal screw. Being a hexagonal screw means that if the tool is rotated 30 degrees in each direction, the tool can if the position is correct, be inserted no matter

what orientation the screw is in. After the tool is inserted into the screw, the robot must be able to detect if the tool is correctly inserted. This detection is done by looking at the external torque being applied to the tool. This is done in roughly the same way as [5].

While the peg-in-hole device primitive is mostly hardcoded, the tool rotation margin during the search can be changed depending on the type of screw. Lastly, while the robot can detect if it has correctly inserted the tool into the screw, there can still be errors resulting in the robot not inserting the tool into the screw. To make sure that this does not happen during the execution of the unscrewing skill, a recovery/error state is created. A flowchart of the device primitive can be seen on Figure 5.4.

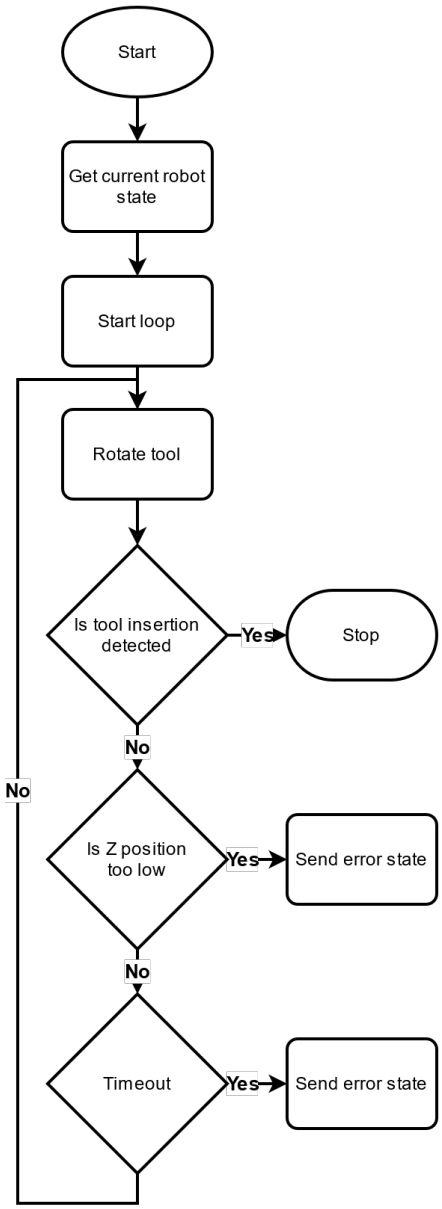


Figure 5.4: Flowchart of the peg-in-hole device primitive.

As seen on the flowchart, the error state depends on the tool z-position or a timeout. The limiting position on z is calculated based on the initial z position and the height of the screw, which enables the robot to detect if the tool falls off the screw. The timeout is

added due to the tool sometimes being stuck on the screw.

5.1.3 Unfastening

The unfastening of the screw is a simple part of the execution once the tool has been inserted into the screw. This is due to the unfastening, essentially, being about turning on the screwdriver (Explained in section 4.3).

However, for the purpose of testing the system on different length screws, some form of detection is needed for the completion of the unscrewing. For this reason, some data is collected during the unfastening device primitive. The data collected for the detection is; The wrench on the z-axis, the force on the z-axis and the position on the z-axis.

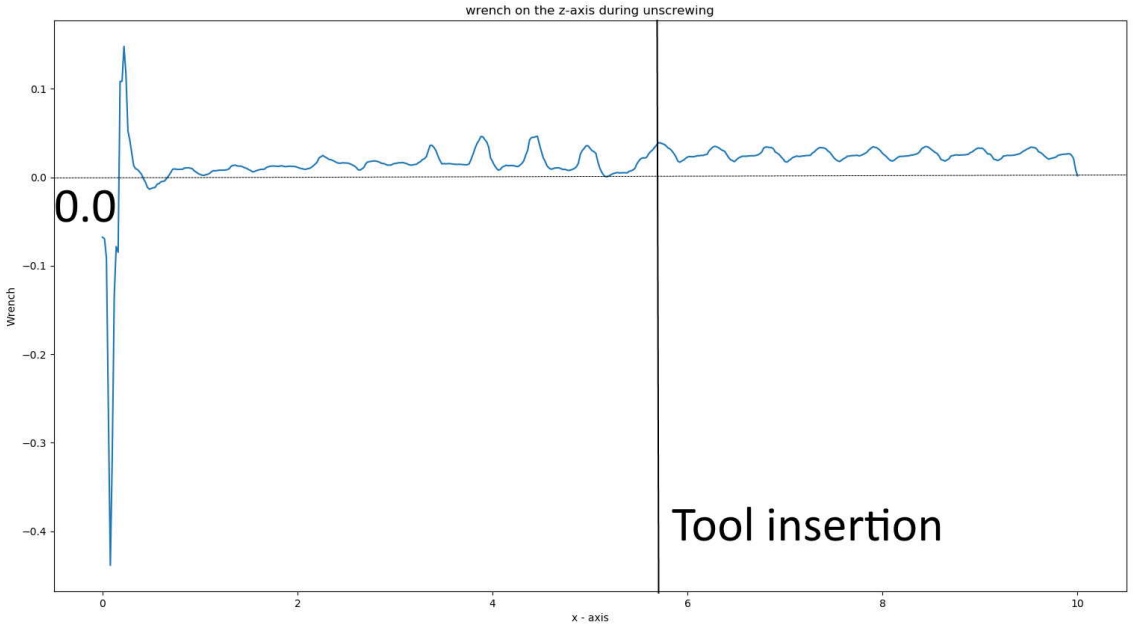


Figure 5.5: Graph with time on x and the external wrench measured on the z-axis on y.

As seen on Figure 5.5, unfastening the screw fully shows a small oscillating pattern. However, since the wrench is higher and lower at points during the unfastening process, other ways of detecting the unfastening are considered. These other ways being the external force on the z and the position on z.

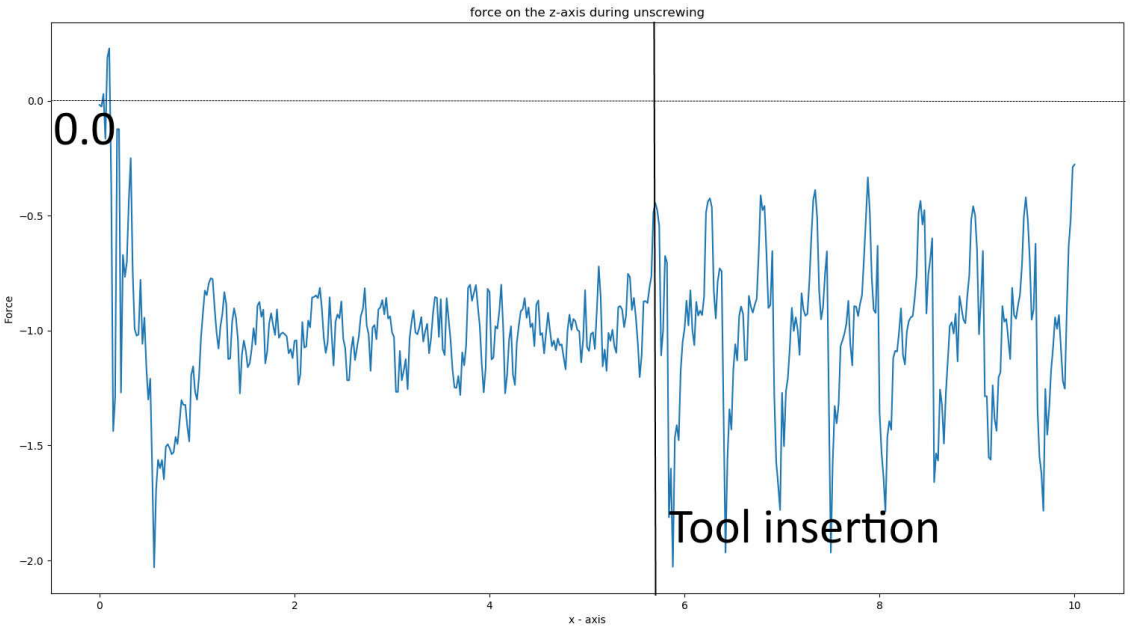


Figure 5.6: Graph with time on the x-axis and the external force measured on the z-axis on the y-axis.

Firstly, Figure 5.6 shows that after the screw has been fully unfastened, an oscillation starts, which can be used to detect the unfastening of the screw, as it is a consistent pattern on the graph.

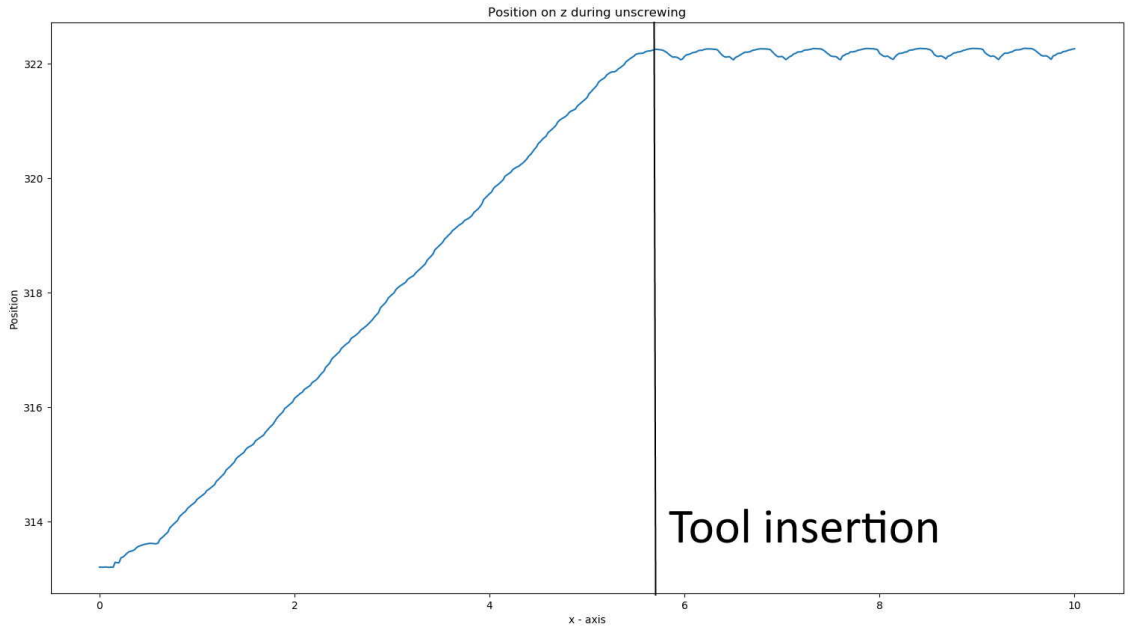


Figure 5.7: Graph with time on the x-axis and the tool z position on the y-axis.

Lastly, Figure 5.7 shows that the tool position rises steadily during the unfastening, while the position starts oscillating once the screw is fully unfastened.

The detection method chosen for the project is to use the position on z, where a simple detection can be created. This detection is done by saving the highest point reached on the z-axis and monitoring how much the position drops. If the drop is over the chosen

threshold, a counter is bumped and a timer is started. If this occurs two times within a second, the screw is deemed unfastened. A flowchart of how the full primitive is done can be seen on Figure 5.8.

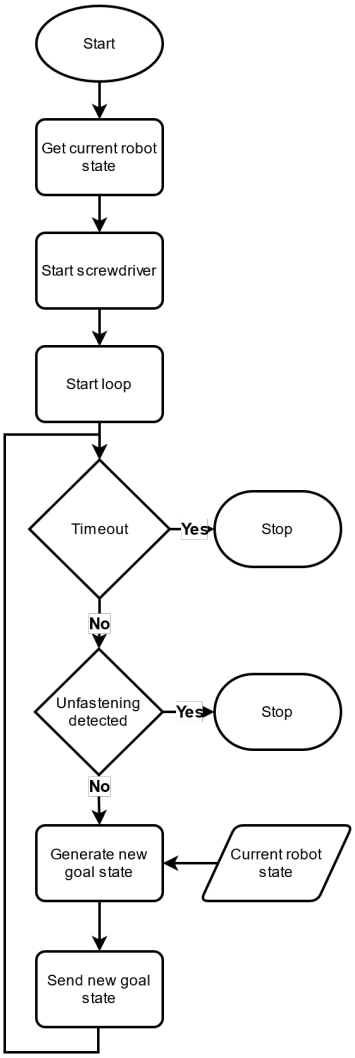


Figure 5.8: Flowchart of the unfastening device primitive.

As seen on the flowchart, a goal pose is also sent to the robot during the unfastening of the screw. This is done to ensure a constant downward force instead of an increasing force.

With this detection, the unfastening can be detected, which is shown in subsection 6.1.3. This enables the robot to unfasten screw due to the unfastening being detected more efficiently.

5.1.4 Tool removal

The last step of unfastening a screw is to remove the tool from the screw. This step, while it seems simple, can be challenging.

The challenging part of removing the tool from the screw is that the tool can often get stuck in the screw when rotating. While this step is simple once the screw is fully unfastened, it can damage the tool if the screw is not unfastened. Because of this issue, a device primitive for removing the tool from the screw is created. A flowchart of the primitive can be seen on Figure 5.9.

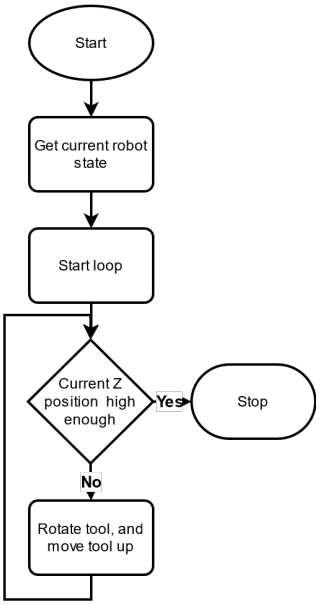


Figure 5.9: Flowchart of the tool removal device primitive.

As seen on the flowchart, the primitive tries to remove the tool from the screw until a predefined Z position is reached. This is done to ensure that the robot can move unobstructed to the approach point of the next screw. It can also be seen that the tool is rotated as it is moved upwards. This is due to the tool's tendency to get stuck in the screw during the unscrewing or tool insertion.

This enables the tool to be removed from the screw, in the case of the screw not being fully unfastened or not at all.

5.2 Teaching the skill

To unfasten screws while not using vision for detecting the screw positions, the positions must be supplied to the robot. This is done through the SBS teaching interface (Explained in section 3.4).

As seen in Table 5.1, there are 8 parameters needed for the execution of the skill. Of these, 6 of the parameters are specified when creating the task for the robot, while the 2 frame parameters are taught to the robot.

	Name	Type	User Input	Description
x1	ScrewFrameJoint	ParamVector	Teach	Target joint positions
x2	ScrewFrameCartesian	ParamVector	Teach	Target EE pose
x3	Velocity	Double	Specify	Velocity
x4	Lin	Boolean	Specify	Joint or Cartesian space movements
x5	OrientationLock	Boolean	Specify	Make tool orientation free
x6	SearchAngle	Double	Specify	Peg-in-hole rotation span
x7	ForceLimit	Double	Specify	Tool contact search force-limit
x8	MultiScrew	Boolean	Specify	Multiple screw execution

Table 5.1: Input parameters for the execution of an "Unscrewing" skill

After the offline parameters are specified and the teaching of the online parameters begins, the robot is stiffened, while it waits for the worker to begin the teaching of screw positions. After the worker applies a force to the end-effector to begin teaching the screw positions, the robot loosens the stiffness of the impedance control to allow the worker to insert the end-effector into the screws. To instruct the worker to insert the tool into the screws, the teaching interface is used, as seen on Figure 5.10.

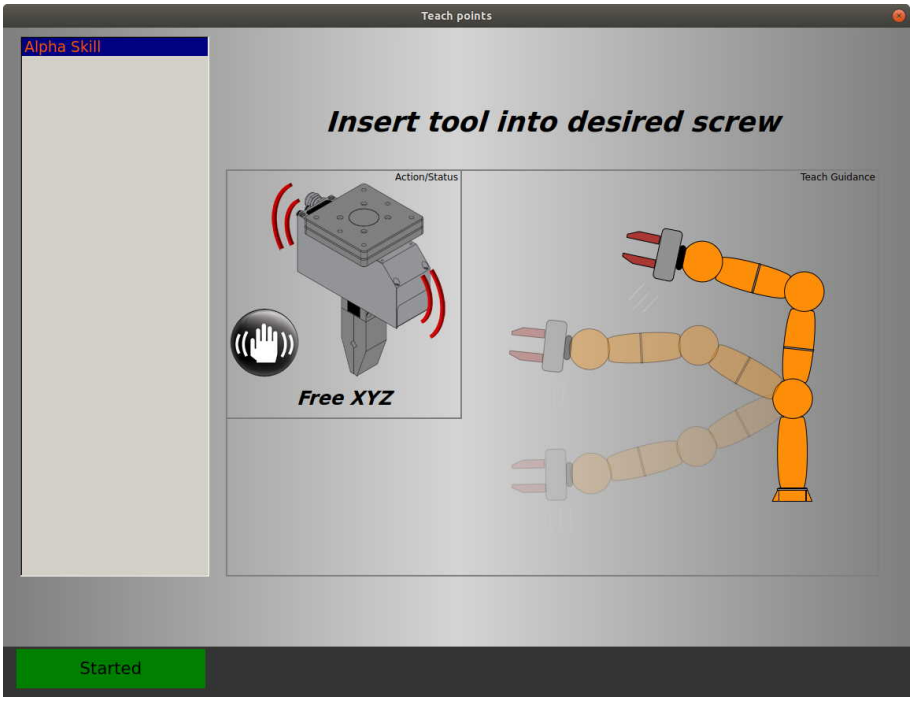


Figure 5.10: SBS teaching interface instructing the worker to insert the tool into the desired screw.

After the tool has been inserted into the screw, the teaching interface detects when the robot has been stationary for three seconds. After this has been detected, the current robot position is saved into the online parameters, after which the worker is instructed to rotate the tool using the interface shown in Figure 5.11.

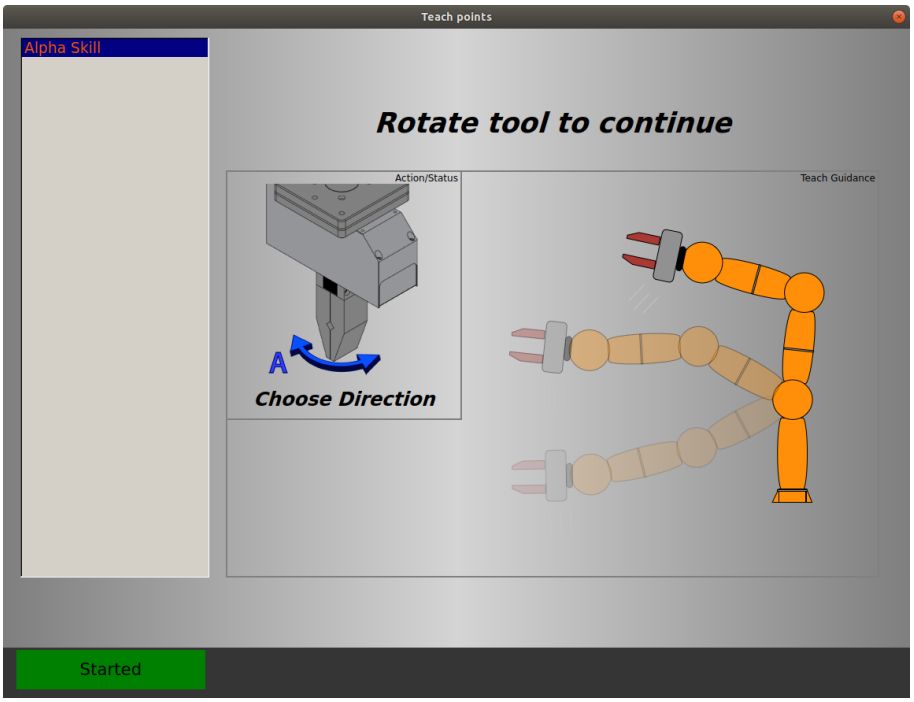


Figure 5.11: SBS teaching interface instructing the worker to rotate the tool.

Once the SBS teaching interface detects the rotation, a warning is given to the worker due to the robot being about to move, after which the robot removes the tool from the screw. After the tool is removed from the screw, the teaching interface stiffens the impedance control to allow the worker to choose between teaching more screw positions and stopping the teaching using the interface shown in Figure 5.12.

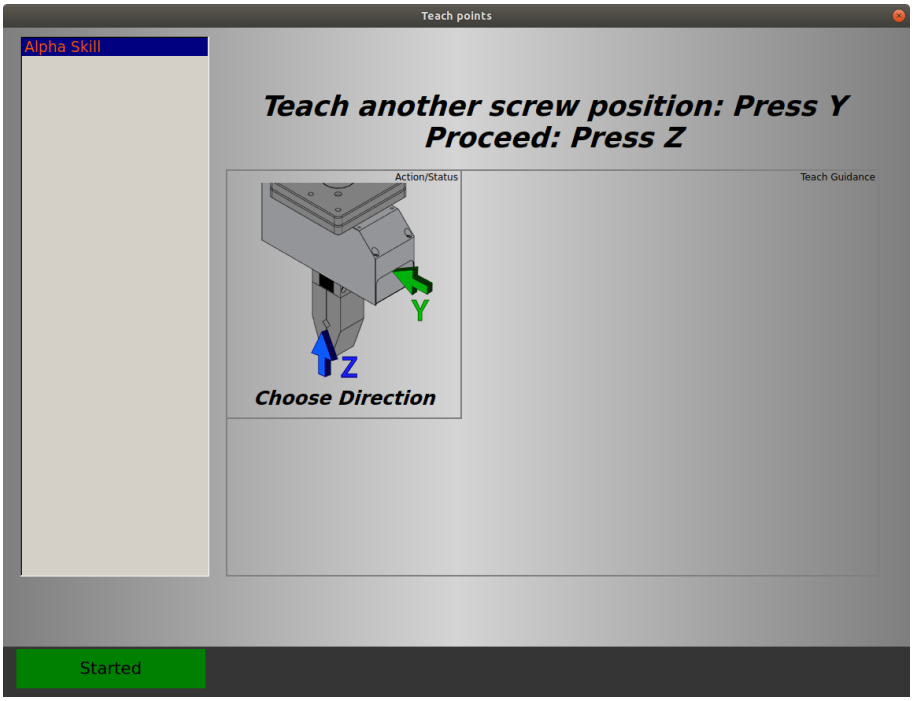


Figure 5.12: SBS teaching interface instructing the worker to decide between teaching more positions, or stopping the teaching.

After the teaching of the screws is done, the positions are saved to be used for later execution of the task created.

5.3 Sunrise Servoing

The Cartesian impedance control mode used in the project is done using the Sunrise servoing package from KUKA [2]. It enables the use of soft-realtime control for the KUKA LBR iiwa.

From the Sunrise servoing package, two control modes are chosen to be used for the project, these being the standard SmartServo class and the SmartServoLIN class. They control the robot using joints and frames respectively for controlling the robot. It does this by executing the movements asynchronously. Both classes have the impedance control mode built-in, enabling the robot to be compliant during the execution of the unscrewing skill.

The two SmartServo classes, both share problem, which occurs when using soft-realtime control. This issue is called the seconds hand effect. This effect is visible when commands are not given to the robot fast enough, which makes the movement look jittery. This effect can be seen on Figure 5.13.

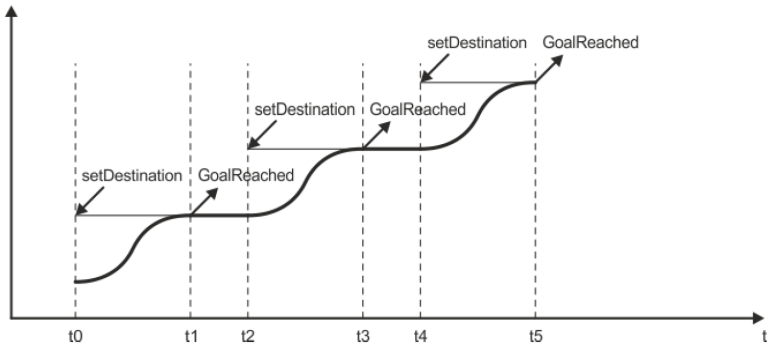


Figure 5.13: Figure showing the effect of the "seconds hand effect", made by KUKA [2]

As seen on the figure, if a new goal position is not sent fast enough, the robot slows down and even reaches the goal, resulting in a movement that looks jittery. However, as seen on Figure 5.14, the effect can be fixed by sending the commands to the robot faster or having larger distances between the goal commands.

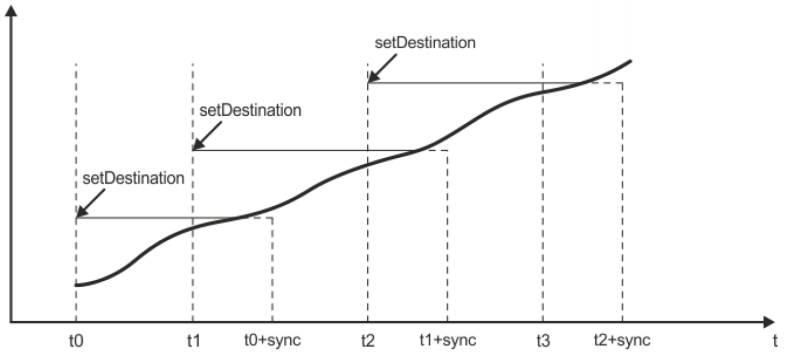


Figure 5.14: Figure showing the movement without the effect, made by KUKA [2]

Another way that this issue can be circumvented, is to lower the maximum acceleration and velocity of the robot so that it takes longer to go between the goal positions. An entirely different way of making the problem smaller is to use the SmartServo impedance control, which can also make the movements appear smoother. For this project, the impedance control mode is in use for the entire unscrewing process.

5.3.1 SmartServo Impedance Control

The impedance control mode found in the SmartServo classes can be controlled in an easier way, compared to using the standard impedance control, as the standard impedance control works during a single movement at a time. However, by using the SmartServo classes, the control mode can be initiated and used until the SmartServo runtime is stopped.

When using the SmartServo and SmartServoLIN classes for impedance control, some parameters are needed for the control mode to be initiated. These different parameters are the stiffness parameters and the tool parameters. The tool parameters are the transformation from the flange of the KUKA iiwa to the tool and the center of mass and mass of the tool. All of this is used for the gravity compensation for the robot. Once the tool is specified, the next step is to create the SmartServo runtime and control the robot using the goal positions. A flowchart of how the impedance control is done can be seen on Figure 5.15.

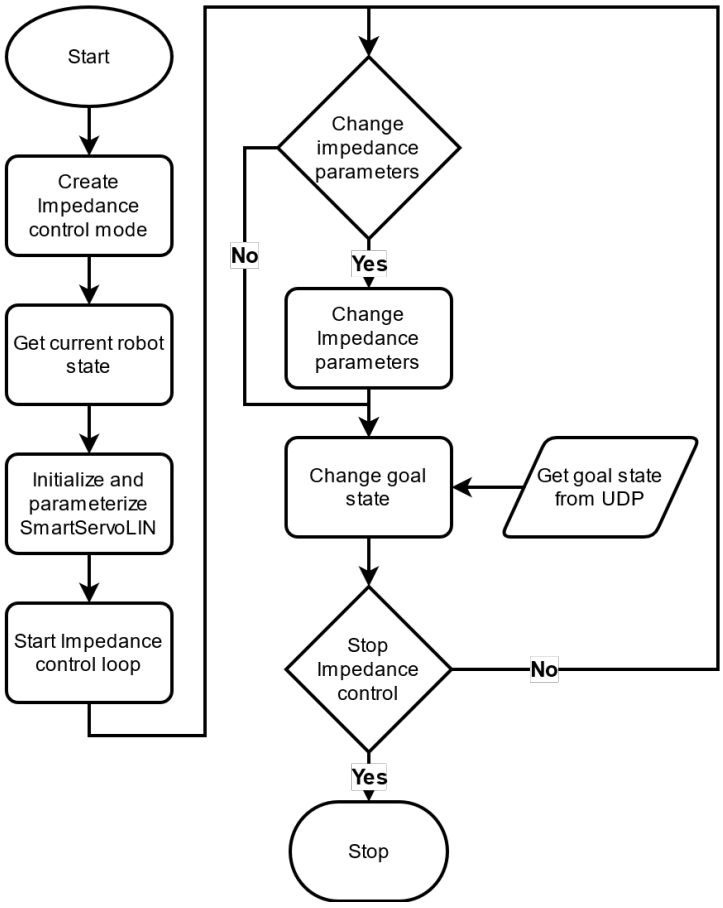


Figure 5.15: Flowchart of the impedance control function

As seen on the flowchart, the controller starts by creating the impedance control mode after which, the SmartServoLIN instance is created and parameterized. After the creation of the SmartServoLIN, the impedance control loop is initiated. In this loop, a decision function is created to allow the impedance control parameters to be changed during the runtime of the robot movements. After the decision function, the new goal state is read from the UDP connection, which is used to change the goal state of the robot.

This chapter explored the task of unfastening screws by breaking down the task into more doable skills and primitives for the robot. The teaching of the screw positions using the SBS teaching interface was explored. Finally, how impedance control is used was explored.

6 Evaluations

This chapter explores the evaluation of the full system and the individual parts of the system. To do this, the tests are done in two parts, these being the individual device primitives and the full system tests.

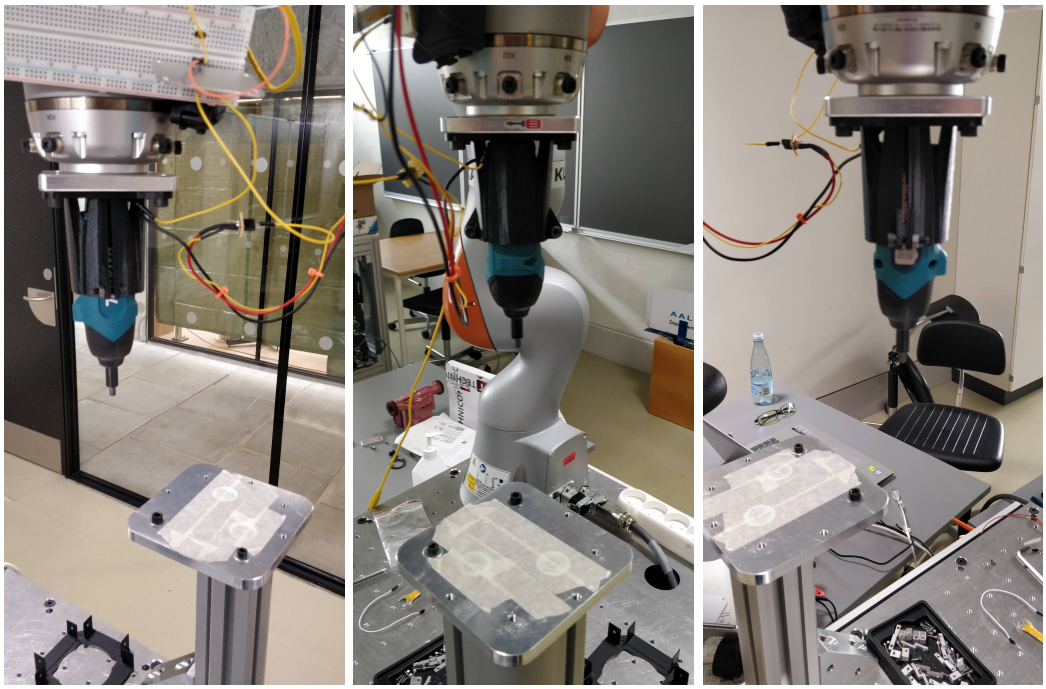


Figure 6.1: Test setup environment showing the robot and the test platform

The test setup, as seen on Figure 6.1, uses a test platform with the screws fastened into it to allow for a more consistent and static test setup. The test platform is lifted from the table, as this allows the robot to reach the screws, as the work-space of the robot cannot properly reach the table it is placed on.

The requirement for the full system test is to have a 95% success rate in unfastening the screws on the test platform. For this reason, individual tests are done using each device primitive to determine the best parameters for each primitive, which in turn allows the full system to have a higher success rate. A hypothesis is created for the tool insertion device primitive, which is that the initial rotation direction affects the outcome of the tool insertion. This hypothesis is tested during the task execution tests.

6.1 Device Primitives

As the SBS uses device primitives to define the skills for the robot, the different device primitives have to be evaluated to see if they perform as they should. The device primitive used for moving the robot in Cartesian space is however not evaluated as this is the basis for the other primitives.

The tests of each device primitive is done by testing different parameters. Because of this, only one parameter is changed at a time for each test. This is done to evaluate how each parameter affects the primitive.

6.1.1 Tool Contact Search

This primitive makes the robot's end-effector go downwards until an external force on the z-axis, after which the robot stops the downwards movement. This test consists of trying the primitive with different parameters to see the end result. The device primitive can be seen on Figure 6.2.

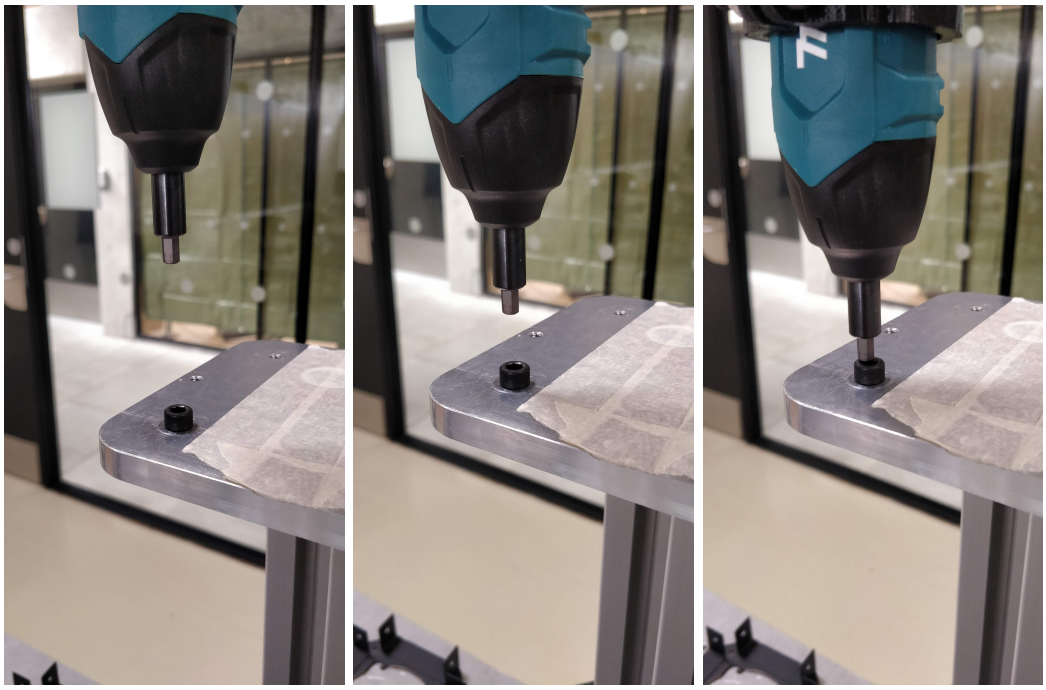


Figure 6.2: Pictures showing the tool moving downwards until contact with the screw.

The different parameters which are tested are the end-effector velocity and the external force needed to stop the movement. For the purpose of testing each parameter separately, only one parameter is changed at a time. The initial parameters for the test are a relative velocity of 25% of the deemed maximum velocity and a force limit of -0.5 N on the z-axis. The test is deemed successful if the robot correctly stops after contact with the screw.

Velocity	Force Limit	Result
25%	-0.5 N	0 / 10
50%	-0.5 N	0 / 10
75%	-0.5 N	0 / 10
100%	-0.5 N	0 / 10
25%	-1.0 N	9 / 10
50%	-1.0 N	10 / 10
75%	-1.0 N	10 / 10
100%	-1.0 N	10 / 10
25%	-1.5 N	10 / 10
50%	-1.5 N	10 / 10
75%	-1.5 N	10 / 10
100%	-1.5 N	10 / 10

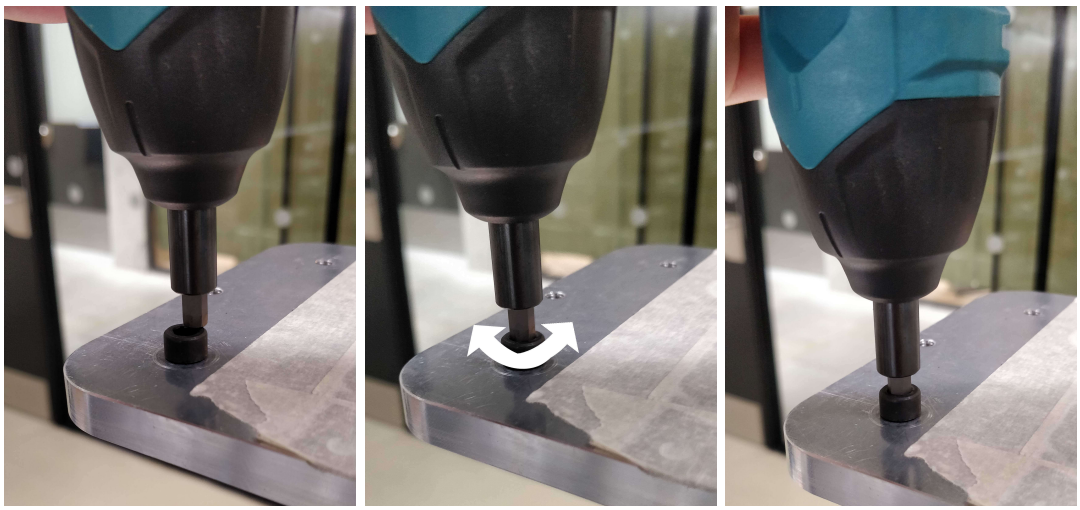
Table 6.1: Test results of the tool contact search

The results of the tests for this device primitive, shown in Table 6.1, shows that when the limiting force is too low, the upwards force exerted by the impedance control mode, is enough for the robot to think that it has hit screw. Due to this, the limiting force is increased, after which the success rate becomes 100%.

This means that in the full skill test, the external force limit is chosen to be -1.5 N, as this works using both high and low velocity for the device primitive.

6.1.2 Peg-in-Hole

The peg-in-hole primitive uses the same strategy used in [5]. However, due to the tool and control strategy being different, a new test is conducted to choose the correct impedance parameters for the problem. The device primitive can be seen on Figure 6.3.



(a) Tool after the tool contact search

(b) Picture with the rotation used for the tool insertion, overlayed with white arrows.

(c) Tool inserted into the screw

Figure 6.3: Pictures showing the tool being inserted into the screw

For this test, the stiffness parameters for the x-axis and the y-axis are changed simultaneously to ensure the stiffness is the same in the XY-plane. The z-axis stiffness is changed to test whether different downwards forces makes the peg-in-hole primitive more successful. The initial Cartesian position is static throughout the test. This test uses the tool contact search for the initial approach towards the screw. The starting stiffness for the XY-plane is chosen to be 1000, which is a high stiffness, meaning that the tool is less likely to deviate from the desired position, while the stiffness on the z-axis is chosen to be quite low to not apply too much force on the tool during the execution.

The test is deemed successful if the robot can successfully insert the tool into the screw and detect that the insertion is done correctly. If the robot takes longer than 15 seconds to insert the tool, the test is deemed unsuccessful.

X and Y stiffness	Z stiffness	Result
1000	100	0 / 10
1000	200	3 / 10
750	100	9 / 10
750	200	10 / 10
500	100	10 / 10
500	200	10 / 10

Table 6.2: Test results for the peg-in-hole impedance test

The test results seen in Table 6.2, show that with too high stiffness on the xy-plane, the tool can not be successfully inserted if the tool hits the side of the screw-head. This results in the success rate is very low for higher stiffness, while the success rate increases as the stiffness on the XY-plane decrease. In the end, the insertion reaches a 100% success rate for the Cartesian position chosen for the test.

Recovery

Due to the tool insertion, or peg-in-hole, having a high chance of failure, a recovery action is made. This recovery action detects if the tool gets too low on the z-axis, meaning that the end-effector missed the screw entirely. It works by asking the operator/worker near the robot to move the end-effector into the screw to ensure that the tool is correctly inserted. Once the tool is inserted, the robot continues with the unscrewing skill.

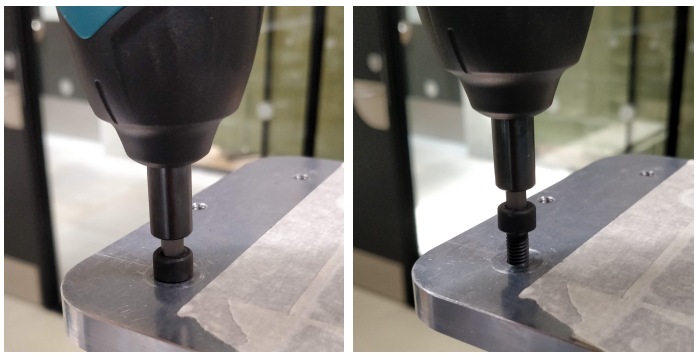
For the purpose of making sure that this is correctly detected, with no false positives, a test is conducted. During the test, the end-effector is pushed off the screw to ensure it does not get inserted into the screw. The test is deemed successful if the failure is correctly detected. A picture of the tool having missed the screw can be seen on Figure 6.4.

Figure 6.4: Picture showing the tool after missing the screw

The test was done 50 times, out of which the robot detected the failure in 100% of the tests. This means that the recovery mode can be expected to work if the initial state of the end-effector is that the tool is in contact with the side of the screw-head.

6.1.3 Unscrewing

For the unscrewing test, the initial state of the test is that the tool is inserted into the screw. This means that the success criteria is whether the robot can correctly detect if the screw is fully unscrewed. This detection is described in subsection 5.1.3. Once the robot has detected that the unscrewing is done, the screw is examined to see if it is fully unfastened. The device primitive can be seen on Figure 6.5.



(a) Picture before the unfastening.

(b) Picture after the unfastening.

Figure 6.5: Pictures showing the screw being unfastened

If the screw is fully unfastened after the detection, the test is deemed successful. In the 20 tests done, the robot succeeded in detecting the unscrewing, giving the detection a 100%

success rate.

6.2 Task evaluation

The task is evaluated in three ways, the first of which is the teaching, as this is how the robot gets to know the positions of the screws. Second, the task execution is evaluated. Finally, the error/recovery state is evaluated.

For the evaluation of the unscrewing task created in the project, there are 4 screws screwed into the test platform, as seen on Figure 6.6.

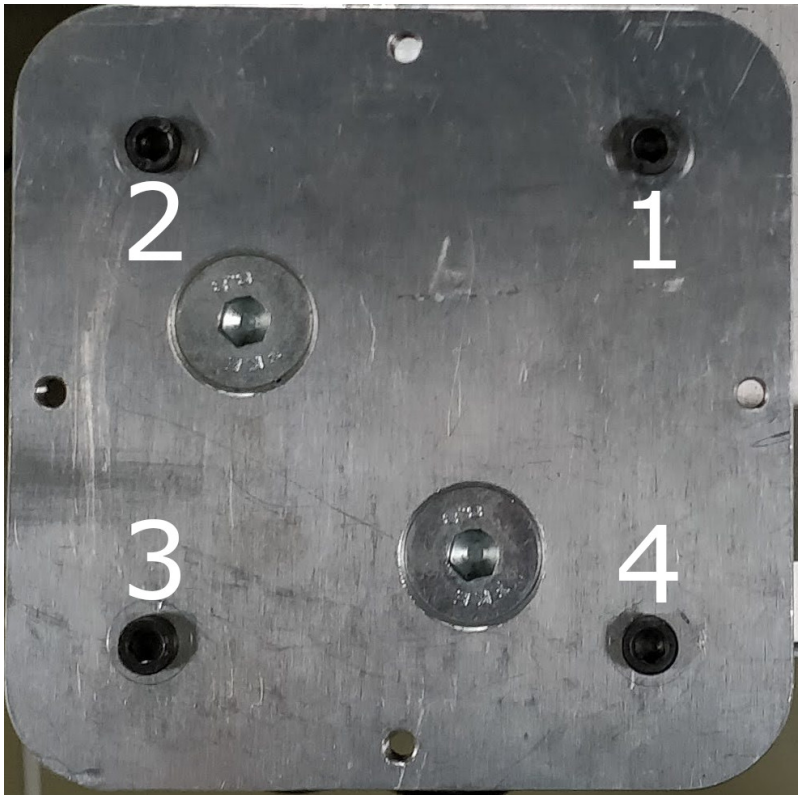


Figure 6.6: Test platform with screws numbered, with the screw 1 being closest to the robot.

The test platform is set up, so that screw 1 is closest to the robot, while screw 3 is the farthest from the robot. Screw 1 and 2 are both 12 mm long screws, while screw 3 is a 6 mm long screw and screw 4 being the longest of 20 mm. This gives the system 3 different screw lengths to be evaluated from.

6.2.1 Teaching

To evaluate the teaching of the unscrewing task, the four screws are taught to the robot 10 different times to test the accuracy of the teaching. The teaching is done 10 times and saved into different task files, which will be used during the execution and error state evaluation.

Once the teaching is done, the positions are checked using the test-run function in the SBS interface to evaluate the positions saved by the SBS. The initial teaching of the robot

shows that screw 3 and 4 have a better Cartesian position saved for the screws, while screw 1 and 2 are off-centred. This problem seems to be due to the nature of the tool not being static enough, as the tool can be moved slightly.

After the information on the screw positions is saved by the SBS, it is processed which can be seen in Table 6.3.

	Screw 1	Screw 2	Screw 3	Screw 4
Average	-308.24 308.47	-307.62 394.87	-392.99 395.38	-393.92 309.97
Minimum	-309.75 307.83	-308.71 393.74	-394.05 394.64	-395.06 309.52
Maximum	-307.56 309.33	-306.66 395.71	-392.23 396.65	-393.16 310.67
Deviance	2.197 1.498	2.057 1.964	1.827 2.011	1.896 1.147

Table 6.3: Information about the Cartesian screw positions saved by the SBS.

In the table, the average, minimum, maximum and deviance can be seen. It can be seen that the saved position can vary by up to 2 millimeters, which can cause the robot to fail with the insertion of the tool during the execution of the skill. This deviance is due to small variances in the orientation during the teaching of the screw positions, as well as the screwdriver being able to move inside the tool.

6.2.2 Execution

To test the execution of the unscrewing task, each of the task files taught to the robot is executed 5 times. The system is then evaluated on how many screws are removed with and without having to ask the operator/worker for help to insert the tool into the screws. This means that the robot will try to unfasten 20 screws in total for each of the task files.

Test nr.	Automatic	Help	Failed
1	17 (4)	2	1
2	16	4	0
3	17 (1)	3	0
4	13 (1)	5	2
5	8	11	1

Table 6.4: Table showing the results of the task execution test. Automatic means that the robot needed no help, Help means that the worker was needed, and Failed means that it failed to fully unfasten the screw. The number in parenthesis means that the screw was tightened too hard, meaning that it could not be unfastened.

As can be seen in Table 6.4, the results show that the robot can, for the first three task files, manage to unfasten the screws automatically in 85%, 80% and 85% of the times, while it only failed once with the first task file. The fourth test failed two times to fully unfasten the longest of the four screws due to the unfastening detection looking for an oscillation, which also appears when a long screw is being unfastened. The fifth and last test failed to perform the tool insertion 11 times, which can be attributed to two causes, both of which are caused by the screwdriver being able to move slightly in the 3D print.

To test the hypothesis of whether the initial rotation direction for the tool insertion changes the outcome of the device primitive, the last 5 tests are done using the opposite rotation of the tool.

Test nr.	Automatic	Help	Failed
1	16	2	2
2	20	0	0
3	15	5	0
4	19	0	1
5	19	0	1

Table 6.5: Table showing the results of the task execution test. Automatic means that the robot needed no help, Help means that the worker was needed, and Failed means that it failed to fully unfasten the screw.

As shown in Table 6.5, the first and third tests performed worse than the other 3 tests. This difference is due to the tool having moved in the 3D print, which causes the execution to fail. For the 3 other tests, they performed the tool insertion significantly better than the tests done in Table 6.4. The reason for this is that the initial rotation in the tool insertion device primitive moves the tool in a different way, depending on the initial rotation direction.

Overall, the system succeeded in automatically unfastening the screws in 80% of the tests. The system required help from the worker in 16% of the tests, which means that the system achieves an overall success rate of 96%. This means that the full system is deemed successful, having more than a 95% overall success rate.

6.2.3 Error/recovery states

The final test revolves around the robot falsely detecting the tool contact or tool insertion failing. This is then evaluated on how the robot responds to the error.

The test is done by interrupting the robot during the two device primitives that use detection to end the primitives. These two primitives are the "tool contact search" and the "tool insertion" primitive. For the tool contact search primitive, the robot is interrupted by stopping the robot before the tool hits the screw. For the tool insertion primitive, the tool is grabbed to make the robot think that the tool is inserted.

The system is then evaluated, on which of two states it enters based on the interruption. These two states are the recovery/error state of the system, and the second state is to continue the task without stopping. Each of the primitives is interrupted 10 times.

Device primitive	**Recovery/error**	**Continue**
Tool contact search	10	0
Tool Insertion	0	10

Table 6.6: Table showing the results of interrupting the robot during the tool contact search and tool insertion device primitives.

As Table 6.6 shows, the robot will go into the recovery/error state when interrupted during the tool contact search. In contrast, the robot continues with the execution of the task when interrupted during the tool insertion primitive.

7 Discussion

In this chapter, the evaluation of the robotic system is reflected upon. Both the successes and issues with the system are analysed in order to list some of the challenges, which could be done in future work.

7.1 Robot connection

The robot UDP connection, was used due to the FRI connection not being able to fully control the robot. Because of this, the UDP connection was created to send commands to the robot, which made controlling the robot using an external PC possible. Through this connection, the robot is able to send feedback to the external PC, as well as receiving commands. However, even though the UDP connection enables fast control of the robot, it does come with its own issues, which should be taken into account.

The big issue with a UDP connection is the possibility of losing packets during the communication between the external PC and the controller. When this issue happens, the robot command can be misinterpreted, which makes the robot move in odd ways. This made it so that the robot would change the orientation of the end-effector and the goal position. This proved to be an issue during the testing of the task execution, as whenever a move command was started, there was a chance of this issue being present, which would result in the current test being reset. This, in addition to needing a reset, could, if the tool is inserted into the screw, potentially damage or destroy the tool. It can also slightly move the screwdriver in the 3D print, which can change the results of the peg-in-hole device primitive.

This is expected to be due to the UDP connection sending and receiving the packets on the same port. It could also be due to the robot state being read before the values from the robot is read, as this would result in the same behavior. For future work, this issue should be explored and resolved.

7.2 Peg-in-hole

The peg-in-hole evaluation behaved as expected, where a too high stiffness on the XY-plane makes it so that the tool is not inserted. However, with a lower stiffness, the tool is able to move in the XY-plane, enabling the system to have a higher chance of inserting the tool correctly into the screw. This means that with the correct stiffness parameters, the system was able to insert the tool into the screws correctly.

As the peg-in-hole is done using positions taught by the worker, the testing of the primitive and task execution show promising results. However, if the screws are deformed due to wear and tear or somehow not in the pre-programmed positions, the system would fail. However, due to this being a known problem during the creation of the system, an error/recovery state was created to get around this issue, which performed as expected. As a side-effect of the implementation, it even made it possible to interrupt the robot during the downwards movement towards the screw.

The simplicity of the primitive, while it works for the test platform, is mostly due to the time constraints with the robot due to the Covid-19 situation.

A more complex searching algorithm should be implemented for future work, as the current algorithm only rotates the tool while applying a downwards force. This could be done using a spiral searching algorithm, as done in [4]. Another thing that could be done is to add a camera to the tool to check for the screw in these cases.

7.3 Powered screwdriver

A 3D print was designed to hold the screwdriver in place for the project, which performed as expected during the evaluations. The 3D print was created as a prototype to be able to test the execution of the unscrewing task. As the screwdriver itself is controlled using an Arduino, it made controlling the tool with ROS as simple as sending a ROS message.

However, as shown in the evaluations of the system, the tool used for the project had two issues during the evaluations. The first of which is how the screwdriver is held in place, as it enabled the screwdriver to move slightly inside the 3D print, which affected the task execution results. The second issue of the tool is that the system is liable to fail to unfasten the screws due to the screws being tightened too much. This issue can be solved by simply using a more powerful screwdriver.

Another thing for future work is to create a more static tool, which makes the teaching of positions more accurate. It could improve the peg-in-hole performance as well. One last improvement for the tool would be to use the KUKA Flange connectors to control the screwdriver. This would make the tool easier to control, while the tool setup would be easier as well, as the cable for the Arduino can make the impedance control behave differently due to the cable pulling in different directions.

7.4 Unscrewing

The unscrewing detection made performed better than expected when used on different length screws. The longest of the screws was expected to produce false positives, whereas it could correctly detect the unscrewing in most of the cases.

The reason for the longer screw being expected to produce false positives is that oscillations on the Z position, which is used for the unscrewing detection, could be produced when unfastening long screws. While this was detected falsely during the task evaluation, the detection could be made better for future work of the project.

A way to make the detection better could be done using the external force on z, which also shows a possibility of being used. Another strategy entirely could be to use machine learning for this detection.

7.5 Error/Recovery state

Though the error/recovery state was implemented for the peg-in-hole to detect if the tool was correctly inserted into the screw, it seemed to solve another problem entirely. The problem mentioned here is that the robot can be interrupted during the execution during two of the device primitives.

These primitives are the tool contact search and the peg-in-hole. As this detection was done using the Z position of the tool, interrupting the tool contact search makes the robot think that the tool has come into contact with the screw. This means that the peg-in-hole primitive is called. But because of the tool being interrupted above the screw, the tool will, due to the downwards force, move downwards, triggering the error/recovery state.

This means that the robot enters the error state when interrupted. For future work, an additional error state should be implemented, as when interrupting the rotation of the peg-in-hole, the robot detects it as the tool is correctly inserted. For this reason, the position on Z could be used to check if the tool has actually been inserted.

8 Conclusion

This report proposed a robotic system using a KUKA LBR iiwa, an Arduino Mega 2560 and a powered screwdriver.

The system was integrated into the SBS to allow for less experienced workers to work with the robotic system while also allowing the system to be easily reconfigured for other purposes.

The system managed to use teaching to get the screws' positions while achieving a 96% success rate for the system. This proves that with some optimization on the system, the success rate could become higher, while also doing the unscrewing faster.

In conclusion, the system works as a proof of concept. It shows that using the SBS for controlling the robotic system can prove to make the system simpler to use and more user-friendly. In addition, it showed that by using impedance control, controlling the velocity of the robot is possible, as well as making the robot compliant to the worker in the workspace

Bibliography

[1] Casper Schou. Easy reconfiguration of modular industrial collaborative robots. Technical report, Aalborg University, 2016.

[2] KUKA. *KUKA Sunrise.Servoing 1.14*. Pub KUKA Sunrise.Servoing 1.14 (PDF) en, 2017.

[3] Grundfos. Grundfos - about us. `https://www.grundfos.com/about-us.html`, 2020. Accessed: 27-05-2021.

[4] Jiayi Liu, Zude Zhou, Duc Truong Pham, Wenjun Xu, Chunqian Ji, and Quan Liu. Collaborative optimization of robotic disassembly sequence planning and robotic disassembly line balancing problem using improved discrete bees algorithm in remanufacturing. *Robotics and Computer-Integrated Manufacturing*, 61, 2020. ISSN 0736-5845. URL `\url{http://www.sciencedirect.com/science/article/pii/S0736584518300516}`.

[5] Guilherme Mateus Martins and Jacob Krunderup Sørensen. An impedance control driven robotic disassembly platform using deep-learning for adaptive screws detection. Technical report, Aalborg University, 2020.

[6] Jun Huang, Duc Troung Pham, Ruiya Li, Kaiwen Jiang, Dalong Lyu, and Chunqian Ji. Strategies for dealing with problems in robotised unscrewing operations. Technical report, University of Birmingham, 2021.

[7] J. Li, M. Barwood, and S. Rahimifard. Robotic disassembly for increased recovery of strategically important materials from electrical vehicles. *Robotics and Computer-Integrated Manufacturing*, 50:203 – 212, 2018. ISSN 0736-5845. URL `\url{http://www.sciencedirect.com/science/article/pii/S0736584516301004}`.

[8] N. M. DiFilippo and M. K. Jouaneh. A system combining force and vision sensing for automated screw removal on laptops. *IEEE Transactions on Automation Science and Engineering*, 15(2):887–895, 2018.

[9] Daniel Schneider, Elmar Schömer, and Nicola Wolpert. A motion planning algorithm for the invalid initial state disassembly problem. In *2015 20th International Conference on Methods and Models in Automation and Robotics (MMAR)*, pages 35–40, 2015.

[10] Wei Hua Chen, Gwendolyn Foo, Sami Kara, and Maurice Pagnucco. Automated generation and execution of disassembly actions. *Robotics and Computer-Integrated Manufacturing*, 68:102056, 2021. ISSN 0736-5845. URL https://www.sciencedirect.com/science/article/pii/S0736584520302672.

[11] Behzad Sadrfaridpour and Yue Wang. Collaborative assembly in hybrid manufacturing cells: An integrated framework for human-robot interaction. *IEEE Transactions on Automation Science and Engineering*, PP:1–15, 09 2017.

[12] Quan Liu, Zhihao Liu, Wenjun Xu, Quan Tang, Zude Zhou, and Duc Truong Pham. Human-robot collaboration in disassembly for sustainable manufacturing. *International Journal of Production Research*, 57(12):4027–4044, 2019. URL https://doi.org/10.1080/00207543.2019.1578906.

[13] Panagiota Tsarouchi, Alexandros-Stereos Matthaiakis, Sotiris Makris, and George Chryssolouris. On a human-robot collaboration in an assembly cell. *International Journal of Computer Integrated Manufacturing*, 30(6):580–589, 2017.

[14] Frank Wallhoff, Jürgen Blume, Alexander Bannat, Wolfgang Rösel, Claus Lenz, and Alois Knoll. A skill-based approach towards hybrid assembly. *Advanced Engineering Informatics*, 24(3):329–339, 2010. ISSN 1474-0346. URL https://www.sciencedirect.com/science/article/pii/S1474034610000406. The Cognitive Factory.

[15] Serena Ivaldi, Sebastien Lefort, Jan Peters, Mohamed Chetouani, Joelle Provasi, and Elisabetta Zibetti. Towards engagement models that consider individual factors in hri: On the relation of extroversion and negative attitude towards robots to gaze and speech during a human–robot assembly task. *International Journal of Social Robotics volume*, 9, 2017.

[16] S.M. Mizanoor Rahman and Yue Wang. Mutual trust-based subtask allocation for human–robot collaboration in flexible lightweight assembly in manufacturing. *Mechatronics*, 54:94–109, 2018. ISSN 0957-4158. URL https://www.sciencedirect.com/science/article/pii/S0957415818301211.

[17] Kathrin Wegener, Wei Hua Chen, Franz Dietrich, Klaus Dröder, and Sami Kara. Robot assisted disassembly for the recycling of electric vehicle batteries. *Procedia CIRP*, 29:716 – 721, 2015. ISSN 2212-8271. URL \url{http://www.sciencedirect.com/science/article/pii/S2212827115000931}. The 22nd CIRP Conference on Life Cycle Engineering.

[18] Esther Alvarez-de-los Mozos and Arantxa Renteria. Collaborative robots in e-waste management. *Procedia Manufacturing*, 11:55 – 62, 2017. ISSN 2351-9789. URL \url{http://www.sciencedirect.com/science/article/pii/S2351978917303372}. 27th International Conference on Flexible Automation and Intelligent Manufacturing, FAIM2017, 27-30 June 2017, Modena, Italy.

[19] L. Johannsmeier, M. Gerchow, and S. Haddadin. A framework for robot manipulation: Skill formalism, meta learning and adaptive control. In *2019*

International Conference on Robotics and Automation (ICRA), pages 5844–5850, 2019.

[20] Casper Schou, Mikkel Hvilshøj, and Christian Carøe. Intuitive programming of aimm robot. Technical report, Aalborg University, 2012.

[21] Joshua Frank. Ros serial arduino library. `https://github.com/frankjoshua/rosserial_arduino_lib`, 2020. Accessed: 18-05-2021.

[22] M. Safeea and P. Neto. Kuka sunrise toolbox: Interfacing collaborative robots with matlab. *IEEE Robotics Automation Magazine*, 26(1):91–96, March 2019. ISSN 1070-9932.