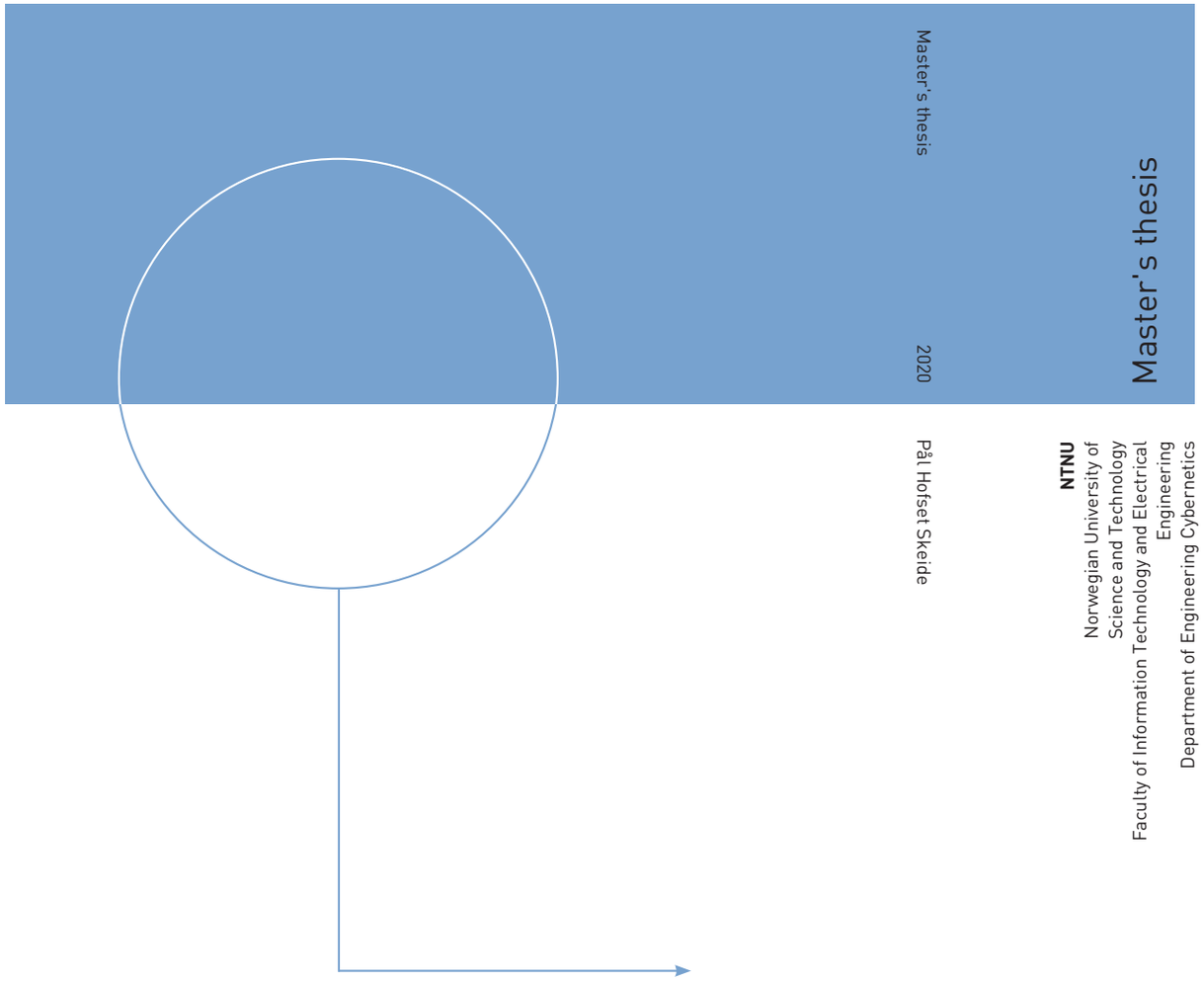


Pål Hofset Skeide

Automating tank operations in smolt production – A concept study for automating tank cleaning using robotic arms

June 2020

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

Automating tank operations in smolt production – A concept study for automating tank cleaning using robotic arms

Pål Hofset Skeide

Industrial Cybernetics
Submission date: June 2020
Supervisor: Martin Føre
Co-supervisor: Dr. Eleni Kelasidi

Norwegian University of Science and Technology
Department of Engineering Cybernetics

NTNU
Norges teknisk-naturvitenskapelige universitet

Fakultet for informasjonsteknologi, matematikk og elektroteknikk
Institutt for teknisk kybernetikk

MASTER'S THESIS

Name of the candidate: Pål Hofset Skeide

Fag: Engineering Cybernetics

Oppgavens tittel (norsk): Autonome karoperasjoner innen smoltproduksjon – Et konseptstudie for å automatisere tankrengjøring med robotarmer

Oppgavens tittel (engelsk): Automating tank operations in smolt production – A concept study for automating tank cleaning using robotic arms

Background:
There is presently a trend in the aquaculture industry with operations and production methods becoming less reliant on manual labour and experience-based reasoning, and more based upon objective indicators, automation principles and decision support systems. This harmonises with the philosophy of Precision Fish Farming (PFF) where new technology and automation principles are employed to improve human control over the biological processes in different phases in the aquaculture production cycle. Although most of the weight gain in salmon production is achieved during the sea-based ongrowing phase, the success of the production also depends strongly on the quality of the fish when transferred to sea-cages. This is determined by how well the fish are managed during the smolt production phase preceding the ongrowing phase. Smolt production is conducted in land-based tank facilities and covers the period from hatching until the fish undergo smoltification (i.e. a metamorphosis adapting them from living in fresh water to handling sea-water).

Together with an industry consortium of technology providers and aquaculture production companies, SINTEF and NTNU recently received funding from the Research Council of Norway for the project AUTOSMOLT 2025. The ultimate vision of AUTOSMOLT 2025 project is to adapt and develop a holistic solution for the next generation of smolt production by applying the principles of PFF at different stages of the smolt production cycle, thus bringing smolt production closer to realization within the framework of Industry 4.0. This entails increasing the level of autonomy and objectivity in smolt production operations to reduce dependencies on manual labour and subjective assessments, and to improve accuracy, precision and repeatability.

This master's thesis is associated with the AUTOSMOLT 2025 project, and will focus on specifying and developing new solutions for autonomous operations in smolt production facilities. The work will feature an overview of state-of-the-art and current practice in the smolt industry and a literature review of existing solutions that may be stated as autonomous. Based on this, a concept will be developed focusing on solving one or several concrete operations/challenges that are currently manual in the smolt production industry. This solution will be implemented in a relevant framework for developing control methods for robotic systems, and demonstrated and tested virtually in a this environment to evaluate how the concept solved the chosen tasks.

The project will contain the following elements:

- Literature study covering core aspects such as:
    - State of the art and methods in smolt production technology
    - Existing autonomous solutions for smolt production

    - Autonomous solutions in comparable industry segments (e.g. marine aquaculture, agriculture, oil and gas)
- Develop a specification for concept for autonomous tank operations by using the literature study to identify:
    - Operation(s) (e.g. feeding, sorting, grading, vaccination, transferring, cleaning and disinfection, and removal of dead fish and waste) that are good candidates for being solved with autonomy
    - Autonomous tools (e.g. robotic arms) that are suitable components for solving one or several of the chosen challenges
    - Path planning strategies that can be used to solve the chosen challenges
    - Relevant frameworks for implementing, modelling and virtually testing the autonomous solutions
- Implement geometric models of the operational environment (e.g. a tank for smolt production) and model the autonomous tool (e.g. robotic arm) in the chosen development framework
    - Specify robotic manipulators based on outcome from literature study
    - Specify tank dimensions and properties using data from partners in AUTOSMOLT 2025
    - Implement control system components needed to solve the chosen operations
- Develop path planning to test the implemented concept for automation
    - Verify basic functionality through unit and systems testing
    - Run virtual experiment to explore concept performance when following paths for a selection of operations that considered possible to automate through the concept
- Based on the above, describe general procedure for efficient modelling and implementation of robotic systems for aquaculture purposes on the chosen implementation platform

Thesis start: 06. January 2020

Delivery deadline: 15. June 2020

Delivered at the Department of Engineering Cybernetics

Main supervisor: Martin Føre
Co-supervisor: Dr. Eleni Kelasidi, SINTEF Ocean

Trondheim, 14. January 2020

Faglærer

Summary

This thesis aims at exploring the possibility of using a novel mobile robotic manipulator system to automate identified operations within smolt production facilities. The work has been done as part of a masters thesis at NTNU Trondheim in collaboration with SINTEF Ocean.

A literature review within the field of smolt production and related industries with particular emphasis on a technological level and possibilities within automation has been conducted. Based on these findings and a comparative study of available commercial robotic solutions within relatable fields such as agriculture, oil and gas and various cleaning operations a requirement specification was defined to be used as a basis for the chosen robotic system.

The manipulator part of the defined novel robotic system was created and simulated in Gazebo using the online Robot Operating System (ROS) based environment ROS Development Studio. The development included Computer Aided Drawing (CAD) work, Unified Robot Description Format (URDF) descriptions, ROS configurations of manipulators, simulation environment and controls, and finally simulations and extraction of results.

Ultimately, Matlab was used for constrained multi-goal path planning of identified operations within smolt production. This included the development of an approach for defining a coarse workspace using screw theory and uniform Monte Carlo method, and algorithms for creating paths of MoveIt compatible poses. Path following was then introduced to demonstrate different operations such as cleaning, feeding and gripping using the ROS compatible motion planning framework MoveIt.

This thesis has laid a good foundation for further development and research within the use of robotic systems in smolt production by creating a generic approach for defining and simulating manipulators within Gazebo simulation environments and mapping of where robotic intervention efforts should be applied.

Keywords: Smolt, automation, SolidWorks, ROS, Gazebo, MoveIt, C++, Matlab, workspace, multi-goal path planning.

Sammendrag

Denne oppgaven tar sikte på å utforske muligheten for å bruke et nytt mobilt robot system for å automatisere identifiserte operasjoner i smoltproduksjonsanlegg. Arbeidet er gjort som en del av en masteroppgave ved NTNU Trondheim i samarbeid med SINTEF Ocean.

Et litteratursøk innen smoltproduksjon og lignende næringer er gjennomført. Teknologisk nivå og muligheter innenfor automatisering var spesielt vektlagt i søket. Basert på funnene fra litteraturen og en sammenligning av tilgjengelige kommersielle robotløsninger ble det definert en kravsspesifikasjon som videre ble brukt som grunnlag for det valgte robotsystemet.

Robotarmen av det definerte robotsystemet ble laget og simulert i Gazebo ved hjelt av det nettleserbaserte Robot Operating System (ROS) utviklingsmiljøet, ROS Development Studio. Dette innebar dataassistert design (CAD), Unified Robot Description Format (URDF) beskrivelser, ROS-konfigurasjoner av robotarm, simuleringsmiljø og kontrollere, og til slutt simuleringer med tilhørende resultater.

Matlab ble brukt til flermåls- baneplanlegging av identifiserte oppgaver innen smoltproduksjon. Dette inkluderte utvikling av en metode for å definere et ytre arbeidsområde (workspace) ved bruk av skruteori og uniform Monte Carlo metode, og algoritmer for å lage baner som var kompatible med det ROS kompatible bevegelsesplanleggings rammeverket, MoveIt. Videre ble bane-følging introdusert for å demonstrere forskjellige oppgaver som rengjøring, mating og griping ved hjelp av ROS og MoveIt.

Denne avhandlingen har lagt et godt grunnlag for videre utvikling og forskning innen bruk av robotsystemer i smoltproduksjon ved å skape en generisk tilnærming for å definere og simulere robotsystemer i Gazebo-simuleringsmiljøer og kartlegge hvor roboter bør brukes.

Nøkkelord: Smolt, automatisering, SolidWorks, ROS, Gazebo, MoveIt, C++, Matlab, workspace, multi-goal path planning.

Preface

This master's thesis was written as a part of a 2-year master's degree within Industrial Cybernetics at the Faculty of Information Technology and Electrical Engineering at Norwegian University of Science and Technology (NTNU). The thesis is a part of the Autosmolt2025 project and was supervised by Martin Føre, and co-supervised by Eleni Kelasidi.

The basis for this project stemmed from my desire to explore the applicability of robotic systems within new industries and challenges that could lead to increased autonomy and wealth creation. I will therefore like to thank my supervisor Martin Føre for giving me the opportunity to explore this subject and in no small degree, dictate the content of this project.

First and foremost, I would like to thank my supervisors for guidance and support throughout this project, giving me constructive feedback and ideas to be explored in the thesis. I am also grateful for the support and openness received from the Autosmolt2025 partners, Blueprintlab and Artec Aqua.

Personally, I have learned a lot from writing this thesis, both from a technical, industrial and structural perspective. First of all, I have learned a lot about smolt- and land-based production, both with regards to state-of-the-art operations, present challenges and possible solutions. From a technical perspective, I have learned how to describe, research, construct and simulate robotic solutions with an intention in mind using open-source software. Finally, I have learned much about research methodology, working with industry contacts and personal structuring to keep up the progress in such a project. I hope this thesis can be a step in the direction of increasing the level of autonomy in smolt production facilities, and that stakeholders can find the output of this thesis useful.

Contents

Summary i

Sammendrag ii

Preface iii

Table of Contents iv

List of Tables vii

List of Figures viii

Abbreviations x

1 Introduction 1
- 1.1 Introduction to smolt production . . . 2
- 1.2 Challenges and demands in land-based fish farming industry . . . 6
  - 1.2.1 Smolt production site information . . . 6
  - 1.2.2 State of the art technological development in the industry . . . 7
  - 1.2.3 Factors that advocate automation . . . 8
  - 1.2.4 Operations in smolt production . . . 9
  - 1.2.5 Selected operations facilitated for automation . . . 10
- 1.3 Related industries . . . 13
  - 1.3.1 The rest of the salmon farming chain . . . 13
  - 1.3.2 The agriculture industry . . . 13
  - 1.3.3 Oil and gas industry . . . 15
  - 1.3.4 The manufacturing industry . . . 15
  - 1.3.5 Urban cleaning solutions . . . 16
  - 1.3.6 Underwater solutions for cleaning operations . . . 17
- 1.4 Adhesion principles - A review . . . 18
  - 1.4.1 Summary and choice of adhesion principle . . . 19

1.4.2 Literature review within selected adhesion principle 20
1.5 Summary 21

2 Selection and setup of autonomous system and simulation environment 23
2.1 Underwater manipulator basics 23
2.2 Existing manipulator solutions 27
2.2.1 Summary and choice of manipulator 34
2.3 Selection and setup of simulation software 34
2.3.1 Robotic simulation software 34
2.3.2 ROS 37
2.3.3 CAD-modelling of tank environments, generic and specific manipulator 39
2.3.4 From CAD to URDF description 42
2.3.5 Setup in simulation environment 44
2.4 Summary 51

3 Theory, control system components and tools used in the implementation 52
3.1 Linear algebra: Transformation matrix 52
3.2 Kinematics using screw theory 53
3.3 Path- and trajectory planning 55
3.4 Motion planning 57
3.4.1 ROS trajectory controller 58
3.4.2 The Move Group Node 59
3.4.3 MoveIt planners 60
3.5 Summary 63

4 Motion planning and control of generic robotic manipulator 64
4.1 Configuration of MoveIt 64
4.2 Programmatic control of MoveIt using C++ 67
4.3 Implementation of generic manipulator and verification of concepts . . . 71
4.3.1 Setup in Matlab and data retrieval from ROS 72
4.3.2 Tuning and test of joint trajectory controller for better response . 75
4.4 Manipulator workspace 81
4.5 Summary 85

5 Motion planning and demonstration of case studies with the Reach Bravo 86
5.1 Configuration of specific manipulator 86
5.2 Path generation in Matlab 89
5.2.1 Cleaning operation 94
5.2.2 Feeding operation 95
5.2.3 Gripping operation 98
5.3 Waypoints concept implementation 99
5.4 Results and demonstrations 100
5.4.1 Simulation of cleaning operation 100
5.4.2 Simulation of feeding operation 104
5.4.3 Simulation of gripping operation 108

5.5 Discussion 112

6 Conclusion and further work **114**

Bibliography **116**

Appendices **128**

A ROS code **129**

B Matlab code **141**

C C++ code **152**

D Contents of digital attachment **158**

E Interview with Artec Aqua **159**

F Converting CAD to URDF using SolidWorks plug-in **163**

List of Tables

1.1 Operations in smolt facilities 10

2.1 Pros and cons of hydraulic manipulators. 25
2.2 Pros and cons of electric manipulators. 26
2.3 Possible manipulators fit for operations in smolt facilities. 33
2.4 Robotic simulation software. 35

List of Figures

1.1 Salmon lifecycle of scottish farmed salmon for visualisation 3
1.2 Future innovations for salmon farming 5
1.3 Autosmolt 2025 concept 12
1.4 Magneto configurations 17
1.5 Thesis overview. 22

2.1 6 DOF revolute representation 24
2.2 Two of the interesting manipulators from Sivčev et al. review. 28
2.3 UMA - Underwater Modular Arm from Graaltech. 28
2.4 Reach Bravo from Blueprintlab. 29
2.5 SeArm from NTNU outspring Searo Underwater Robotics. 30
2.6 BionicMotionRobot from project partner Festo. 30
2.7 CDHRM manipulator from Chinese research group. 31
2.8 ROS Master node communication. 37
2.9 Typical tank designs in smolt facilities. 39
2.10 Draft one of generic robotic manipulator. 40
2.11 Specific manipulator (Reach Bravo from Blueprintlab). 41
2.12 SW2URDF definition of base for the Bravo manipulator. 42
2.13 Completed SW2URDF setup with the respective defined joint coordinate systems for the Bravo manipulator. 43
2.14 RViz visualisation of generic manipulator. 49
2.15 *Rqt* publisher in the graphical tool window. 50
2.16 Initial simulation environment with generic manipulator. 50

3.1 visualisation of transformations 53
3.2 Motion planning hierarchy 57
3.3 ROS controllers 59
3.4 Move Group Node (MGN). 60
3.5 Basic rapidly-exploring random tree. 62

4.1 Path for a half circle motion. 73

4.2 Generic manipulator in the midst of a circular path motion. 74
4.3 Joint positions for the circular motion (PID controller). 76
4.4 Joint velocities for the circular motion (PID controller). 77
4.5 Joint positions for the circular motion (PD controller). 79
4.6 Joint velocities for the circular motion (PD controller). 80
4.7 Generic manipulator definitions. 82
4.8 Workspace generic manipulator points. 84
4.9 Workspace generic manipulator poses. 84
4.10 Workspace generic manipulator poses within bounds. 85

5.1 Joint definitions of BluePrintLab's Reach Bravo manipulator. 87
5.2 Reach Bravo with cleaning tool. 88
5.3 Bravo with feeding gun. 88
5.4 Bravo with cleaning tool. 89
5.5 Specific manipulator (Bravo) definitions. 90
5.6 Workspace for the Reach Bravo manipulator consisting of points. 91
5.7 Workspace Reach Bravo manipulator with alpha shape. 92
5.8 Exception handling. 93
5.9 Path planning for autonomous cleaning operations. 94
5.10 Spiral path. 95
5.11 Feeding path with constant height. 96
5.12 Feeding path with projected z value. 97
5.13 Feeding path using nearest neighbor. 97
5.14 Grasping path. 98
5.15 Cleaning simulation, ongoing. 100
5.16 Path following of cleaning operation 101
5.17 Joint positions for the cleaning simulation. 102
5.18 Joint velocities for the cleaning simulation. 103
5.19 Feeding simulation, ongoing. 104
5.20 Path following of feeding operation. 105
5.21 Joint positions for the feeding simulation. 106
5.22 Joint velocities for the feeding simulation. 107
5.23 Gripping simulation, ongoing. 108
5.24 Path following of gripping operation. 109
5.25 Joint positions for the gripping simulation. 110
5.26 Joint velocities for the gripping simulation. 111

F.1 Initial setup assistant for SolidWorks to URDF plug-in for the Bravo manipulator 164
F.2 SW2URDF definition of base for the Bravo manipulator 164
F.3 Configuration of joints 165
F.4 SW2URDF joints configuration 166
F.5 SW2URDF links configuration 166
F.6 Completed SW2URDF setup with the respective defined joint coordinate systems for the Bravo manipulator 167

Abbreviations

ROS	=	Robot Operating System
URDF	=	Unified Robot Description Format
RAS	=	Recirculating Aquaculture System
MAB	=	Maximum Allowed Biomass
FTS	=	Flow Through System
TRL	=	Technology Readiness Level
DOF	=	Degrees Of Freedom
HSE	=	Health Safety and Environment
SDF	=	Simulation Description Format
SRDF	=	Semantic Robot Description Format
CAD	=	Computer Aided Design
TRL	=	Technology Readiness Level
ZPDM	=	Zero Pressure Difference Method
DOF	=	Degrees Of Freedom
GUI	=	Graphical User Interface

Chapter 1

Introduction

The salmon farming industry is rapidly shifting from small-sized countryside facilities to larger and more industrialised facilities and solutions, both for the land- and sea-based segments. Salmon is a premium and sought-after source for protein, and up until now, only a few countries have been lucky enough to have the geographical facilities to produce salmon due to discerning in terms of surroundings, especially temperature. Therefore, most salmon farming has been done within certain latitude bands in the northern and southern Hemisphere such as in Norway, Chile, Canada and Scotland [1]. These countries have been fortunate, and Norway as the most significant player in the Atlantic salmon market stood for export of 1.1 million tonnes of salmon corresponding to a value of 72.5 billion NOK in 2019 [2].

Traditional salmon farming consist of both land- and sea-based stages whereas all juvenile fish is hatched and grown in land-based facilities up until they become a certain size before they go through a metamorphosis and gets ready for their sea-based growing phase. This is just like a wildlife salmon would be grown out in their freshwater lake of origin before moving to the big and scary saltwater ocean to fulfil their life cycle. The sea-based growing phase is in constant threat of external factors such as sea lice, algae blooming and escapees through net-pen holes which in Norway accumulated to 171 thousand reported escapees of bred salmon in 2019 [3]. Also, there are many challenges associated with competing claims for useful areas along the coastlines of the dominating supplying countries, which is limiting the growth in sea-based production. Therefore, there has been a shift towards increased efforts within post-smolt production and full-scale land-based production in recent years [4]. This has resulted in enormous spending on new, large and more technological smolt production facilities for increasing the land-based phase of the salmon farming, hence decreasing time in the sea and the threat from mentioned external factors, and maximise utilisation of the maximum allowed biomass (MAB). Also, there have been significant leaps within full-scale land-based salmon farming requiring even bigger facilities that can ultimately make it possible to farm salmon anywhere in the world. This last segment has seen great interest in the US and China, to name a few.

In this environment of ever-increasing size of facilities, a lot of the traditional manual operations could and should be automated to increase efficiency and profit. Automation could also cope with some of the tedious and hazardous operations to get a better working environment for the employees such as carrying or cleaning tasks. Also, potential risks to the fish welfare due to sub-optimal manual operations can be reduced. There are little available information related to fish welfare parameters but it is found that the mortality rate varies greatly in between facilities, pointing out production and routines as the cause of the large differences [5]. It is therefore of great interest to explore requirements, specifications and possible solutions to automate some if not all manual operations by applying robotics.

1.1 Introduction to smolt production

A normal land-based smolt production site typically consists of three dedicated areas or units, namely the hatchery, start feeding and growth feeding. These production units are typically divided into individual departments for hygienic and practical reasons as mentioned in the interview with Artec Aqua in Appendix E. Naturally, the sizes of these areas and their corresponding tanks increase with the increased size of the fish. For land-based salmon farming a new stage namely the grow-out stage can be added to the list. This stage can be subdivided into different units based on the size of the fish. This section will present a short overview of a typical salmon farming life cycle and trends in the industry. Since there is lack of extensive research and published articles on adapted practices and use of technological tools in smolt production the cited sources are mostly based on public information from firms within the segment, industry partners of Autosmolt2025 project and web articles.

Overview of a salmon farming life cycle
A typical farmed salmon life cycle can be schematically illustrated as seen in Figure 1.1 and will now be explained using on the following sources [1], [6], [7], [8], [9] and project work in the Autosmolt2025 project [10]. This figure is borrowed from Scottish Fish Farms. Therefore, the illustrated months does not necessarily apply for the different stages. The first stage in the process is the fertilisation and hatching of eggs. The eggs and sperm have their origin from bred fish, called "brood-stock" either locally by the facilities own inherit board or bought from external providers specialised in salmon breeding. This first process stage happens in a secluded area at the facility. It is a semi-automatic process where hatched eggs automatically end up in a batch (hatching tub) where it moves on to the next stage, which is called yolk sac feeding. In this stage, the now called alevin lives on a nutrient-rich sac that hangs below its body (the yolk sack is the orange pouches under their bellies and are often called "lunch bags"). They are bad swimmers at this stage, mostly staying in the bottom of their tanks. The lunch bags last a month typically (4-6 weeks) before the alevin evolves into the third stage, namely the initial pellet feeding stage where the fish is called fry and is about 2-4 cm in length. In this stage, the juvenile fish start to accept pellets as feeding and are moved into small tanks. Once the fish gets black marks on their sides, the fish has entered the fourth stage where they are called parr and are ready

for their extensive freshwater growth. At this stage, they are moved into larger tanks. Once the parr has reached a weight of 60-100 grams, they undergo a transformation process to get ready for seawater life. This process is called smoltification and is governed by light and temperature conditions which are typically controlled in the facility to optimise this transformation. The process causes major physiological and morphological changes in the fish to be able to tolerate a saltwater life. After this transformation, the smolt is typically transferred to sea cages for saltwater growth up until slaughter weight of 4-6 kgs. The total freshwater cycle takes approximately 10-16 months, whereas the following seawater phase lasts around 12-24 months, resulting in a total production cycle length of an average of three years. The seawater phase is estimated to be the most vulnerable phase due to uncontrollable challenges.

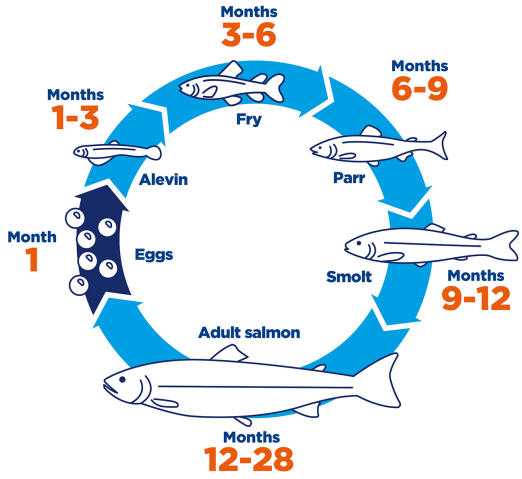


Figure 1.1: Salmon lifecycle of scottish farmed salmon for visualisation borrowed from Scottish fish farms homepage [7].

The initial growth phase in seawater is called the post-smolt phase. A lot of effort and work is being made within this phase to move it inside facilities to increase the contained and protected growth phase of the fish. This land- or sea-based phase takes place in large tanks since post-smolt growth typically goes from the initial smolt size of 60 grams and up to 1000 grams before final sea cage placements. Much effort is being put into the last part of this growth phase testing brackish water using recirculating systems (RAS) or partly sea-based cages by Nofima under their CtrlAQUA centre of research [11] and other institutions and industrial players. The intention of the increased time in land-based facilities is to have larger and more robust smolt put out in the sea cages and thereby have a shorter

production time in the environment where it is exposed to the highest risk in the production cycle. If fish is placed at sea at 1000 grams instead of traditionally 100 grams, the sea growth period can be reduced from today's typical 16-22 months down to 10-11 months [12], thus increasing production per sea-based facility which is strictly regulated in terms of MAB (maximum allowed biomass). Also, the hypothesis is that the sea transferred fish will be more robust and therefore have a lower mortality rate than "normal" sized smolt. Another strong incentive for increased land-based fish farming is that the production permits for land-based farming are free since 2016 [13], only needing local approval while sea-based permits typically costs between 120-250k per tonne of biomass [14]. The lowest prices in this price range were the original fixed prices for permits sold by the Norwegian government whilst the interval up to 250k was obtained from an auction of permits in 2018 by Directorate of fisheries. Due to this high cost of sea-based production permits, it is a necessity to optimise the utilisation of available biomass by always having a full cage of healthy fish with a short turnaround.

The industry trend is turning towards this way of optimising production and large industry players especially in countries with a challenging geographical base such as USA (south) are working extensively to produce full-grown salmon in land-based RAS facilities such as in [15], the world's largest land-based facility for farmed salmon placed in Florida. Due to the prices of production permits, there has also been increasing interest in full-scale land-based facilities in Norway. As of 19^{th} of May 2020, Salmon Evolution started construction of what is to be Europe's largest land-based salmon farming facility with a capacity at full-scale of 36k tonnes [4]. Andfjord Salmon is also constructing their new land-based facilities in Vesterålen where they are utilising a best of both worlds approach using nutritious arctic seawater in their land-based farming facilities [16]. One of the players that have come the longest in the land-based segment within the borders of Norway is Nordic Aquafarms which by May 2020 delivered its first batch of slaughtered full-scale farmed salmon [17]. Followed by a lot of other players in the market, this will get much focus in the years to come even though there are numerous uncertainties regarding the actual capabilities of land-based salmon farming since it is still early in the development stage. From CtrlAQUA's report from 2017, the trends in the industry are presented, and it can be clearly seen that the trend pushes towards multiple and diverse post-smolt growth strategies together with land-based farming as seen in Figure 1.2.

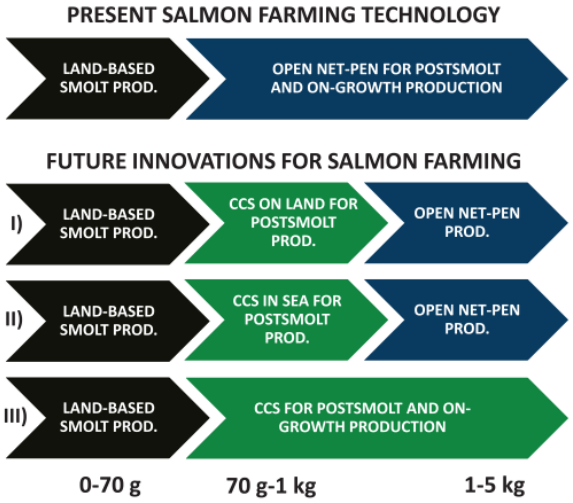


Figure 1.2: Future innovations for salmon farming. Illustration: @ Bendik Fyhn Terjesen/CtrlAQUA SFI [18].

This increased land-based focus has resulted in many investments in huge land-based facilities such as the multi-sectioned state-of-the-art facility of Bakkafrost at the Faroe Islands with a tank volume of 29 km^3 [19]. Alternatively, the new projected facility of Salmar to be built from 2021 in Verran [20] which will be the worlds largest smolt production facility when completed. Sinkaberg-Hansen is constructing a new massive post-smolt facility at 13k square meters intended for smolt within the size range of 400-1000 grams [21]. These considerable investments in land-based facilities result in a large backlog for suppliers of RAS facilities such as AkvaGroup [22] and ScaleAQ [23] to name a few.

Ernst & Young looked into the available system solutions and technology in land-based fish farming in their annual aquaculture analysis from 2019 [24]. The bottom line of their report was that there is a definite shift towards the use of RAS compared to FTS (flow-through systems) from a technological perspective. Also, there is a clear trend of significant investments in grow-out land-based salmon farming outside of Norway, hence farming closer to consumer markets, and that this is the way to go from a global environmental and social-economic perspective. This shift towards more extensive use of large RAS facilities is a paradigm shift in the industry and has been going on for a while, but exploded in recent years.

AkvaGroup's annual report shows a substantial increase in orders for land-based farming facilities. Almost half of their backlog is land-based RAS facilities, and a considerable increase is expected in the coming years [22]. These significant investments within the land-based segment are set to create a whole new market for technological solutions such as continuous monitoring and controls, but also for autonomous solutions for close to all operations due to increase in both sizes of facilities, tanks and not at least fish.

1.2 Challenges and demands in land-based fish farming industry

This section presents a review of smolt production facilities. Starting off with the design of a typical facility before moving on to what and how operations are done today, what could be automated and why, and ultimately describing these challenges that could be solved by an autonomous system.

1.2.1 Smolt production site information

The tanks in salmon production facilities are mainly circular, even though some suppliers operate with octagonal shaped tanks. The tanks are mostly made of fibreglass-reinforced plastic (polyester) with a gel coated surface; hence they are not ferromagnetic. Diameter and height vary a lot based on site and purpose stated by the project partner Brimer AS. From a survey done in [10], this was enlightened by project partners that tank sizes could correspond to diameters of respectively 5-6 meters for empty tanks and up to 20 meters at operational Flow-through system (FTS) facilities, while it is planned for circular tank sizes up to Ø28 meters or about 40x40 meters for octagonal-shaped tanks in various facilities that are under construction. ScaleAQ is operating with octagonal shaped tanks called OptiTank for optimal hydraulics [25] as installed at Tverrvågen in Frøya, a facility owned by Vikan Settefisk. From an interview with Artec Aqua documented in Appendix E, the statement of varying shapes and sizes of tanks were supported. Either way, as an example for smolt production from birth to grow-out phase (0-200 grams) it was stated that the tank environments through the cycle could typically be; hatching cabinets, Ø5, Ø8 and Ø10-15 meters in diameters of the tanks. For larger post-smolt it was stated that typically tanks of diameter Ø28 meters such as the once at Salmon Evolution were used. Almost all of their projects were using circular tanks, but this was dependent on customer needs.

As for general facility design, the interviewees in Appendix E point out that most of their new facilities are centrally or remotely controlled but do not necessarily use a dedicated control room. The control and monitoring are typically connected with digital pads that can be connected from anywhere or more dedicated pads at stations such as the tanks. Facilities are typically split into dedicated sections for the different stages of the fish such as a section for hatching, juvenile growth and grow-out. However, there is no correct answer for the exact sectioning. The sections typically consist of large industrial halls with all necessary equipment mounted such as tanks, pipes, hoses, pumps, etc. If possible, it is always tried to keep the path from the tank to the ceiling clean of objects to mitigate unnecessary threats to fish welfare such as falling objects or leakages.

The land-based farming sites vary greatly in size. However, the ever-increasing cross-section area of the facilities makes it time-consuming, hence expensive to have personnel walking in between workstations. This militates in favour of multiple dynamic robotic systems stationed inside a specific work area doing multiple necessary tasks.

1.2.2 State of the art technological development in the industry

The technological development within the smolt segment of the salmon farming industry has been focused almost exclusively on systems for quality monitoring and recirculation of water up until recently. These features are of course a necessity for the possibility of doing land-based fish farming at all, but after quite a few critical mass deaths in RAS facilities [26], these systems are looking ever better. Unfortunately, in older facilities, little focus has been targeted on implementation of autonomous systems and most operations are done by manual means. Even though the use of autonomous solutions has increased in new facilities, there is still room for improvement.

Much work is currently being done within the implementation of sensor technology to increase monitoring and insight of fish health and growth. This is done by optimising living conditions such as water speed in the Fitsmolt research project [27], or estimate average weight of fish inside the tank (biomass measurement) done using the Akvavision system from Nofima and Vard fisheries [28]. Also, as can be seen from the last couple of year's Tekset [29], a conference for technology and innovations in the fish hatching industry. The suppliers and players in the market are struggling to fully implement and optimise the RAS facilities, acquire the right competence (increased use of technology) and implement new sensor software and technology. The same can be concluded from the conference meetings; Smolt production in the future [30], a conference on recirculating aquaculture. There is little to no direct literature on automation or robotic solutions in the smolt production industry. This confirms the prejudice that the industry is mainly based on manual or semi-automatic operations. Nevertheless, from the interview conducted with Artec Aqua a little bit different viewpoint were found as seen in Appendix E. They are a relatively new contractor in the market and have delivered quite a few fish farming facilities in recent years. They are lately experiencing that the producers are willing to invest in autonomous solutions where value is created instantly, such as for monitoring, feeding and a variety of transportation solutions.

There are a lot of large players in the process equipment industry for land-based fish farming such as Scale Aquaculture, Optimar AS, Skala Maskon AS, AkvaGroup, Vard Aqua AS with more. Common for all of these players is that their designed solutions within the smolt production segment often are aimed towards specific operations. In the knowledge of the author, there exist no industrial system for addressing the whole or multiple parts of the production system when it comes to autonomy and optimisation of the operations even though this is a fast-growing field. A facility that is closing in on the goal of full autonomy and become a lodestar in the business is Vikan Settefisk [31], a facility designed and developed by the project partner Scale Aquaculture. The level of technological implementation at different smolt facilities varies greatly. However, most new facilities are using automatic or semi-automatic solutions for some operations. For vaccination, vaccination machines such as the one from Skala Maskon AS [32] or Aqua Culture Supply [33] are used. Further, a pump system for the transportation of fish in between tanks such as [34] (or rudimentary and homemade solutions) are typically in place. Finally, a mechanised system for feed transport and feeding such as the system from [35] that uses a dispenser and a screw to spread out pellets along a linear path over the tanks are used. Besides, it is common that

the tanks are equipped with collection points for dead fish often in drain boxes that have to be emptied manually, or they are directly transported to a cubicle. Andfjord Salmon has chosen a novel tank cleaning- and monitoring solution developed and delivered by Mørenot Robotics for their new facilities in Vesterålen [36]. They aim at using high tech features in their production and has therefore opted for electrically driven ROV's for daily cleaning and inspection of the base and walls of tanks. The goal is to achieve continuous cleaning of the tank surfaces and prevent formation of sediments, biofilms and propagation of potentially pathogenic microorganisms, hence maintain good fish welfare in the tanks. The ROV's will be connected by cables to the surface area, use thrusters for propulsion in the water environment and be equipped with either suction tools for sludge removal, brushes or inspection tools (cameras). It is intended to use 5 ROV's per tank, two specialised for base operations (sludge), two specialised for wall operations (brush) and one for inspection purposes (camera). Other automated solutions within the smolt segment that should be mentioned is the automatic roe farming robot from Alvestad [37], named *AutoTend*, their novel RAS system for the hatching phase named *KUBEhatch*, the water quality monitoring system from Blue UNIT [38] and a variety in nozzle systems for tank cleaning from HL. Skjong [39].

1.2.3 Factors that advocate automation

A survey presented at the smolt production conference [30] by CtrlAQUA [40] on tanks and tank hydraulics points towards a markedly increase in the use of larger culture tanks in land-based RAS facilities. Also, they state that tanks are operated at a lower intensity per unit flow (compared to 10 years ago) to optimise water quality throughout the tanks. They also present the challenge of concurrently clean settleable solids in the large tanks, pointing towards an increased need for a cleaning solution that can work during production or the need for more frequent cleaning.

The trend with an increase in both facility- and fish size will create substantial challenges and potential problems for manual work methods. Therefore, it is of interest to explore the possibility of creating an autonomous robotic system to relieve some of the cumbersome or tedious work operations from today's operator. Such systems might increase the efficiency of each operator as well as free up time for essential tasks related to fish welfare or other tasks that require reasoning. The increased use of RAS facilities and sensor technology also demands a higher technological understanding of the smolt farming operators of the future, which also will be beneficial for doing corrections and maintenance on a possible robotic system.

Labour-related costs constitute one of the most significant expenses in smolt production. Looking at a profitability assessment in the salmon and trout industry from the Directorate of Fisheries from 2015 [41], the labour costs are the most considerable expense in the production cost of each smolt and fry, and the third-highest cost if only the smolt are accounted. In 2018 the number of employees in the smolt production industry was 1773 [42], assuming 30% of these employees working in the administration and an annual gross salary of 542 760 NOK as the average salary for aquaculture workers in 2018 [43], the expenses accumulate to approximately 673.6 MNOK. In other words, creating robotic so-

lutions capable of handling unnecessary manual operations would mean enormous cost savings since some of these working hours can be used more wisely on tasks for increased fish welfare and other tasks. The implementation of robotic solutions will contribute to new positions within the smolt farming segment, both more technological operator work requiring higher tech-competence and possibly increase positions within biology, R&D and fish-welfare. Also, the risk barriers of the working environment will increase, resulting in a safer workplace for all operators.

Based on the stated facts it is of great interest to create new technological tools and systems such as a robotic solutions for solving the regular and time-consuming operations in a more efficient and possibly precise way than done today. Using the extensive work by SINTEF Ocean in the Autosmolt2025 project and findings in this report, some of the typical operations in smolt facilities and their level of autonomy are presented and discussed in the next section. It is reasonably apparent to believe that these operations, in addition to some extensions, are also present in full-scale land-based fish farming facilities.

1.2.4 Operations in smolt production

All identified operations within smolt (or land-based) production are presented in table 1.1. It is natural to believe that some if not all of these operations, will also be applicable in full-scale grow-out facilities. It is worth mentioning that the identified operations with their corresponding level of autonomy are mapped discretely through project partners, available producer information and interview. The level of technological utilisation for the different operations might vary significantly from facility to facility, especially for new facilities that are either under construction or recently started producing. Another note on this topic is that there is little to no available literature on the field, which is one of the main reasons for the initiation of the Autosmolt 2025 project. It is also worth noting that a lot of producers, suppliers or project coordinators are not publically branding their most novel solutions in detail.

From a survey done within the Autosmolt2025 project, the following was highlighted when it comes to the general level of autonomy or potential level based on today's available solutions. The feeding operation is for most facilities partly automated, either using a feeding system (often a screw principle) that is manually filled or using a centralised system such as the new feeding systems from Vard Aqua [44]. Their main system, the exact feeding robot is an automatic precision feeding robot that is transported using a ceiling mounted system. Therefore, it is not stated as a critical operation even though some of the survey respondents mention this and the feeding decision basis as important areas of improvement. The project partners state the removal of dead fish as an essential task to address, especially connected with a tracking system for relevant data. Also, the removal of other waste is identified as a possible area for autonomous solutions. The grading, sorting, transferring, and vaccination operations either is or could be automated using existing commercial solutions. Nevertheless, existing transportation solutions are identified as a significant stressor for the fish; hence it is desired to opt for more cautious solutions if possible. The vaccination operation is typically involved in the transportation part as an extra stoppage point where there exist fully automated vaccination solutions for most fish

sizes. Cleaning operations are implemented in very diverse ways both with regards to intensity, form and surroundings. A few facilities aim to have fully automated solutions that will operate when the tanks are empty of water. Some are cleaning manually within the same time frame and surroundings, while some are manually cleaning while biomass is inside the tanks using brushes. Others are using a combination of these strategies. The time spent on cleaning operations can vary greatly based on approach and intensity. Either way, it is mentioned as one of the most critical operations to automate, reflecting the importance of the outcomes of this thesis aiming to implement and demonstrate an automated and modular cleaning robotic systems for smolt production units.

The findings from the interview with Artec Aqua in Appendix E underlines most of the presented statements even though they present a higher level of autonomy for their new facilities. On this note, it is worth mentioning that Artec Aqua in a new contractor that has been part of many new projects in recent years, hence might not be too involved in the segment of older facilities that are still operating. Their viewpoint was that available centralised feeding solutions were working great without manual intervention. The solutions for dead fish removal was adequately working requiring only manual removal from a container which is included in their fish welfare monitoring routines. Further, the cleaning is mostly done manually today because of costly cleaning solutions in comparison with the versatile manual labour. The importance of not making scratches inside the tank walls could not be stressed enough, which in terms will increase the requirements of the transportation of a robotic system.

Based on these findings, the identified operations have been outlined in table 1.1 consisting of the name of the operation, a short description of available solutions, the level of autonomy that available solutions offer, and an identified potential for an increased level of autonomy or need for new novel solutions. For full survey details, please look closer into released articles connected to the Autosmolt2025 project [10].

Operation	**Description**	**Level of autonomy**	**Potential [1-6]**
Feeding	Feeding screw or other system	(Semi) automatic	4
Grading and sorting	Sorting solutions	Automatic	2
Vaccination	Vaccination machine	Automatic	1
Transferring	Various pump and hose systems	Automatic	2
Tank cleaning and disinfection	Various methods	Mostly manual	6
Pipe cleaning and disinfection	Not done	N/A	5
Removal of dead fish and waste	Manual and lures	Manual/Automatic	4

Table 1.1: Operations in smolt facilities.

1.2.5 Selected operations facilitated for automation

Cleaning and disinfection of production units and equipment are essential for biosafety and hygiene. As mentioned in [45], the hygienic procedures are primarily a benefit to fish welfare. They can only be a danger to welfare if cleaning is done while the fish is in the system and if residues of potentially harmful substances remain in the water. The challenges in such cases are physical damage, stress associated with the disruption and

effects of toxic chemicals. The report also states that there are signs that daily or repeating disorders are less harmful than rare and persistent disorders. It is believed that the fish tend to adapt to regular disorders, which is pointing towards that a "continuous" cleaning solution could be a more gentle approach than moving the fish in between tanks for cleaning. Another interesting statement in the article that points towards the same is a potential problem with gill pathology after hygienic procedures. Some often used chemicals in cleaning procedures can damage gills. A reduction in the use of disinfectant and detergents by continuously cleaning can decrease the likelihood of this happening even though this is not necessarily possible at all stages due to the biological risks between batches. The cleaning and disinfection of tanks in the production facilities is today done as described in 1.2.4. Cleaning in the production cycle is typically done manually by scrubbing the water layer edge of the tanks, or the biomass is moved to a separate tank while cleaning is completed, else it is not done at all. In between production cycles, the tanks are emptied, and a total washdown is done. For this, cleaning personnel is entering the tank with equipment such as high-pressure water guns and brushes to clean the tank walls. This imposes the workers of a hazardous environment when they are using chemicals to clean the concentrated tank environment. Besides, the tanks are huge, and there is no straight forward way of reaching hard-to-reach spots efficiently resulting in time-consuming cleaning.

The Norwegian veterinary institute did a study on mortality data and juvenile fish welfare in freshwater facilities in 2019, resulting in the following interesting facts [5]. The highest mortality was found to be for fish below 3 grams and may be due to relocation from hatching units to the small tanks (open environment with no hiding and constant light) for first feeding. In order for all the fry to have equal opportunities to grow, the right amount of feed must reach the fry at their location. Survey respondents from the industry mentioned that overfeeding could easily create muddy water in the tanks, which in turn could cause inflammation of gills and increased mortality. Insufficient feeding, however, will result in uneven development of fry and thereby cause an increased need for sorting. This identified point in the fry phase is also believed to be relevant for the other phases of the smolt production cycle. A more precise and customised feeding solution could result in more uniform growth of the biomass and cope with unnecessary feeding and sorting resulting in muddy waters; hence increased need for cleaning and separation.

In addition to the two defined operations, it was of interest to look into an operation that could be duplicated for other situations. Therefore it was chosen to look at the operation related to the removal of dead fish and waste to, for instance, be able to use similar control approaches for, i.e. movable camera monitoring inside the tank. The selected operations to address a novel robotic solution for was therefore condensed down to the numbered objectives presented next:

1. Cleaning and disinfection: Creating an autonomous cleaning solution of the tanks that optimally can work while the biomass is inside the tanks to increase uptime, decrease the use of chemicals and ultimately remove a hazardous work station for manual workers.

2. Feeding: Creating a more precise and controlled feeding solution than today's mechanical solution, being able to reach or specify feeding patterns.

3. Removal of dead fish and waste: Create a solution that in addition to the tasks mentioned above has a vast reach and possibility to cope with floating waste and dead fish that can create a lousy tank environment.

When it comes to sorting, vaccination, transport or similar tasks, these were found to be adequately automated or could possibly be automated using existing industrial solutions. These tasks were found to be inappropriate for a portable robotic system intended in this thesis. When it comes to the pipe cleaning task, this was also found to be infeasible for the intended robotic system due to the desired size for solving the other objectives. A conceptual figure of the intended outcome of the Autosmolt2025 research project that this thesis is a part of can be seen in Figure 1.3. It is natural to believe that it is impossible to create one robotic system that can cope with all of the challenges within a smolt production facility. Therefore it was chosen to aim at defining one robotic system that could do selected tasks that in a way, are similar.

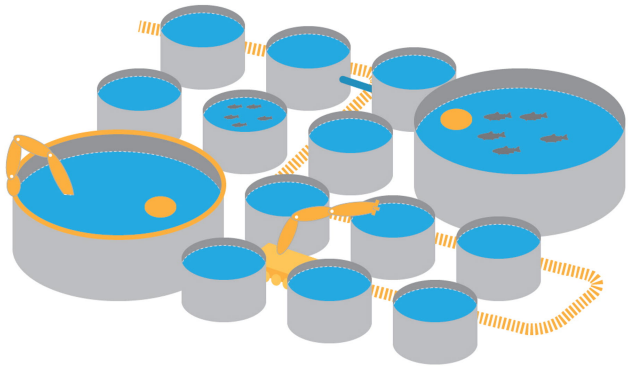


Figure 1.3: Autosmolt 2025 concept [10].

Even though the chosen operations in some ways are similar with regards to using a robotic system for solving a trajectory following problem for cleaning, feeding or gripping (removal), this thesis has a particular emphasis on the first-mentioned operation, namely the cleaning task. This has been emphasised due to the identified potential for autonomy and the hazardous characteristics of the operation. In addition, solving this problem could give substantial synergies into other operations where the created robotic system could easily be adapted also to solve these operations using different end-effector tools.

1.3 Related industries

When exploring the possibilities of increasing autonomous operations in a field such as the smolt production industry where there has been little to no implementation of autonomous or robotic solutions it is of great interest to explore technological adaptations done in similar industries and look to more technologically advanced industries to find potential applicable solutions or ideas.

1.3.1 The rest of the salmon farming chain

An obvious starting point is to look within the rest of the salmon production cycle for related technologies to be used in addition to the already stated solutions in section 1.2.2. A related product is the possibility to use similar technology to the net cleaner from Akva-Group which is a net cleaning tool coupled with an ROV that uses high-pressure water to clean the net underwater and a collector solution called the Spider to collect fouling and pump it to the ground from where it is operated [46]. This last part must be a natural part of the potential cleaning solution for the tanks to avoid muddy water during cleaning to spare the fish from unnecessary danger. In 2019, SINTEF ocean with Nina Bloecher in the lead [47] did a study of net cleaning principles in aquaculture regarding what type of cleaning was most gentle and effective. Their study showed that cavitation-based cleaning (using brushes) was a very promising principle for cleaning of biofouling due to the gentleness on the coating and satisfactory efficiency with regards to biofouling removal. It is natural to believe that this could also be applicable inside tanks to avoid unnecessary wear and tear on the gel coating of the tanks inside. Cavitation based cleaning will, in addition, most likely yield less turbulence in the water and facilitate for an easier collection of dirt than using high-pressured water. Most of the other operations in the production cycle of salmon are either done manually or semi-automatically. Typical for the slaughtering process is a lot of customised automatic solutions for, i.e. categorisation, be-heading and filleting while most other operations are done manually, as presented in previous work by the author, Skeide 2019, [48]. When it comes to the net-pen growing phase of the fish, most of the operations are done similar to the smolt production, only more extensive use of robotic solutions (even though mostly manually controlled) for underwater operations such as inspection, monitoring, cleaning etc. have been applied.

1.3.2 The agriculture industry

Another interesting and similar industry to look into is the agriculture industry that also involves consumer goods. An extensive literature review was conducted by Jostein Vik et al. [49] regarding new technology in the agriculture industry released in early 2020. Most of the technology presented is directed towards precision farming or industry (agriculture) 4.0 with a particular emphasis on data collection and analysis. The presented technology solutions show that the use of big data and IoT (internet of things) from sensor technology is the most mature technologies while robotic solutions for precision farming are ambitious and are further away from being industrially applicable. Nevertheless, it is these robotic solutions that are of most interest to the topic of this thesis for reaching a higher degree

of autonomous operations in the smolt production industry. Robotic solutions in the agriculture industry are often divided into their specific category, which is either robots for specific operations or versatile robots that can cope with numerous different challenges. Most robotic solutions within the agriculture industry are customised for their specific operating conditions; hence they are typically created for operations within a relatively flat surface using wheels for mobility and are not meant for excessive water contact. Nevertheless, it is interesting to research the level of autonomy in this related industry and find inspiration, especially within robotic operations related to robot vision, picking motion and solutions for efficient manipulator reach. The identified robotic solutions within these respective categories will now briefly be presented, starting with the robots for specific operations.

In the segment of custom robots for specific operations, there have been quite some developments both within robots for commercial and professional use. In the sub-segment of commercial or lightweight agriculture robots, it is worth mentioning the Farmbot [50]; a Cartesian X-Y-Z based open-source robot that can plant, weed and water inside its workspace. It is a commercially available product but is very limited due to its strict workspace and does not take outside conditions into account. Nevertheless, it is an exciting idea to create a similar rail-based system for reaching and doing repeating tasks. Another interesting concept is the use of something similar to the solar-powered garden weeding robot Tertill [51], which is a small-sized weatherproof robotic system that patrols the garden and uses mechanical principles for removing weeds. It works much like a robotic lawnmower where it chops all growing plants that is less than a centimetre. Using similar principles for a continuously cleaning and monitoring system in tank environments by moving around on its surfaces will yield minimal disturbances. Finally, the TerraSentia [52] is a small-sized monitoring robot that is measuring plants heath, size, biomass and diseases using visual cameras, LIDAR and other onboard sensors. Using similar sensors on a dynamic robot system within smolt tanks could benefit fish- and facility health.

There has also been much work done within more professional or industrial solutions in the agriculture industry such as the Agrobot [53], an autonomous strawberry picking robot with up to 24 independent robotic arms. Each of the arms has an integrated camera to analyse the state of close by strawberries to categorise if they are ready for harvest. The controls and vision of the multiple dynamic arms for harvesting the correct berries can be of great interest in a gripping solution in smolt production. Similarly, the Norwegian based robot system Asterix [54] could be of interest for precise robot vision that it uses for precision spraying of weed.

When it comes to the versatile robotic solutions most of these are either a supplement or a replacement of the traditional tractor to handle multiple of the large scale and demanding operations that need to be done in an industrial-sized agriculture farming site. It is, therefore, only of interest to explore the modular-based solutions including some robotic interaction with the crops. The Norwegian developed solution named Thorvald from Saga Robotics [55] is one on the solutions with the highest TRL (technology readiness level). The solution is modular and can be customised to fit multiple plant rows, single row and

be based on a variety of equipment to be used. The solution is at first intended for disease management and harvesting of fruits and vegetables. In addition to this solution, there is an English company called Small Robot Company [56] that is working on some exciting designs of versatile robotic solutions that are formed as wheeled spider robots such as their concepts named Dick and Harry.

1.3.3 Oil and gas industry

The oil and gas industry is an industry where a lot of novel solutions, and automation is happening due to the transition to industry 4.0. Especially on the Norwegian continental shelf, most players are aiming at moving many of the offshore stationed personnel onshore in order to reduce costs and increase HSE (health, safety and environment). Robotic and autonomous solutions play a vital role in this transition. To this date, the focus has been on more autonomous operations such as monitoring and controls of drilling and wells. However, a lot is happening, especially withing the inspection and monitoring segment where robotic solutions are being adapted. Examples of novel and industrialised approaches are, i.e. the bold attempt by Cognite to use Boston Dynamics robot Spot for inspections in the harsh environment offshore [57], the use of the snake robot Eelume (NTNU outspring project) for inspection and maintenance subsea [58], or OceanTech's robotic solutions for cleaning, inspection, repair and modifications in the dangerous splash zone of offshore installations [59]. A common characteristic of most robotic solutions for the oil and gas industry is that they are made for operations in harsh environments, hence almost all robotic manipulator solutions found in the industry are hydraulically powered, pressurised and often heavy. This, unfortunately, implies that there are few directly suitable solutions to be used in smolt facilities. Regardless of this, the use of the Eelume AUV and the robot Spot for inspection and maintenance in such a demanding environment offshore most likely will pave the way for similar but more simple solutions in production facilities onshore.

1.3.4 The manufacturing industry

SPARC (The partnership for robotics in Europe) created in late 2016 a multi-annual roadmap [60] about robotics to 2020. The roadmap provides an overview of technical and market details regarding robotics in most large industries such as manufacturing, healthcare and agriculture. The report pinpoints the potential for lightweight and low-cost compliant manipulators and development of mobile manipulators to be used in especially the manufacturing industry, pointing towards possibly more manipulators coming to the market soon. In general, the manufacturing industry, and especially the automotive industry is by far the largest market for industrial manipulators accounting for $1/4$ of the market in 2017 [61]. Even though this market is vast for the manipulator suppliers, the solutions are dominated by massive and stationary manipulators along an assembly line for specific operations such as assembly, welding and spray painting. Due to the high volumes pushed through the assembly lines, there are most of the time, no need for dynamic robotic systems. Therefore, most of these solutions are not directly applicable in a complicated environment such as in smolt facilities at the moment.

This thesis has a special emphasis on the upper part of the system, and it is therefore assumed that the system for movement is in place and working as desired. Such a combined system, as mentioned above, are often called a collaborative Mobile Industrial Manipulator (CMIM). Yang et al. [62] did in 2020 a comprehensive review of the development of such robots and their future outlook. CMIM's are expected to get its breakthrough in industry 4.0 especially into other industries than the once where it is already in use to some degree, which is the three industrial areas of logistics, manufacturing and assembly. CMIM has been implemented in these areas since it is very suitable and convenient for purposes such as transportation and to perform "pick and place" tasks on an industrial and flat floor environment.

1.3.5 Urban cleaning solutions

To delve into industries coping with more specific and similar problems, it is of interest to look into available literature within cleaning solutions both in air and water. Due to urbanisation, cities are growing more rapidly than ever. Due to pressure in the housing market, skyscrapers with increasing height are built. Working in the extreme heights of these buildings is seen as hazardous, and robotic solutions have therefore been requested. Due to vast variations in architecture, robots must be quite versatile to be able to both move around and reach difficult spots.

If airborne drone solutions are neglected, there are many different approaches tested to solve skyscraper cleaning problems. Kite robotics has made a cavitation based cleaning robot that is controlled by positioned wires for traverse movement [63]. Skyline robotics has gone in a more traditional direction using already installed scaffolds as a workstation for their dual robot configuration of industrial KUKA robots that uses water and a cavitation based method to clean skyscraper windows [64]. Serbot's Gekko facade cleaning solution [65] that is a fully automated cleaning robot is using suction cups as adhesion and propulsion using a rotating brush with uniform pressure for cleaning. The adhesion principle of this robot would be of great interest if it were to work as good under-water as in air surroundings. The robot uses a "Gekko"-principle where suction cups are arranged in a D-form making it able to move both straight ahead and rotate efficiently by choosing if the inner or outer part of the mirrored D-shapes is active.

In addition to the mentioned cleaning solutions, there have been some other solutions such as the bulky robotic solution of Pufeng Intelligent technology in China using suction cups as adhesion and various cleaning principles [66]. The novel biped cleaning robot designed and experimentally tested by a research group in Japan [67], or another bio-inspired biped wall-climbing robot called W-Climbot from a Chinese research group [68]. Also, the novel underwater crawling cleaning-robot designed by a Swedish research group [69] that is double linked biped underwater cleaning robot is impressive. This robot uses suction cup adhesion and actuators to move around using its high-pressure water gun for doing cleaning operations underwater.

1.3.6 Underwater solutions for cleaning operations

Other prominent candidates for inspiration with special emphasise on underwater cleaning is commercially available hull- or pool cleaning solutions or specialised inspection robots. For pool cleaning solution, most of the identified solutions are similar to the Mørenot robotics cleaning solution, only smaller and cheaper looking, typically customised for the consumer market; hence not being too expensive. Nevertheless, some heavyweight sludge cleaning solutions are commercially in use for sludge removal in large water tanks such as Scranton Robotics basin cleaning solution [70]. The system is composed of a compact underwater robot with brushes and a connected sludge pump. The hull cleaning segment has long been dominated by manual divers or old and outdated scrubbing solutions. In recent years this has changed rapidly with the increased focus on cost savings and reduced CO_2 emissions. This has resulted in a lot of new and influential players in the market such as the joint venture between Jotun and Kongsberg creating their new hull cleaning solution that uses brushes and magnetic adhesion to manoeuvre along the hull of ships [71]. Most of the available solutions for hull cleaning are working without any fouling collection, and all use ferromagnetic adhesion for movement. Therefore, these solutions must be seen as inspirational when it comes to design and not necessarily functionality for application within smolt production. When it comes to ferromagnetic adhesion systems, there are quite some available solutions used for inspection and monitoring purposes such as the novel multi-limbed inspection robot Magneto from Nexxis [72] that can be seen in Figure 1.4 or other players offering more conventional inspection robots using wheeled magnetic adhesion such as the inspection systems from General Electric [73]. The mentioned multi-limb robot was customised to be able to cope with obstacles as can be seen in the figure, which is an alluring property for application within smolt facilities.

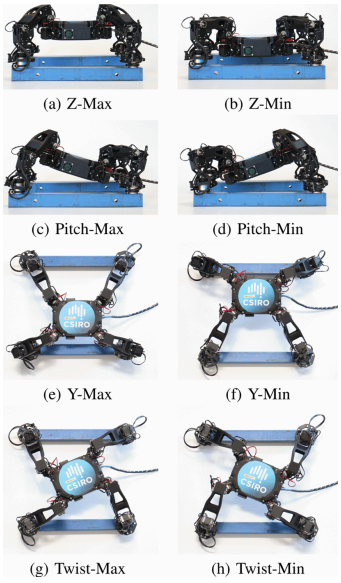


Figure 1.4: Magneto configurations borrowed from [72].

Even though most of the industrial solutions from the similar industries are not directly applicable due to the wet climate or the complex environment of operations, it is of interest to connect some of the solutions from the various industries that stand out. Based on the identified operations and the initial thoughts of a novel robotic system described in section 1.2.5 it has been identified a possible novel solution composed of a dynamically moving base coupled with a lightweight manipulator. In this regard, the lower part is intended to be similar to the Magneto or other designs mentioned in section 1.3.5, while the manipulator for doing operations that can involve everything from monitoring using similar features as described in section 1.3.2 to intervention tasks is to be defined. It is intended that the manipulator can be equipped with a variety of end-effectors for solving a variety of tasks inside its task space, as seen in the typical manufacturing industry. Either way, to make it possible to do multiple of the identified operations, the robotic system has to be movable. Ultimately this has to be done in a variety of environments that can be wet, steamy or dry. Therefore, it has been conducted a literature review on possible adhesion principles and existing industrial solutions. The base or movable frame of the robot has not been further emphasised in this thesis due to time constraints. However, it will be a natural and necessary part if such a novel robotic system as described is to be created for industrial use. Therefore, the literature review is conducted, and the transportation system is assumed to be developed and available.

1.4 Adhesion principles - A review

Even though this thesis does not explicitly aim to investigate the problem around the transportation around in the facilities or tanks, it is of interest to look into the limits or requirements this potentially can cause for a robotic solution to be applicable; hence discovering further work to be done.

There are several propulsion principles in a water-surface movement problem. These are described shortly, and their applicability with regards to the mentioned novel system will be discussed briefly. A principle is chosen as the best fit for the requirements and literature related to this principle is presented more thoroughly.

1. Mechanical force adhesion

Mechanical adhesion is typically used where there are available gripping spots for an articulated arm to hold on to, which generally is the case for many ROV intervention operations. These operations are typically done by an operator whereas, the operator locks one of the ROV's manipulators to a gripping spot while doing intervention work with the other manipulator. Other possibilities are to use mechanical nano principles as the gecko-arm's micro needles at their hands, but this is found to work poorly in wet climates and often cannot withstand more force than the robots own lightweight construction as [74]. Another option is to use wires or rails for movement as the mentioned solution for the Kite robotics solution for cleaning of skyscraper facades [63], or the typical rail-based manipulator solutions as seen in manufacturing engineering, often with an X-Y-Z functionality similar to a traverse crane.

2. Magnetic force adhesion

Magnetic adhesion uses a physical principle that requires the surface of a ferromagnetic material. The material type allows the creation of a robust normal force, hence adhesion between a permanent magnet and the ferromagnetic surface. This is by far the most used adhesion principle in hull cleaning or inspection applications for large ships as mentioned in section 1.3.6. The principle, if applicable, results in a reliable and fast adhesion and is almost unaffected by its surroundings.

3. Thrust force adhesion

Thrust force adhesion is not directly an adhesion method. However, by using the thrust of thrusters and fins, an underwater ROV/AUV can in principle "walk" on the surface underwater doing submerged surface objectives such as cleaning. This principle is mainly being used for service inspections of horisontal or vertical submerged surfaces but has been used for net cleaning applications in the salmon farming industry by for example AkvaGroup [46] and Ocean Innovations [75]. The same principle applies for flying robotic systems as drones that can be used to move along walls or objects even though this is often not desired due to imposing dangers to the flying object.

4. Suction force adhesion

Suction adhesion uses the principle of pressure difference to produce adhesion between typically a cup and a surface. The principle involves removing fluid from inside the cup, creating a lower pressure inside the cup than for the surrounding environment. This pressure difference causes either suction of the environmental fluid (leakage if there are openings in the barrier) or an increased vacuum producing a normal force for the cup to stick to the surface; hence adhesion. There are numerous ways of creating this adhesion. However, the most common ways for underwater use is to use an electrical vacuum generator or an external hydraulic vacuum generator, often connected to a top site pump. Suction adhesion can work on arbitrary surfaces of different materials and roughness. Nevertheless, there are some significant drawbacks. The vacuum generators need a significant amount of time to create enough vacuum for the adhesion force to be sufficient. Another considerable drawback is the mentioned leakage problem in the barrier of the suction cups. Small gaps in the seal can cause a drastic loss of adhesion force.

1.4.1 Summary and choice of adhesion principle

As a summary of the adhesion principles and their applicability in smolt production facilities, it is evident that magnetic force adhesion will not work for most of the tanks that are made of fibre-glass reinforced plastic as mentioned in section 1.2.1. Thrust adhesion has a potential application within the water environment of the tanks, and similar industrial solutions are available out of the box. Nevertheless, this principle can potentially have complications with regards to unnecessary stress on the fish and complicated controls, especially in combination with a robotic manipulator as outlined in [76] and [77]. Also, thrust adhesion will limit the robotic system to operate exclusively in water, which is undesirable. Mechanical adhesion is widely used in manufacturing using rails due to its reliable and straightforward mechanism. The use is typically for doing operations within

an open free space with the shape of a box such as for doing welding or assembly operations. The installation of such rails is costly and is placing significant demands with regards to facility design and installation since the rail systems probably must be installed along the roof. Also, there are important requirements for the robustness of the connection of the robots that are typically mounted upside down and are heavy. Therefore, it seems like a far fetched solution, at least for installation in existing facilities that cannot be customised for the rail systems. The remaining adhesion principle of suction adhesion is therefore chosen as the most suitable principle to be used even though it has its imperfections. The versatility of suction force adhesion, if correctly configured, making it possible to overcome various surface materials and coping with different surroundings is a great advantage. Some typical design choices have to be made when using suction force adhesion for a moving robot system. The necessary suction force is a product of the weight of the robot system and external factors such as surrounding fluid, flow, obstacles and surface. Besides, numerous suction force principles can be applied, some embedded that can create the forces necessary inside its own body and others that are connected to pumps and generators at the top site. The latter principle is probably the most interesting since the robot system, regardless has to be connected to a pump in order to transport sludge away from its path. The next paragraphs will shortly describe some interesting articles about suction force adhesion principles that can be of inspiration when designing the movable base part of the intended robotic system.

1.4.2 Literature review within selected adhesion principle

Suction adhesion has gotten increased attention lately, especially for use in robotics. A research group from Zhejiang University in China explored in early 2020 the possibility of creating a vacuum suction unit based on the zero pressure difference method (ZPDM) [78]. The principle is based on creating a rotating water layer at the boundary of the cup to eliminate pressure differences that are typically present between the vacuum and the environment due to surface roughness. Their experiments showed that a ZPD suction unit of about 0.8 kg could generate a suction force of over 245 Newton on rough surfaces with a power consumption of less than 400 Watts. In comparison, a traditional suction unit of the same size would need a colossal vacuum pump weighing multiple kilograms consuming several kilowatts to generate and maintain the same suction force because of considerable leakages.

The research group also designed and tested a hexapod with the developed ZPD units at each foot. The robot itself weighed 16.5 kg, and a payload of 11.5 kg was added. A single suction unit generated a suction force of about 500 N on the tiled wall, and the measured maximum friction force between the suction unit and the tiled wall was approximately 200 N. Since at least three of its feet were anchored on the wall simultaneously, the maximum payload of the robot was about 43.5 kg [78]. The mentioned main limitation of the ZPD method is the demand for high-density fluids such as water to be used as a medium in the sealing layer. This limitation does not apply for the use in underwater or smolt production facility applications where there is easy access to water. Due to this and the promising research results, this could be an excellent method to experiment further with, especially for underwater use and for the mobility part of a potential robotic system.

A Russian research group looked into the use of a gas-water ejector device for vacuum contact of an underwater climbing robot in 2017 [79]. The robotic system consisted of two platforms, one (outer) platform for movements in X-Y direction and one (inner) platform for rotations about the Z-axis Each platform has several legs made of pneumatic cylinders for traverse movement and a set of suction cups using the "gas-water"-ejector principle for suction force adhesion. Their experimental results were promising for low depth applications but had problems with transient time for suction cup attachment with increasing depths.

Another interesting climbing robot, even though for skyscraper purposes, is the Sky cleaner 3 created in 2006 by a research group from China [80]. This robotic system is very similar to the one described previously. However, it is exclusively based on an X-Y principle of movement using vacuum suction pads for adhesion of the arms. It features 14 suction pads, and it is stated that the system can carry a payload of approximately 60 kg, including its bodyweight of 45kg. This is probably under ideal conditions with glass as the operating surface and air as the surrounding fluid.

There also exist some interesting research within bio robotic underwater adhesion mechanisms that are adapting some of the powerful evolutionary adhesion mechanisms of underwater creatures such as the Remora suckerfish [81] or the northern clingfish [82]. Also, it could be of interest to explore the work of Seibel and Adami that researched onboard pneumatic pressure generation methods [83] even though their work has particular emphasise towards soft robotics. These research projects have shown promising results, but to date, none of these are commercially available.

1.5 Summary

This chapter has laid the foundation for an intended robotic system by defining desired and feasible operations for the system to solve based on possible increased value creation and removal of tedious tasks. Similar robotic systems have been researched to find inspiration or directly applicable solutions from related industries.The intention of the robotic system is to create a versatile robotic system that can do multiple of the identified operations from section 1.2.5 such as cleaning underwater, feeding and potentially manipulation operations as grabbing dead fish. For doing this, the system must be able to move freely in the facility regardless of surface and environment. Therefore, a novel concept of using a multi-limbed base with suction adhesion for mobility was chosen. The modelling, controls and simulations of such a system is a complex and difficult task. Therefore this thesis aims at exploring if the manipulator part could be able to complete the identified tasks, given that the system for mobility was in place. For this to be proved it is aimed to construct, implement and simulate a fixed robotic manipulator doing some of the specified tasks.

This thesis is schematically outlined in Figure 1.5. Chapter 1 consisted of a literature review within the field of land-based salmon farming with a special emphasise on smolt production. This included challenges, current state, technological level, identified operations

facilitated for increased autonomy, related industries for inspiration, and an autonomous robotic concept for solving the operations. Chapter 2 consists of an extended literature review within underwater manipulators, robotic simulation software and initial setups and test of the chosen software which belong to the two leftmost squares in the overview. The review presents basic underwater manipulator theory, available manipulators, a comparison and a chosen manipulator for the simulations of the identified operations. Chapter 3 presents relevant theory and research about the used tools in this thesis which is identified as the blue text involved in multiple parts of the thesis. The main takeaways from this chapter are related to workspace generating, path- and motion planning, and controls implementation in ROS. Chapter 4 presents an approach for implementing programmatic control using MoveIt, C++ and Matlab for multi-goal path planning using information from the previous chapter. This correlates to the second box from the left. As described, this general approach is validated through the use of a generic self-designed manipulator. The chapter also presents basic workspace generation using Matlab and screw theory as defined in the third box in the top row. Further, chapter 5 presents a utilisation and extension of the previous chapters where the identified manipulator along with the operations are simulated using constrained multi-goal path planning and path following with the presented tools, only improved. This corresponds with the last box in the prerequisites half and the end product. Finally, chapter 6 presents the conclusion and further work of this thesis.

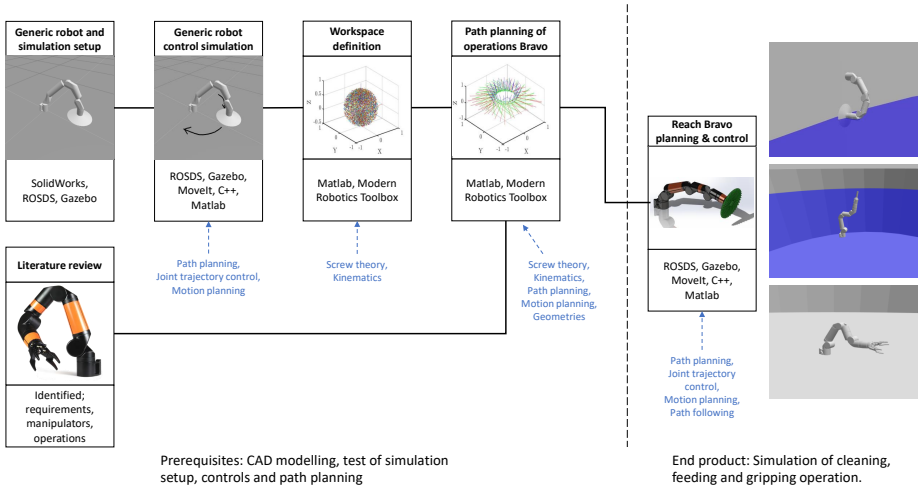


Figure 1.5: Thesis overview. Each box constitutes one or multiple parts of this thesis. A box consist of a describing name, illustration and tools or discoveries covered in the part. The blue text indicates what theory is applied in the parts.

Chapter 2

Selection and setup of autonomous system and simulation environment

To further develop the superfluous concept from the previous chapter towards simulations, some basic requirements and choices have to be defined. First, the underlying theory related to underwater manipulators is presented. Further, requirements are set for the ideal manipulator, and a review is conducted on existing manipulator solutions, whereas the identified best fit is chosen for the final simulations in this thesis. By using the chosen manipulator as inspiration, a generic manipulator is designed for test simulations and simple changes in linkage lengths. Finally, simulation software is chosen, the manipulators are modelled, described in the simulation environment, and a simple approach for conducting simulations with the generic manipulator is presented.

2.1 Underwater manipulator basics

This section presents an introduction to underwater manipulators, including requirements and configurations. In the mechanical design of underwater manipulators, the typical specifications are based on operating depth, surrounding fluid, tasks to be done, reach, lifting capacity and wrist torques [84].

Underwater manipulators are most commonly constructed by materials with high strength and corrosion resistance to withstand the underwater pressure and avoid possible leakages. The most common materials are metal alloys such as titanium (Ti 6-4), anodised aluminium alloys (5083, 6082 T6, 6061 T6, 7075, T6 and A356), stainless steel alloys (316, 630 and 660), and plastics (polyethene or polyoxymethylene) [84] [85].

The size or length of a manipulator is described by a parameter called "reach" which represents the total length of the whole manipulator kinematic chain from its base to the end-effector point which in terms is the interaction point between the manipulator and ob-

jects to be manipulated. The maximum reach along with the possible motions of the joints define the manipulators' workspace, which can be described as a set of points that can be reached by the robots end-effector [86]. Typical reach for different degrees of freedom (DOF) industrial manipulators ranges between 0.5 and 2.5 meters.

The majority of underwater manipulators are created for ROV/AUV mounting and operations. The coupling between the underwater vehicle and the manipulator result in complex controls. Therefore, most of these manipulators are designed with few degrees of freedom (DOF) and a near-natural buoyancy to not interfere with the control of the vehicle itself. Underwater manipulators are therefore typically designed with 3 to 6 DOF where the manipulators with few DOF often are used for simple inspection purposes. In contrast, the manipulators with the most DOF are used for intervention tasks while stationary locked, typically by mechanical adhesion. Three DOF are sufficient for obtaining an arbitrary position in the workspace, while 6 DOF are sufficient for obtaining both an arbitrary position and orientation [87]. A normal simplification of a 6 DOF manipulator is to interpret the manipulator as an arm constituting the positional component and a wrist constituting the orientational component. A standard industrial configuration of a robotic manipulator is a 6 DOF revolute arm where the three first joints represent the arm, and the three last joints represent the wrist as can be seen in Figure 2.1. Manipulators with more DOF are not commonly used, but some do exist, particularly within research. True seven-or-more DOF manipulators are said to be inherently redundant from a kinematic point of view [88]. This means that such a manipulator possesses more joints than those strictly necessary to execute a given configuration. The extra joint provides the robot with an increased level of dexterity that may be used to avoid singularities, joint limits, and workspace obstacles. Also, it can help minimise joint torques, energy use or optimise task solving based on desired performance indexes [89].

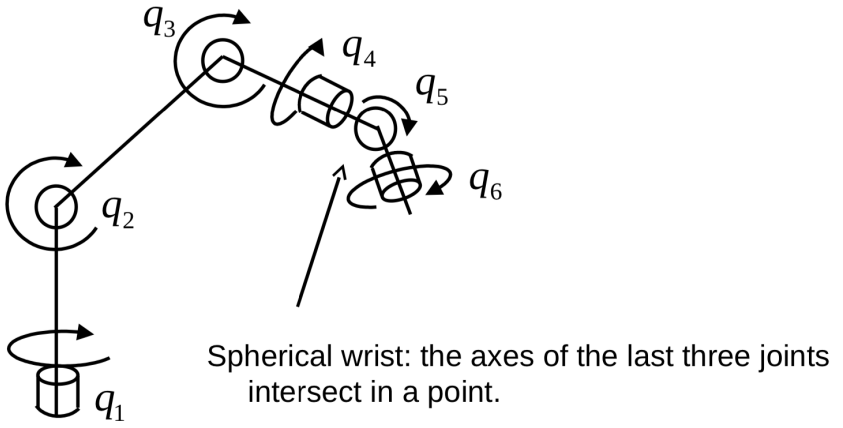


Figure 2.1: 6 DOF revolute representation from Modern Robotics book [87].

There are mainly two types of actuator principles that are used in robotic manipulators. These are hydraulic (or pneumatic), and electric, where the first-mentioned is the most commonly used manipulator type. The different types will now be presented with their respective pros and cons with regards to operations in smolt production.

1. Hydraulic manipulators

Hydraulic manipulators use incompressible fluids (typically oil) for force transmission in its actuators. The hydraulic fluid is transported from a reservoir to actuators through flexible hoses or rigid pipes by a hydraulic power unit (HPU) that consists of an electrically driven pump and a pressure regulator [84]. The motion of hydraulic systems is controlled by the flow of hydraulic fluid, which is typically regulated by valves of the types; directional control, proportional or servo. Pneumatic actuated manipulators works on the same principle as the hydraulic manipulators, only using gas (typically air) as the fluid medium.

Pros	Cons
High power to weight ratio (payload capability) without gears and levers. Ratio: 1-3	Leakages
Few parts	Poor positioning accuracy (sensitivity)
Compact	Challenges with control of interaction force
Inherently pressurised, not susceptible to seawater intrusion	Often large and heavy due to auxiliary equipment
Inbuilt protection against overload	Often expensive
HPU can be placed topsite	

Table 2.1: Pros and cons of hydraulic manipulators.

2. Electric manipulators

Electric manipulators are driven by electricity, either by a battery pack or by cable. These manipulators most commonly use brushless DC electric motors with harmonic drive gears featuring low backlash and large reduction ratio [84]. Electric manipulators for deepwater use are often filled with oil to counteract seawater intrusion and work as cooling and lubrication. This last point could be avoided for low depth manipulators, which will be beneficial in a work area with strict hygienic requirements.

Pros	Cons
Precise motion control	Low power to weight ratio. Ratio: 0-1
Precise force/torque control	Complex (needs gears and levers)
No leakages (if not oil filled)	Needs high degree of isolation
Facilitated for autonomy due to precision	Does not handle long-lasting overloads
Compact design	

Table 2.2: Pros and cons of electric manipulators.

As can be seen from table 2.2 and 2.1, there are both desired pros and undesired cons for both of the two manipulator types for use in smolt facilities. When choosing between the two configurations, there will always be a trade-off regarding what characteristics are most critical for the operations to be done. In a closed environment where the robotic system will be in direct contact with living creatures, that besides will end up on people's dinner tables, the requirements regarding HSE aspects are at the highest priority. Therefore, hydraulic manipulators are typically disregarded since they impose a significant threat of potential oil leakages even though they are not oil pressurised. There are, however, some available hydraulic manipulators that use "food oil" for force transmission but are often a costly affair. Nevertheless, even though the hydraulic manipulators offers far higher power to weight ratio and compact design, these leakages are highly undesirable even for vegan oils. Also, hydraulically controlled manipulators have generally poor accuracy, which could be a problem when an increased level of autonomous operations are desired. Based on these essential shortcomings for which the electrical manipulator fulfils, it was chosen to go for an electrical manipulator system regardless of their lower power to weight ratio and more complex mechanisms. These shortcomings were identified as less problematic since operations within the specified environment typically will not require large force or heavy intervention tasks.

Even though it is identified that an electrical manipulator might be beneficial for the chosen operations in the smolt facilities, these come with varying size, shapes and characteristics. Therefore it is of interest to narrow down the search by applying some more requirements for the ideal robotic system. Since it is desired to have a versatile system that can do multiple operations with high precision and efficiency, along but lean manipulator that does not occupy unnecessary space or result in clumsy control is wanted. Therefore it is desired to have an "as long reach" as possible to for example be able to reach a significant cross-section when doing cleaning operations. Also, the manipulator should be as lightweight as possible not to impose unnecessary challenges to the controls of the movable base and adhesion technique used. The weight is always in a trade-off with the power that is needed to complete the desired tasks. In addition to these requirements, it is of interest to have a robust system that requires minimal maintenance and does not impose any threat to its surroundings, either through direct contact, leakages or by noise disturbance.

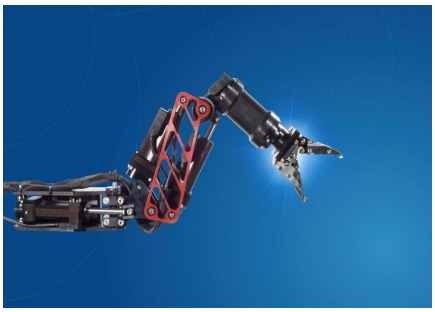
Based on the identified tasks and requirements with regards to various tank designs, fish welfare and feasibility found in section 1, the following requirements are set for the manipulator of the CMIM in falling order of importance beneath. These will underlie the choice of the most appropriate manipulator or lack of such industrial solutions. Based on the literature review and considering the needs of smolt production units, the following derived requirements for the development of an underwater robotic arm suitable for the defined operations in section 1.2 with particular emphasis on cleaning tasks are listed below.

1. Long reach
2. Lightweight
3. Fit for underwater use
4. Electric
5. Robust and reliable
6. Little motor noise and vibrations (preferably not conventional servos)

2.2 Existing manipulator solutions

Sivčev et al. [84] did an extensive review into the available underwater manipulators with a particular emphasis on manipulators for underwater vehicle mounting in 2018. As it emerges from their findings, only 7 of their total 47 identified commercial manipulators (or 15%) were electric. Even though there exist some industrial hydraulic manipulators using food grease as their fluid medium such as the Fanuc robot *LR Mate 200iD* [90], these usually are made for food or wet room operations, and not for underwater applications. The hydraulic manipulators from the review were disregarded for the applications in this thesis due to HSE reasons even though some of them could fit other mentioned requirements.

Eca Group's Hytec series [91] represents the main share of electric manipulators in this review with their 5 models. Disregarding their heavy alternatives (Eca Hytec Arm 7E and 7E mini) weighting 50+ kg in air, there are three alternatives left. These are the 5E series (normal, mini and micro). The models are of different configurations with 4 DOF and weight in air ranging from 27 to 10kg, and a reach of 1 to 0.64m. The ECA Hytec 5E Mini can be seen in Figure 2.2a. In addition to these manipulators, the review mentions *Ansaldos MARIS 7080* [92], a heavy (65 kg in air) electric manipulator which will not be further addressed because of its size and weight, Ocean Innovation Systems *BES-500*, a lightweight (15 kg in air) 4 DOF manipulator with a reach of 0.7m, and *Graal Tech's UMA* [93] that will be further addressed beneath. Ocean Innovation Systems and its manipulator has now been acquired by TMI Orion and is now called just **EMA** for electrical manipulator arm [94] and can be seen in Figure 2.2b.



(a) ECA 5E Mini borrowed from [91].

(b) TMI Orion EMA borrowed from [94].

Figure 2.2: Two of the interesting manipulators from Sivčev et al. [84] review.

Graaltech Underwater robotics has made an underwater modular arm, called the **UMA** [93] which is of great interest since the reach is one of the critical characteristics of the ideal manipulator solution. This manipulator is made up of modular joints and links such that it is reconfigurable. Electrical brushless motors generate the manipulator motions in the joints, and all the joints share the same mechatronic interface. At its base configuration called the TRIDENT arm, the manipulator has 7 DOF, a weight of 28kg in air, 14 kg in water, 1 meter of reach and a lift capacity of 10kg at full reach. The manipulator is as its name implies made for underwater use, especially for underwater vehicle mounting, but has also been used on outdoor mobile platforms.

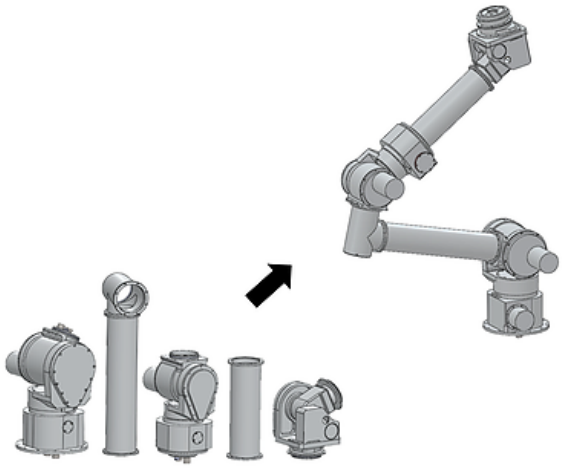


Figure 2.3: UMA - Underwater Modular Arm from Graaltech [93].

Blueprintlab released a new electric underwater manipulator at the Underwater Interventions in 2020 [95] called the Reach Bravo manipulator (formerly called Reach 7). The manipulator is fully electric, has a depth rating of 300 MSW (meter seawater), is based on revolute joints with 6 DOF (7-functional), weights 8kg in air and 4.5 kg in water, a lift capacity of 10 kg at full reach in its base configuration (reach of 0.9m), and is made

of AL6061 (anodised aluminium). The manipulator is relative cheap at 25K USD and is initially intended for underwater vehicle mounting and a master-slave controller.

Figure 2.4: Reach Bravo from Blueprintlab [95].

A research group at NTNU [96] have created a novel subsea electric low-cost solution for underwater inspection, maintenance and repair operation, called **SeArm**. The manipulator has its origin from an identified gap of knowledge and lack of solutions for precise, lightweight and strong (high power output) underwater manipulators that could be mounted onto ROV's. Hence, the research group created a precise underwater electric manipulator with moderate payload capabilities, as seen in Figure 2.5. Even though this manipulator was created for mounting on an underwater vehicle, it could be a great fit for use in a smolt production environment. The manipulator is designed to be completely modular such that it can be configured to fit size and performance requirements based on the application. The modules are made up of a stainless steel cylinder sealed with an o-ring lid. All modules contain an electric servo motor that creates a torque about the joint axis. The modules are connected by hollow plates of stainless steel that provides a waterproof connection for the servo cable that goes through the whole kinematic chain (servo motors are connected through a "Daisy chain", only one cable needed to control the manipulator). The connection comes in arbitrary lengths where the torque limit of the servos governs the maximum length. The weight limits are also a function of the desired configuration.

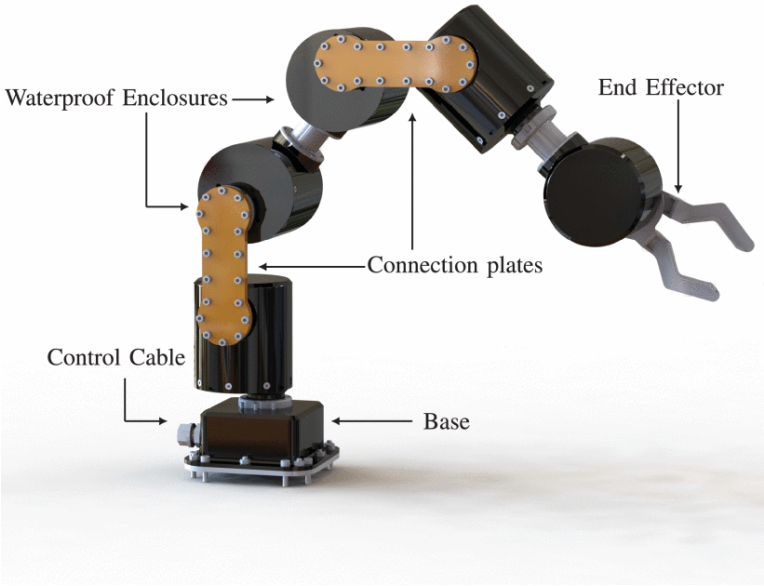


Figure 2.5: SeArm from NTNU outspring Searo Underwater Robotics borrowed from [96].

An interesting bionic inspired robotic manipulator that could be applied for underwater applications is Festo's **BionicMotionRobo**, a lightweight pneumatic robot with a natural movement pattern [97]. The manipulator has 12 DOF, a power-to-weight ratio of around 1 (3kgs:3kgs) and a maximum length of 850 mm. The manipulator is initially not configured for underwater use but since it is pneumatically controlled and already has a textile cover as seen in Figure 2.6 it should be possible to replace this cover with a waterproof one.

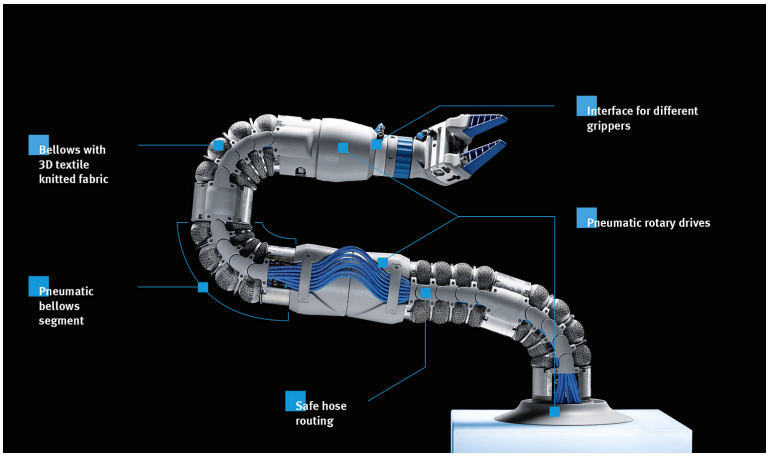
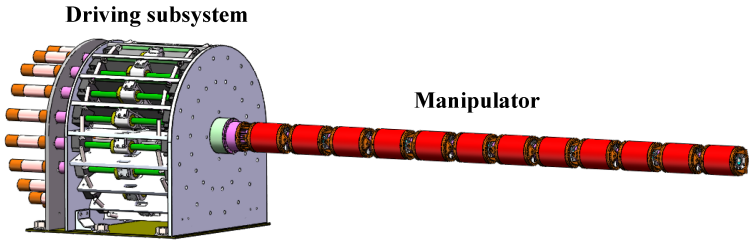


Figure 2.6: BionicMotionRobot from project partner Festo [97].

Another concept that is similar to the mentioned bionic inspired manipulator from Festo is the concept called cable-driven hyper-redundant manipulators (CDHRM). A CDHRM is a snake-like robotic manipulator that is modular built up by configurable joints that are controlled cables passing through the tubular structure from the base to the end-effector. This modular build-up makes it possible to increase or decrease the joint numbers, hence length and other characteristics of the manipulator based on the task requirements. Such a manipulator was designed and tested by a research group from China in 2019 [98], seen in Figure 2.7. The manipulator configuration showed promising results within crash avoidance, and the lightweight attribute relative to other manipulators but several issues were identified with special regards to mechanical design, kinematics and control. Especially the controls are problematic since three cables control the motion of each joint with two DOF's. Due to this, there exist complex mapping relationships and hyper-redundancy among the cables, joints and end-effector, which results in the complex kinematics and control. Nevertheless, this could be an interesting manipulator concept for the future.



(**a**) A general mechanical design of a cable-driven hyper-redundant manipulator (CDHRM).

Figure 2.7: CDHRM from Chinese research group [98].

From a more traditional industrial perspective, there might be some appropriate manipulators out there that are not customised for underwater operations. Nevertheless, it is undoubtedly possible to adapt some of these robot arms to handle underwater operations of not too high depths. This might result in cheaper manipulator systems since the volume of traditional robotic arms is very high relative to customised underwater manipulators. For this to be the case, the manipulators must fulfil all of the mentioned requirements in section 2.1 and preferably beat the manipulators mentioned earlier that are out of the box designed for underwater use.

The **Gen 3 Ultra lightweight robot** delivered by Kinova [99] is a lightweight manipulator created for education and research purposes initially with built-in functionality for ROS, Matlab and Simulink supporting advanced programming environments for C++ and Python. When it comes to the characteristics of the manipulator, it has 6 (7) DOF, a weight in air of 7.2kg, a reach of 0.9m and a payload limit at full extension of 2kg.

Kassow Robots [100] from Denmark is a robotic manipulator developer that specialises in strong, fast and simple electric manipulators. Two of their manipulators could be great candidates for the use in smolt facilities with the implementation of seals and covers to

withstand small depths. These are the **KR810** and the **KR1205** with the respective characteristics of 7 DOF for both, 0.85 and 1.2m reach, a weight of 23.5 and 25kg, and payload at full extension of 10 and 5 kg.

Most industrial manipulators are made to stand still at one dedicated work station, hence a standard "lightweight "arm for industrial use in manufacturing, packaging or handling weights around 100kg such as Fanuc's paint robot P-40iA. These manipulators are way too heavy for movable operations in a challenging environment unless large railways are to used.

To wrap this up, the explored robotic solutions that are close to, or commercially available and found to be best in their "class" have been assembled in table 2.3 on the next page. When it comes to the traditional industrial arms not customised for underwater use, these are found to be unfavourable relative to their underwater facilitated counterparts within the same category regarding price, weight and lift capacities. Therefore the customisation for underwater use and actual use has been disregarded as can be seen in the table that summarises the most interesting robotic manipulators for the use in smolt facilities. As it emerges from the table, there is always a trade-off that has to be made when designing or choosing a robotic manipulator, especially with regards to reach, weight and lift capacity. The table presents manufacturer, model name, DOF, weight in air and water, lift capacity at full extension, depth rating, maximum reach, power source, primary material, control principle and price. Not all details have been found. Therefore, some elements might be valued with a minus sign, or a question mark if it appeared uncertain.

Table 2.3: Possible manipulators fit for operations in smolt facilities.

Manufacturer	ECA Group	ECA Group	ECA Group	Graal Tech	TMI Orion	Blueprint lab	Searo	Festo
Model	Hytec Arm 5E	Hytec Arm 5E Mini	Hytec Arm 5E Micro	UMA (Trident conf.)	EMA	Reach 7	SeArm	BionicMotionRobo
Actuation	Electric	Electric	Electric	Electric	Electric	Electric	Electric	Pneumatic
DOF	4	4	4	6	4	6 (7)	4 ++	12
Weight in air [kg]	27	23	10	28	15	8	2.4	2.95
Weight in water [kg]	18.5	15	2.7	14	8	4.5	~0	–
Lift capacity at full ext. [kg]	25	25	10	10	16	12	1.3 (2.4 in water)	3
Depth rating [m]	6000	3000	300	100	500	300 (to 6000)	100	–
Max reach [m]	1	0.85	0.64	1	0.7	0.85	0.58	0.85
Power source	24-30VDC	24-30VDC	24-30VDC	24VDC	24VDC	24-40VDC	12VDC	PU 3 bar
Material	Al 6082 T6	Al 6082 T6	Al 6082 T6	Al	Al 5083, PE	Al 6061	SS	Knitted fabric ++
Control	Proportional (?)	Proportional (?)	Proportional (?)	Position	Rate	Position	Position	Festo Motion Terminal
Price [$]	~40k	~40k	~25k	~75k	~30k	~25k	–	–

2.2.1 Summary and choice of manipulator

As can be seen from table 2.3, all of the identified manipulators have a limited reach (<= 1m) in their base configuration. They are all lightweight relative to industrial- and hydraulic manipulators, and they either are or can be easily customised for underwater use. The manipulators from ECA are identified as too heavy due to the vast depth rating and undesired little DOF for versatility. The same lack of versatility applies to the Orion manipulator. When it comes to the extreme DOF of the Bionic Motion Robot from Festo, there are quite some uncertainties with regards to increased lift capacity and modules, and the underwater use with embedded controls. Finally, three remaining manipulators stand out. These are the two modular robots **UMA** from Graal Tech and **SeArm** from Searo that can be configured for more extended reach in a trade-off against lift capacity, and the lightweight **Reach Bravo** manipulator. The modularity will be beneficial since it is of great interest to have sufficient reach to be able to do multiple of the identified operations. Due to the high price on the UMA and uncertainty around the SeArm, these candidates should be put on hold even though they technically fit with most requirements. The remaining candidate is then the **Reach Bravo** from Blueprint lab due to its high degrees of freedom, commercial availability and decent pricing. In addition to fulfilling most of the specification requirements with decent reach, being lightweight, facilitated for underwater use and electric. The friendly pricing could result in the possibility of buying larger quantities, hence install and operate multiple systems simultaneously resulting in increased efficiency and reduced need for travel in-between work stations. This seems like the best option to date and will be the chosen manipulator to be tested in the following sections.

2.3 Selection and setup of simulation software

This section will describe the approach used to create a basic design of a manipulator and a tank environment using CAD software and how this can be transferred to a Gazebo simulation environment. This will lay the basis for the rest of the thesis work which includes implementation of controls in the simulation, programmatic control of the manipulator and multi-goal path planning and simulation execution of these paths to demonstrate identified operations in smolt facilities.

2.3.1 Robotic simulation software

A robot simulation software is a tool created to replicate real-world robotic operations by taking environmental, surroundings and physical factors into account. This represents a simple and efficient way to test and validate novel complex systems, platforms or prototypes, hence resulting in lower risk, development costs and time of system integration [101].

The landscape of available robotic simulation software is large and solid industrial players such as ABB, Siemens and Fanuc with more, operate with their brand-specific simulation tools. Other providers such as RoboDK, Octopuz, Vortex and Delmia are not brand specific, offering a vast amount of different robots available for simulation. Nevertheless, all

of the above-mentioned simulation software comes with a substantial investment cost and most of this software are strictly regulated with regards to model modifications and setup of control algorithms. It was therefore of interest to explore the possibilities of using an open-source robotic simulation software where these capabilities were present and make the further work independent of costly software investments.

As mentioned by Staranowicz and Mariottini in their survey of robotic simulation software from 2011 [102], there are many providers also of open-source robotic simulation software. Their survey concluded that the software combination of Player/Stage/Gazebo (PSG) and Robot operating system (ROS) was the two most widely used and best software on that date. Gazebo now comes with ROS integration and is widely used in academia. Another research group from Universidade Federal de Pernambuco (UFPE) in Brazil did an analysis and comparison of available robotic simulation software in 2019 [101]. The research group first did a general review of available solutions before narrowing down to test and analyse the three most commonly used simulation software in the robotics community, namely; V-REP, Gazebo and Unity. The latter mentioned simulation software is mostly used in the gaming industry and does not present the same amount of robotic tools as the two others and was therefore discarded.

Much work has been done in the field of robotic simulation recently, and some other highly rated open-source software are Webots and OpenRAVE. A comparison of different software is presented in table 2.4 and the chosen software to be used in this thesis is then described. The table shortly presents the developer of the software, the available licensing, what physics engines it is compatible with, the primary scripting language, what scripting languages there are available API's for and what middleware is typically used for controlling and connecting simulations to hardware. Finally, the experienced complexity based on research about their user-friendliness and facilitation for a novice user, hence access to highly valued documentation and tutorials are presented.

Software	**Developer**	**License type**	**Physics Engine**	**Script language**	**External API**	**Middleware**	**Complexity**
Gazebo	Open robotics foundation	Apache 2.0: Free	ODE, Bullet, DART, Simbody	C++	C++	ROS, Player, Socket	3
V-REP	Coppelia-Sim	GNU GPL: Partly free academic	ODE, Bullet, Vortex, Newton	LUA	C/C++, Python, Java, Matlab	Socket, ROS	3
Webots	Cyberbotics Ltd.	Apache 2.0: Free	Fork of ODE	C++	-//-	Socket, ROS, NaoQL	4
OpenRave	OpenRAVE Commu-nity	GNU LGPL: Free	ODE, Bullet	C++, Python	C/C++, Python, Matlab	Socket, ROS, YARP	5

Table 2.4: Robotic simulation software.

Gazebo and V-REP were compared together with another simulation environment named the ARGoS simulation environment by Pitonakova et al. in 2018 [103]. They made a

feature and performance comparison of the three software for different situations with particular regards to swarm simulations (multiple robots). Disregarding the ARGoS software since it is designed for swarms and not presented in this thesis, the research group concluded that V-REP is the most suitable simulation software for high-precision modelling of robotic applications. This includes various industrial applications where only a few robots are required to operate at the same time while Gazebo outperformed V-REP for larger simulation environments and several robots. Nevertheless, their experiments showed that the usability of Gazebo was relatively poor. While Gazebo can import 3D meshes, there are no editing options, making it difficult to alter and optimise models. Another problem was the interface that had several issues and failed to follow established conventions. Difficulties with installations of dependencies for many of its third party models were also mentioned.

In the mentioned analysis of 3D simulation software providers at UFPE [101], V-REP, Gazebo and Unity were tested by robotic simulation experts that concluded with a small victory to V-REP. The reason for the victory was mostly because of simple usability with more options directly in the software while Gazebo does require more user-defined editing of used files. Nevertheless, their bottom line findings were that V-REP was the most straightforward software to operate which could result in time savings when developing, but that this would come at a significant investment price due to licensing (if used commercially). It is anyhow entirely possible to use the open-source Gazebo software with an expected increase in development time. Therefore, development time, investment and easiness should be carefully considered when choosing between the aforementioned simulation software.

The Graphical User Interfaces (GUI) of the different simulation environments were examined through papers, available videos, their official documentation and by test access. By inspection, it was found that the V-REP and Webots had the most intuitive GUI with lots of functionalities while Gazebo requires much manual work with defining everything through description files (like XML files). The OpenRave simulation environment seemed promising, but lacked the amount of community and documentation. Either way, as can be seen from table 2.4, all of the Simulation environments use ROS as middleware. Therefore it was decided to turn things around and start with the ROS (the robot operating system) which is a set of software libraries and tools for building robot applications. In that occasion, a web-based studio for ROS code development was found, namely the *ROS Development Studio*. This is a browser-based interface that works on any operating system without any installations, which was an invaluable feature to be able to work from anywhere. The development environment is delivered by a company called The Construct [104] and it comes with many tools pre-installed such as Gazebo, RViz, jupyter, OpenAI and Tensorflow. The company is working to increase the use of ROS and is operating its own Academy and community. Access is free up until 30 hours a month for basic GPU usage but could be increased for a small sum. Even though the ROS setup can run any of the simulation software described, it is mainly facilitated for the use of Gazebo, which therefore ended up as the obvious choice. The system setup for this project is described as using a cloud-based environment corresponding to a Linux based Ubuntu 16.04 with ROS Kinetic and the use of Gazebo 7 for simulations.

2.3.2 ROS

The Robot Operating System (ROS) is a flexible and dynamic open-source framework for writing robot software. It consists of tools, libraries and conventions that aim to simplify the task of creating complex and robust robot behaviour across a wide variety of robotic platforms [105]. ROS is often called a meta-operating system for robots that are based on collaboration and development from its worldwide community. Since it is a collaborative development platform, many of the "standard" repositories are in the development phase and patched regularly. This means that a lot of the available and used functions are prone to changes, hence there could be a lack of up-to-date tutorials or documentation available. ROS was built from the ground up by a community of fellow robot enthusiasts to encourage collaborative robotic software development. As a result, the founders are hoping that experts in various fields will take advantage of each other's expertise to create new state-of-the-art software. For example, by matching scientists within mapping, navigation, path planning and computer vision to build upon each other and facilitate for new scientific leaps.

ROS can be exploited by anyone through some of the mentioned software platform suppliers and is free to use. ROS does not necessarily require extensive knowledge within programming and robotics due to a lot of available tutorials and information regarding the use and implementation of various topics such as mobile robot systems or manipulators. Nevertheless, there are some essential terms in ROS that one should know. ROS is based on nodes, messages, topics and services which will now be presented shortly. The ROS architecture is built around the ROS Master, which is the core of all connections in ROS. The Ros Master often called the ROS core, provides naming and registration services to the rest of the nodes in the ROS system. It tracks all publishers, subscribers and services. The ROS core works as a mapping, market place or a telephone book, enabling individual ROS nodes to locate each other through advertising (publishing) and requesting (subscribing) to the ROS core (or master node) as can be seen in Figure 2.8. This enables the advertising and requesting nodes to locate each other and then communicate on a peer-to-peer basis once connected.

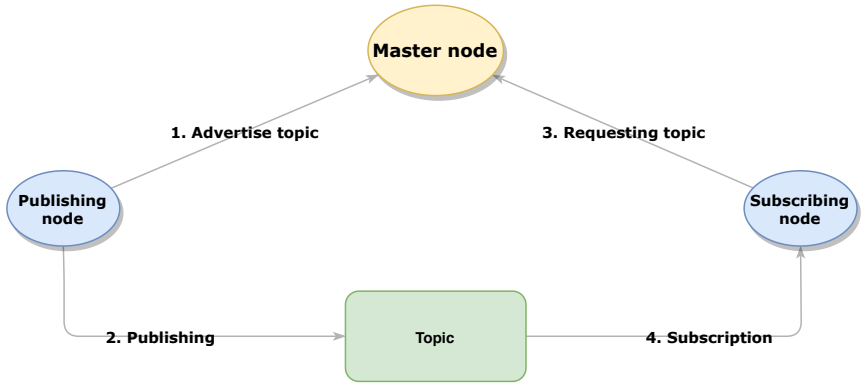


Figure 2.8: ROS Master Node communication inspired by the ROS wiki [105].

In ROS, functionalities are broken down into multiple chunks that communicate through messages. Each of these chunks is called nodes and would typically run separately. Messages can be of any data structure, typically standard primitive types (i.e. integer, floating-point, doubles, boolean) or arrays of primitive types. Nodes send messages by publishing them to a specific topic where the topic is defined as the name of a mapping, identifying the corresponding content (i.e. Float) of the named topic. There may be multiple publishers and subscribers for a single topic, and nodes may publish or subscribe to multiple topics. This varies significantly with the intent of the nodes or topics. For example, there should not be multiple publishers to a topic regarding the position of a dynamic robotic system that is used by other nodes. This may lead to uncertainties and jumps in position values. Requests and replies are made using services which consist of a paired message structure where a client node sends a request message (3), waiting for a reply message from the service providing node (4) [106]. For this reply to be feasible, a publishing node must have advertised (1) and started publishing the correct topic (2) on beforehand. Else, the master node will have a hard time connecting the publishing and subscribing node. The rest of the ROS architecture will not be further addressed since this thesis has a greater focus on the utilisation of pre-existing software plugins rather than creating new ROS plugins, clients or services.

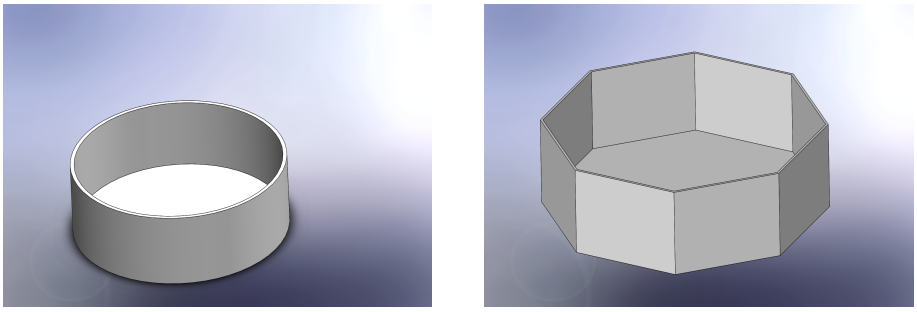
For defining a robot in ROS, the robot must be configured in its robot description file format Universal Robotic Description Format (URDF) which is an XML file format used in ROS to describe all elements of a robot [107]. For ROS to be able to compute collision checks, path planning and in general operate a manipulator, it needs to know the manipulator characteristics, what it looks like and how it behaves. Therefore it needs to know the characteristics of the joints, links and the dependencies between them. In addition to the URDF file, it is often of interest to include a Semantic Robot Description Format (SRDF) which is a representation of semantic information about a robot. This format represents information about the robot that is not included in the URDF file but could be useful for a variety of applications such as motion control using the MoveIt motion planning framework [105]. This typically involves dependencies between joints to be moved together, specifying joint groups and definition of links that cannot collide such that the software can neglect calculating collision checks between the given links.

Gazebo is as mentioned a 3D dynamic simulator with the ability to simulate populations of robots in complex indoor or outdoor environments. The software offers life-like physical simulations, a suite of sensors and interfaces for both users and other programs. Gazebo is used for testing robotic algorithms, design of robots and performing regression testing with realistic scenarios [107]. The combination of the Gazebo simulator environment and the ROS interface to the robot creates a powerful robot simulator with many possibilities. For robot simulations to be feasible in a Gazebo simulation environment, the robot description (URDF) must be properly described for simulation purposes. This is done by adding more physical properties to the URDF file such as, i.e. inertia, visuals, joint actuators. The "converted" or redefined URDF file is often called a Simulation Description Format (SDF), similar naming as for the Semantic Robot Description Format (SRDF), but these should not be mistaken.

2.3.3 CAD-modelling of tank environments, generic and specific manipulator

For the mechanical modelling of typical tank environments and both a generic and specific robotic manipulator, the Solidworks CAD software was chosen due to licensing at NTNU and the availability of documentation. The Computer-Aided Drawing (CAD) is not especially emphasised in this thesis since it is a degree within industrial cybernetics. The author is a former mechanical engineer with significant experience using CAD software. This part will, therefore, emphasise the connection between the CAD software and the ROS environment, which is more of a programming and simulation related task. Using the approach described should result in fast and straightforward future implementation and use of the described software combination for any robot system.

Two simple tanks were designed for use in the simulation environment. The tanks were constructed with an inner diameter of 28 meters, which is the largest tanks that the Autosmolt2025 partner Brimer AS delivers. The first tank was designed as a simple circular tank which is the most common shape as mentioned in section 1.2.1 while the second tank was designed with an octagonal shape which is also a common shape used in smolt production. The two tanks can be seen in Figure 2.9 and was constructed using simple extrude functionality in SolidWorks (SW).



(a) Circular tank, inner diameter: 28 m. **(b)** Octagonal tank, inner diameter: 28 m.

Figure 2.9: Typical tank designs in smolt facilities.

Further, a generic manipulator was designed using basic robotics design as described in section 2.1 and using the chosen *Reach Bravo* robot from section 2.2 as inspiration to obtain similar attributes, but keep the options open for simple changes in link lengths. The resulting first draft of this manipulator can be seen in Figure 2.10 including an arbitrary first draft cleaning tool attached as the end-effector tool. The manipulator consists of 6 revolute joints and links in between them. The first or zeroth link is the base link which in this first draft results in a normalised length equal to 155mm from the mounting of the base link and up to the second revolute joint (J2). The normalised length from joint 2 to joint 3 is equal to 290mm. These two lengths are identical to the first couple of joints of the Reach Bravo manipulator while the rest of the joint distances are a bit increased for this generic manipulator. Specifically, the normalised length from J3 to J5 is 220 mm, and the length from J5 to the midpoint of the cleaning tool (end effector) as L6 is equal to 367

mm. Including the displacement of joint 2 from the base frame centre axis (J1), the reach of this configuration using the midpoint of the cleaning tool as the end-effector point is equal to 922mm. It is worth mentioning that the end-effector cleaning tool in Figure 2.10 was created quickly for visualisation purposes early in the progress of this thesis. The exact dimensions and setup of the manipulators will be closer explained once they are to be controlled in simulations. The generic manipulator is created in such a way that the mid placed links can easily be increased in length, hence increasing the reach and change characteristics.

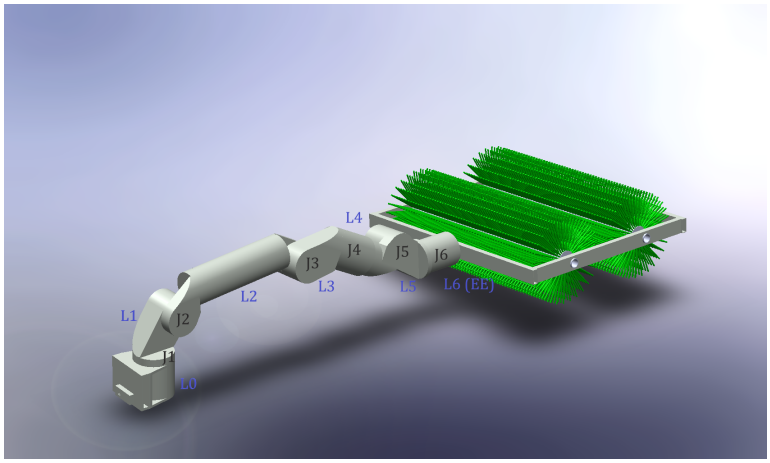


Figure 2.10: Draft one of generic robotic manipulator.

The initial thought for the specific manipulator from Blueprintlab, the Reach Bravo robot, was to draw the whole robot from scratch using SW and available dimensions. This was not necessary after a brief mail correspondence with their helpful business development manager which willingly shared the STEP files of the Bravo robot [108]. By importing the STEP files and saving them in SW, the files are converted to a SolidWorks file format containing a "stupid body" without specific features. It is possible to use feature generation when importing STEP files which will ultimately result in more defined features to work with, but also often result in various error messages. The importing step can be done using other software suppliers or SW as used in this thesis. Due to the lack of some geometric features, especially for the orange covers that can be seen in Figure 2.11, it was hard to assemble with the rest of the parts. This was solved using a pattern connected to the parent part of the covers but ultimately does not yield any difference other than visual quality in the simulations. The rest of the connections or mates as it is called in SW were straight forward for connecting circular connections with each other, resulting in the assembly seen in Figure 2.11. One important note for this part is to be sure to mate the parts in a manner corresponding to the desired freedom of each connection, i.e. resulting in a revolute, prismatic or other freedom description of the joint connections. This can, for example, be done by coinciding two circular profiles with each other and mate them together with a concentric mate, resulting in freedom corresponding to a revolute joint. Also, it is wise to mate the base frame of the manipulator to the world coordinates with the correct

orientation. A correct base mating will result in a typical robot description that is intuitive to understand making connections, tuning and simulations easier.

Finally, end-effector tools to serve the applications (cleaning, feeding and gripping) defined in section 1.2.5 was constructed in SW using the same principles as mentioned. The iterated cleaning tool can be seen in Figure 2.11. The feeding cannon was constructed using basic CAD modelling approaches while the gripper was assembled similarly to the rest of the provided manipulator from Blueprintlab. All of the end-effectors were created with a mounting part for simple connection to the manipulators and will be presented in section 5.1.

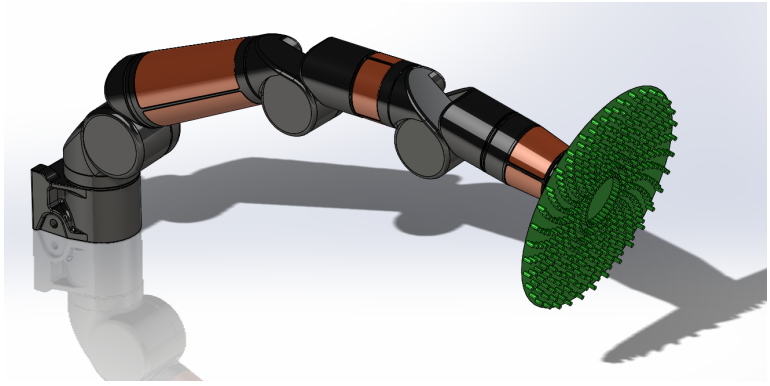


Figure 2.11: Specific manipulator (Reach Bravo from Blueprintlab).

The next step in simulation preparations is to import and recreate the CAD modelled systems in simulation software. In order to do this, the created CAD models in Solidworks has to be described in URDF as mentioned. There are a couple of ways to do this. Either, the links can be exported as STL (STereoLithography) files for geometry recreation and all properties can be defined, or existing help tools can be used. The first approach is done by importing the STL-files, define their positions and dependencies using link- and joint definitions manually. Else, a tool developed by the ROS community can be used in order to create URDF files directly from SW parts or assemblies. This software, created by Stephen Brawner, is developed as a plug-in for SolidWorks version 2018 and above and works on Windows 64bit [109]. The plug-in is intuitive and straightforward to use for defining links and joints based on the assembly tree structure in SW. Joint- and link properties can be set for all parts before the finished URDF package including meshes, textures and robotic description (URDF) is created. If the automatically created meshes (STL-files) are created with too many details, these can be simplified using other open-source software as MeshLab [110] or Blender [111]. In this theses, the manipulators were created with quite basic geometric features; hence this was not necessary. Nevertheless, this might be necessary in order to avoid slow or buggy simulations and calculations for more complex simulations. Especially for collision computations, it is normal to create simple geometric representations of the links for collision checking algorithms instead of using the often complex STL files.

2.3.4 From CAD to URDF description

This section will present an approach briefly to quickly transform the created CAD files into a URDF structure that can be used in ROS. This will include a brief introduction to the installation and use of a chosen tool that interacts with SolidWorks, making the complexity of the transition minimal. For a more thorough guide, please see Appendix F.

The installation of the plug-in for SolidWorks to URDF conversion is straightforward using the setup and the installation tutorial in [109]. As the installation tutorial states, it is possible to install the add-in using the compiled source and Visual Studio. However, since the pre-compiled installation works correctly, this is a simpler and faster way of setting up the plug-in. First, the user has to install the .Net Framework of V4.7.2 or above, typically the latest version is installed if the user or computer has been using Visual Studio. The second step is to follow the "Download Installer" link or to locate the Github repository of the SW2URDF exporter [112] where an executable setup file can be downloaded and run to install the plug-in.

The conversion from SW to URDF is quite straight forward using the mentioned tool but is a critical part for further simulations and should, therefore, be laid some time and effort into doing it right the first way around. When the plug-in is started in Solidworks, the assembly models are automatically rebuilt and prepared for URDF configuration. If the connections between the different links are properly defined, hence reflecting the intended joints, the menu can be followed by adding new linkages and joints from the base and up until the desired configuration. An important note is that the manipulator should be set in the desired home configuration upon generation of coordinate frames for the joints. Also, it is important to note that the model tree should be used to choose the links as seen in Figure 2.12.

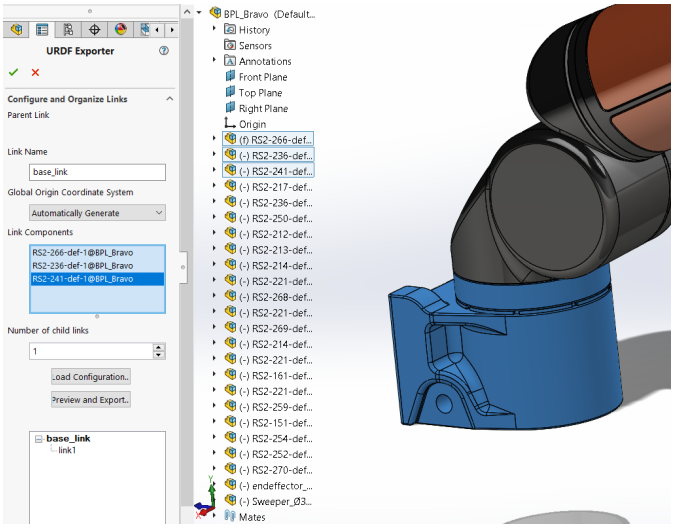


Figure 2.12: SW2URDF definition of base for the Bravo manipulator.

Once the full manipulator is properly configured in its home position, the coordinate systems and helplines can be automatically generated. If desired, the coordinate systems can be changed if a specific representation is desired such as redefining z-axis of joints to correspond with the Denavit-Hartenberg convention (DHC) or the Product of Exponentials (PoE) as will be more thoroughly presented later on. Finally, properties can be set for the links and joints and the STL files with its corresponding URDF formats can be exported and integrated into ROS. The completely defined Reach Bravo manipulator using the plugin can be seen in Figure 2.13.

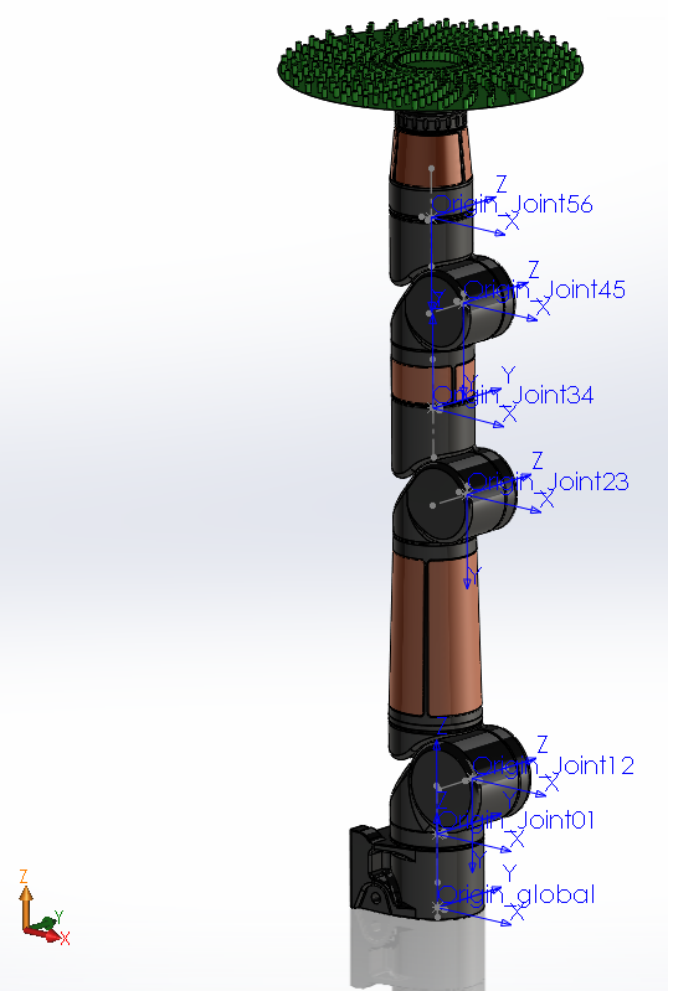


Figure 2.13: Completed SW2URDF setup with the respective defined joint coordinate systems for the Bravo manipulator.

2.3.5 Setup in simulation environment

If a proper description of the robotic system to be simulated is available, then it can be imported and implemented into the chosen simulation environment. In this thesis, the ROSDS has been used. Therefore, the following described approach for defining the setup might not be fully compatible with other possible setups, but should, in general, be very similar for all Linux based setups. This section is split into two parts, one describing a variety of implementations of the manipulator in the software setup, and another part describing the procedure of running the setups. This part might be obvious for users with Linux or ROS experience, but helpful for beginners. The code will not be described line-by-line. For more information and code specific description, please see the attached files in the Appendices.

The following part involves the setup of three different work-packages such that future users can choose what package to use based on their goal. The first package involves the implementation of the URDF created in the previous section and to set up an RViz configuration, a ROS visualisation environment, to verify that the manipulator has been configured correctly, both in terms of placement of links and joints, and their attributes. The second package involves the implementation and spawning of the created manipulator in a Gazebo simulation environment with working actuators, and the third package describes how to create a primitive Gazebo simulation environment. Also, a brief section for running the mentioned packages is presented.

Setup of visualisation in RViz

For creating a simple visualisation of the manipulator system, the first step is to use a terminal (or shell) and navigate to a chosen folder for where the project is to be created and create a package for the robot description. Using the command in line 2 of listing 2.1 will create a package named ROBOTNAME_description only dependent on the URDF type as specified. This is a shortcut for the creation of the underlying files that the system needs to compile the created package. The two files that are created are the CMakeList file for the compilation of the necessary C files to run and link the created package, and a package file that is building and implementing related packages called depends related to the constructed package. These files will not be highlighted further, and where it is necessary to change these files, the actions will only be shortly presented. See attached ROS projects for more specific code.

```
1  cd simulation_ws/src/
2  catkin_create_pkg ROBOTNAME_description urdf
```

Listing 2.1: Creating package in ROS as file location

The next step is to create folders for the definition and executions of the manipulator package. Namely, it is of interest to create a folder for the URDF description, the meshes and a folder for launch files. In the URDF folder, a xacro file should be created containing the description of the manipulator in URDF as created in the previous section. It is possible to use multiple xacro files, creating macros such that generic links, joints and transmissions

can be created for tidy code. This has not been emphasised in this project. At this point, the predefined URDF file from the previous section can be implemented almost directly, but there are some minor notes. The path of the meshes should be redefined to be found in the working directory. Also, the scaling factor of the imported meshes should be checked when running initial visualisations since ROS is using meters by default and the scale can vary based on SW setup. If the manipulator does not spawn for the naked eye, the scaling factor should be changed to 0.001. Also, to make the code intuitive, it is advised to move the description of the joints in the "right" spot in between its respective links, and finally give all of the links and joint clean and describing names. A short excerpt of the URDF code for the generic manipulator can be seen in listing A.1 (Appendix). As seen in the code excerpt, the description of each link includes a description of inertial and collision properties for simulation purposes and visual properties that are self-explaining. The mass and inertia properties are of high importance to be correct for the simulation to act natural.

Nevertheless, the values of the parameters are often incorrect from the SolidWorks models due to faulty model definitions. Therefore, these values have been set to a "uniform" character to avoid extensive wobbling or collapse in the simulations. This is especially important for the inertia matrices of the links. The joints are described in a straight forward way with a definition of their origin, the axis of rotation, related links, limits and dynamics. The limits have been set to utilise the full workspace of the manipulator while the other properties, effort, velocity, damping and friction are set to arbitrary values initially. These properties are what defines how the manipulator acts in the simulation, i.e. if the weight of the links or the inertia matrix is wrong (too heavy) concerning the allowed effort of the joints, the manipulator will not move in a "real" Gazebo simulation. For describing the links and joints only for visualisation purposes, some of the properties defined in the listing is not needed, hence it is enough with a URDF format for this part. However, an SDF format is required for the subsequent sections. For a full description of how to define links and joins for different uses explore links and joints in the ROS wiki [105].

After the manipulator has been properly described everything is facilitated to create a visualisation of the manipulator using RViz. A dedicated launch file must be created to initialise the visualisation of the robotic system. Such a launch file collects the description of the robot (URDF), reads the joint values, and shows this in the RViz plug-in. The visualisation is spawned with a predefined joint state publisher to publish desired joint values using a graphical user interface (GUI) in the RViz window as can be seen in listing A.2 (Appendix).

The visualisation can then be used to test and reconfigure the URDF file if necessary. This typically involves checking joint directions and limits. An important note is that the lower and upper limit defined in the URDF must be strictly increasing, i.e. defined from -1 to 1 even though the *SW2URDF* plugin indicates the use of lower bound 1 and upper bound 1. For testing, it is smart to comment out most parts of the robot and configure piece by piece from the base to the end effector. For pure visualisation purposes of the manipulator, verify properties, or test different configurations, the RViz visualisation is undoubtedly enough. If it is of interest to simulate the robot system in a simulation environment, then

the next sections will be of interest. Nevertheless, it is always wise to do this first step for verification purposes before moving onto more complex parts.

Import and setup of manipulator in Gazebo environment

The first steps in the setup of a manipulator in a Gazebo simulation are similar as for the RViz visualisation, only creating a launch file with a few other characteristics such that the robot is spawned and connected to the simulation environment. In the launch file, placement and orientation of the base to be spawned in are defined by creating the arguments for *x, y, z, roll, pitch* and *yaw*. This is ultimately enough to spawn the visualised robot in the Gazebo environment using the accompanied code in listing A.3 (Appendix). However, without defined transmissions (motors in other words) and controllers for the joints, the robot will collapse in the simulation. Therefore it is of interest to define controllers for the joints as can be seen in listing A.3 (Appendix). In this code excerpt, the robot is loaded and spawned with a defined position and orientation of the base link. Further, the controllers are loaded and spawned from a config folder containing the controller definitions. Lastly, an *rqt* node is spawned to be able to efficiently publish desired joint angles to the position controllers during runtime. *Rqt* is a framework in ROS that implements various GUI tools in the form of plugins to be used for simple UI commands or visualisation of data.

As mentioned in the last paragraph, controllers and transmissions that are replicating actual hardware in the simulation environment must be defined and implemented. ROS has all necessary libraries for this available such as a pre-defined joint state publisher and controllers that can be implemented by adding them to the dependencies of the package as can be found in the file structure of Appendix D. Once the dependencies are added, transmissions must be created for each of the joints in the URDF file replicating motor hardware. The transmissions have in this project been defined of the ROS type *EffortJointInterface* as seen in listing A.5 (Appendix), meaning that they are controlled by effort applied to the joints which are typical for electrical motors. Since simple inputs at this stage typically are defined as joint positions, the controllers "behind" the direct interface does the calculations for converting and applying the necessary efforts to obtain desired joint positions. Other joint interfaces are also available such as position- or velocity joint interfaces. It is possible to implement a mechanical reduction such as gears, but in this thesis, no such attribute has been applied. The last thing that is missing is the controllers described in listing A.3 (Appendix). Joint controllers can be defined in various ways, either creating self-made controllers using published data from the manipulator or by utilising existing functionalities within the ROS libraries. In this project, the available libraries have been used. For initial tests in this section, the standard positional controller using effort commands have been implemented together with a state publisher for feedback as can be seen in the joint controller excerpt in listing A.6 (Appendix). The effort controller type uses standard PID gains for obtaining desired control characteristics. The PID controller values can be tuned using the simulation environment and tracking response for different values using a *rqt_gui* that is defined in listing A.3 (Appendix). This was not done in this thesis since fine-tuning only is strictly necessary if the robot description is correctly defined.

If the presented necessities have been configured, there is only one thing missing, the connection between the Gazebo simulation environment and ROS. This connection is applied by adding the code in listing A.7 (Appendix) to the URDF file resulting in a fully functional robot in the simulation environment. Note in this part is that the URDF/SDF description of the manipulator most likely needs some tweaking concerning the inertia, mass, effort, damping and friction for the simulation to work correctly. This is because the simulation wants to be as real as possible. Therefore, it is hard to clearly define a "generic" robot system for simulations that will work 100% at the first try. The robot should be iteratively spawned while tuning the mentioned parameters to get get a working robotic system in the simulation. Low joint effort limits are a common source of error if the robot is not able to move. This is due to a resulting low allowed torque on the joints if the effort limits are too small. Using the initial defined inertial matrices and weights from the *SW2URDF* plugin is not necessarily right, and it is, therefore, necessary to tune these parameters as well as the PID controllers and effort limits. In this thesis, the inertia matrices have been set to a low and uniform value, and the link weights have been set to arbitrary levels that are feasible — the better the model, the better simulation results.

A closing remark on the Gazebo simulation part is that it is often wise to include a virtual link and joint to fix the base of the manipulator to its desired position in the URDF such that it does not overturn in the simulation. This is done by implementing the code in listing A.8 (Appendix). If a more complex system is to be simulated, a manipulator can, for example, be connected to a moving base using a virtual joint in the planning group that will be described later.

Setup of simulation environment in Gazebo

A user-defined Gazebo simulation environment is typically created as a dedicated package where the specifications for the simulation environment such as gravity, objects, light sources are defined in a world file. Also, a corresponding launch file for the initiation and spawning of the self-made world is included. The launch file for the world consists of arguments that define the different choices that come along with the creation of Gazebo worlds using the built-in gazebo_ros module. As can be seen in listing A.9 (Appendix), the arguments are defined in the first part of the code before they are implemented into the gazebo_ros launcher to overwrite the default empty Gazebo world with its standard configurations. For overwriting the standard empty world configuration, a world file must be defined appropriately. A world file is defined in an SDFormat, including the whole simulation environment with all its attributes and models. Only a simple environment has been created in this project consisting of a light source, a tank and a simple cylinder of water for visualisation purposes as can be seen at the end of this section. The code for this environment can be seen in listing A.10 (Appendix) and involves the mentioned models. Models can be included from open model repositories such as the one of the Open Source Robotics Foundations [113], or self-made models can be included within a created repository. A model consists of a config file as seen in listing A.11 (Appendix) which describes what is modelled and by who, and an SDF file describing the visual, collision, friction and other properties as seen in listing A.12 (Appendix).

Running Simulations

Running the created packages, either the RViz visualisation, the Gazebo world or spawning the robot is all done using the shell. In general, CTRL+C is used to terminate whatever process the terminal is working on, which is handy for killing off processes before rerunning simulations or cases.

1. Running RViz visualisation

RViz visualisations are initiated by running its corresponding launch file which will spawn the visualisation in the *Graphical tools* window as can be seen in Figure 2.14. The package (or folder) containing the visualisation must be compiled the first time at creation as seen in line two of listing 2.2.

As can be seen in line 8 of listing A.1 (Appendix), an argument has been commented out. This is the reference to a configuration file that should be created after the RViz visualisation has been defined as preferred. Once the RViz is launched, a fixed frame must be defined (the base or world frame). Further, the robot model must be added using the *Add* button. Besides, it is of interest to add axes systems for the global frame by adding *Axes* and *TF* from the list such that changes in robot poses can be spotted. At this point, the joint state publisher can be used to confirm that the links and joints are appropriately described. The joint state publisher is found in the same window of the graphical tools. Once satisfied, the configuration can be saved, and the previously mentioned argument can be moved from the comment and into the RViz node, which will then spawn the saved configuration next time the visualisation is launched. There is a known bug when launching the RViz simulation, that is that the planning request size needs to be changed from 0 to 0.1 or a small number to make the end effector "point" smaller and increase the quality of the planning scene visualisation.

```
1  cd simulation_ws/
2  catkin_make
3  roslaunch FOLDER RVIZ_FILE.launch
```

Listing 2.2: Running RViz visualisation.

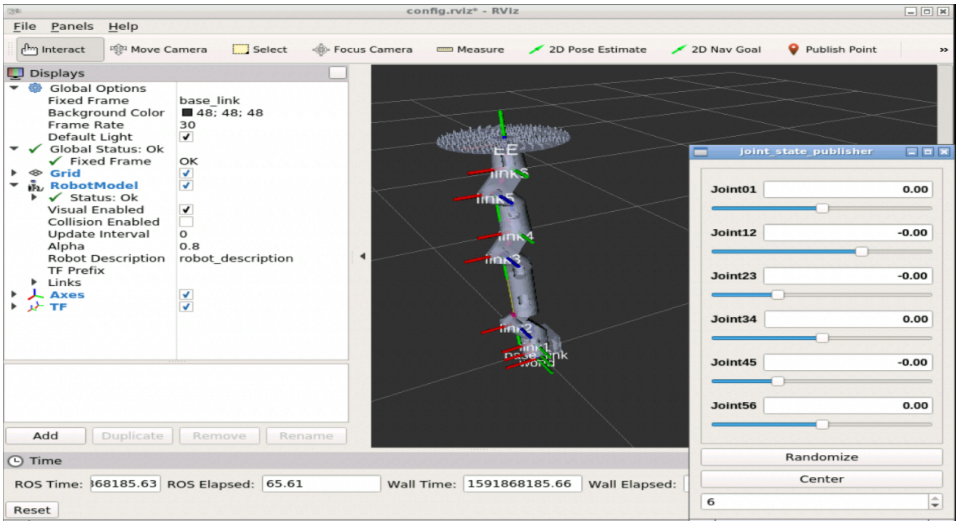


Figure 2.14: RViz visualisation of generic manipulator.

2. Spawning a robot system in a simulation environment

The spawning of a robot in a Gazebo environment follows a couple of steps. The first step is to initiate a Gazebo environment by launching a world file in the same way as launching other files in the shell. The next step is to *roslaunch* the spawn file for the robot that was created in part two previously. As mentioned, it is wise to use the *rqt* message publisher in the Graphical tools window to check whether the manipulator is appropriately defined for simulations before inserting it into other simulation environments. In the *rqt* publisher, the joint commands can be found in the drop-down menu to the left in Figure 2.15 and added to the list using the plus sign. By activating these commands, the data type beneath the expression column can be defined. This corresponds to the joint angles in radians which can be tested during runtime. These values can also be set using the shell directly, but this is cumbersome. If the manipulator cannot complete the given joint angles, either the PID gains or the factors described previously should be inspected, especially the effort limits of the joints. If the controls work as expected, everything is set for implementing more advanced controls.

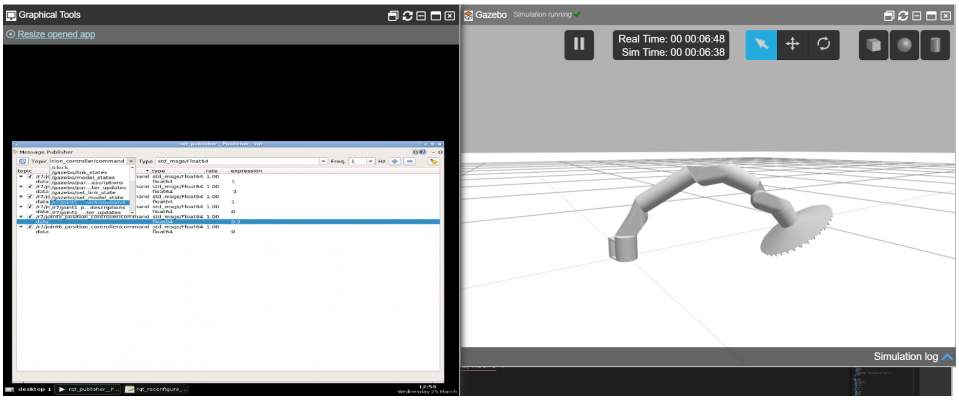


Figure 2.15: *Rqt* publisher in the graphical tool window.

3. Launch Gazebo world

A Gazebo simulation can be initiated using the GUI of ROSDS by choosing the desired world to spawn, or it can be launched using its defined launch file. These approaches will start the simulation and connect it to the Gazebo window with its world-specific attributes such as gravity. Using one of the stated approaches will result in the environment, as shown in Figure 2.16 for the defined tank environment where only the spawn arguments in listing A.3 (Appendix) has been changed for the robot to be spawned at the tank wall.

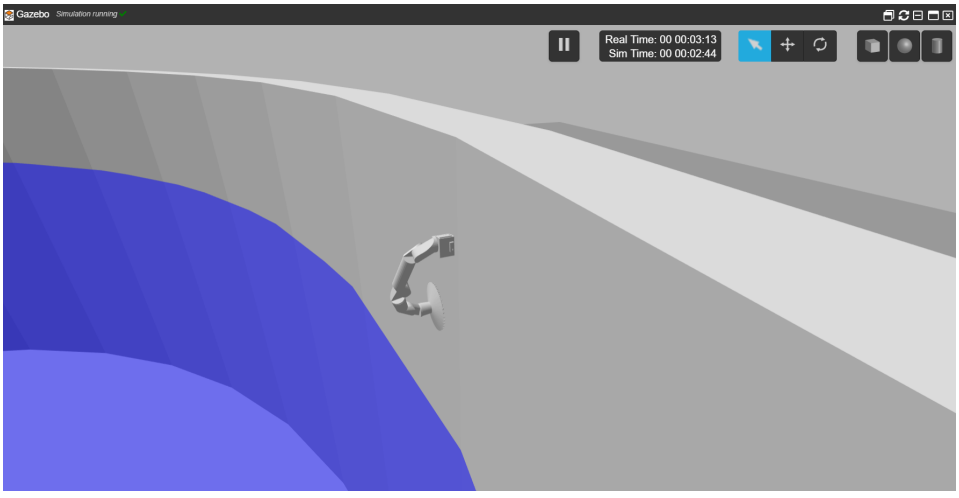


Figure 2.16: Initial simulation environment with generic manipulator.

2.4 Summary

This chapter has presented an approach for setting up an arbitrary simulation using ROS, Gazebo and SolidWorks for basic simulation and control of a generic manipulator. Also, a review has been conducted on available underwater manipulators that could fit the requirements defined by the selected operations in section 1.2.5. The next natural step will be to implement more advanced controls and a motion planning framework for the generic manipulator before moving on to the favourable chosen manipulator in order to simulate the cleaning, feeding and gripping operations in a tank environment. In this thesis, the available motion planning framework MoveIt has been used for motion planning. This has been done using the C++ API of the framework for programmatic control where multi-goal paths, generated in Matlab, have been used as input to simulate the tasks. A workspace was generated using screw theory and forward kinematics as a part of the multi-goal path planning to constrain it within certain bounds for feasibility before implementation in the motion planning framework which requires feasible inputs.

Chapter 3

Theory, control system components and tools used in the implementation

This chapter includes the necessary theoretical parts to realise constrained multi-goal path planning in Matlab using screw theory and basic path planning for the selected operations. Also, some theory regarding motion planning for the implementation of the MoveIt framework in ROS is presented.

3.1 Linear algebra: Transformation matrix

A transformation matrix is a representation of a combined orientation and position of a rigid body according to Lynch and Park [87]. To define this representation it is normal to use a rotation matrix R $\in$ SO(3) to represent the orientation of the body frame $\{b\}$ in the fixed frame $\{s\}$ and a vector p $\in \mathbb{R}^3$ to represent the origin of $\{b\}$ in $\{s\}$. This results in the following general transformation matrix

$$\mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{p} \\ \mathbf{0}^T & 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_1 \\ r_{21} & r_{22} & r_{23} & p_2 \\ r_{31} & r_{32} & r_{33} & p_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.1}$$

In general, a transformation matrix can be used to transform coordinates, directions or objects from one representation to another as seen in Figure 3.1. In robotics, transformation matrices are used to describe the position and orientation of joints (or links) with respect to one another. The most common variant is the "total" transformation matrix describing the end-effector position and orientation with respect to the base frame. It is this transformation matrix or decomposition into multiple transformation matrices that are used

for controlling the manipulator to desired poses. A frame and a coordinate system is, by definition, the same thing and might be used interchangeably in this thesis.

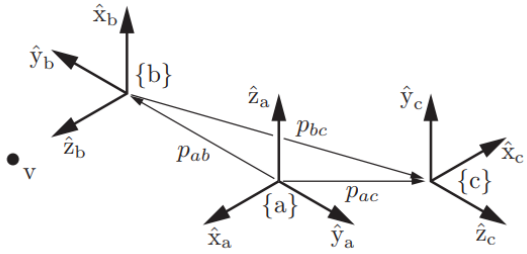


Figure 3.1: visualisation of transformations borrowed from the Modern Robotics book [87].

3.2 Kinematics using screw theory

A manipulator consists of a series of rigid bodies (links) connected through kinematic pairs (joints). Joints can mainly be of two types: revolute or prismatic (disregarding ball and socket joints) where a concatenation of such single DOF joints forms what is called a kinematic chain [87]. One end of the chain is constrained to a base, while the other end is called the end-effector (or tool), allowing manipulation of objects within the manipulator's workspace. From a topological viewpoint, the kinematic chain is termed open when there is only one sequence of links connecting the two ends of the chain. Alternatively, a manipulator contains a closed kinematic chain when a sequence of links forms a loop [88], which will not be regarded in this project.

There are different approaches (conventions) available to compute kinematics, where the Denavit-Hartenberg (DH) convention is the most used convention in the robotics community [114]. This convention uses a minimum of parameters for each joint to describe the robot state but requires strict definitions of the joint frames, which are not always intuitively placed. Another convention with growing popularity is the Product of Exponentials (PoE) convention using screw theory to describe robot states by using geometric interpretation intuitively. This approach only uses two reference frames, the base frame and e nd effector frame, and a geometric interpretation from the use of screw axes of each joint to determine motion impact of the joints. The PoE formula is directly derived from the exponential coordinate representation for rigid-body motions [87]. Screw based kinematic modelling is often found to be more complex than the DH convention, but also more flexible; hence provide advantages in modeling and analysis of kinematic chains [114].

The key concept behind the PoE Formula is to regard each joint as applying a screw motion to all other links, hence creating a screw axis through each joint, either perpendicular of the links for revolute joints or aligned with the links for prismatic joints. For doing computations using the PoE formula, some initial choices must be made. First, a fixed base

(or spatial) frame $\{s\}$ and end-effector (or body) frame $\{b\}$ must be chosen. These frames can be chosen arbitrary, but they are often chosen such that the z-axis of the base frame is pointing upwards, and the z-axis of the end-effector is pointing "outwards" targeting intervention (or pick-up) points. The last initial condition is to define the home position of the robot by setting all joint values equal to zero at that configuration. This transformation (or configuration) is written as $M \in SE(3)$. It denotes the configuration of the end-effector frame relative to the fixed frame in its zero position consisting of a rotational and translational part, ultimately yielding a homogeneous transformation matrix as can be seen in equation 3.1. A dynamic chain must not necessarily be described from base to end effector and might as well be a description in between selected links of a dynamic chain.

Once the initial choices are made, the forward kinematics for an open-chain robot with n-DOF (typically corresponding to joints) can be computed in either the base or the end-effector frame. To calculate the forward kinematics of an open chain using the PoE formula in base or end-effector frame, the following elements are needed:

1. Base and end-effector axes chosen arbitrary.
2. End-effector configuration $M \in SE(3)$ when the robot is at its home position.
3. The screw axes expressed in either the base frame denoted $S_1, ..., S_n$ or the end-effector frame denoted $B_1, ..., B_n$, corresponding to the joint motions when the robot is at its home position.
4. The joint variables $\theta_1, ..., \theta_n$

For the PoE computations with regards to the base frame (which is the most usual approach), suppose that joint n (the last joint) is displaced to some joint value θ_n. The end-effector frame M will then undergo a displacement of the form:

$$T = e^{[S_n]\theta_n} M \tag{3.2}$$

Where $T \in SE(3)$ is the new configuration of the end-effector frame and $[S_n] = (w_n, v_n)$ is the screw axis of joint n expressed in the fixed base frame.

Further, if it is assumed that joint $n - 1$ is also allowed to vary, then this has the effect of applying a screw motion to link $n - 1$, which is connected to the end-effector frame via joint n. This will result in the following displacement of the end-effector frame

$$T = e^{[S_{n-1}]\theta_{n-1}} (e^{[S_n]\theta_n} M) \tag{3.3}$$

Computing this for all joints in between the base and the end-effector frame ultimately results in the Spatial PoE formula of all joint values θ:

$$T(\theta) = e^{[S_1]\theta_1} \dots e^{[S_{n-1}]\theta_{n-1}} e^{[S_n]\theta_n} M \tag{3.4}$$

For computational purposes, the screw axes S_i and B_i can intuitively be defined from a geometrical representation of a manipulator, a dimensional sketch in its home position. The following is defined for the components of a screw axis; the angular component w_i

is the unit direction vector of the z-axis through joint i represented in the chosen frame, while v_i is called the linear component which is the linear velocity of a point in the chosen frame. It is then easy to identify these screw axes by visual inspection, defining the z-axis (axis of joint rotation) for each joint and visualise a turntable of the joints intersecting the chosen frame to define the linear component. A typical screw axis for a revolute first joint mounted above a base frame is defined in equation 3.5 consisting of an angular component through the z-direction of the base frame and zero angular velocity impact on the base frame since the centre of the screw axis of the joint coincides with the base frame axis. Doing this for all of the joints makes it possible to define a matrix of the screw axes, often called an *Slist*.

$$\mathbf{S_1} = \begin{bmatrix} w_1 \\ v_1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{3.5}$$

As for the DH convention, there are a lot of available computational tools built to serve the purpose of solving problems related to kinematics, dynamics and everything else of interest within the robotics universe. Along with the in-depth explanation and use cases for the PoE convention, the Modern Robotics [87] book used in this thesis comes with a toolbox of functions supporting Matlab, Mathematica and Python.

3.3 Path- and trajectory planning

Path planning is a subproblem of the more general motion planning problem that is concerned with finding a feasible collision-free path between a start- and goal configuration. This subproblem usually does not take into account the dynamics, duration and constraints regarding motion and control inputs. If a time tag is specified for where the end-effector should be at each duration in time, a path is converted to a trajectory. A trajectory consists of a purely geometric path describing waypoints of a manipulator, and time scaling that specifies the times for when the waypoints are to be achieved [87]. In this thesis, only the directly geometrical path planning has been emphasised. Some work has been laid into the use of trajectory generating and trajectory tracking in the simulation part, but only using already built-in tools together with the generated paths. Paths are often described as a curve that is a function of a parameter, typically zero to one, where this parameter describes each configuration (denoted c_i) along the path as can be seen in equation 3.6. A path should be described within the configuration space of a manipulator for it to be feasible. The configuration space is defined as the space of all possible positions and orientations, hence all possible configurations with regards to allowed joint values. The workspace is defined to be the subset of the configuration space that the end-effector can reach, hence the outer edge of the configuration space [87].

$$\mathcal{P} : [0, 1] \to \mathcal{C} \quad where; \quad \mathcal{P}(0) = c_{start}, \quad \mathcal{P}(1) = c_{end} \tag{3.6}$$

A path is typically defined by a significant amount of points along the path, making it by definition continuous. Nevertheless, a path can be coarsely described by fewer points making it less restrictive for the application of use in trajectory tracking. As mentioned, a trajectory can be described as a path where the parameter is a function of time and not a "percentage". Therefore, it is possible to define a path by using initial trajectory generation, only disregarding the duration parameter.

There are a lot of different approaches to create and implement trajectories. Some of the most well-known approaches are trapezoidal velocity, polynomial and spline, which are presented below.

Trapezoidal velocity trajectories are piecewise trajectories of constant acceleration, zero acceleration, and constant deceleration. This leads to the characteristic trapezoidal velocity profile, and a "linear segment with parabolic blend" (LSPB) or s-curve positional profile. These characteristics make them relatively easy to implement, tune, and validate against requirements such as position, speed, and acceleration limits which are often preferred [115]. Due to this, the trapezoidal velocity trajectory generation is the most used method for initial testing purposes.

Polynomial trajectories interpolate a line between two waypoints using polynomials of various orders together with boundary conditions at each end. The conditions must be of one degree higher than the polynomial order. The most common polynomials are cubic (3^{rd} order) and quintic (5^{th} order) with their respective boundary conditions at each endpoint consisting of position and velocity, or position, velocity and acceleration respectively. Higher-order polynomials can be used but are usually not desired due to computational burden. Polynomial trajectories are useful for continuously stitching together segments with zero or nonzero velocity and accelerations since acceleration-based profiles are smooth unlike with trapezoidal velocity based trajectories. However, validating acceleration-based trajectories are harder because boundary conditions have to be set instead of directly tuning maximum velocities [115]. This may lead to overshooting between trajectory segments.

Spline trajectories are also piecewise combinations of polynomials. Unlike polynomial trajectories, which are polynomials that are time relative (one dedicated polynomial for every segment), spline trajectories are based on polynomials in space that can be used to create complex shapes. The time aspect is added when the end-effector is to follow the resulting splines at a uniform speed. There are several types of splines, but one commonly used type for motion planning is B-Splines (or basis splines). B-Splines are parameterised by defining intermediate control points that the spline does not precisely pass through but rather is guaranteed to stay inside their defined convex hull. These control points can be tuned freely to meet motion requirements within the bounds without worrying about overstepping the boundary points [115].

These approaches are limited to 3D position trajectories and do not include end-effector orientation. The orientation part of a pose is often a challenge due to multiplicity of so-

lutions for different orientations using, i.e. Euler angles. Therefore quaternions are often used to represent orientation unambiguously. A technique for representing orientation efficiently is called Spherical Linear Interpolation (Slerp). This technique finds the shortest path between two orientations with constant angular velocity about a fixed axis [115]. Using Slerp together with one of the above-stated trajectory generation approaches can efficiently create a trajectory or path for implementation in the ROSDS software programmatically. The mentioned approaches are all available in the Matlab toolbox, Robotic System Toolbox. They could be a great way of offline programmatically set the waypoints that are needed for MoveIt to compute trajectories using the cartesian path planning function. This function needs feasible and relatively close waypoints describing the core position and orientation through a described motion. MoveIt is primarily a kinematic motion planning framework, that is, it plans for joints or end-effector positions without emphasising velocity and acceleration. However, the MoveIt framework comes with the possibility of utilising post-processing to parameterise kinematic trajectories for velocity and acceleration constraints. It is worth noting that time parametrisation; hence trajectory following should not be emphasised unless a robotic system is correctly described. As long as not all parameters for the robot is set entirely right, it is better to stick to path following, which has been the chosen approach in this thesis.

3.4 Motion planning

Motion planning is defined as the problem of finding a robot motion from the start- to goal state that avoids obstacles and satisfies constraints such as joint- or torque limits [87]. Motion planning can typically be broken down into a hierarchy consisting of path planning, trajectory planning and trajectory tracking whereas the basic path planning deals with generating a feasible path (multiple points) from the start- to end state, trajectory planning deals with generating the time-based schedule for how to follow the path using defined constraints on, i.e. position, velocity and acceleration, and finally, trajectory following deals with the actual implementation of the planned trajectory using controllers to execute the desired trajectory within sufficient accuracy as is visualised in Figure 3.2.

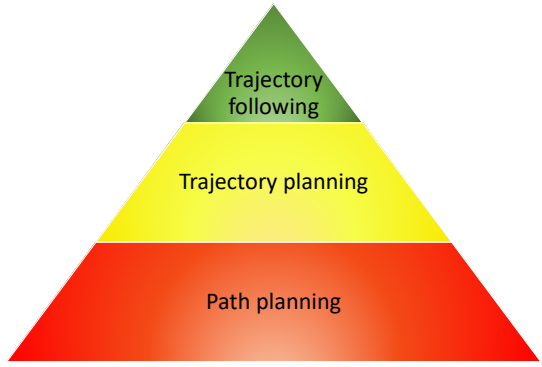


Figure 3.2: Motion planning hierarchy.

The basis of this hierarchy regarding path planning was presented in the preceding section. Therefore, this chapter will focus on the two latter parts of the hierarchy consisting of the joint-space trajectory planning and trajectory tracking using the ROS compatible motion planning library MoveIt. This was chosen since path planning, and path following are the topics that have been emphasised in this project. A note regarding this is that trajectory planning and following can be used in a segmented way to represent simpler path following by solving multiple A to B tasks. Since the path following was the main goal in this thesis, it was found suitable to use an existing framework for motion planning and available controllers to implement and test the paths created for simulating the identified tasks. The motion planning framework is therefore presented together with some of its available path planning algorithms and ROS controllers for implementation.

3.4.1 ROS trajectory controller

The ROS environment brings a vast range of opportunities to control the manipulator once it has been appropriately configured as done in section 2.3.3. Even though there have been created many tools for planning and controls of robotic systems by the large open community such as [116] or [117], the MoveIt package [118] is acknowledged as the most stable, versatile and user-friendly tool available. The MoveIt package was therefore chosen for the motion planning and controls of the manipulators in this thesis. This section will constitute the brief dynamic control description since this has not been the main emphasis in this thesis.

The joint trajectory controller (JTC) is one of the main building blocks of the use of MoveIt with robotic systems. The JTC supports multiple trajectory representations, but a default spline interpolator is typically used. This interpolator has a set of strategies that are applied based on the waypoint specifications. Either a linear, cubic or quintic strategy is utilised as mentioned for trajectory planning previously. The linear strategy does only take position into account. Therefore this trajectory representation only guarantees continuity at the position level, yielding trajectories with possible discontinuous velocities at waypoints which can lead to controller failure. The cubic strategy takes position and velocity into account, which guarantees continuity at the velocity level. The last trajectory representation of this controller, the quintic, takes position velocity and acceleration into account, bringing a guarantee of continuity at the acceleration level which is the desired continuity for a trajectory following simulation or real-life robot motion. This last representation is the one used by the JTC in most cases where the trajectory can create a path based on position, velocities and accelerations. Following the quintic line segment, the code for generating the trajectory segments and implementations, checks if these parameters are present at each timestep, then the controller chooses which representation to use based on the available information. As for the implementation and completion of the trajectory generated, there are five different controllers (for each joint) possible; position-, velocity-, effort-, position- + velocity- and position- + velocity- + acceleration controllers. It is important to note that the chosen controller type must correspond with the hardware interfaces chosen for the robot. The ROS control scheme can be seen in Figure 3.3 and describes the basic control scheme. First, a controller manager is initiated and loaded with the chosen controller type(s). The controller(s) in this controller manager is then loaded

into the controller hierarchy where parameters are set from its setup file (typically PID gains, publishing rate and other constraints based on chosen controller). Further, the controller(s) are communicating with the hardware interface using a resource layer to convert signals into for example torque commands and feedback of joint positions. This is the input and output layer of the controllers to steer the simulated robot where the hardware interface is the connection with the simulated (or real) robot. Using a connection between the hardware interface and a real robot makes it possible to control a digital twin (the simulation) and real hardware as showed in [119]. In this thesis, the joint trajectory *ROS interface* is used by the controller manager using effort controllers with corresponding PID gains. This controller is using the hardware interface of the actuators to communicate with the robot hardware which in this thesis ends with the simulation even though it is possible to fully integrate it with a real robot as seen in the last part of the scheme.

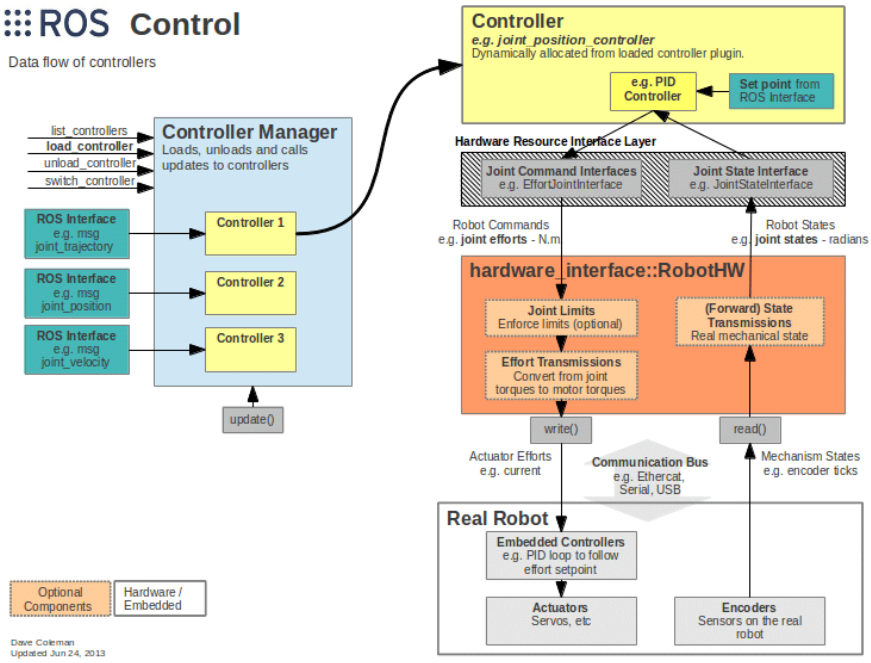


Figure 3.3: ROS controller scheme [120].

3.4.2 The Move Group Node

The move_group node (MGN) that the whole MoveIt package is built upon can be schematically outlined, as seen in Figure 3.4. An MGN is a part of the final package that comes from running the MoveIt setup assistant. The setup assistant uses the URDF description and given inputs through the setup stages to define a semantic robot description format (SRDF) which describes semantic information about the robot. This description includes

information that the URDF does not contain such as chained links, a grouping of links and joints, virtual joints and collision properties. Using this and multiple other auto-configured files, a Move Group can be created that is inherently connected to the input robot description. Further, connecting the created MoveIt package to a robot simulation opens up the possibilities that come with the package. The MGN offers a lot of predefined actions as services to get information about the robot state continuously or control it in various ways such as the use of cartesian paths, joint position- and velocity input or pose definitions. Also, obstacles can be added to the planning scene to limit the room of motions of the manipulator, hence making the node plan for the obstacles present in the scene. Since there are some known bugs and limitations with the cartesian path planner in MoveIt, there is ongoing research on a project for performing path planning on "under-defined" Cartesian trajectories, named Descartes [117]. This is to-date not a finished product but is being developed by ROS industrial and the community, showing promising results for offline programming of manipulators.

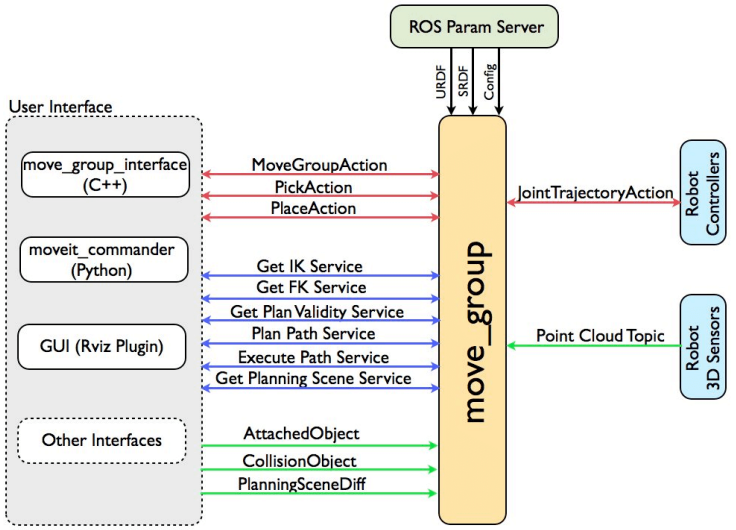


Figure 3.4: Move Group Node [118].

3.4.3 MoveIt planners

The MoveIt software is designed to work with a variety of motion planning frameworks to be able to make use of specialised planners for different operations and to compare performance. There are currently three available frameworks for integration with ROS, but customised motion planning plugins can be made and integrated if desired. The documentation, integration and availability of the available motion planning frameworks vary greatly, and some of the frameworks do need a binary install. In descending order of popularity and support within the community, the available planners are; Open Motion Planning Library (OMPL), Stochastic Trajectory Optimisation for Motion Planning (STOMP) and

Covariant Hamiltonian Optimisation for Motion Planning (CHOMP) where the OMPL is the default used in the MoveIt setup. These motion planners or planning libraries will be discussed briefly before more thorough research within the chosen planning library is presented.

OMPL is an open-source motion planning library that primarily consists of multiple randomised sampling-based motion planners. The planners in the library are said to be abstract; that is, the planners are general and have no concept of what a robot is. Therefore the planners themselves have no code for performing collision checking or visualisation. Therefore they have to be connected to such code in the front end of applications [121]. Nevertheless, this is the best-documented planning library available and is highly integrated into MoveIt. The library consists of multiple planners that are divided into two categories, geometric planners and control-based planners. The geometric planners only account for the geometric and kinematic constraints of a system while control-based planners take into account eventual differential constraints that are present. These planners rely on state propagation instead of simple interpolation to generate motions. Within these two categories, there are lots of different algorithms with different characteristics which should be examined closer. The research group from Rice University [121] that created the OMPL conducted a review in 2012 describing the library, its functionalities and future possibilities. Even though the library has grown significantly since then, both in users and functionalities, there are some key takeaways. The library was designed in a way that facilitates contributions from other motion planning researchers and is constructed such that all planners are defined generically. This was done such that the different planners can be efficiently benchmarked or tested against each other and finally designed for simple integration with other software packages such as ROS.

STOMP is an optimisation-based motion planner which is based on [122] and is an algorithm for planning smooth trajectories, avoiding obstacles and optimising constraints such as minimising torques, i.e. creating a trajectory with minimal energy usage [123]. The integration of this planner is a work in progress and have, therefore, not been further explored.

CHOMP is a novel gradient-based trajectory optimisation procedure that is based entirely on trajectory optimisation. That is, the algorithm uses covariant- and functional gradient approaches to the optimisation stage of motion planning instead of the standard approach for high-dimension motion planners that separate trajectory generation into distinct planning- and optimisation stages. The planner can handle infeasible or naive paths by pulling the trajectory out of collision territory while simultaneously optimising dynamic quantities such as joint velocities and accelerations [124]. This planner is of great interest within the robotics community. Unfortunately, the integration of this planner into MoveIt is also a work in progress but could be built from source if desired.

Due to the support and full integration of the OMPL, this stood out as the obvious choice in this thesis since integrations and test of different planners was not the main priority. As mentioned, the OMPL library comes with a vast amount of different planners with differ-

ent characteristics. It was therefore opted to either test the readily available planners or find previous research within the field for choosing one or multiple appropriate planners within the library.

A research group from Sevilla [125] did a review on planning techniques for a multi-rotor UAS equipped with a manipulator's arm using the different solvers of the MoveIt plugin of Gazebo/ROS and comparing their planning times, failed attempts and execution times. Based on iterative simulations, the RRT-connect algorithm (Bidirectional Rapidly-exploring Random Trees) were found to be the most suitable algorithm to be used for both the UAS and the manipulator. The algorithm presented the shortest path planning time; the least failed planning attempts and similar execution times as the other algorithms.

"*This planning algorithm combines the major advantages of the RRT algorithms, a random fast exploration, with the Connect heuristic, finding a fast solution connecting the goal with the actual pose through launching an RRT tree from each point.*" - Ragel et al. [125].

This matches the description from the OMPL documentations itself which is that RRT-connect (the bidirectional version of standard RRT) usually outperforms the original RRT algorithm which is a simple single query planner that is easy to understand and implement. This planner was, therefore, the initially chosen planner used in the simulations in this thesis. The simple one-directional RRT principle, as described by Kuffner and LaValle [126] can be seen in Figure 3.5.

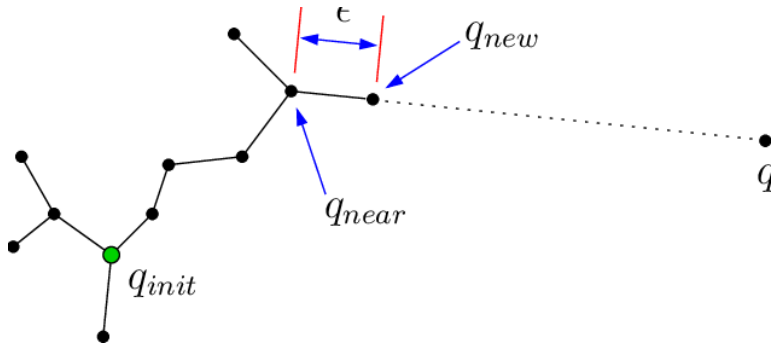


Figure 3.5: Basic rapidly-exploring random tree [126].

Even though the RRT Connect algorithm was found to be the fastest and most reliable planner for various tasks, some typical drawbacks were defined by Strandberg in [127]. The "narrow passage problem" is stated as a typical drawback for various types of RRT planners. He states that RRT algorithms typically throw away potentially valuable samples along the way if the active tree(s) is not able to connect correctly. This is typically the case for narrow passages where the randomly exploring nodes will collide with obstacles or infeasible territory. It is also noted that goals should not be set too close to each other when using RRT algorithms due to query inefficiency. This paper presents a possible solution

to some of the shortcomings with the RRT Connect algorithm by creating local RRT's to cope with the narrow passage problem. Nevertheless, this has not been put to the test in this thesis since the OMPL was chosen. However, it is stated that the use of traditional Probabilistic Roadmap Methods (PRM) could be a great substitute if RRT algorithms were to fail. Such algorithms are available in the OMPL.

3.5 Summary

This chapter has presented the necessary theory and researched tools to be able to implement more dedicated simulations. More dedicated simulations will involve multi-goal path planning in Matlab using screw theory and motion planning using MoveIt. What is presented in this chapter will be utilised in the coming chapters. First, for the previously described generic manipulator, and finally, for the Reach Bravo manipulator completing presented operations.

Chapter 4

Motion planning and control of generic robotic manipulator

This chapter describes the approach for implementing more sophisticated controls than the simple position controllers described in chapter 2. This includes the configuration of MoveIt for the manipulators, a functional library for robot manipulation tasks customised to run on top of ROS, programmatic control of the manipulators using the C++ API of MoveIt, multi-goal path planning in Matlab, workspace generation for the manipulator and a validation step for testing and tuning controls in simulations. The aforementioned is first done in a general way for the generic manipulator before it in a straight-forward way is implemented with the specific manipulator to solve the identified tasks in the next chapter.

4.1 Configuration of MoveIt

The configuration of the MoveIt package will not be emphasised significantly since there are a lot of sources available online and an excellent tutorial in the official documentation at [118]. Nevertheless, a short description of the setup will be provided before a more thorough explanation of the connection between the configured MoveIt package and the Gazebo simulation configured in section 2.3.5 will be presented. This is done because it seems like one of the more complicated steps of the implementation of controls using this approach. Once the setup and connection have been obtained, it is possible to control the manipulator using the move_group node (MGN), either using C++, Python or the GUI in RViz. For the implementation of a path- or trajectory following, the node should be controlled programmatically since the GUI has limited capabilities and works best for validation purposes using pre-defined configurations. The use of C++ has been emphasised in this thesis, but the use of C++ vs Python is very similar in characteristics, though Python is often associated with being simpler.

When a manipulator or robotic system has been completely defined in URDF format as described in section 2.3.4, a MoveIt package can be efficiently created using the MoveIt setup assistant spawned in the graphical tools window using the *roslaunch* command in listing 4.1. The use of a MoveIt package is typically referred to as using the node, MoveGroupNode (MGN), MoveGroup interface or the move group. It could vary throughout this thesis based on the context.

```
roslaunch moveit_setup_assistant setup_assistant.launch
```

Listing 4.1: Spawn MoveIt setup assistant.

The first part of the setup is to locate the URDF definition in its file location and to import this configuration into the setup assistant. Next, a self-collision matrix must be defined manually or by using the default self-collision matrix generator which in terms searches and tests pairs of links if they can directly collide with each other. If links are identified for example adjacent; hence not able to collide, the collision checking between these links are disabled, resulting in a decreased motion planning processing time. The sampling density specifies the number of random positions to be checked for self-collision. A higher number will increase computation time in the setup assistant, but increase precision, i.e. secure that link pairs that should not be disabled will be disabled. Nevertheless, for a simple manipulator, there will be no difference in using the minimum default value of 10 000 checks versus a more significant number.

The next part is to define virtual joint(s). Virtual joints are primarily used to attach the robotic system to the world. Therefore it is often defined as one virtual joint for the robotic system, either of type fixed for a stationary robotic system or a floating virtual joint for a dynamic robotic system. The next step is to configure the planning groups for which the node is to compute and implement the desired motions. Planning groups are used to describe different parts of the robot semantically. The parts can typically be a planning group for the virtual joint (if the robotic system is dynamic, else this can be neglected), a dynamic chain such as a robotic manipulator, or an end-effector if this is to be controlled, i.e. using a gripping tool. The most efficient way of defining a robotic arm is to describe it using the option of defining a dynamic chain from the base up until the last link before the end-effector (or also add the end-effector if it is supposed to be fixed to the arm). Further, predefined poses such as various "home" positions can be configured and stored within the package that is being created. The package can then be configured for the designed robotic system and placed in a chosen workspace.The setup assistant creates a standard ROS package with all needed files such as the *package.xml* and *CMakeLists.txt* files in addition to launch- and config folder containing all needed files for virtual control of the manipulator in RViz visualisation. This involves a file for semantic information about the robotic system (SRDF), planning plugin (default using the OMPL Open motion planning library for the defined planning groups), kinematics (what solver and settings to be used), joint limits (additional or overwriting information about the joints) and some unconnected controllers to be used for testing purposes only. The defined robotic systems planning package can be tested launching the demo file in the created package. Nevertheless, the simulations will not be carried out in the Gazebo simulation environment since the MGN

has not been connected to the controllers spawned in the Gazebo simulation environment. There are multiple ways of configuring this connection. However, in this thesis, it has been emphasised to use a step by step solution creating new controller files and not tweak the auto-generated files. This will now be presented thoroughly.

To implement specific waypoints, poses or joint state configurations in the simulation environment, the MoveIt configuration must be connected to the actual simulated (or real) robot hardware interface. This is done by reconfiguring or replacing the joint controllers from section 2.3.5, the position controllers of each joint, to something called a joint trajectory controller. This type of controller is connected to all of the joints and makes it possible for the node of the MoveIt configuration to move the different joints simultaneously. The whole MoveIt configuration is built around this controller for the robotic system to be able to move smoothly and for the MoveIt to be able to solve IK and apply them to the actual robot. This type of controller requires a new file for describing the control implementation, which is a bit different from the simple positional controllers. This is done by defining one unifying controller of the type *JointTrajectoryController* including a list of the joints, possible constraints for the controller and the PID gains for the different joints. Such a controller can be based on effort, position, velocity or a combination of these controller types that have to be specified for each joint in the URDF file (i.e. the transmissions). A joint trajectory controller has an action interface of the type *FollowJointTrajectory* which is the action interface needed for the MoveIt configuration to be able to control the real or simulated robot. The controller type is made for executing joint-space trajectories on a group of joints where trajectories are specified as a set of waypoints to be reached at specific time instants for which the controller attempts to execute if the mechanisms allow it. The trajectory points consist of positions (joint angles), and velocities and, if applicable, accelerations. The new trajectory controller can be defined as seen in listing A.13 using effort controllers for the transmissions. If another transmission type is desired, either a new URDF file can be created, or the previously used transmissions can be redefined. Either way, if another transmission type such as position joint interface or the described effort joint interface is used, the launch file of the controllers has to be rewritten to spawn the defined joint trajectory controller in listing A.13 instead of the position controllers used in the previous chapter.

Some debugging that can come in handy is to launch the Gazebo environment and check the *rostopic list* for what topics are being published. If all topics needed are available, everything is facilitated for connecting the joint trajectory controller with the topic called *follow_joint_trajectory*. This is done by adding a file for the controllers in the MoveIt package as seen in listing A.14 (Appendix). The next step is that the controller manager of the MoveIt node must be told the name of the *Action Server* that was just created (the topic of each joint). This is done by adding a file for the joint names to be identified. Also, the auto-created controller manager file should be filled with the launching of the MoveIt controllers as seen in listing A.15, making it possible to apply trajectory control during runtime.

The "backend" of the MoveIt package will then be able to connect and execute commands

on a spawned robotic system using the approach described in section 2.3.5. The only thing missing is a way of actually implementing the package with the simulation. This is done by creating an implementation (launch) file which ultimately spawns the *Move Group Node* with all its inherent functionalities. Launching this file after the robot with its joint trajectory controller (JTC) spawned will start the *MoveItSimpleControllerManager* and load the JTC interface defined in the controller file. The implementation file launches the visualisation (RViz) of the planning interface, the planning context and the Move Group Node. Inside the implementation file, it might be necessary to remap the joint states in the simulation to the source list to obtain the necessary joint states for feedback.

The configuration can be tested by launching the created files in a structured manner as can be seen in listing 4.2. By using different shells, the two launch files should be run after a Gazebo simulation is initiated. It is advised to start off with an empty Gazebo simulation and then implement other scenarios such as self-defined worlds from section 2.3.5 later. Once the launch files are started, the manipulator is spawned in the Gazebo environment as in section 2.3.5. The MoveIt node is then initiated, and the Graphical tools can be opened. Further, the planning framework can be used in the GUI setting different poses using the start and goal state inputs. By using these inputs, motions can be planned in the RViz visualisation and executed in the Gazebo simulation. This setup offers direct access to utilising all of the features in the MoveGroup Node using programming, which is described in the next section.

```
1  roslaunch ROBOT_FOLDER SPAWN_FILE.launch
2  roslaunch MOVEIT_FOLDER MOVEIT_EXECUTION_FILE.launch
```

Listing 4.2: Running MoveIt GUI.

4.2 Programmatic control of MoveIt using C++

The mentioned approach works as a validation step for the setup of the MoveIt node and should, therefore, be done before any programmatic control is to be implemented. Once the MoveIt implementation is validated through tests using the GUI, it is facilitated for taking the next step into programmatic control. Programmatic control is necessary for simplicity and efficiency when testing and implementing new goals, trajectories and paths in the simulation environment. Programmatic control of MoveIt will facilitate the use of other software tools such as Matlab for path planning and possibly controls. In this thesis, the focus has been on controlling the manipulator programmatically offline for fulfilling multi-goal paths created in Matlab for specific operations in smolt production as a proof of concept. By this, it is meant that the goal was to implement paths containing "waypoints" of poses defined in the task space (position and orientation) by using the MoveIt framework for the actual motion planning and implementation. The programming language was chosen to be C++ due to knowledge and the availability of documentation.

The C++ programming in ROS follows a typical object-oriented characteristic by defining a header file for the declaration of classes and functions (or methods), a cpp file for the

implementation of the functions and finally a run (or main) file for the execution of the functions and classes in the main loop. In C++, defined functions (naming used in other programming languages) are called both functions, methods or member functions. In this thesis, functions have been mostly used, but some places another naming might appear. As a start in implementing programmatic control, it is possible to neglect the two basic files (header and construction) and create only a main run file directly. This can quickly become messy as functionality and complexity is increased and was therefore discarded. For generic and user-friendly code, it was decided to insert all of the functions into a dedicated planning class and to use a switch in the main script to run the different functionalities from a shell during runtime easily. There are a few prerequisites when creating a cpp work package that needs to be in order before any code development can be done. This will now be presented and discussed, before an excerpt from the first programmatic control code will be presented and explained. For full code, see Appendix D.

As for any ROS package, the package has to be built together with its required dependencies. This is done in the same manner as for other created packages only creating a new package dependent on *roscpp*, a package needed for C++ communication with ROS. The setup can be defined as in listing 4.3 where FOLDER_NAME is the name of the new C++ based package.

```
1 cd catkin_ws/src/
2 catkin_create_pkg FOLDER_NAME roscpp
```

Listing 4.3: Create C++ package.

All pre-made and necessities such as functions, objects and message types are typically included in the header file for use in customised classes and functions. The community is continuously developing new tools that are open-source and available for use by merely installing the files and including them in the header file if an offline Linux based approach is chosen. If an online environment, such as the ROS Development Studio is used, most of the well-known plugins are directly available. To work with the MoveIt configuration or the move group node, the API documentation of the MoveIt ROS planning interface is of great use [128] [118]. The C++ syntax will not be explained in detail, and only a small excerpt of the developed code is presented in Appendix C defining a rough sketch of how to set up the programmatic control of the MoveIt group using a header, cpp and run file. For full code, see attached files in Appendix D. In the example for the generic manipulator, three files have been created in the source folder of the created MoveIt package. These files are the header file (planning.hpp), the configuration or source file (planning.cpp) and the implementation file (run.cpp). This file setup is also used for the specific manipulator code that can be found in the zip attachment in D.

The header file is a file for minimally declaring all functions only describing their return type (typically void for functions doing work and not returning anything), name and inputs. In the start of the header file seen in listing C.1, the necessary header files are included to be able to work with and communicate with the MGN and other standard ROS libraries. To start with, the created class named *PlanningClass* has been defined within a

namespace corresponding with the robot name. This is done for the possibility of creating multiple instances of the same class with the same names of functions and variables. It is not a necessity to use namespaces, only a precaution for future usability. Further, the class is constructed, and the move group initialised using an in-line initialiser list. Note that the public move_group is initialised using the private constant name corresponding to the name of the MoveGroup created using the MoveIt setup assistant. The rest of the private features of the class are related variables of the move group, either it is generated plans stored and accessed using the my_plan variable, or interaction with the planning scene using the virtual_world variable. Upon construction of the class and initialisation of the move_group object, some parameters of the object can be set. This might include allowed replanning, how many times the motion planning library should be able to try and solve a given problem, or the declaration of the pose reference frame for calculations to be the base frame of the manipulator. Changing these default values in the C++ files or other properties in the ROS files will require a rebuild of the package for the changes to be applied.

The cpp file with the corresponding name of the header file is the file for the actual implementation of the functionalities within the declared functions in the header file. This file can be looked at as a fill to the declared frame in the header file as seen in listing C.2 (Appendix). In this file, it is not standard to include other files than the namesake header file. However, if additional functionalities are required such as input/output streams for reading files or similar, these can be included in this file. Using the same namespace as defined in the header file, the functionalities of the void functions from the header file can be constructed by creating their bodies within the curly brackets. The described functions in the construction file in listing C.2 (Appendix) will now be shortly described to give a picture of what capabilities lay within the *MoveGroup*.

The first function (*goToPoseGoal*) takes a pose of the Moveit type found within its geometry messages as input. This consist of a position part of x, y and z coordinates described in the base frame and an orientation part based on a quaternion definition of w, x, y and z. These parameters of a pose can be defined as seen in line 25-31 in listing C.2 (Appendix). Quaternions are not easily understood or visualised since it is described in 4 dimensions [127]. Therefore it is often wise to use visualisation tools for help with describing the orientation of poses, for example, using Matlab's function *tform2quat* together with Euler angles transformation *eul2tform* that is more intuitive. If a well-defined pose is inserted into the function, the sub-function of the move group node, namely the *setPoseTarget*-function is fed the pose. This function connects to the planning interface and sets the pose as the next target exactly the same way as a pose target can be set using the GUI in RViz, ultimately feeding the target to the private variable *my_plan*. Next, the planning function of the move group is initiated. By working behind the scenes, this function uses the chosen planner connected to the Moveit node to try to plan a trajectory from the current pose to the input pose, returning a Moveit message. It is, therefore, added a check for the success of the planning. If not, an error will be thrown. Finally the move group function *.execute()* executes the plan that has been "uploaded" to the move group if it is valid, else nothing will be executed in the simulation. The trajectory planning deals with path planning from

initial state to goal state and trajectory planning using a polynomial approach as described in section 3.3. The execution command corresponds with the trajectory tracking part, implementing the motion in the simulation using the transmissions and controllers.

The next function implements the *cartesian path planner* function of the Moveit group, following multiple desired waypoints (poses) to its goal state. This is an excellent function for simple path following applications, but could also be used for more complex trajectory tracking applications. Firstly, a vector is defined containing the desired waypoints. Also, some parameters related to the constraints of the trajectory following are defined; this can also be defined globally in the initiation of the move group. Seen from the *Pose* description in the last paragraph, manually creating waypoints is a tedious and laborious task due to many lines of code and the inexplicable quaternions, especially for more complex paths. Therefore, it was decided to use Matlab for the path planning and definition of the poses. This will be further explained in the subsequent sections. Nevertheless, if poses are defined in csv format or similar it can be efficiently read into the C++ script and used to create the waypoints as seen in lines 33-62 of listing C.2 (Appendix).The last step is to utilise the move group function *computeCartesianPath*, which takes the desired poses as inputs together with end effector step size (the resolution of the trajectory in meters). Also, jump threshold (maximum allowed change in distance in the configuration space, preventing jumps in IK solutions) and a trajectory of the MoveIt message type *RobotTrajectory* for where the computed trajectory is to be placed must be specified. This function returns a double between 0 and 1, defining the percentage of the trajectory the planner was able to compute. There are numerous sources of error related to this "black box" trajectory computation. However, the function is typically not able to compute the full trajectory if waypoints or trajectory points in between the waypoints are infeasible. Another problem is identified for a case where the current state is too far away from the first waypoint resulting in a significant "jump". Once the trajectory is completed, it is inserted into the plan of the move group using its related trajectory function. From there on, the planned trajectory (stored in the move group's plan variable) can be executed using the function with the identical name. It is worth noting that there are significant shortcomings within the functionalities and the debugging capabilities of the cartesian path planner of Moveit, which is discussed in the final part of this thesis.

To be able to run these defined functions during runtime, an executable run file has to be defined and adorned with the mentioned functionalities. In this main file, a ROS node has to be initiated to be able to perform any remapping that is provided in the command line. There are many ways of doing this, but the described way in line 7 of listing C.3 (Appendix) is the most straight-forward way. A ROS node has to be initialised before using any other part of a ROS system. A node handle is the main access point of communication with the initiated ROS system. An asynchronous spinner is initiated and used in order to cope with the challenge of a regular synchronous spinner that callbacks can end up in an "order book" waiting for the previous callback to be completed. An asynchronous spinner is a threaded spinner and will therefore give each callback their dedicated thread if possible. The next part is merely checking for console input. *Argc* is the count of arguments, and *argv* is the command-line arguments where the program itself always is the first argu-

ment, hence *argc* is always greater or equal to one. Therefore, when user input is added when running this script, the *argc* will increase to 2 and move on, else the instructions on how to use the script will be printed followed by error messages. As seen in the listing, the usage of the C++ functionalities are dictated by the following rules; run n, where n is the second argument in *argv* that is used as an integer in the switch to choose which function to run. This approach is defined by the build of the package, which will be discussed in the next paragraph. If the input in the console is as expected, an instance of the class is created (line 19) for interaction with the running simulation. A complete simulation must be active in the background, including the right robot with its spawned controllers and MoveIt configuration as defined in the previous sections. The final step in this script is to apply the input argument into the switch and running the created functions.

Even though all files have been properly defined without any bugs, they need to be defined as executables such that they can be executed during runtime. This can be done using commands in the shell but is more clearly done by changing the package files. In this step, it might be necessary to add additional packages if other plugins have been used in the scripts.For the full implementation, see appendix D. Once the setup is complete, the code can be built using the command *catkin_make* in the correct workspace, and run in the console using *rosrun* as seen in listing 4.4. This is done as follows; *rosrun* the chosen executable script in the planning folder (here; run which has been added as a target link executable) and choose input argument. In this listing, the input argument is set to 1, resulting in the execution of a go-to pose goal command defined in line 33 of listing C.3 (Appendix).

```
1  cd catkin_ws
2  catkin_make
3  ## WAIT FOR BUILD ##
4  rosrun MOTION_PLANNING_FOLDER run 1
```

Listing 4.4: Building and running C++ scripts.

There are numerous possibilities within the *MoveGroup Interface* such as moving to a pose goal (position and orientation), moving to a joint state target, computing a cartesian path in 3D space defined by waypoints (ultimately trajectory tracking, but can work as path following too, which will be further explained), redefining planning scene interface by adding obstacles of various shapes and sizes, and a lot of other possibilities. In this thesis, the focus has been on utilising the cartesian path planner, and pose- and joint goal functionalities of the MoveGroup interface.

4.3 Implementation of generic manipulator and verification of concepts

There are multiple ways of generating a robot path. It can be online and dynamic, offline or a combination of the two. In this thesis, the path planning in operating space has been done offline by using some of Matlab's great features for trajectory generation. It was chosen

to define the multi-goal paths in the operating space, i.e. by defining multiple waypoints consisting of end-link position and orientation values. This was chosen since it is intuitive to work with and easy to plot. Also, the MoveGroup is compatible with inputs on the form of cartesian position and quaternion orientations.

4.3.1 Setup in Matlab and data retrieval from ROS

The multi-goal path planning was done using Matlab functions for creating homogeneous transformation matrices (*trvec2tform*), the Euler angles transformation (*eul2tform* for specifying the intuitive orientation of transformation matrices and the trajectory generator (*transformtraj*) for creating multiple transformation matrices in between two homogeneous matrices. This latter Matlab function is not strictly necessary since each of the defined waypoints (homogeneous transformations) can be set individually, but is nice for generating more precise paths or when entire trajectory generation is desired. In addition to these functions, the *plotTransforms* function for plotting homogeneous transformations in 3D space was used to visualise the generated waypoints. This function takes a 7x1 array as input consisting of position (x, y and z) and a quaternion (w, x, y and z) which can be obtained from each transformation matrix by using *tform2trvec*, a function for extracting a translation vector from a homogeneous transformation matrix, and *tform2quat* for extracting a quaternion representation from the same matrix.

By using the mentioned functionalities, a path defining half a circle was constructed as seen in listing B.1 (Appendix) resulting in the plotted waypoints seen in Figure 4.1. The Matlab script starts by initiating the figure with some initial parameters. The next part involves creating a trapezoidal velocity profile for later use in the creation of the trajectory (or path in this case). This part ultimately defines how many waypoints are to be created between two homogeneous transformation matrices. By using the defined velocity profile s, sd and sdd (position, velocity and acceleration) for the points, the *transformtraj* function can create the trajectories in between the transformation matrices. Here, the trapezoidal velocity profile has been chosen, but any of the velocity profiles in section 3.3 or others can be used, generating different characteristics of the generated paths. In this thesis, the trajectory generation functions have been used to create multiple transformation matrices, which ultimately results in all of the exported poses after the transformation matrices have been converted to arrays consisting of position and quaternion orientation. In this process, the possible timestamps of the different poses are discarded; hence the output is a list of waypoints (poses) on the form that MoveIt uses.

As a test of the mentioned approach for creating compatible paths for MoveIt, a simple algorithm for creating a half-circle multi-goal path based on the desired radius, number of imposed trajectory points and height were defined as seen in listing B.1 (Appendix). This algorithm was supposed to replicate a simple cleaning motion to validate the path generation, and the programmatic controls integration. Initially, a starting pose is generated, plotted and stored in an array consisting of the poses. Further, a generated θ list containing angular values in radians is looped through and in an iterative manner, new poses are generated and stored in the array with the specified interval of angle.

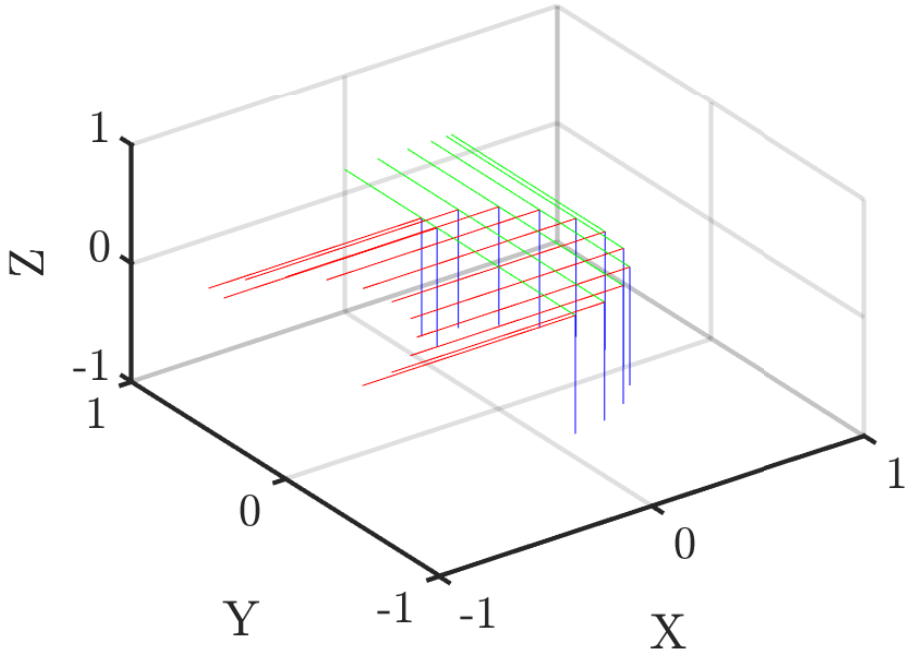


Figure 4.1: Path for a half circle motion (red, green and blue correspond to the *x*-, *y*- and *z*-axis respectively, defining a pose).

Even though a generated path is containing only feasible poses, there are some easy ways to make sure the waypoints are processable for MoveIt. As mentioned, all points must be strictly increasing with both time and placement; hence it is smart to loop through the generated waypoints from the path to remove potential duplicate points. This is especially common if more than one waypoint is created between the homogeneous transformations as specified in line 1 of listing B.1 (Appendix). Therefore, a Matlab script was developed to remove these duplicates called *cleanDups.m*. This script takes a matrix as input, removes duplicate rows with approximately the same points (similar first three elements of a row by four decimals) and returns the cleaned matrix as seen in listing B.2 (Appendix). Once the matrix is cleaned, it can be safely used in the ROS setup, hopefully resulting in less inexplicable errors. In this project the created paths have been exported to csv using the Matlab function *writematrix* and exported to the workspace of the ROS setup, namely into the browser environment of ROS Development Studio.

Using the programmatic approach for implementation of a path from a csv file in listing C.2 (Appendix), the test path was successfully loaded, planned and executed using the joint trajectory controller and the Open Motion Planning Library with its RRTConnect planner. The implementation on the spawned generic manipulator positioned in the origin of the Gazebo simulation environment mid-motion can be seen in fig 4.2.

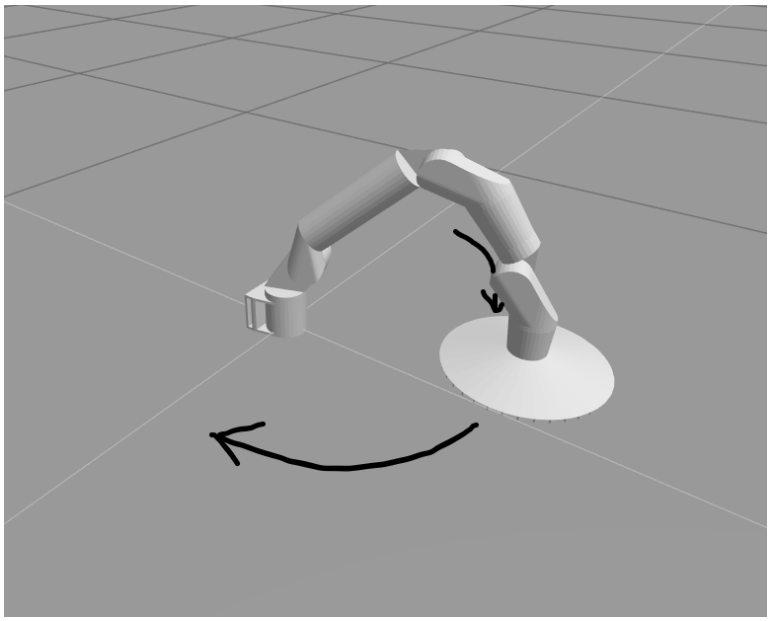


Figure 4.2: Generic manipulator in the midst of a circular path motion.

From the Gazebo simulation window, the path following seemed quite good based on qualitative observations. Nevertheless, it is a necessity to be able to check the actual behaviour versus the desired. The ROS community has created a package that is built to do precisely this, store and also play simulation data. This package is called *Rosbag* and can be operated either using the command window simultaneous with running simulations or programmatically using C++ or Python. Initially, the command window approach was used for simplicity. The Rosbag plugin is a builtin feature in the ROSDS. Using other offline solutions may require installation. If the package is installed, the recorder can be started using the command lines in listing 4.5 where at first a new temporary folder is created, the active file location moved there, and the recorder is started for the desired topics. In this case, the joint trajectory controller that is used for controlling the manipulator using MoveIt is publishing its states at each time step. These states comprise the desired, actual and error joint positions and velocities. The states also include effort and acceleration for each joint, but these are not used in practice. It is therefore, only of interest to compare the input position and velocity for each joint at every timestep. The Rosbag recording can be stopped by using the kill command in ROS, CTRL+C in the same shell.

```
1  mkdir ~/bagfiles
2  cd ~/bagfiles
3  rosbag record -O BAG_NAME /ROBOT_NAME/
       joint_trajectory_controller/state
```

Listing 4.5: Initiation of Rosbag.

Matlab is compatible with ROS and has therefore also functions for working with and interpreting Rosbag files. The functionalities of Rosbag reading in Matlab are extensively documented, and only a short description of how it has been used will therefore follow. A Rosbag consists of the topics that are stored inside the file. By using selection criteria such as topic and time interval, the stored topics were separated and condensed into smaller, more manageable datasets since the topics are stored at a high frequency, including many messages. Further, the messages were extracted from the topics into an array of objects with their respective features. This large amount of data in the messages were further processed by extracting the position, velocities, timestamps and so on to matrix form for more straightforward processing. In the final datasets, the desired values correspond with the planned values determined by the planning interface as discussed previously, and the actual values correspond with the recorded joint states (the feedback). In this regard, the manipulator uses the error in positions and velocities at each time step as input to the PID controllers for following the desired path.

4.3.2 Tuning and test of joint trajectory controller for better response

The simulation results regarding the joint positions and velocities for experimentally set PID gains can be seen in Figure 4.3 and 4.4. The plots are running from 160 to 200 seconds which is the Gazebo simulation running time. The path is sent to the motion planning node after approximately 172 seconds, where the motion involving the waypoints is calculated as a trajectory. The torques for fulfilling the position and velocity requirements at each timestep are then applied. As can be seen from the plots, the manipulator can follow the desired path almost flawlessly with some minor problems with the velocity for some of the joints when complex motions involving multiple joints at the same time is being completed. Almost no overshoot or standard deviation can be spotted in the plots suggesting that the integral and derivative gains are properly weighted relative to the proportional gain.

From the initial draft of the generic manipulator in Figure 2.10, the "tasks" of the different joints can be identified. These tasks reflect the identified behaviour in the plots also whereas the first joint is rotating the whole manipulator in the half circular pattern, joint 2, 3 and 5 are controlling the pose of the end-effector relative to the first joint, and joint 4 and 6 corrects deviations in the angle of the end-effector. The functionality of these last joints are not very important in this simulation since an end-effector tool for cleaning most likely will be a continuously rotating tool. These joints are therefore not directly "used" in this operation as can be seen from the minimal changes in the value of joint 4 and the simple twist of joint 6 turning π^o. The oscillating behaviour of joint 2, 3 and 5 is not expected from a path planning point of view since it should be possible to find an initial pose and use joint 1 for rotations to fulfil the simple cleaning path. Nevertheless, it is believed that the MoveIt planner is planning for such an oscillating motion since the generated desired trajectory coincides with the actually implemented trajectory of the manipulator. This should be closer examined, but the effects are small and hard to observe in simulations. In this experiment, the PID gains were set experimentally by applying a standard rule of thumb to assign a high proportional gain, a thousand part integral gain and a ten thousand part derivative gain. The values were set arbitrarily such that the Gazebo model was able to overcome the simulation environment (the gravity) and the weight of

the joints during the first validation steps in section 2.3.5. It is worth noting that also the weight of the links has been set randomly with the basis of keeping the total weight of the manipulators at relatively low weight (10 kgs) and increased weight towards the base. PID tuning will play a crucial role if a correctly described manipulator is to be simulated. This also applies for all other properties such as joint limits, effort and velocity limits. In this experiment, the proportional gain of the rotating joint connected to the base and the first link was set to 5k while the rest were set to 2k . Even though some velocity and joint limits were set initially in the URDF configuration, these values can be overwritten in the joint limits file such that no limits are set for the controllers. This was done in this initial simulation since it was supposed to be a proof of concept. Therefore some "strange" poses were possible to obtain even though they are not feasible in the real world.

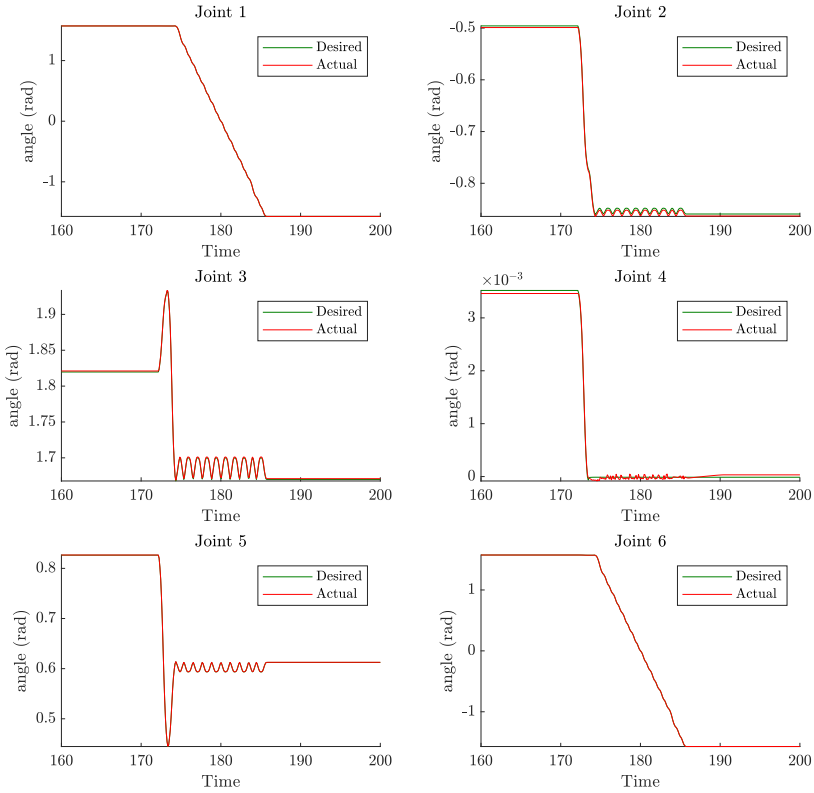


Figure 4.3: Joint positions for the circular motion (PID controller).

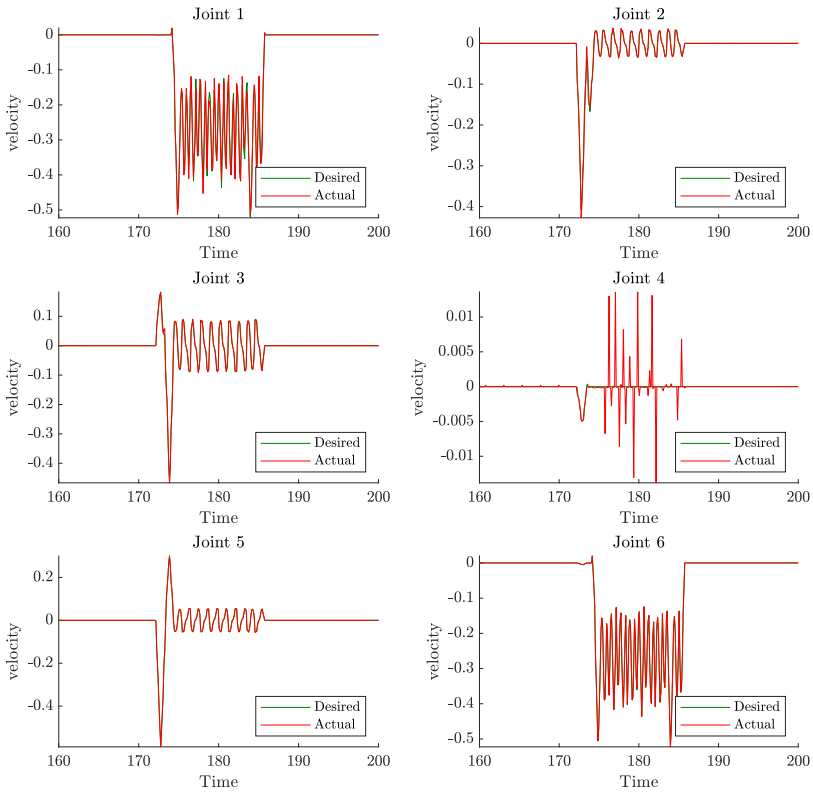


Figure 4.4: Joint velocities for the circular motion (PID controller).

Since the effort controllers used in this project was not found to be explicitly defined, it was of interest to run some tests to see how well the controllers performed with increasing velocity and changed PID controller gains. Using high values on the controller gains either made the simulated robot wobble or resulted in error messages that the initial states deviated from some expected value. Besides, if the manipulator were able to fulfil its intended motion, often error messages regarding the final goal position tolerance were published. Testing with small values on the proportional gain resulted in a collapsing manipulator that could not fulfil any motions. Using the arbitrary set gains as described previously as a basis and experimenting with increasing proportional gain, various derivative and integral effects showed that typical PID controls theory also applies for the effort controllers used in this thesis [129]. The simple P-controller worked surprisingly well, following the intended path almost perfectly, similar to the PID gained situation. Either way, the simple setup in this simulation reflects an ideal environment where only gravity and robot weight is putting demands on the controls; hence reflecting the excellent result of the simple P-controller. If more complex simulations (dynamic modelling of external environment and

factors) is to be considered in the future, more advanced control concepts must be considered for sure. The same can be said for most controllers involving both proportional, derivative and integral effect without excessive high gains, hence resulting in a stable manipulator. In "stable", it is meant that the manipulator would not shake in the simulation; hence it would be uncontrollable using the MoveIt node due to ever-changing initial state. The shaking results in pretty small deviations in the joint values, but these values are large enough for the MoveIt to decline requests due to deviations between actual and expected starting position of the joints.

When running a PD controller with a proportional gain of 10k for joint 1 and 5k for the rest of the joints with a derivative gain of 5 for all joints, a small standard deviation was spotted for joint 4 and 6 with regards to position while large deviations with regards to joint velocities were found for all joints as can be seen in Figure 4.5 and 4.6. When it comes to the simulation, the manipulator was able to complete the path as planned with no problem. The motion was experienced as a bit hackier than for other tests. A standard deviation is expected with the lack of integral effect, and as can be seen in the joint velocities, most of the deviations are standard deviations, hence an offset with the same characteristics. The extreme deviations for joint 4 and 6 must be propagated errors that are being multiplied and never corrected. Either way, the initial experimentally chosen gains for the PID controllers using gains of 2k, 0.2 and 2 for all joints except for the first joint with its dedicated 5k, 0.5 and 5 corresponding to the P, I and D gains worked out nicely for the arbitrarily defined manipulator systems. The proportionally increased gains for the first joint was set for the robot to be able to overcome gravity at extreme positions such as a wall hanging configuration with the manipulator at full stretch downwards. This configuration imposes a substantial challenge for the manipulator to overcome gravity and change to other configurations.

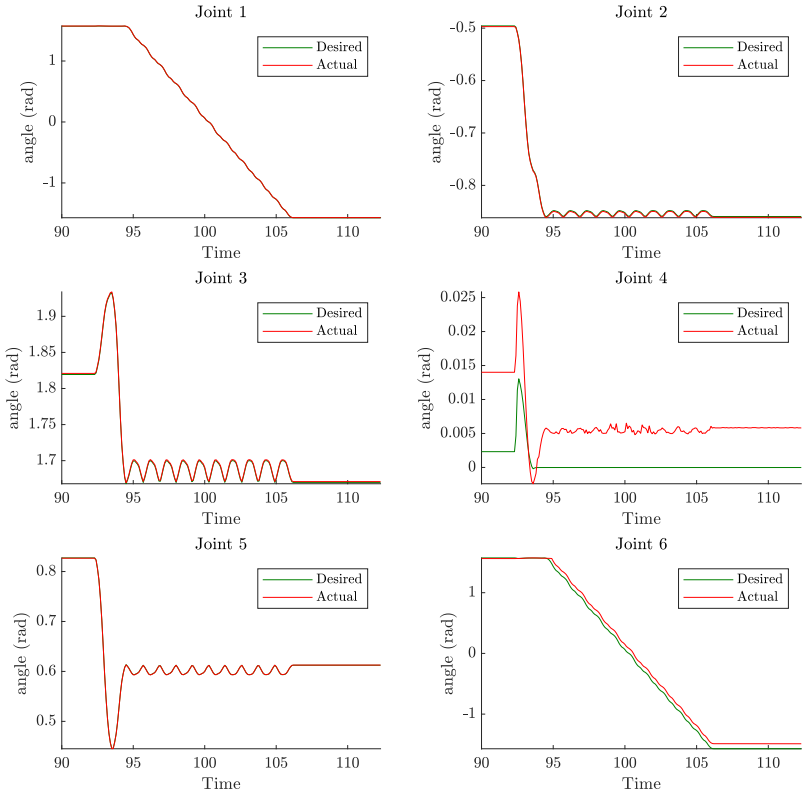


Figure 4.5: Joint positions for the circular motion (PD controller).

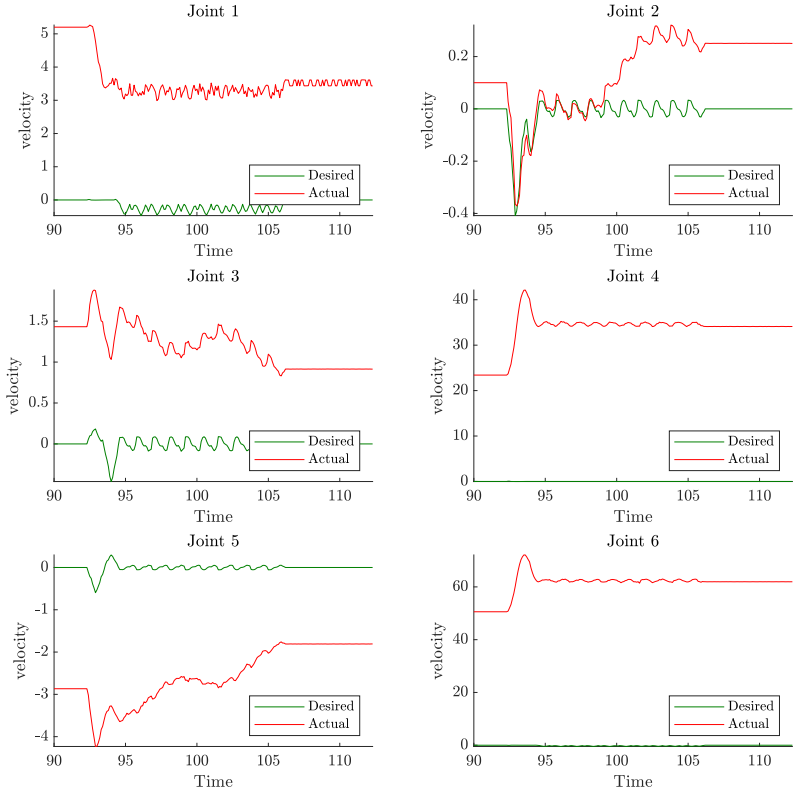


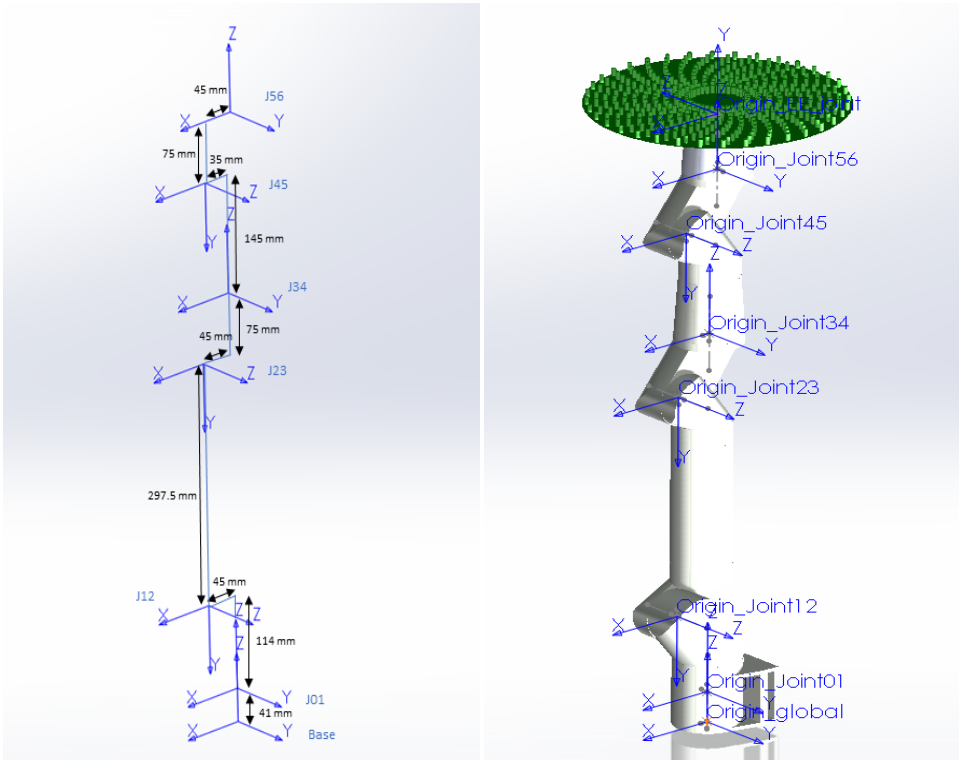
Figure 4.6: Joint velocities for the circular motion (PD controller).

4.4 Manipulator workspace

Doing path planning by intuition is laborious and inefficient. It was therefore decided to make use of a simple workspace [87] in the Matlab path planning work defining the outer reach of the manipulator. This would make it easier to create the desired paths and avoid a number of infeasible poses. After examining the ROS and URDF world, no direct way of extracting the workspace envelope of a custom manipulator was found, only redefining the URDF and using a bridge to OpenRave would seem to solve it. A plugin for workspace generating was found to be wanted in the ROS community and will surely be available in the future. Nevertheless, due to the lack of this tool, it was decided to use the approach as described by Yazdanpanah in [130] using the uniform Monte Carlo method and screw theory to create a simple workspace representation.

The generic manipulator consists of 7 links, namely the base link, link 1 to 6 and the end effector, and 6 joints in between the links. A workspace is usually describing the maximum reach of the end effector in all directions, often defining a sphere. For motion control it is reasonable to plan and execute motions for the same end-effector point within the boundaries of the workspace, operating within its configuration space. The usual approach in the industry is to create a workspace envelope where also limits regarding minimal "inside" reach and torque limits for doing operations is taken into account. This is a complicated task, and with the experimental approach of the manipulators in this thesis, it was decided to use a more simple, but sufficient approach with defining a sphere describing the maximum reach of the manipulators. The typical convention when defining a workspace is to use the end effector tool (such as the cleaning tool) as the maximum point of reach. Even though this is the normal approach both in academia and in the industry, it was decided to use a slightly other approach in this thesis. Due to the configuration of the MoveGroup as mentioned earlier, it was chosen to plan and control the kinematic chain from the base frame to the 6^{th} link to have the opportunity of easy and fast switching between various "dummy" end effectors. Therefore, the workspace, motion planning and controls have been applied from the base frame up to link 6.

The joints, links and dimensions of the home configurations of the generic manipulator can be seen in Figure 4.7. This configuration is the basis for the workspace calculations done in Matlab as can be seen in listing B.3. The Denavit-Hartenberg convention is commonly used for describing the state of a robotic manipulator using a minimum of parameters for each joint. However, it requires strict definitions of reference frames. The use of screw theory or the Product of Exponentials (PoE) approach only uses two reference frames (base and end) and a geometric interpretation from the use of screw axes (i.e. the z-axis through each joint) as mentioned in section 3.2.



(a) Dimensions. **(b)** visualisation.

Figure 4.7: Generic manipulator definitions.

When working with the PoE convention, the first thing is to define the home configuration of the end effector. Here the coordinate system of the 6^{th} link or the connected joint (J56) have been used. The home configuration is defined as a transformation matrix from the base (or the first joint) frame consisting of the rotations and translations necessary to transform the starting frame to the end frame. In the Matlab calculations, the first joint was set as the starting frame, and a constant height from the base to this starting frame was added. This yields the exact same results as using the base frame as the starting frame. The next part of the PoE approach is to define the screw axes of all joints with one degree of freedom. A screw axis is defined as a column vector of 6 elements as described in section 3.2. By using the right-hand rule after defining the direction of the screw in the top three elements of the screw axis, a 2D "turntable" can be drawn and used to find the linear velocity at the origin of the start frame geometrically. This linear velocity can be described in multiple directions but is normally described along one or two directions perpendicular to the screw direction. The screw direction is a normalised directional vector based on the angle of rotation of the joint relative to the start (spatial) frame. In this thesis, the SolidWorks software was used to obtain values by using the measure tool in between the joints. Using the defined z-directions for each joint made it simple to create the necessary matrices for computations.

If the manipulator is correctly described in the PoE convention on the form as seen in equation 4.1 and 4.2, the forward kinematics can be efficiently solved using the Modern Robotics Matlab toolbox [87]. This toolbox contains a vast amount of functions related to typical robotics application using the PoE convention. Only the spatial forward kinematics function has been utilised in this thesis to find end effector configurations. This has been done multiple times using the uniform Monte Carlo method, ultimately creating an experimental workspace consisting of fuzzy points. The uniform Monte Carlo method deals with creating a specified number of random values in between defined bounds. The uniformity defines that all outcomes are of the same probability.

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & -0.01 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0.706 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.1}$$

$$\mathbf{S} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & -0.114 & -0.411 & 0 & -0.631 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.010 \\ 0 & 0.045 & 0.045 & 0 & 0.035 & 0 \end{bmatrix} \tag{4.2}$$

The full code for the workspace visualisation can be found in B.3. In short, the joint limits are first set to represent the possible joint angles. Then a vector is created for each joint of specified size, *N*, containing uniformly distributed values in the range of motion for each joint. Using the forward kinematics solver from the toolbox in a loop for all created instances will result in a transformation matrix at each iteration. The computational and visualisation time varies greatly with the number of iterations and the chosen way of plotting. Plotting only the positional arguments will yield a point-based visualisation of the workspace as seen in Figure 4.8 which does not take orientation into account. Plotting each transformation matrix as described previously (using *plotTransforms* instead of *plot3)* can be beneficial for a small number of iterations or for getting a better grasp of the orientational possibilities as seen in Figure 4.9. With an increasing number of computations, both of these representations get out of hand, and "view" commands in Matlab has to be used for re-orienting the plots. Often it is only a small part of the workspace that is of interest. Therefore, it was created a filter such that only the transformation matrices within specified bounds were plotted as seen in Figure 4.10. The full code can be found in Appendix D.

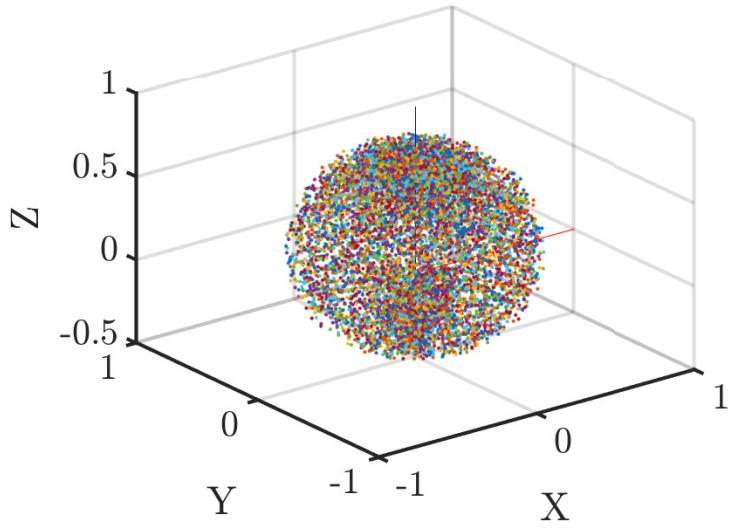


Figure 4.8: Workspace generic manipulator points.

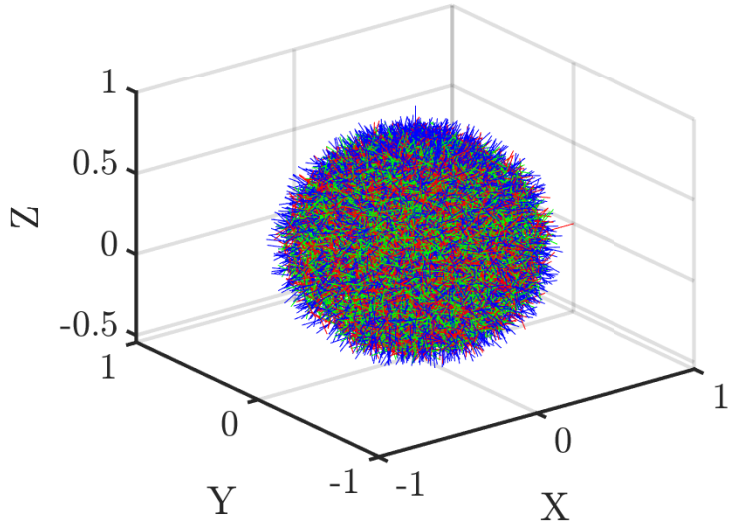


Figure 4.9: Workspace generic manipulator poses (red, green and blue correspond to the *x*-, *y*- and *z*-axis respectively, defining a pose).

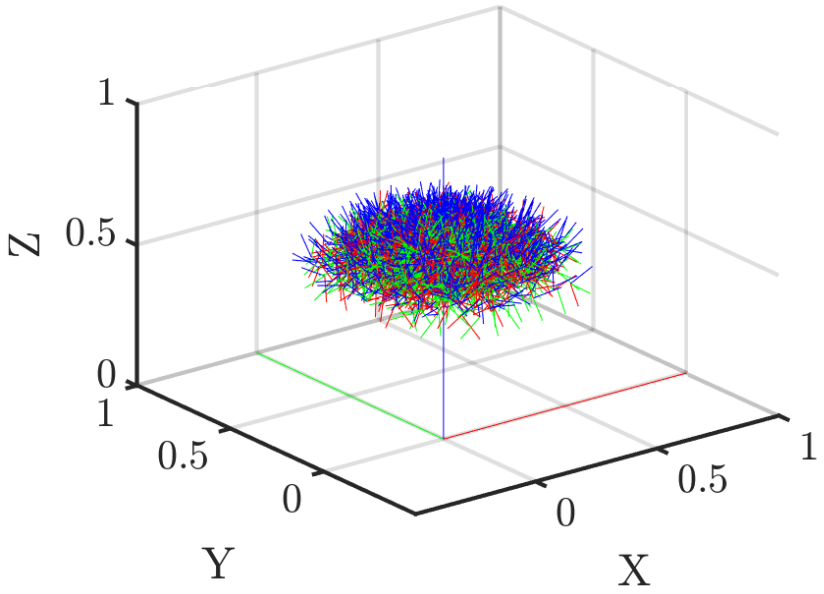


Figure 4.10: Workspace generic manipulator poses within bounds (red, green and blue correspond to the *x*-, *y*- and *z*-axis respectively, defining a pose).

4.5 Summary

After investigating the theoretical approaches adopted in this thesis, a generic model was developed in a virtual environment, including simple control concepts for path following of an underwater robotic arm. Further, an approach was presented for creating a workspace that could be used for constrained path planning. The simulation of the generic manipulator worked as intended, and a controller was tuned for a fast response. The validation of the approach created for simple simulations facilitates for other robotic simulations with increased complexity. A natural next step was therefore to use the described approaches for the chosen manipulator from section 2.2.1 and simulate the identified operations.

Chapter 5

Motion planning and demonstration of case studies with the Reach Bravo

This chapter includes multi-goal path planning, simulations and tests for selected smolt operations by using the created setup presented in the previous chapter 4. The desire of using the same MoveGroup node for all of the different end effector configurations worked out perfectly after some minor changes. It is worth noting however that if the end effectors are to be controlled in the simulations, i.e. turning cleaning tool or a functioning gripper, these have to be included in the MoveGroup that is created. It is therefore advised to create multiple URDF's with their respective end effector configurations. Nevertheless, in this thesis, the same MoveGroup has been used for all of the end effectors by changing the end of the URDF description in between runs (i.e. tool shifts). In addition, the connection between the unoriginal end effectors and the last link (link6) has been added as an exception to the collision checker in the SDF file of the MoveGroup. This is due to some problems with minor collisions when adding other STL files than the original used as the end effector. The fixed end effector joint and the collision properties of the unoriginal end effectors can, of course, be adequately placed, but this solution presents an easy and fast approach for implementing various simple end effectors.

5.1 Configuration of specific manipulator

Following the approach defined in section 2.3.4, 2.3.5 and chapter 4 for the generic manipulator, it is a straight forward task to create a fully functional simulation for any manipulator. By using this approach, the simulation configuration was created for the Reach Bravo manipulator from BluePrintLab as can be seen with its defined joints in Figure 5.1. This manipulator will be used for the rest of the simulations in this thesis. Some minor challenges might emerge if the approach and code part are used directly without any intu-

ition. Typical problems are related to the URDF definition, wrong configuration of MoveIt or not launched Nodes that is necessary for the simulations.

Nevertheless, using the approach for a very similar manipulator resulted in rapid development and more focus could be aimed at code development, making the "programmatic control code" for the generic manipulator better, and implement different end-effectors to simulate the identified operations. In this regard, the end-effector connection was set to be fixed without any mechanisms for simplicity. By creating a MoveIt Group to control the arm from the base link and up until the last link made it possible to use the same planning group for various fixed end-effector tools as described in the last section.

Figure 5.1: Joint definitions of BluePrintLab's Reach Bravo manipulator.

It was aimed at simulating the operations in section 1.2. Using different developed tools (for visualisation purposes) as seen in Figure 5.2 (cleaning), 5.3 (feeding) and 5.4 (grasping), the different operations could be simulated in the created tank environment. There are obvious shortcomings with the presented end-effector tools such as the small reach of the gripping tool, relatively small cross-section of the cleaning brush and the infeasibility of the *Il tempo gigante* inspired feeding gun. The thought behind these tools was to make a simple circulating brushing tool that could be connected to top side pumps for transport of sludge. The visualisation of this connection has been neglected in this thesis. Further, the feeding cannon is thought to be connected to existing feeding dispensers at the facilities. By using suction force and a funnel, the cannon can retrieve feed from the dispenser and spray this feed in a pre-configured path reaching the entire tank cross-section, or it could be connected to a decision-based system for pointing out where feed is needed.

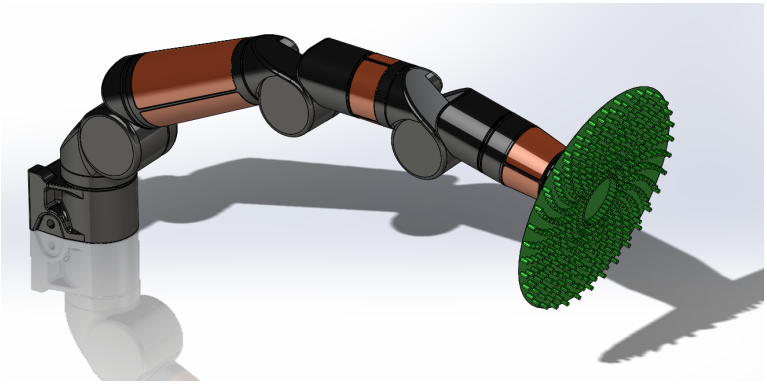


Figure 5.2: Bravo with cleaning tool from [108].

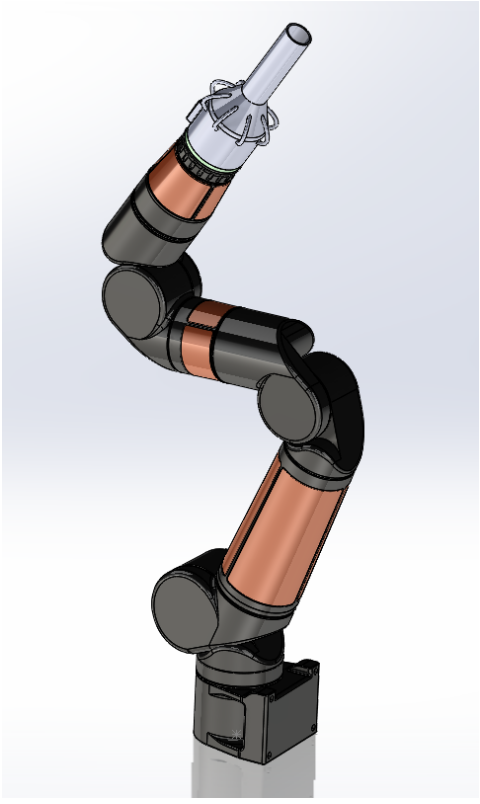


Figure 5.3: Bravo with feeding gun.

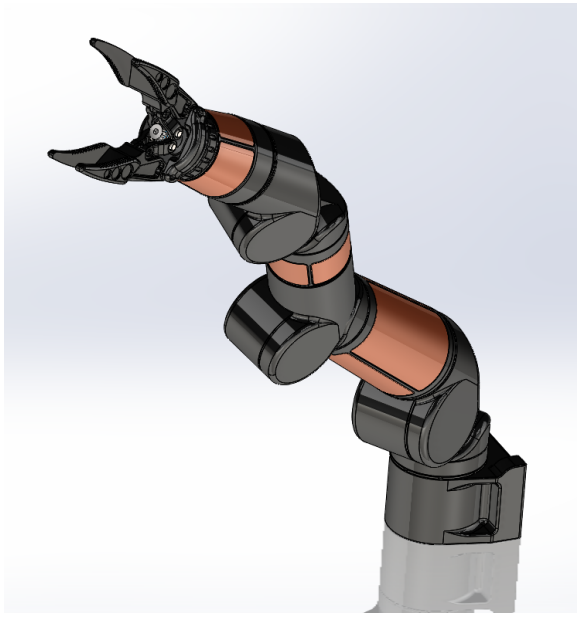
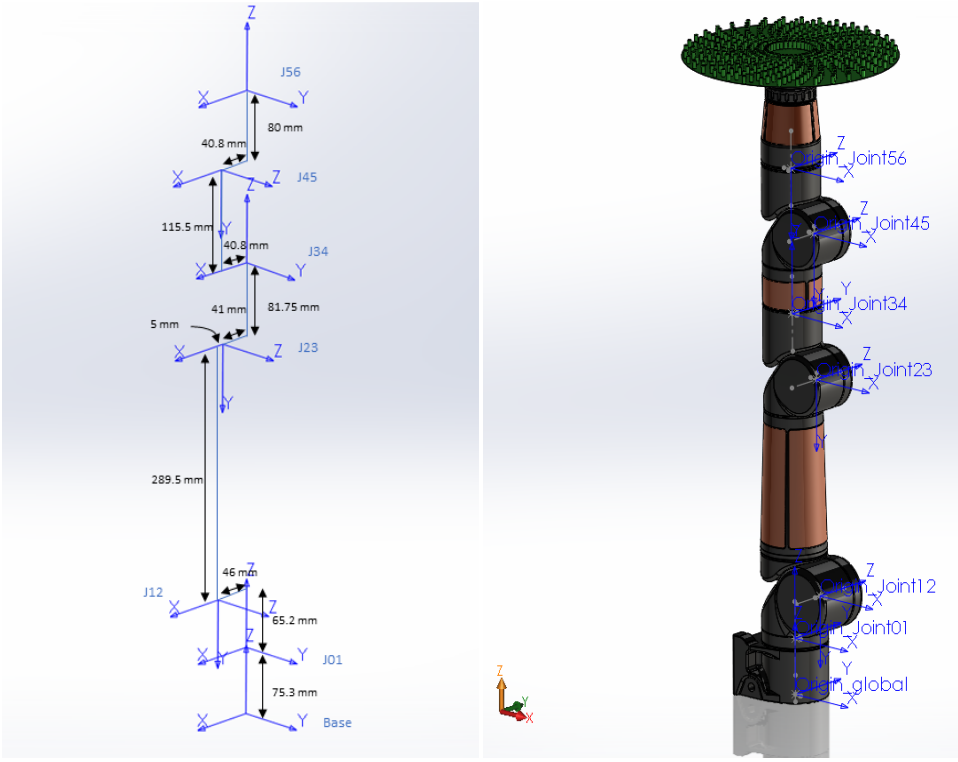


Figure 5.4: Bravo with cleaning tool.

5.2 Path generation in Matlab

Building on the experience from the simple path planning in the operating space for the generic manipulator it was developed motion plans for the Bravo robot for doing an extended half circle (cleaning operation), a feeding operation using a spiral motion and a random grasping motion.

As for all motion planning, it is an advantage to know where exactly it is possible to execute a motion. Using the same approach as in section 4.4 with the dimensions for the Bravo manipulator defined in Figure 5.5a, equation 5.1 and 5.2, the workspace was computed as can be seen in Figure 5.6.



(a) Dimensions. **(b)** visualisation.

Figure 5.5: Specific manipulator (Bravo) definitions.

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -0.004 \\ 0 & 0 & 1 & 0.632 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.1}$$

$$\mathbf{S} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & -0.065 & -0.355 & -0.003 & -0.552 & -0.004 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.046 & 0.041 & 0 & 0.041 & 0 \end{bmatrix} \tag{5.2}$$

The fuzzy workspace representation of points cannot only be used as a pointer to where a path should be created but can also be used actively in the planning; hence moving from simple open path planning into constraint-based path planning. There are numerous ways of doing this. In this thesis, the emphasis has been laid into creating a simple sphere

representation of the workspace. That way, desired poses can be efficiently checked if they are inside the workspace and if not, be transformed within the feasible territory.

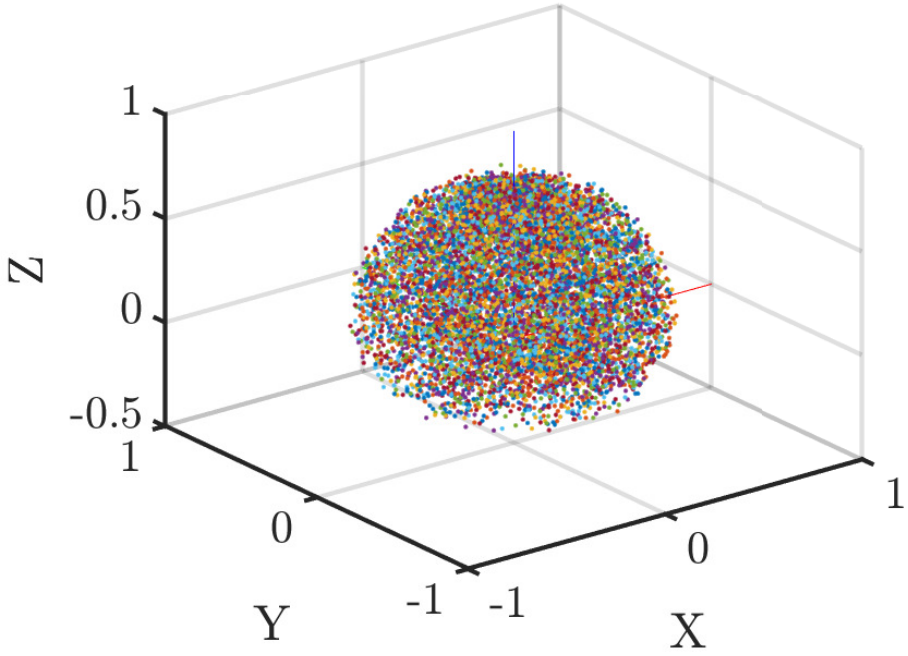


Figure 5.6: Workspace for the Reach Bravo manipulator consisting of points.

If each step of the path generation would have to compare its positional properties with the vast amount of points describing the workspace as seen in Figure 5.6, it would take an immense amount of time. Curve fitting for the points to create a simple sphere definition was therefore the chosen approach. There are many available curve or surface fitting functions in Matlab, but the sphere fitting function created by Alan Jennings [131] seemed like the best option based on great feedback on Mathworks. The function takes a list of fuzzy points described in cartesian coordinates x, y and z as input and returns the centre point and the radius of a sphere that is found by using the least-squares method for all of the points in the input list. Using the built-in function in Matlab for creating a sphere and scaling this unit sphere by the returned radius from *SphereFit* yields a representation of the workspace sphere with a slight offset. This is where the returned centre coordinates come into play. The sphere is still a heavy computational burden. Therefore it is possible to create an alpha shape from this sphere, including the offset such that it is placed correctly. An alpha shape is a polygon or a polyhedron created from points in either 2D or 3D. For this purpose, the sphere was redefined to cartesian coordinate points and fed into the *alphaShape* function with the centre-offset creating a simple polyhedron which has helpful functions that can check if a point is inside the shape (*inShape*) or find the nearest point on the edge of the shape (*nearestNeighbor*). The resulting alpha shape for the workspace

boundaries can be seen in Figure 5.7 with a safety factor of 10% applied. It is easily observed that the boundary shape will be too little with respect to the actual possible reach of the defined manipulator. This can easily be changed by applying a different scaling factor on the radius such that the boundary gets closer to the actual max reach. It might be worth noting that an alpha shape can be created directly using the fuzzy points from each transformation. This will, however, yield a rock-like shape with more "extreme" points allowed which can result in too much slack on what points go through this filter. Since this workspace only takes positional arguments into account, some infeasible poses regarding orientation will inevitably pass through.

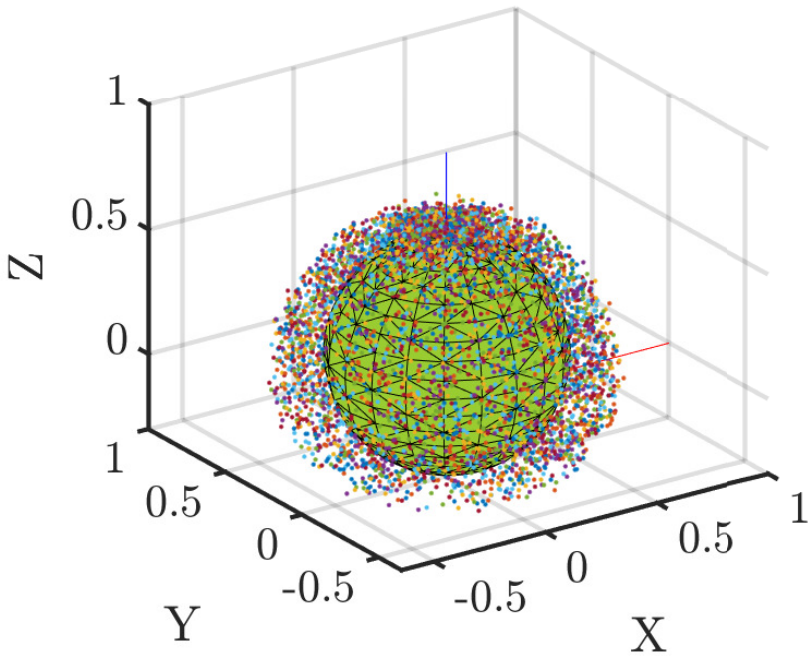


Figure 5.7: Workspace Reach Bravo manipulator with alpha shape.

The functionality of the alpha shape for checking if a point is inside the workspace is of great use as a boolean variable. However, even more, critical is the algorithms or exception handling for when the boolean is false, and the point is outside the defined workspace. Since most of the paths to be defined in this thesis are working in the upper part of the manipulators reach, this was emphasised when creating the exception algorithms. More specifically, it was expected that initially defined paths would be within the maximum x and y values and possibly above allowed z values in the cartesian space. With this basis, two exception algorithms were defined. The most obvious candidate was the direct application of the alpha function *nearestNeighbor* which returns the index related to the point that defines the closest vertex in the alpha shape to the actual point as can be seen in

yellow in Figure 5.8. This is done by computing the absolute distance between the given point and each of the vertices in the alpha shape. A disadvantage with using this directly is that an initial path often is defined in 2D, i.e. for a circular motion that is defined above the actual reach of the manipulator. It is often desirable to keep the x and y values of these points for a full path implementation. Therefore, a second approach was defined which involves using the equation of a sphere to compute the z value of given x and y values as seen in 5.3 and visualised in red in Figure 5.8. This approach is independent of created points which will yield a more precise projection of a point onto the workspace. These two exception handles have been used further in the path planning where applicable, often in conjunction.

$$\begin{aligned} (x-a)^2 + (y-b)^2 + (z-c)^2 &= r^2 \\ z &= \sqrt{r^2 - (x-a)^2 - (y-b)^2 + c} \end{aligned} \tag{5.3}$$

Where; x, y and z is the position of the point and a, b and c describes the center of the sphere in cartesian space.

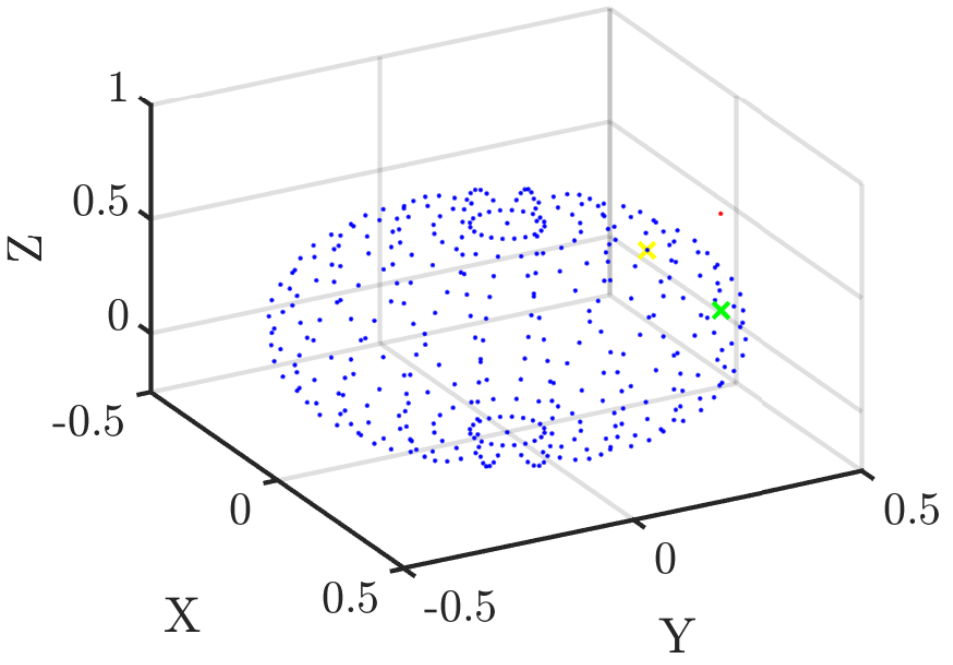


Figure 5.8: Exception handling.

5.2.1 Cleaning operation

The intended cleaning operation is a simple operation with regards to planning since the working area is said to be along a simple 2D surface within the workspace of the robot. This is a natural assumption for cleaning operations done in large tanks where the curvature is minimal, hence can be neglected. For smaller tanks with steeper curvature, this surely has to be taken into account when path planning is done. Either way, the case for this simple cleaning of large tanks is done within the "good" configuration space of the manipulator where torque- and joint limits will not be tested. Therefore it was decided to use a similar approach to the generic test in section 4.3, only increasing the complexity a bit to simulate a broader cross-section of cleaning in front of the robot. This was done by adding the reverse θ angles to the vector of angles, hence have a θ vector moving from -π to π and back to -π, and using different radiuses in the loop where the transformation matrices were created. This resulted in the path seen in Figure 5.9 where it can be seen that no emphasis has been laid on the angle about the tool's z-axis. This is because the last joint (Joint56) is a continuous joint that can have whatever angle about its z-axis (axis of rotation). The defined parameters in this configuration are the height above the base frame (here 0.15m), outer- (0.5) and inner radius (0.3m). Also, a start pose is defined as an initial state. This is not strictly necessary but has been done for all for the paths created, mainly for visualisation purposes. The full code of this path definition can be seen in listing B.4 (Appendix).

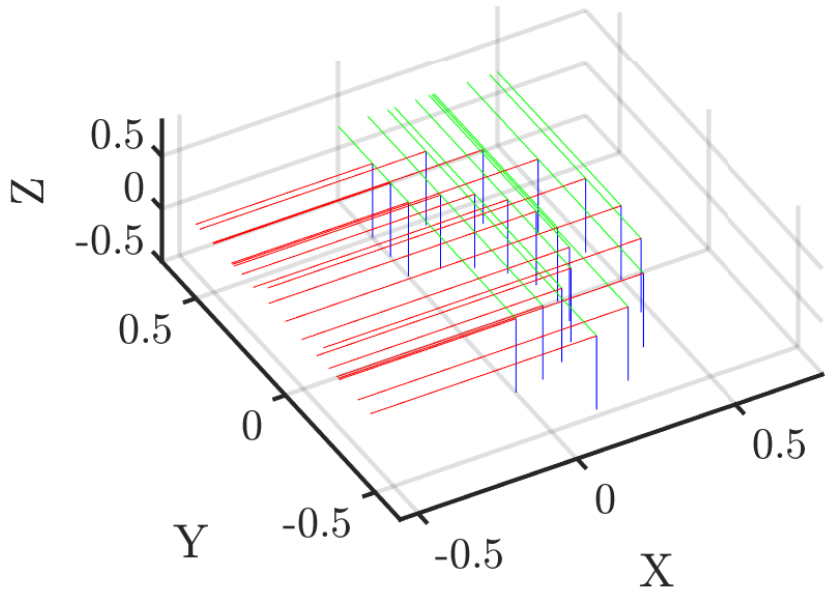


Figure 5.9: Path planning for autonomous cleaning operations (red, green and blue correspond to the x-, y- and z-axis respectively, defining a pose).

5.2.2 Feeding operation

The defined feeding operation in this thesis is a bit more complicated than the cleaning operation due to its requirements of different orientation and position for the gun to be able to reach a more significant part of the tank environment. For solving this operation, it was assumed that the robot could be stationed in the centre of the tank, either on a pilar, footbridge or something similar. There are numerous ways of solving the problem of reaching the whole surface area using a feeding cannon such as creating different circles with varying orientation, spiral motion or others. Either way, these configurations are more demanding than a cleaning procedure due to its often unnatural poses with the end effector (here; feeding cannon) aiming upwards. These poses are typically closer to singularities and weak configurations. Therefore, it was aimed at using some of the tools described in section 4.4 for projecting a spiral path onto the workspace, hopefully resulting in exclusively feasible poses. In addition, a simple 2D spiral was created well within the workspace defined in Figure 5.7. A spiral motion can be derived in various ways, and the most known spiral motion is the Archimedean spiral using the polar equation. In this thesis, however, the spiral equation was defined such that the inputs for creating the path consisted of an outer and an inner radius, the number of desired revolutions and points per revolution. This was the preferred approach since it is more intuitive and yields better control of the waypoints created since it is ultimately the waypoints to be used in the ROS software that is important, not the continuity of the plotted shapes or how nice it looks. Figure 5.10 shows a simple created spiral of points using the following inputs; outer radius = 0.5, inner radius = 0.2, height = 0.5, revolutions = 3 and points per revolution = 25.

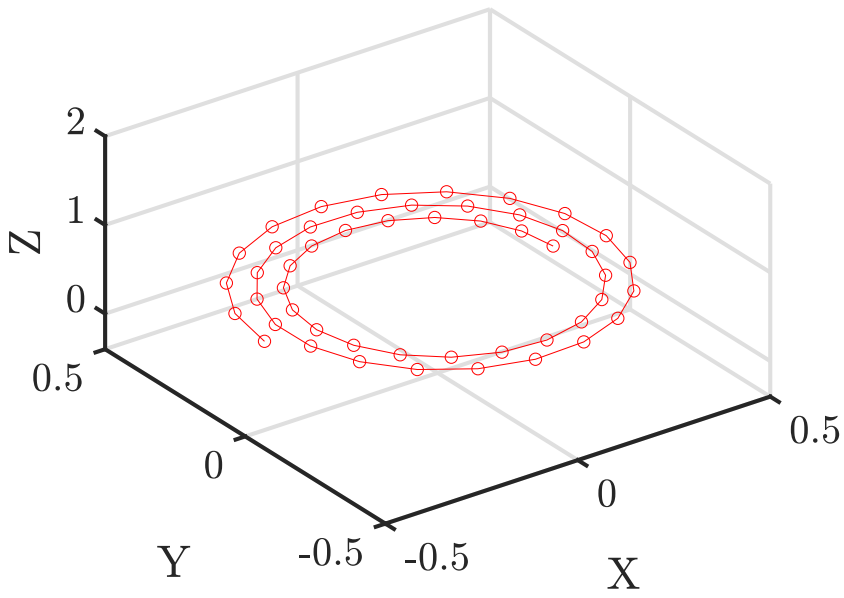


Figure 5.10: Spiral path.

Using the same approach as for the cleaning path only applying the spiral equation for the x and y positions, a constant z position set to the desired height and a varying Euler angle transformation resulted in the waypoints plotted in Figure 5.11. The Euler angle transformation uses the four tangent $arctan$ function with the cartesian position as input to get the desired end-effector orientation. The following parameters were used for the path planned in Figure 5.11: constant height of 0.4 which is well inside the workspace, an outer radius of 0.45, an inner radius of 0.3, 2.5 revolutions and 20 points per revolution. These first created waypoints might yield a very "compact" motion of the robot since it is created very tightly towards the base of the robot which most robots do not favour much due to challenging configurations because of proximity between the en-effector and the base. Therefore, using a height of 0.6 meters, the waypoints outside the defined workspace were projected down into the workspace as can be seen in Figure 5.12. By using the nearest neighbour of the alpha shape, the spiral points were moved to their closest corresponding vertex as can be seen in Figure 5.13. The full code can be found in listing B.5, B.6 and B.7 (Appendix). These approaches come from quite some different path planning principles, whereas the first is open unconstrained path planning, while the second and third option is ways of constrained path planning. Either way, none of these approaches have taken into account the desire to reach the whole cross-section of the tank. Nor have they taken into account possible orientation problems for the manipulator. Therefore, using assumptions about the feeding gun, a path should be created for reaching all of the cross-section in the tank. Also, the implementation of an inverse kinematic solver should be applied to not only constrain the position within the workspace but to constrain the orientation of the poses created as waypoints. This will yield a much better and robust path of waypoints to be implemented in the robot, and a lot of unnecessary trial and error will be removed.

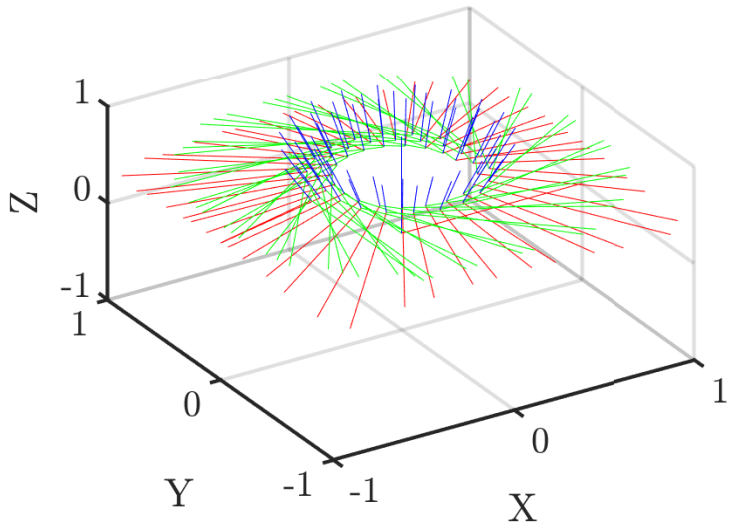


Figure 5.11: Feeding path with constant height (red, green and blue correspond to the *x*-, *y*- and *z*-axis respectively, defining a pose).

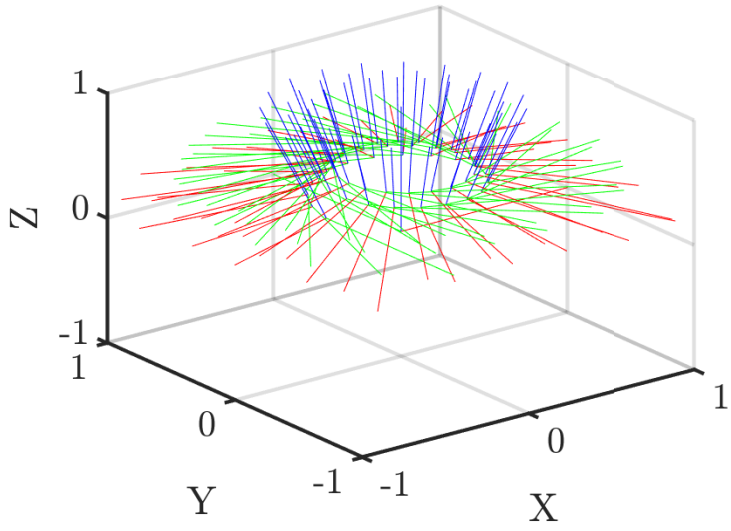


Figure 5.12: Feeding path with projected z value (red, green and blue correspond to the *x*-, *y*- and *z*-axis respectively, defining a pose).

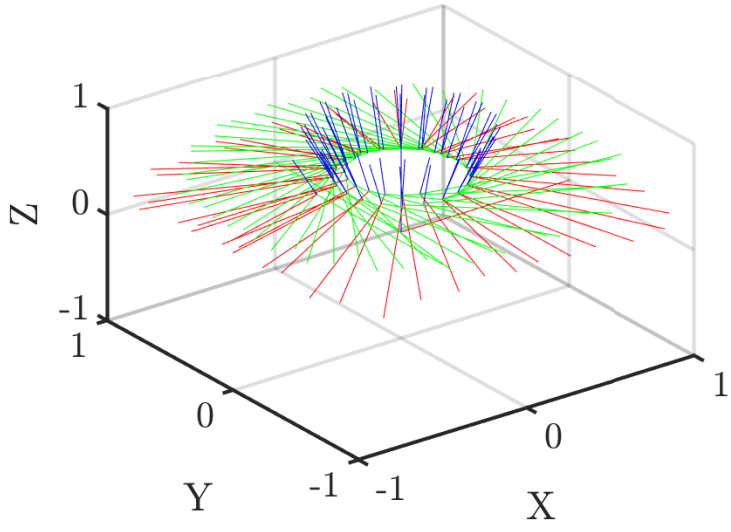


Figure 5.13: Feeding path using nearest neighbor (red, green and blue correspond to the *x*-, *y*- and *z*-axis respectively, defining a pose).

5.2.3 Gripping operation

Gripping as an operation is in terms, not the most straightforward task of completing. Ultimately, the manipulator should move to a specific approach pose, gently moving in towards the target pose, grip the object and then manipulate it or move it to another goal position before leaving the object. Inside a smolt facility, this operation will be much more demanding than an offline stationary grasping operation. By this, it is meant that a grasping operation for, i.e. removing dead fish within a tank will require an extended reaching tool with a net or grabber. This extension in reach will surely complicate the dynamics and path planning. Also, robot vision and dynamic controls will be required to adequately cope with this challenge in such a dynamic environment with moving fish and water steams. This is therefore a natural part of further work. Nevertheless, in this thesis, a simple path in 3D, replicating a grasping scenario for removing a close by dead fish has been created. The path was created well inside the workspace and consisted of a simple 3D motion from the base frame's y axis to the x axis with decreasing height z, resulting in a goal state for where to tentatively leave the dead fish. The waypoints of this pose can be seen in Figure 5.14 and the full code can be found in listing B.8 (Appendix).

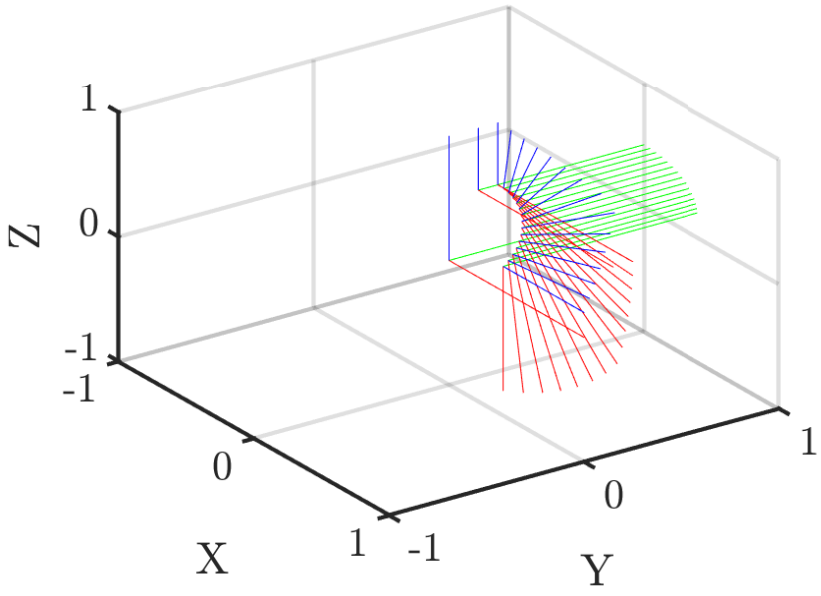


Figure 5.14: Grasping path (red, green and blue correspond to the x-, y- and z-axis respectively, defining a pose).

5.3 Waypoints concept implementation

Even though all of the created waypoints for the paths defined in the previous chapter is within the shrunk workspace of the manipulator, there is no guarantee that the orientation of the waypoints is feasible. This is usually because of joint limitations from a strict visualisation point of view, but for actual robot implementations, torque limits and other hardware limitations might also apply. In this thesis, the simulation part has been done with no hardware in the mix. Therefore, it was emphasised to test and verify the created paths in the created simulation environment using the MoveGroup node described in section 3.4. The C++ code developed for the generic manipulator had its obvious drawbacks such as not being able to check specific points of input paths defined in csv format, extracting the active pose, moving to specific joint angles from input csv file and lastly running a variety of csv files in the cartesian path planner or point by point without having to build the package in between each run. These shortcomings were solved by creating more generic functions and increasing the size of the switch for being able to run multiple tests in a single build. The main class with its methods can be found as an excerpt from the header file in listing C.4. The rest of the code can be found in the attachments in appendix D.

By using the defined methods, the switch for inserting choices to be run interfacing with ROS was extended to contain a case for implementing the path from its csv file as a cartesian path, waypoint for waypoint, and lastly to compute and execute specific waypoints of the input files. Therefore, the run commands of the executable file path as described in section 4.2 were increased to take two inputs as seen in listing 5.1. Where m is the selection of which case to run in the switch and n is specifying either the interval of waypoints to be iterated if the function for looping through all waypoints is chosen, or it represents the specific waypoint in the list of waypoints to be run.

```
rosrun motion_planning run m n
```

Listing 5.1: Running C++ script.

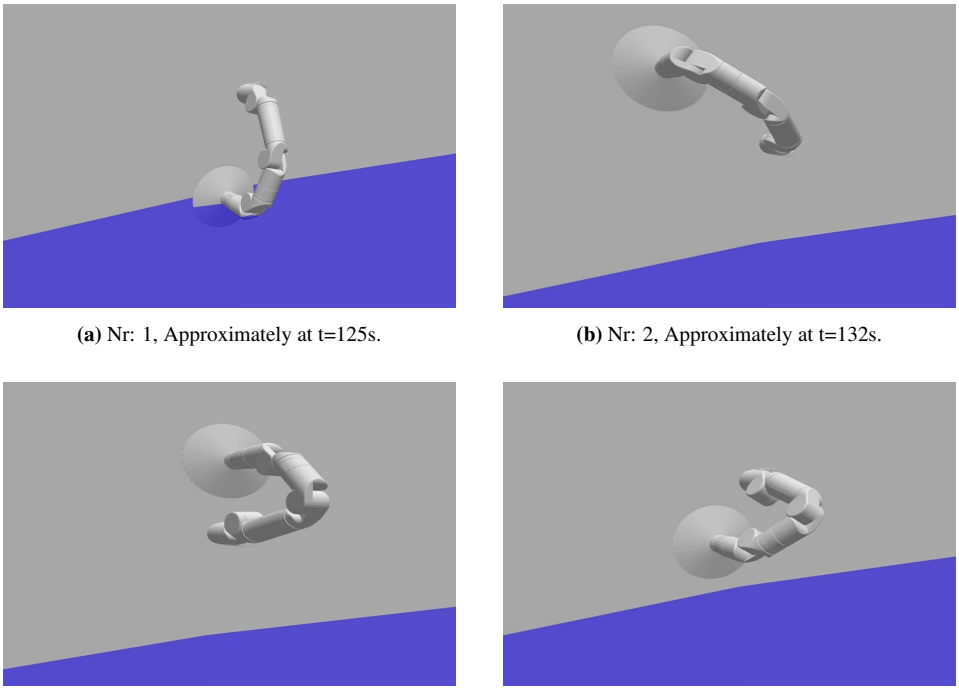
This whole part was especially necessary due to the named shortcomings of the cartesian path computations of the MoveGroup. Even though all inserted waypoints were well-defined and within a feasible distance of each other, the trajectory generator often failed completely in computing the trajectories. There are a lot of open issues about this plugin on the ROS community pages, and it is often advised to use the more stable functions for moving to known poses or single pose computations, which was used in this thesis to experimentally test the created waypoints. With this extended programmatic control in place, one could move on to simulate the aforementioned paths, interesting joint configurations or poses in an efficient manner. The proceeding presents the simulations and results of the three chosen operations.

5.4 Results and demonstrations

This section presents the simulations of the planned operations from the preceding sections including documentation of their fulfilled actions by plotting the positions and velocities of the joints for the actual and desired values, and end-effector path following to validate the simulations.

5.4.1 Simulation of cleaning operation

The main desire for all of the simulated paths in this section was to efficiently implement them using the cartesian path planning feature of the MoveGroup and execute the generated trajectories in the simulation environment. This worked as expected for the quite simple path created for the cleaning operation seen in Figure 5.15. Corresponding with the planned path from section 5.2.1, the simulated manipulator fulfils a cleaning path consisting of two half circles in the water layer area covering a cross-section of about 0.63 m^2 ($\frac{1}{5}\pi\ m^2$) using the created cleaning tool with a radius of 0.3 m. The path following can be seen in Figure 5.16, yielding a very nice tracking of the intended waypoints. The planner of the cartesian path generator had some problems if the current pose were at a significant distance from the initial waypoint of the input path, but using exception handling in the C++ code and moving to this initial waypoint in a pre-computational operation solved this.



(a) Nr: 1, Approximately at t=125s.

(b) Nr: 2, Approximately at t=132s.

(c) Nr: 3, Approximately at t=137s.

(d) Nr: 4, Approximately at t=142s.

Figure 5.15: Cleaning simulation, ongoing.

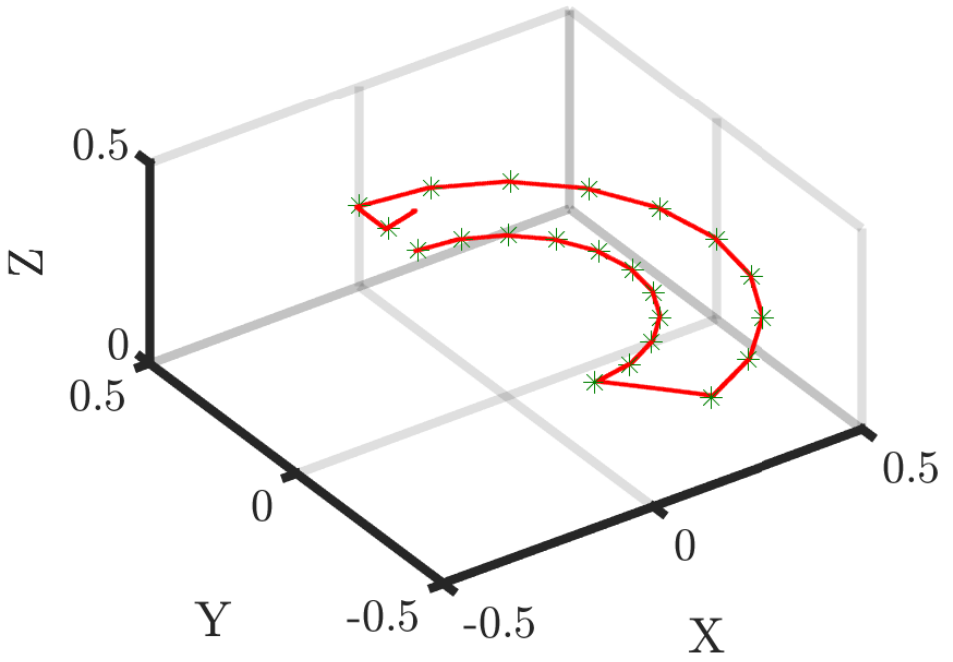


Figure 5.16: Path following of cleaning operation (green stars are waypoints and red dots are recorded end-effector positions).

From the plots in Figure 5.17 and 5.18 it is observed a very similar behaviour as for the simple test case done for the generic manipulator in section 4.3.2. This is natural since this cleaning operation in a way consists of two simple half-circular motions that are merged. Besides, the Reach Bravo manipulator has been given almost identical parameters as for the generic manipulator except for link lengths and design. Using the same PID tuning, therefore, yield very similar positional and velocity plots of the joints. The first joint dictates the main rotation of the manipulator chain that can be seen to rotate from the initial position of π to $-\pi$ and back again in the cartesian space. At around 120 seconds, a spike can be spotted for joint 2, 3 and 5. This is the motion to reach the initial waypoint and preparing for the path following. Further, these joints, mainly dictating the pose of the manipulator, are following the desired (or planned) path almost perfectly. As described for the generic test case in section 4.3.2, it is believed that the MoveIt planner is planning for an oscillating motion as spotted since the generated desired trajectory coincides with the actually implemented trajectory. This might be happening due to high velocity limits, controller gains or other parameters that are used by the MoveIt framework for solving the motion planning problems. This should be closer examined, but the effects are small and are hard to observe in simulations. A very small over-/ undershoot can be spotted for joint 4 dictating rotation of the end-effector which is neglected in these simulations since it is believed to use a continuously turning cleaning tool at the end-effector. Anyway, Joint 4

and 6 dictates the rotation of the end-effector, which can be seen from the path planned in Figure 5.9 has been kept constant in these simulations. Therefore, it is also expected to observe small very small values for the position and velocity of joint 4 and a controlled position (and velocity) for joint 6, keeping the orientation of the end-effector stationary throughout the planned motion. The commented plots validate that the described joint trajectory controller, in fact, provides a working controller for position and velocity.

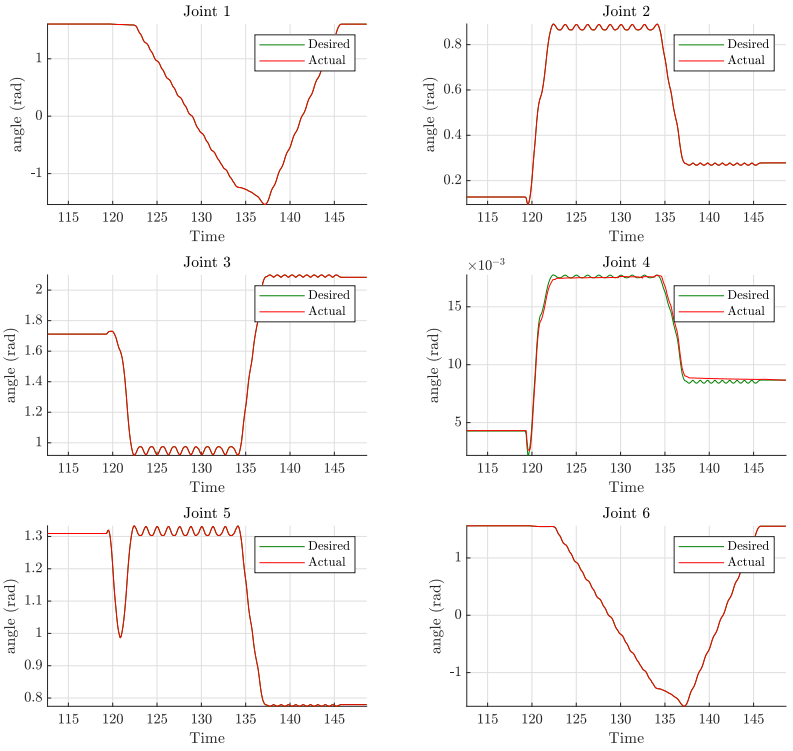


Figure 5.17: Joint positions for the cleaning simulation.

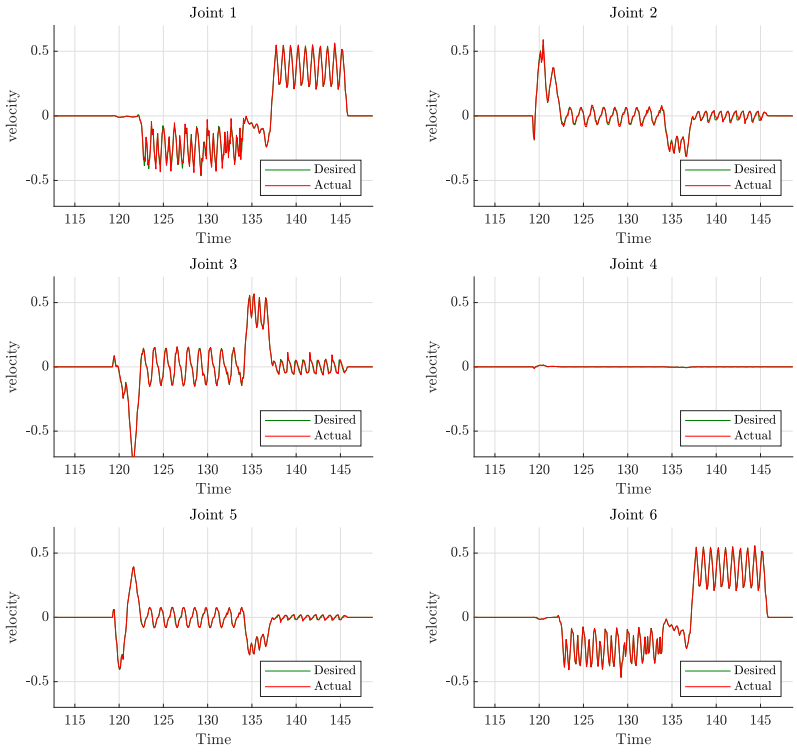
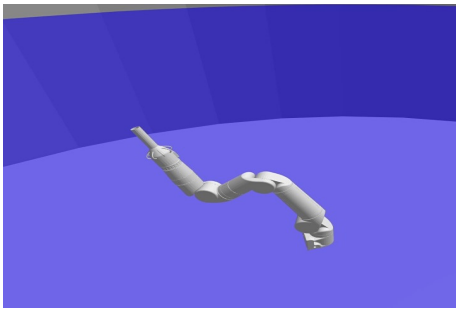


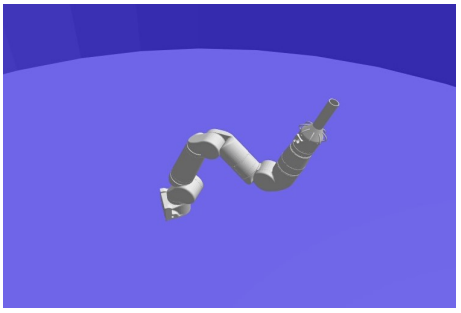
Figure 5.18: Joint velocities for the cleaning simulation.

5.4.2 Simulation of feeding operation

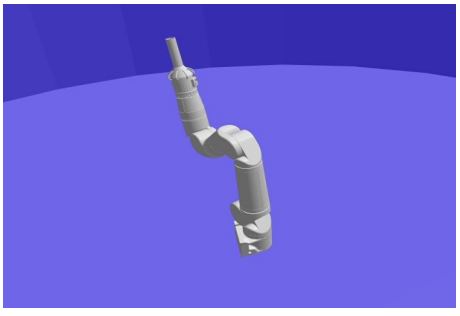
For the first defined feeding path with constant height, it was found that some of the last poses in the path, close to the z axis of the base frame was the root to some problems. Quite a few tests were applied for different inner radiuses and heights. As expected, the closer the points were to the centre axis, the higher requirement for correct z-position was observed. Nevertheless, keeping the inner radius at a soberly value, all of the waypoints were feasible when running them one by one. Even though all waypoints were feasible and within close distances, the *cartesian path planner* was not able to compute the path through numerous attempts. Therefore, it was decided that it was adequate to use the created function for looping through pose goals for validating the path following. Nevertheless, it would be nice to implement or make use of other controllers for smoother path following and this is suggested as topics for future work. A visualisation at four time steps in the path following simulation can be seen in Figure 5.19 whereas it is started at the outer point of the spiral and works itself inwards with time while keeping the angle of attack of the feeding gun relative to the z axis of the base constant. Figure 5.20 presents the input waypoints and the recorded behaviour of the manipulator. The motion (in red dots) was performed piecewise, solving many small motion planning problems in between the waypoints. Therefore, the simulation is observed as hacky even though it cannot be spotted from Figure 5.16 since breaks due to the computation of the different segments are plotted on top of each other.



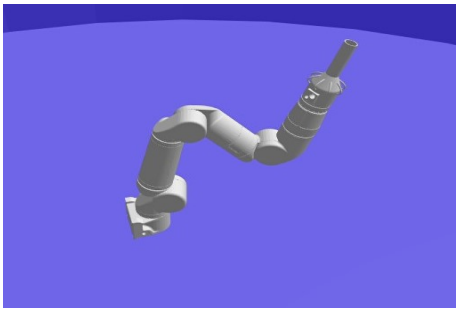
(a) Nr: 1, Approximately at t=65s.



(b) Nr: 2, Approximately at t=72s.



(c) Nr: 1, Approximately at t=86s.



(d) Nr: 1, Approximately at t=95s.

Figure 5.19: Feeding simulation, ongoing.

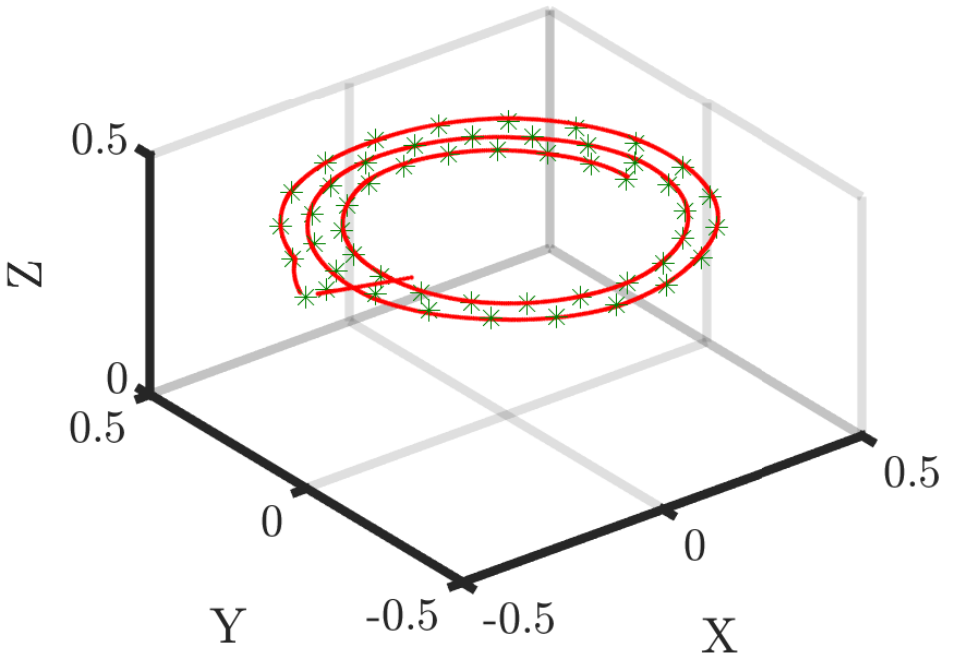


Figure 5.20: Path following of feeding operation (green stars are waypoints and red dots are recorded end-effector positions).

Due to the use of the created method for moving to defined pose goals, the motion of the manipulator is hacky as expected. This is observed on the behaviour of joint 1 in Figure 5.21 and 5.22, whereas a stairway characteristic can be seen in the plotted position and a bang-bang like characteristic can be seen for the velocity of the first joint. The spiral motion is validated by looking at the decreasing joint angle for the first joint, which dictates the rotational motion. The plotted lines of desired versus actual in this context are expected to be identical since both values are returned from the trajectory controller.; hence the segments computed at each waypoint results in a narrow window for errors to occur when the segments are to be followed afterwards. As seen in Figure 5.21, joint 2, 3 and 5 are mainly dictating the pose of the arm, consequently changing at each new calculation. This happens due to the allowed deviation in position and orientation of the end-effector, together with the structure of the used planners in MoveIt. The planners are always finding an approximate solution within its constraints which can result in different configurations for the same problem. For this path, where the orientation is changed with the rotation of the manipulator in the world frame as seen in Figure 5.11, joint 4 and 6 dictating the orientation of the end-effector are initiated to its starting position but are from there on kept approximately constant. The velocity profiles of the other joints are also here following the desired (planned) perfectly, which is natural due to the segmented computations and implementations that are completed. The same behaviour, problems and

solutions applied to the two other defined feeding paths only resulting in more significant changes in the position and velocity values of joint 2, 3 and 5 due to larger changes in z-values.

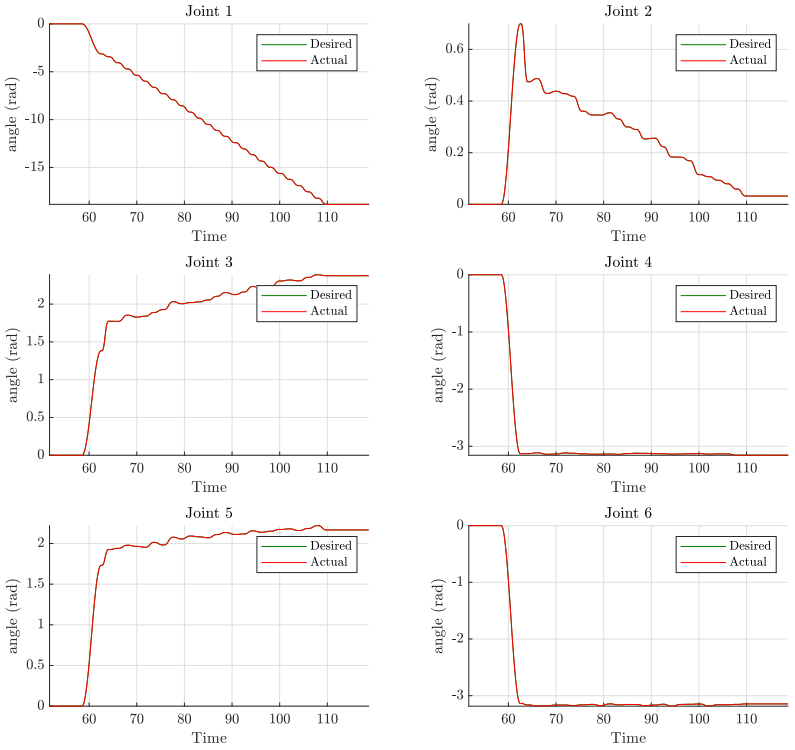


Figure 5.21: Joint positions for the feeding simulation.

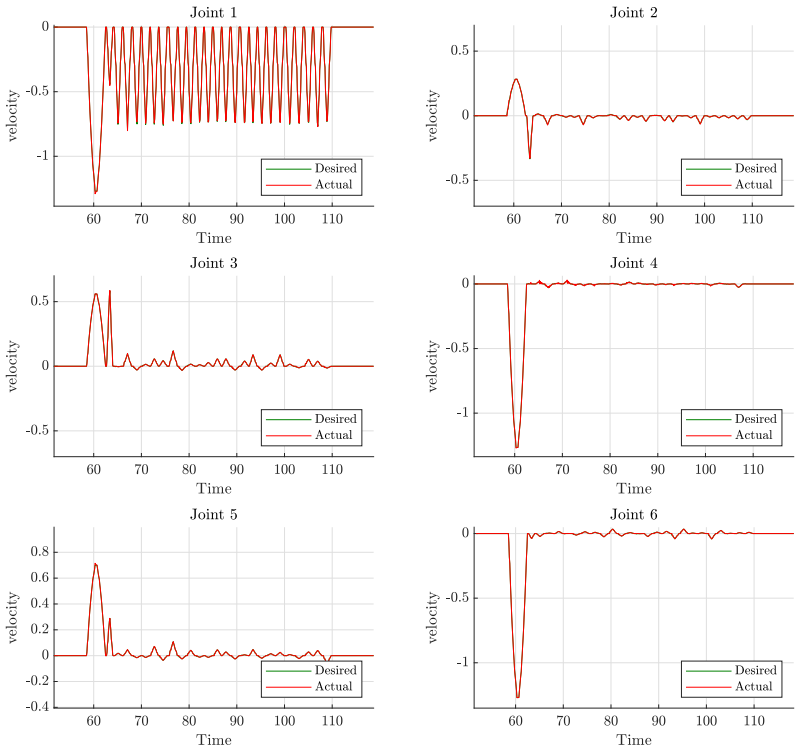
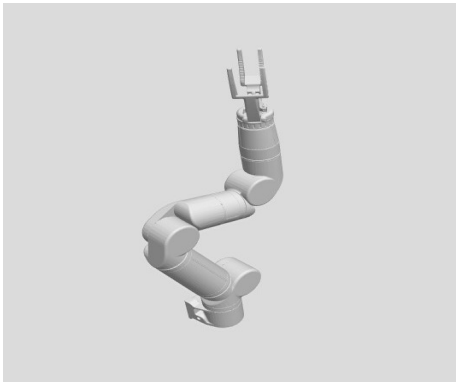


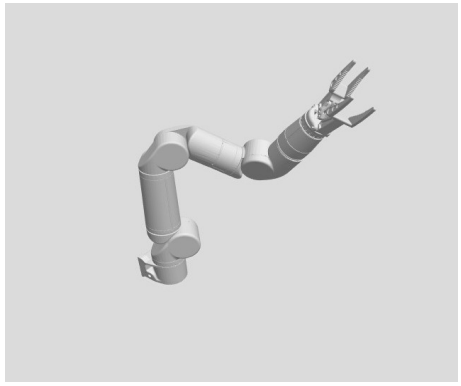
Figure 5.22: Joint velocities for the feeding simulation.

5.4.3 Simulation of gripping operation

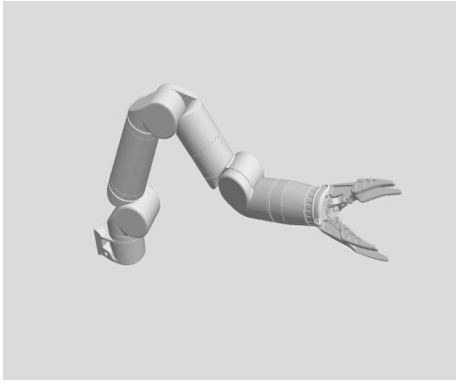
The gripping path was as mentioned in section 5.2.3 created in a few simple steps defined by positional references and varying Euler angles between 0 and π for the end effector orientation relative to the base frame. Testing all of the waypoints one by one went well, but as for the feeding paths, the *cartesian path planner* was not able to compute this "simple" path. Like for the feeding operation, the simulations were carried out using the pose goal functionalities resulting in the simulated grabbing motion placed at the bottom of the tank, as seen in Figure 5.23. Figure 5.24 presents the recorded motion of the manipulator (in red dots) and the planned waypoints (in green stars). This validates the intended path following. As for the simulated feeding operation, this motion was also observed to be hacky. In Figure 5.24, it is observed that the actual motion is not tracking the intended waypoints too precisely. As described for the feeding operations, the MoveIt planner with its chosen parameters is probably the reason. Nevertheless, it is found that the manipulator follows the planned path sufficiently for the intention as a proof of concept.



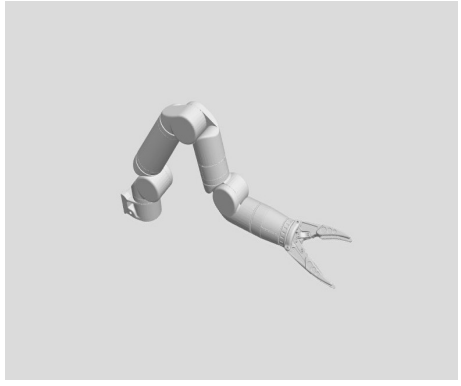
(a) Nr: 1, Approximately at t=59s.



(b) Nr: 1, Approximately at t=63s.



(c) Nr: 1, Approximately at t=70s.



(d) Nr: 1, Approximately at t=75s.

Figure 5.23: Gripping simulation, ongoing.

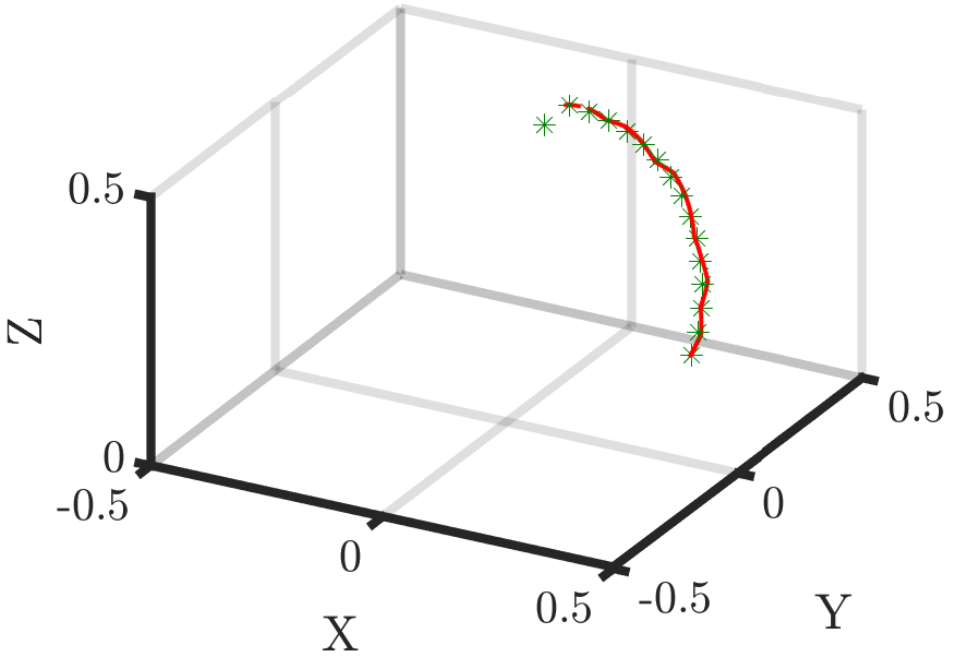


Figure 5.24: Path following of gripping operation (green stars are waypoints and red dots are recorded end-effector positions).

Similar to the feeding operations it can be seen from fig 5.25 that the manipulator is moving from a completely unfolded state and into the initial start state of the gripping operation at 55 seconds into the simulation. The manipulator is then positioned at a 90 ^{o} angle relative to the base frame where it was intended for the manipulator to grab something. Further, the first joint controls the manipulator from its angular start state to the final state directly in front of its base frame (at around 75 seconds) where a drop-off was intended. The same behaviour as in the feeding simulation is observed for joint 2, 3 and 5 where hacky variations in value can be seen in the plotted positions and velocities. In this path following task, the orientation of the end-effector is changed at each waypoint, as seen in Figure 5.14. It is therefore natural to see a corresponding stairway characteristic for the position of joint 6 as for joint 1 in this simulation. Joint 4 is seen to work together with joint 6 to define the orientation along the path. Finally, it can be seen from the velocity plots that the trajectory controller is working as expected for this segmented path as for the feeding motion.

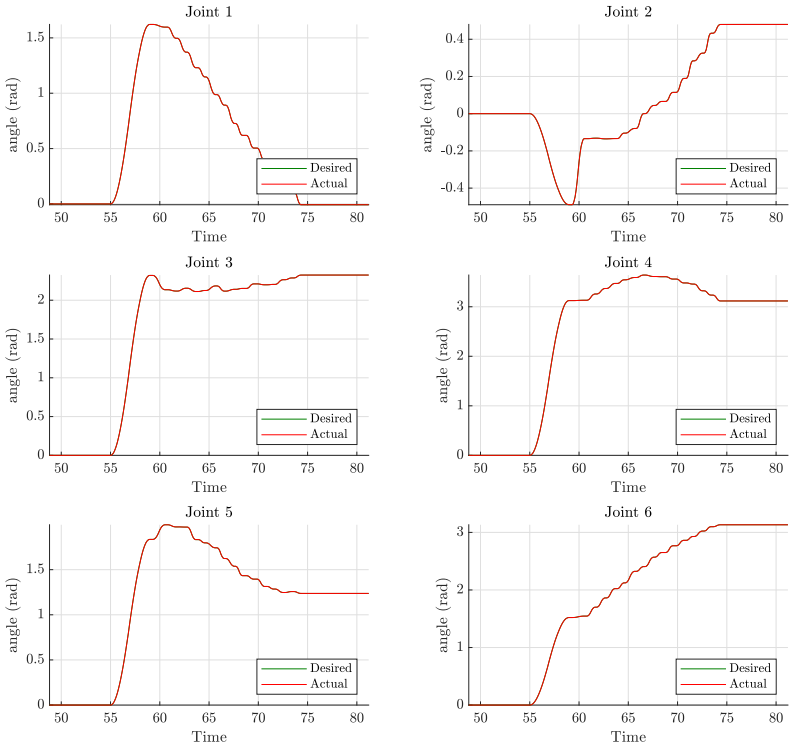


Figure 5.25: Joint positions for the gripping simulation.

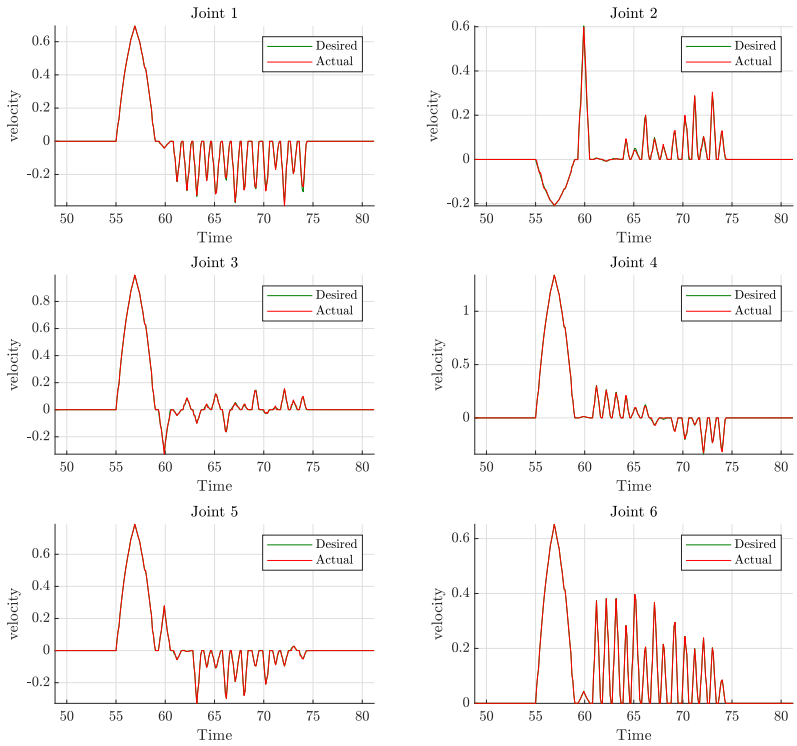


Figure 5.26: Joint velocities for the gripping simulation.

Much effort was laid into understanding and debugging the trajectory controller and the cartesian path planner without necessarily getting to the bottom of it and finding the root causes for failure. Therefore, some of the created paths were implemented on a pose to pose basis instead of running a trajectory. Either way, it was found that the created paths were all mostly feasible with some exceptions regarding initial and final waypoints that went outside the orientational capabilities of the robot. Following the recommendations found in section 3.3, a few PRM planning algorithms were tested on the created paths to experimentally test if these planners would yield a better trajectory generation from the cartesian path planner plugin. Testing the SPARStwo, the PRMstar, and the LazyPRMstar algorithms yielded very similar results to the use of the RRTConnect algorithm. All the mentioned algorithms were able to generate the trajectory for the cleaning motion most of the time. However, all yielded similar results for computing the generated grabbing and feeding motions resulting in generated trajectories of 0-8% completion.

5.5 Discussion

The results in the preceding section verify that the chosen robotic simulation software can indeed be used for rapid experimental development of robotic solutions, and used to control a robot in self-made environments using MoveIt for position and velocity control. The results verify that the Reach Bravo manipulator from Blueprintlab can, at least, be used for operations in static areas and solve a variety of operations using different end-effector tools. Therefore it can be stated that the simulations were successfully conducted with some constraints and limitations even though significant time-consuming challenges appeared along the way. Nevertheless, it is a long way from the simulations in this thesis to an industrial applicable autonomous system. It is not clear if the solution presented in this thesis will be neither possible nor beneficial in the coming years. First of all, the transportation must be solved. The chosen system in this thesis using suction adhesion is probably infeasible at the moment and would require extensive work to be solved. Other solutions, such as the discussed railway system might be preferable but require significant investments.

Even though the simulation software offers many opportunities using open-source packages, some of these have been root to some significant challenges. Especially the MoveIt package ended up being a time thief even though it is widely used and generally well-documented by the community. Following all of the driving rules for the waypoint in the *cartesian path planner* such as removing close to duplicate points, confirming the feasibility of all points and checking for singularities did not necessarily result in a generated trajectory. No direct reason has been found for this. However, it is suspected that one or more joints might be close to singularities along the way. This can indeed make some trajectory points that is to be created between waypoints infeasible or too hard to solve. Another source of error could be the characteristics of the planner(s), which can result in a timeout error if the planner is not able to complete the path within the allowed time frame. The *cartesian path planner* is not created for simple debugging purposes and can therefore often return a generated trajectory of 0-99% without any reason for why it was not able to compute the complete trajectory (or path). Such an incomplete trajectory is

then not possible to execute in the simulations. Regardless of the underlying reason, it is strange that the function is not able to compute simple trajectories for a simple path such as the gripping operations only consisting of feasible points. Nevertheless, this is a known shortcoming in the ROS community, which is under continuous development. Therefore, the robotic simulation software in itself is found to be a great tool for robotics control and simulations and is therefore recommended for future simulations.

As mentioned, the challenge related to implementing the planned paths were solved by using the MoveIt planner for segmented planning and execution. This created a hacky motion but worked well for verifying the planned paths. Nevertheless, there might be better solutions that are directly applicable such as using another inverse kinematic (*IKFast* using OpenRave bridge, *TRAC-IK* or implement a self-made solver) or using other motion planning plug-ins such as *Descartes* [117].

Chapter 6

Conclusion and further work

This thesis has presented a thorough literature review within the field of smolt production with regards to technological level, and general process description, whereas challenging operations that are facilitated for automation such as cleaning, feeding and waste removal has been identified. Comparing the identified operations with relatable industries provided the basis for a novel twofold robotic system comprised of a movable base using suction adhesion and a mounted manipulator for versatility. The latter part of this robotic system was emphasised for simulations in this thesis due to time limitations. Nevertheless, autonomous solutions seems promising for at least some of the operations within smolt facilities but it is not clear if the proposed system in this thesis will be neither possible nor beneficial. This will require further development and case studies but it is safe to say that more autonomous solutions will be implemented in smolt production in the future.

An approach for designing, implementing and simulating robotic manipulators has been presented. This approach includes CAD work, URDF description, ROS configurations of manipulators, simulation environments and controls, and simulations with logged results. This approach can facilitate rapid design, implementation and testing of new manipulator systems and controls.

For validating the possibility of autonomous completion for some of the identified operations in the smolt facilities, programmatic control of the most used motion planning plugin in ROS, the MoveIt package, was developed. Using this motion planning plugin for implementation of path following controls together with multi-goal paths defined in Matlab made it possible to simulate selected operations. A simple and efficient approach for defining the workspace of a manipulator has been developed in Matlab using screw theory, kinematics and the uniform Monte Carlo method. This method was used to create constrained paths within the workspace of the manipulator, hence achieve exclusively feasible points with regards to positional properties. This approach may pave the way for using the intuitive screw theory for efficiently defining workspace envelopes for custom manipulators in the future.

Using the constructed simulation environment, the created manipulator and the generated paths, the selected operations were successfully simulated in a tank environment similar to some of the largest tanks available in Norwegian smolt facilities. The constrained multi-goal paths were successfully executed using simple point-by-point following; hence a proof-of-concept for the identified operations was in a simple manner obtained.

To the author's best knowledge, a study like this thesis has not been reported earlier in the literature. Thus the results of the thesis contribute to lay the foundation for further research and development within the use of robotic systems in smolt production. This can be done using the findings of the literature review or using the developed tools and approaches mentioned together with the further work to take it a step further towards a demonstration stage. Therefore, it is expected that this project can work as a springboard for further development of the mentioned robotic system, other autonomous operation within the segment or other correlated projects.

A next natural step forward is to demonstrate the generated multi goals paths in real life using either a connected Reach Bravo robot or by using a similar manipulator controlled in cartesian space. If the first approach is to be used, the parameters of the robot must be adequately defined to reflect the real-life robot including new PID tuning and implemented torque- and joint limits of the actually connected motors. For the latter approach, this will probably not be necessary, and the paths can be demonstrated and validated directly. This is therefore the recommended approach going forward.

While working with this project, numerous challenges and interesting crossroads appeared. Much possible future work directly based on this project or more generic issues were found. Therefore it was decided to create a twofold list of bullet points, one for future work related to this thesis and one for generic future work that seems promising.

Future work related to this project:

1. Test and compare different planners in the OMPL library and other available and compatible planners in the MoveIt framework.
2. Implement automatic Rosbag recorder in the C++ code.
3. Test and compare other available inverse kinematics solver in the MoveIt framework.
4. Implement Moveit visual tools in C++ code for visualising generated trajectories, waypoints etc. in RViz.
5. Test other path planning or following frameworks than the cartesian path planner plugin, such as *Descartes*.
6. Explore if the *reuleaux* plugin [132] can be used for workspace computation or research other promising approaches.

7. Test hardware corresponding to a simulated manipulator with correct parameters. An interesting approach would be to do something similar to [119], using Matlab for controller integration and testing the hardware for the defined operations in this thesis.

8. Extend the simulation with a mobile base, preferably using suction adhesion.

9. Research robot vision and its possible use for controlling a manipulator in an amphibian environment such as a tank environment.

10. Research and implementation of other control concepts based on optimal motion planning with regard to different parameters such as power consumption or speed.

11. More advanced path- or trajectory following considering a more precise and dynamic environment with disturbances to make the simulations more feasible. This could include some of the aspects of underwater environment, presence of fish and obstacles.

Generic future work:

1. Create a simple and efficient tool/approach for generating a workspace envelope that takes into account torques limits etc. that is easy to implement in ROS.

2. Develop a tool for creating partly Gazebo environments such as a cylinder of water with the correct parameters to replicate a tank environment or similar.

Bibliography

[1] MOWI, "Salmon industry handbook 2019." https://ml.globenewswire.com/Resource/Download/1766f220-c83b-499a-a46e-3941577e038b, 2019. Accessed: 2020-05-10.

[2] Nærings- og fiskeridepartementet, "Press release: Sjømateksport for over 107 milliarder." https://www.regjeringen.no/no/aktuelt/sjomateksport-for-over-107-milliarder/id2684826/. Accessed: 2020-04-10.

[3] Fiskeridirektoratet, "Rømmingsstatistikk - antall og art." https://www.fiskeridir.no/Akvakultur/Tall-og-analyse/Roemmingsstatistikk/Roemmingsstatistikk-antall-og-art, 2020. Accessed: 2020-05-20.

[4] Salmon Evolution, "Salmon evolution har tatt første spadetak – nå starter byggingen av gigantanlegget for oppdrett på land i romsdal." https://www.salmonevolution.no/salmon-evolution-har-tatt-forste-spadetak-na-starter-byggingen-av-gigantanlegget-for-oppdrett-pa-land-i-romsdal/, 2020. Accessed: 2020-05-21.

[5] B. Tørud, B. Bang Jensen, S. Gåsnes, S. Grønbech, and K. Gismervik, "Animal welfare in fish hatcheries," *Norwegian veterinary institute*, vol. 14, 2019. Accessed: 2020-02-05.

[6] Erko Seafood, "Laksens livssyklus." https://erkoseafood.no/laks/, 2020. Accessed: 2020-05-20.

[7] Scottish Seafarms, "Salmon lifecycle." https://www.scottishseafarms.com/sustainability/salmon-lifecycle/, 2020. Accessed: 2020-05-20.

[8] Norwegian seafood council, "Salmon lifecycle." https://salmon.fromnorway.com/sustainable-aquaculture/the-salmon-lifecycle/, 2020. Accessed: 2020-05-20.

[9] M. Skatvold, “Fremtidens landbaserte produksjonsanlegg for settefisk,” Master’s thesis, NTNU, 2012. Accessed: 2020-05-20.

[10] Autosmolt2025, “Autonomous containment- and production systems for smolt and post-smolt production.” https://www.sintef.no/en/projects/autosmolt2025/. Partners: ScaleAquaculture (Project owner), SINTEF Ocean AS (Project Leader), Brimer AS, Sinkaberg-Hansen AS, Salmon Evolution AS, Andfjord Salmon AS, Wago Norge AS, Festo AS, Posicom AS, NTNU, Accessed: 2020-02-27.

[11] A. B. Holan and J. Kolarevic, “Postsmoltproduksjon i resirkulert sjøvann på land,” tech. rep., Nofima, 2020. Accessed: 2020-05-20.

[12] Nofima, “Optimalisert postsmoltproduksjon (opp).” https://nofima.no/prosjekt/optimalisert-postsmoltproduksjon-opp/. Accessed: 2020-03-20.

[13] Det kongelige nærings- og fiskeridepartement, “Forslag til endringer i regelverket for å legge til rette for landbasert oppdrett.” https://www.regjeringen.no/contentassets/ca4d67a7e21341e9a8e4529b14f898e5/horingsnotat-om-landbasert-oppdrett.pdf, 2015. Accessed: 2020-05-21.

[14] Fiskeridirektoratet, “Auksjon av akvakulturtillatelser juni 2018.” https://www.fiskeridir.no/Akvakultur/Tildeling-og-tillatelser/Auksjon-av-produksjonskapasitet/Auksjon-juni-2018, 2018. Accessed: 2020-05-21.

[15] Atlantic Sapphire, “Made in the usa - fresh from florida.” https://atlanticsapphire.com/american-sapphire, 2020. Accessed: 2020-05-21.

[16] Andfjord Salmon, “Andfjord salmon bygger framtidens havbruk og kombinerer det beste fra sjø- og landbasert oppdrett..” https://www.andfjord.no/teknologien, 2020. Accessed: 2020-05-21.

[17] S. Olsen, “Nordic aquafarms: – vi ser for oss å bygge flere anlegg nært de store markedene.” https://ilaks.no/nordic-aquafarms-vi-ser-for-oss-a-bygge-flere-anlegg-naert-de-store-markedene/, 3 2020. Accessed: 2020-05-21.

[18] Nofima, “ctrlaqua annual report 2017.” https://ctrlaqua.no/?publication=ctrlaqua-annual-report-2017, 2017. Accessed: 2020-01-25.

[19] O. A. Saue, “Worlds biggest smolt production facility.” https://salmonbusiness.com/worlds-biggest-smolt-production-facility-scheduled-for-completion-in-2019/, 2018. Accessed: 2020-02-05.

[20] Ilaks, “Salmars nye settefiskanlegg i trøndelag blir verdens største.” https://ilaks.no/salmars-nye-settefiskanlegg-i-trondelag-blir-verdens-storste/. Accessed: 2020-02-28.

[21] Sinkaberg-Hansen, "NÅ er det god fart ved det nye anlegget svaberget smolt!." https://sinkaberghansen.no/full-fart-ved-nye-svaberget-smolt/, 2019. Accessed: 2020-05-21.

[22] Akvagroup, "Akvagroup annual report 2019." https://ir.akvagroup.com/investor, 2019. Accessed: 2020-01-25.

[23] Scale Aquaculture, "Aquaoptima og vikan settefisk inngår avtale om rassamarbeid." https://scaleaq.no/aktuelt/2018/10/aquaoptima-og-vikan-settefisk-inngar-avtale-om-ras-samarbeid/, 2018. Accessed: 2020-05-21.

[24] Ernst & Young, "The norwegian aquaculture analysis 2019." https://www.ey.com/Publication/vwLUAssets/Norwegian_Aquaculture_Analysis_2019/\protect\T1\textdollarFILE/The%20Norwegian%20Aquaculture%20Analysis_2019.pdf. Accessed: 2020-02-26.

[25] Scale Aquaculture, "Optitank: Åttekantede kar for optimal hydraulikk." https://scaleaq.no/produkt/optitank/. Accessed: 2020-05-21.

[26] Stian Olsen, "All fisken døde i mowis nye settefiskanlegg på skjervøy." https://ilaks.no/all-fisken-dode-i-mowis-nye-settefiskanlegg-pa-skjervoy/, 5 2020. Accessed: 2020-06-07.

[27] Nofima, "Improving atlantic salmon smolt robustness to reduce losses in sea by development of screening tests, exercise regimes and markers." https://www.fhf.no/prosjekter/prosjektbasen/900870/, 2016. Accessed: 2020-01-25.

[28] Nofima, "ctrlaqua annual report 2019." https://nofima.no/en/nyhet/2019/04/annual-report-for-closed-containment-aquaculture/, 2019. Accessed: 2020-01-25.

[29] Tekset, "Tekset, conference for technology and innovations in the sea hatchery industry," 2019. Accessed: 2020-02-02.

[30] Nofima, Sunndal næringsselskap, "Smolt production in the future," 2018. Accessed: 2020-02-02.

[31] Redaksjon Kyst.no, "Nytt, helautomatisk oppdrettsanlegg." https://www.kyst.no/article/nytt-helautomatisk-oppdrettsanlegg/, 5 2020. Accessed: 2020-06-07.

[32] Skala Maskon, "Maskon vaccination system." https://en.skalamaskon.no/aquaculture2/vaccination. Accessed: 2020-02-27.

[33] Aqua Culture Supply, "Easyvac 12.00sb." http://aq-supply.net/easyvac%2012.000sb.html. Accessed: 2020-02-27.

[34] Skala Maskon, "Maskon fish pump." https://en.skalamaskon.no/aquaculture2/fish-pump. Accessed: 2020-02-27.

[35] Betten Maskin, "Smolt feeder." https://en.skalamaskon.no/aquaculture2/fish-pump. Accessed: 2020-02-27.

[36] Andfjord Salmon, "Mørenot robotics leverer roboter til andfjord salmon." https://www.andfjord.no/nyheter/morenot-robotics-leverer-roboter-til-andfjord-salmon-as. Accessed: 2020-05-21.

[37] Alvestad Marin AS, "Autotend® er den første roboten i sitt slag som er utviklet for automatisk røkting av egg og yngel." https://alvestad.com/produkter/autotend/. Accessed: 2020-05-28.

[38] Blue Unit A/S, "Worlds most advanced automatic, data visualization system to monitor water quality for ras." http://blue-unit.com/. Accessed: 2020-05-28.

[39] HL. Skjong AS, "Roterende tankvaskedyser." http://hlskjong.no/default.asp?page=9847,9852,9860&lang=1. Accessed: 2020-05-28.

[40] J. Gorle, B. F. Terjesen, and S. Summerfelt, "Technology development in ras: Focus on tank hydraulics," 2018. Accessed: 2020-02-02.

[41] Fiskeridirektoratet, "Lønnsomhetsundersøkelse for produksjon av laks og regnbueørret, 2015." https://www.fiskeridir.no/Akvakultur/Tall-og-analyse/Statistiske-publikasjoner/Loennsomhetsundersoekelser-for-laks-og-regnbueoerret. Accessed: 2020-02-27.

[42] Fiskeridirektoratet, "Excel file with number of employees in juvenile production of trout and salmon, 2018." https://www.fiskeridir.no/content/download/7629/95560/version/30/file/sta-laks-set-3-sysselsetting.xlsx. Accessed: 2020-02-27.

[43] Statistisk sentralbyrå, "Gjennomsnittslønn for havbruksarbeidere 2018." https://www.ssb.no/statbank/table/11418/tableViewLayout1/. Accessed: 2020-02-27.

[44] Vard Aquaculture, "Ny serie fÔringslØsninger fra vard aqua sunndal." https://vardaquaculture.com/nyheter/ny-serie-foringslosninger-fra-vard-aqua-sunndal/, 5 2020. Accessed: 2020-06-07.

[45] C. Noble, J. Nilsson, L. H. Stien, M. H. Iversen, J. Kolarevic, and K. Gismervik, "Welfare indicators for farmed salmon: How to assess and document fish welfare," 2018. Accessed: 2020-02-10.

[46] AquaGroup, "Akva fnc8 2.0." https://www.akvagroup.com/merdbasert-oppdrett/notvasksystemer/akva-fnc8-notvasker, 2020. Accessed: 2020-01-28.

[47] N. Bloecher, K. Frank, M. Bondø, D. Ribicic, P. C. Endresen, B. Su, and O. Floerl, "Testing of novel net cleaning technologies for finfish aquaculture," *Biofouling*, vol. 35, no. 7, pp. 805–817, 2019. PMID: 31538816.

[48] P. H. Skeide, "Methodology for customised offline robotic programming for cleaning of fish processing lines." Accessed: 2020-03-5.

[49] J. Vik, A. Melås, R. M. B. Hårstad, E. P. Stræte, and O. R. Langørgen, "Smart teknologi i landbruket – kartlegging og modenhetsvurdering," 2020. Accessed: 2020-01-28, ISSN:1503-2027.

[50] Farmbot, "Farmbot: Open-source cnc farming." https://farm.bot/, 2020. Accessed: 2020-02-24.

[51] Tertill, "Tertill: The solar powered, weather-proof, garden weeding robot.." https://www.franklinrobotics.com/, 2020. Accessed: 2020-02-24.

[52] EarthSense, "Terrasentia: Small-sized monitoring robot." https://www.earthsense.co/, 2020. Accessed: 2020-02-24.

[53] Agrobot, "Agrobot: robotic strawberry harvesting." https://www.agrobot.com/, 2020. Accessed: 2020-02-24.

[54] Asterix, "Asterix: Detection and spraying of weeds in plant rows." https://www.asterixproject.tech/, 2020. Accessed: 2020-02-24.

[55] Saga robotics, "Thorvald: Autonomous modular robot delivering agricultural services." https://sagarobotics.com/, 2020. Accessed: 2020-02-24.

[56] Small Robotic Company, "Dick: Autonomous non chemical weeding." https://www.smallrobotcompany.com/meet-the-robots#weed-killing, 2020. Accessed: 2020-02-24.

[57] Cognite, "Aker bp and cognite partner to explore the potential of robotics in the oil & gas industry." https://www.cognite.com/newsroom/aker-bp-and-cognite-partner-to-explore-the-potential-of-robotics-in-the-oil-gas-industry. Accessed: 2020-03-03.

[58] Equinor, "Eelume to be piloted at Åsgard." https://www.equinor.com/no/how-and-why/etv-news/eelume-to-be-piloted-at-aasgard.html. Accessed: 2020-03-03.

[59] OceanTech, "Your splash zone expert." https://oceantech.no/. Accessed: 2020-03-03.

[60] SPARC, "Robotics 2020: Multi-annual roadmap for robotics in europe." https://www.eu-robotics.net/cms/upload/topic_groups/H2020_Robotics_Multi-Annual_Roadmap_ICT-2017B.pdf. Accessed: 2020-03-03.

[61] NBC, "Industry research: Manipulators market 2020 report forecast by global industry trends, future growth, regional overview, market share, size, revenue, and forecast outlook till 2024." https://www.nbc-2.com/story/41678971/manipulators-market-2020-report-forecast-by-global-industry-trends-future-growth-regional-overview-market-share-size-revenue-and-forecast-outlook. Accessed: 2020-03-03.

[62] M. Yang, E. Yang, R. C. Zante, M. Post, and X. Liu, "Collaborative mobile industrial manipulator: A review of system architecture and applications," in *2019 25th International Conference on Automation and Computing (ICAC)*, pp. 1–6, Sep. 2019. Accessed: 2020-02-26.

[63] Kite Robotics, "Safe, efficient and sustainable, with crystal clear results." https://www.kiterobotics.com/. Accessed: 2020-01-30.

[64] Skyline Robotics, "The future of window cleaning is here." https://www.skylinerobotics.com/. Accessed: 2020-01-30.

[65] Serbot Swiss Innovations, "Gekko: Surface robot." https://www.serbot.ch/en/home/gekko-principle. Accessed: 2020-01-30.

[66] Pufeng intelligent technology, "Large window cleaning robot." https://www.theb1m.com/video/meet-chinas-window-cleaning-robot. Accessed: 2020-01-30.

[67] S. Nansai, K. Onodera, V. Prabakaran, R. E. Mohan, and M. Iwase, "Design and experiment of a novel façade cleaning robot with a biped mechanism," *Applied Sciences*, vol. 8, p. 2398, 11 2018. Accessed: 2020-02-05.

[68] H. Zhu, Y. Guan, W. Wu, X. Zhou, L. Zhang, X. Zhang, and H. Zhang, "The superior mobility and function of w-climbot illustrated by experiments," 12 2011. Accessed: 2020-02-05.

[69] H. Albitar, K. Dandan, A. Ananiev, and I. Kalaykov, "Underwater robotics: Surface cleaning technics, adhesion and locomotion systems," *International Journal of Advanced Robotic Systems*, vol. 13, p. 1, 01 2016. Accessed: 2020-02-05.

[70] Scranton Robotics, "Industrial tank/basin cleaning." http://www.scantronusa.com/roboticcleaning.html. Accessed: 2020-05-21.

[71] Jotun, "Jotun hull skating solutions." https://jointherevhullution.com/. Accessed: 2020-05-21.

[72] T. Bandyopadhyay, R. Steindl, F. Talbot, N. Kottege, R. Dungavell, B. Wood, J. Barker, K. Hoehn, and A. Elfes, "Magneto: A versatile multi-limbed inspection robot," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2253–2260, 2018. Accessed: 2020-05-21.

[73] General Electric, "Obotic solutions for improved safety, productivity data." https://inspection-robotics.com/. Accessed: 2020-05-21.

[74] R. Chen, L. Fu, Y. Qiu, R. Song, and Y. Jin, "A gecko-inspired wall-climbing robot based on vibration suction mechanism," *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, p. 095440621986904, 08 2019. Accessed: 2020-04-10.

[75] Ocean Innovations, "Stealthcleaner." https://www.ocein.no/en/products/en-stealthcleaner. Accessed: 2020-03-02.

[76] O. A. N. Eidsvik, B. O. Arnesen, and I. Schjølberg, "Seaarm-a subsea multi-degree of freedom manipulator for small observation class remotely operated vehicles," in *2018 European Control Conference (ECC)*, pp. 983–990, 2018. Accessed: 2020-05-20.

[77] B. O. A. Haugaløkken, E. K. Jørgensen, and I. Schjølberg, "Experimental validation of end-effector stabilization for underwater vehicle-manipulator systems in subsea operations," *Robotics and Autonomous Systems*, vol. 109, pp. 1 – 12, 2018. Accessed: 2020-05-26.

[78] K. Shi and X. Li, "Vacuum suction unit based on the zero pressure difference method," *Physics of Fluids*, vol. 32, no. 1, 2020. Accessed: 2020-02-26.

[79] V. Gradetsky, M. Kntazkov, A. Sukhanov, E. Semenov, V. Chaschukhin, and A. Kryukova, "Possibilities of using wall climbing robots for underwater application," pp. 239–246, 10 2017. Accessed: 2020-02-04.

[80] Houxiang Zhang, Jianwei Zhang, Guanghua Zong, Wei Wang, and Rong Liu, "Sky cleaner 3: a real pneumatic climbing robot for glass-wall cleaning," *IEEE Robotics Automation Magazine*, vol. 13, pp. 32–41, March 2006. Accessed: 2020-02-04.

[81] Y. Wang, X. Yang, Y. Chen, D. K. Wainwright, C. P. Kenaley, Z. Gong, Z. Liu, H. Liu, J. Guan, T. Wang, J. C. Weaver, R. J. Wood, and L. Wen, "A biorobotic adhesive disc for underwater hitchhiking inspired by the remora suckerfish," *Science Robotics*, vol. 2, no. 10, 2017. Accessed: 2020-03-02.

[82] P. Ditsche and A. Summers, "Learning from northern clingfish (gobiesox maeandricus): bioinspired suction cups attach to rough surfaces," *Philosophical Transactions of the Royal Society B: Biological Sciences*, vol. 374, no. 1784, 2019. Accessed: 2020-03-02.

[83] M. Adami and A. Seibel, "On-board pneumatic pressure generation methods for soft robotics applications," *Actuators*, vol. 8, p. 2, Dec 2018. Accessed: 2020-05-26.

[84] S. Sivčev, J. Coleman, E. Omerdić, G. Dooly, and D. Toal, "Underwater manipulators: A review," *Ocean Engineering*, vol. 163, pp. 431 – 450, 2018. Accessed: 2020-02-25.

[85] M. Dawidziuk and A. Olejnik, "General construction and classification of manipulators for underwater vehicles," *Polish Hyperbaric Research*, vol. 63, pp. 10–17, 06 2018. Accessed: 2020-03-02.

[86] Y. Cao, K. Lu, X. Li, and Y. Zang, "Accurate numerical methods for computing 2d and 3d robot workspace," *International Journal of Advanced Robotic Systems*, vol. 8, no. 6, p. 76, 2011. Accessed: 2020-02-25.

[87] K. M. Lynch and F. C. Park, *Modern Robotics: Mechanics, planning and control*. Cambridge University Press, 2017. Accessed: 2020-02-25.

[88] B. Siciliano, L.Sciavicco, L. Villani, and G. Oriolo, *Robotics: Modelling, planning and control*. Springer-Verlag London Limited, 2009. Accessed: 2020-02-25.

[89] S. Chiaverini, G. Oriolo, and I. D. Walker, *Kinematically Redundant Manipulators*, pp. 245–268. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. Accessed: 2020-02-25.

[90] FANUC, "Fanuc robot lr mate 200id/7c, /7lc, 4sc." https://www.fanuc.co.jp/en/product/robot/model/f_r_mini.html. Accessed: 2020-03-02.

[91] ECA GROUP, "Subsea electronical manipulator arms." https://www.ecagroup.com/en/solutions/subsea-electrical-manipulator-arms. Accessed: 2020-03-02.

[92] SAUVIM, "Maris 7080 underwater manipulator." https://gmarani.org/na/sauvim/RoboticArm.html. Accessed: 2020-03-02.

[93] Graal tech Underwater robotics, "Uma: Underwater modular arm." https://www.graaltech.com/uma-manipulator. Accessed: 2020-02-27.

[94] TMI-Orion, "Ema - electrical manipulator arm." https://www.tmi-orion.com/en/robotics/underwater-robotics/ema-electrical-manipulator-arm.htm. Accessed: 2020-03-02.

[95] BluePrintLab, "Reach bravo: Underwater manipulator." https://blueprintlab.com/products/reach-bravo/. Accessed: 2020-02-27.

[96] O. A. N. Eidsvik, B. O. Arnesen, and I. Schjølberg, "Seaarm-a subsea multi-degree of freedom manipulator for small observation class remotely operated vehicles," in *2018 European Control Conference (ECC)*, pp. 983–990, June 2018. Accessed: 2020-02-25.

[97] Festo, "Bionicmotionrobot: Pneumatic lightweight robot with natural movement patterns." https://www.festo.com/net/SupportPortal/Files/630184/Festo_BionicMotionRobot_en.pdf. Accessed: 2020-02-25.

[98] J. Tang, Y. Zhang, F. Huang, J. Li, Z. Chen, W. Song, S. Zhu, and J. Gu, "Design and kinematic control of the cable-driven hyper-redundant manipulator for potential underwater applications," *Applied Sciences*, vol. 9, p. 1142, Mar 2019. Accessed: 2020-03-02.

[99] Kinova robotics, "Gen3 ultra lightweight robot." https://www.kinovarobotics.com/en/knowledge-hub/gen3-ultra-lightweight-robot. Accessed: 2020-03-03.

[100] Kassow robotics, "Kr series of robotic manipulators." https://www.kassowrobots.com/products/download-center/. Accessed: 2020-03-03.

[101] M. Santos Pessoa de Melo, J. Gomes da Silva Neto, P. Jorge Lima da Silva, J. M. X. Natario Teixeira, and V. Teichrieb, “Analysis and comparison of robotics 3d simulators,” in *2019 21st Symposium on Virtual and Augmented Reality (SVR)*, pp. 242–251, 2019. Accessed: 2020-05-28.

[102] A. Staranowicz and G. L. Mariottini, “A survey and comparison of commercial and open-source robotic simulator software,” in *Proceedings of the 4th International Conference on PErvasive Technologies Related to Assistive Environments*, PETRA ’11, (New York, NY, USA), Association for Computing Machinery, 2011. Accessed: 2020-02-26.

[103] L. "Pitonakova, M. Giuliani, A. Pipe, and A. Winfield, “"feature and performance comparison of the v-rep, gazebo and argos robot simulators"," in *"Towards Autonomous Robotic Systems"* (M. "Giuliani, T. Assaf, and M. E. Giannaccini, eds.), ("Cham"), pp. "357–368", "Springer International Publishing", "2018". Accessed: 2020-02-26.

[104] The Construct, “Ros development studio.” https://www.theconstructsim.com/rds-ros-development-studio/, 2020. Accessed: 2020-05-15.

[105] Open Source Robotics Foundation, “Ros documentation.” http://wiki.ros.org/. Accessed: 2020-02-26.

[106] J. Chenl, H. Deng, W. Chai, J. Xiong, and Z. Xia, “Manipulation task simulation of a soft pneumatic gripper using ros and gazebo,” pp. 378–383, 08 2018. Accessed: 2020-04-29.

[107] Open Source Robotics Foundation, “Gazebo: Robot simulation made easy.” http://gazebosim.org/. Accessed: 2020-02-26.

[108] Anders Ridley-Smith, BluePrintLab, “Mail correspondance relating step files and pics of reach bravo.” Accessed: 2020-03-15.

[109] Ros Community, “Solidworks to urdf exporter.” http://wiki.ros.org/sw_urdf_exporter. Accessed: 2020-03-10.

[110] P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, and G. Ranzuglia, “MeshLab: an Open-Source Mesh Processing Tool,” in *Eurographics Italian Chapter Conference* (V. Scarano, R. D. Chiara, and U. Erra, eds.), The Eurographics Association, 2008. Accessed: 2020-03-10.

[111] The Blender Foundation, “Blender is the free and open source 3d creation suite.” https://www.blender.org/about/. Accessed: 2020-03-10.

[112] Ros Community, “Solidworks to urdf exporter github repository.” https://github.com/ros/solidworks_urdf_exporter/releases. Accessed: 2020-03-10.

[113] Open Source Robotic Foundation, “Gazebo models.” https://bitbucket.org/osrf/gazebo_models/src/default/. Accessed: 2020-03-25.

[114] C. Rocha, C. Tonetto, and A. Dias, "A comparison between the denavit–hartenberg and the screw-based methods used in kinematic modeling of robot manipulators," *Robotics and Computer-Integrated Manufacturing*, vol. 27, no. 4, pp. 723 – 728, 2011. Accessed: 2020-06-08.

[115] S. Castro, "Trajectory planning for robot manipulators." `https://blogs.mathworks.com/racing-lounge/2019/11/06/robot-manipulator-trajectory/`. Accessed: 2020-04-3.

[116] M. M. M. Manhães, S. A. Scherer, M. Voss, L. R. Douat, and T. Rauschenbach, "UUV simulator: A gazebo-based package for underwater intervention and multi-robot simulation," in *OCEANS 2016 MTS/IEEE Monterey*, IEEE, sep 2016. Accessed: 2020-04-02.

[117] ROS Industrial, "Descartes project: path-planning on under-defined cartesian trajectories." `http://wiki.ros.org/descartes`. Accessed: 2020-05-06.

[118] PickNik COnsulting, "Moveit documentation." `https://moveit.ros.org/documentation/concepts/`. Accessed: 2020-04-02.

[119] S. J. Gandhi, "Design, simulation and testing of a controller and software framework for automated construction by a robotic manipulator," p. 61, 2019. Accessed: 2020-05-14.

[120] M. Rahman, S. H. Rashid, K. R. Hassan, and M. Hossain, "Comparison of different control theories on a two wheeled self balancing robot," vol. 1980, 07 2018. Accessed: 2020-04-17.

[121] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, vol. 19, pp. 72–82, December 2012. Accessed: 2020-05-13.

[122] E. Theodorou, J. Buchli, and S. Schaal, "Learning policy improvements with path integrals," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics* (Y. W. Teh and M. Titterington, eds.), vol. 9 of *Proceedings of Machine Learning Research*, (Chia Laguna Resort, Sardinia, Italy), pp. 828–835, PMLR, 13–15 May 2010. Accessed: 2020-05-13.

[123] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "Stomp: Stochastic trajectory optimization for motion planning," in *International Conference on Robotics and Automation*, 2011. Accessed: 2020-05-13.

[124] M. Zucker, N. Ratliff, A. D. Dragan, M. Pivtoraiko, M. Klingensmith, C. M. Dellin, J. A. Bagnell, and S. S. Srinivasa, "Chomp: Covariant hamiltonian optimization for motion planning," *The International Journal of Robotics Research*, vol. 32, no. 9-10, pp. 1164–1193, 2013. Accessed: 2020-05-13.

[125] R. Ragel, I. Maza, F. Caballero, and A. Ollero, "Comparison of motion planning techniques for a multi-rotor uas equipped with a multi-joint manipulator arm," in *2015 Workshop on Research, Education and Development of Unmanned Aerial Systems (RED-UAS)*, pp. 133–141, Nov 2015. Accessed: 2020-03-23.

[126] J. Kuffner and S. LaValle, "Rrt-connect: An efficient approach to single-query path planning.," vol. 2, pp. 995–1001, 01 2000. Accessed: 2020-05-10.

[127] M. Strandberg, "Robot path planning: An object-oriented approach," 2004. Accessed: 2020-05-10.

[128] Ioan Sucan, "Moveit api." http://docs.ros.org/kinetic/api/moveit_ros_planning_interface/html/classmoveit_1_1planning__interface_1_1MoveGroupInterface.html#details. Accessed: 2020-05-03.

[129] J. Balchen, T. Andersen, and B. Foss, *Reguleringsteknikk*. 2016. Accessed: 2020-05-14.

[130] R. Y. Abdolmalaki, "Development of direct kinematics and workspace representation for smokie robot manipulator & the barret wam," 2017. Accessed: 2020-05-02.

[131] A. Jennings, "Matlab central file exchange: Sphere fit (least squared)." https://www.mathworks.com/matlabcentral/fileexchange/34129-sphere-fit-least-squared. Accessed: 2020-05-08.

[132] A. Makhal and A. K. Goins, "Reuleaux: Robot Base Placement by Reachability Analysis," *ArXiv e-prints*, Oct. 2017. Accessed: 2020-04-27.

Appendices

Appendix A

ROS code

This part of the appendix presents relevant code excerpts related to the ROS implementation. This includes a brief explanation of a URDF description, some launch files for various parts, description of transmissions (motors), controllers, implementation of MoveIt and description files for creating of Gazebo worlds. All of these files are excerpts from the used code in this project which can be found in the attached zip file.

```
<?xml version="1.0" encoding="utf-8"?>
<robot name= "r7" xmlns:xacro="https://www.ros.org/wiki/
    xacro">

<!-- START ROBOT DESCRIPTION-->
  <link
    name="base_link">
    <inertial>
      <origin
        xyz="-0.01526 -2.9742E-06 0.037769"
        rpy="0 0 0" />
      <mass
        value="1.54925" />
      <inertia
        ixx="0.002"
        ixy="0.0"
        ixz="0.0"
        iyy="0.002"
        iyz="0.0"
        izz="0.002" />
    </inertial>
    <visual>
      <origin
```

```
      xyz="0 0 0"
      rpy="0 0 0" />
    <geometry>
      <mesh
        filename="package://r7_description/meshes/base_link.
            STL" scale="1 1 1" />
    </geometry>
    <material
      name="">
      <color
        rgba="0.79216 0.81961 0.93333 1" />
    </material>
  </visual>
  <collision>
    <origin
      xyz="0 0 0"
      rpy="0 0 0" />
    <geometry>
      <mesh
        filename="package://r7_descri ption/meshes/base_link
            .STL" />
    </geometry>
  </collision>
</link>

<joint
  name="Joint01"
  type="revolute">
  <origin
    xyz="0 0 0.041"
    rpy="0 0 0" />
  <parent
    link="base_link" />
  <child
    link="link1" />
  <axis
    xyz="0 0 1" />
  <limit
    lower="-3.14"
    upper="3.14"
    effort="10"
    velocity="1" />
  <dynamics
    damping="0.5"
    friction="0.1" />
```

```
66  </joint>
67
68  <link
69    name="link1">
70    <inertial>
71      <origin
72        xyz="0.020745 0.011274 0.082486"
73        rpy="0 0 0" />
74      <mass
75        value="0.38653" />
76      <inertia
77        ixx="0.002"
78        ixy="0.0"
79        ixz="0.0"
80        iyy="0.002"
81        iyz="0.0"
82        izz="0.002" />
83    </inertial>
84    <visual>
85      <origin
86        xyz="0 0 0"
87        rpy="0 0 0" />
88      <geometry>
89        <mesh
90          filename="package://r7_description/meshes/link1.STL"
              />
91      </geometry>
92      <material
93        name="">
94        <color
95          rgba="0.79216 0.81961 0.93333 1" />
96      </material>
97    </visual>
98    <collision>
99      <origin
100       xyz="0 0 0"
101       rpy="0 0 0" />
102     <geometry>
103       <mesh
104         filename="package://r7_description/meshes/link1.STL"
              />
105     </geometry>
106   </collision>
```

```
  </link>
```

Listing A.1: URDF excerpt describing the setup of the base- and first link with a corresponding joint in between them. Each link is described with its origin coordinate system based on position and orientation relative to the previous link. Further, an arbitrary chosen mass and inertia matrix has been chosen for working simulations where the weight was set based on size of components and a maximum total weight of the whole manipulator of 10kg. The visual and collision parts of the link description defines what is to be presented in the visual windows, and what geometry is to be used in collision calculations. A joint is defined by choosing the type of joint, its placement relative to the parent link, choosing axis of rotation and if desired, chosing limits and dynamics.

```
<launch>
   <param name="robot_description" command="$(find xacro)/
      xacro --inorder '$(find r7_description)/urdf/r7.xacro
      '"/>

<!-- Combine joint values-->
   <node name="robot_state_publisher" pkg="
      robot_state_publisher" type="state_publisher"/>

<!-- Show in Rviz -->
   <node name="rviz" pkg="rviz" type="rviz" /> <!--args=" -
      d $(find r7_description)/launch/config.rviz"-->

<!-- Send joint values using state publisher-->
   <node name="joint_state_publisher" pkg="
      joint_state_publisher" type="joint_state_publisher" >
      <param name="use_gui" value="True"/>
   </node>
</launch>
```

Listing A.2: RViz launch description that presents the spawning of the RViz visualization node and the joint state publisher used to control the joint positions in the visualization.

```
<?xml version="1.0" encoding="UTF-8"?>
<launch>
      <!-- Group all launching to the same namespace (ns)--
         >
       <group ns="/r7">

      <!-- Robot model -->
      <param name="robot_description" command="$(find xacro
         )/xacro --inorder '$(find r7_description)/urdf/r7.
         xacro'" />
      <arg name="x" default="0"/>
      <arg name="y" default="0"/>
```

```
10        <arg name="z" default="0"/>
11        <arg name="roll" default="0"/>
12        <arg name="pitch" default="0"/>
13        <arg name="yawn" default="0"/>
14
15        <!-- Spawn the robot model -->
16        <node name="mybot_spawn" pkg="gazebo_ros" type="
              spawn_model" output="screen"
17             args="-urdf -param robot_description -model r7 -x
                   $(arg x) -y $(arg y) -z $(arg z)
18                 -R $(arg roll) -P $(arg pitch) -Y $(arg yawn
                       )" />
19
20        <!-- Load controllers -->
21        <rosparam command="load" file="$(find r7_description)
              /config/joints.yaml" />
22
23        <!-- Controllers -->
24        <node name="controller_spawner" pkg="
              controller_manager" type="spawner"
25           respawn="false" output="screen" ns="/r7"
26           args="--namespace=/r7
27           joint_state_controller
28           joint1_position_controller
29           joint2_position_controller
30           joint3_position_controller
31           joint4_position_controller
32           joint5_position_controller
33           joint6_position_controller
34           --timeout 60">
35        </node>
36
37        <!-- rqt nodes such that the PID controllers can be
              changed during simulation and use the rqt
              publisher to change values -->
38        <node name="rqt_reconfigure" pkg="rqt_reconfigure"
              type="rqt_reconfigure" />
39        <node name="rqt_publisher" pkg="rqt_publisher" type="
              rqt_publisher" />
40
41    </group>
```

```
</launch>
```

Listing A.3: Spawn file for a simple Gazebo simulation. The robot model is first spawned in a desired pose in an active Gazebo environment. Corresponding joints are then located, loaded and spawned to be able to communicate with the transmission (motors) in the simulation. The last rqt configuration node is an additional node that can be used for tuning during realtime, but is not necessarily used.

```
<build_depend>controller_manager</build_depend>
<build_depend>joint_state_controller</build_depend>
<build_depend>robot_state_publisher</build_depend>

<exec_depend>controller_manager</exec_depend>
<exec_depend>joint_state_controller</exec_depend>
<exec_depend>robot_state_publisher</exec_depend>
```

Listing A.4: Code excerpt from the *package* file describing how to add and build dependencies into the package.

```
<transmission name="trans01">
  <type>transmission_interface/SimpleTransmission</type>
  <joint name="Joint01">
    <hardwareInterface>hardware_interface/
        EffortJointInterface</hardwareInterface>
  </joint>
  <actuator name="motor01">
    <hardwareInterface>hardware_interface/
        EffortJointInterface</hardwareInterface>
    <mechanicalReduction>1</mechanicalReduction>
  </actuator>
</transmission>
```

Listing A.5: Transmission for joint describing a hardware connection such that a simulation or real robot can be controlled.

```
# Publish all joint states
joint_state_controller:
  type: joint_state_controller/JointStateController
  publish_rate: 50

# Position Controllers
joint1_position_controller:
  type: effort_controllers/JointPositionController
  joint: Joint01
  pid: {p: 50.0, i: 5.0, d: 1.0}
```

```
11
12 # Same for rest of the joints...
```

Listing A.6: Short excerpt from the definition of joint position controllers of the effort joint type. A joint state controller is launched for feedback of the actual joint states and position controllers with defined PID gains are spawned for each joint.

```
1 <gazebo>
2    <plugin name="gazebo_ros_control" filename="
        libgazebo_ros_control.so">
3    </plugin>
4 </gazebo>
```

Listing A.7: Gazebo tag in robot description file to connect the URDF file in ROS to Gazebo (cross compatibility).

```
1    <link name="world" />
2
3    <joint name="start_joint" type="fixed">
4       <origin xyz="0 0 0" rpy="0 0 0" />
5       <parent link="world" />
6       <child link="base_link" />
7    </joint>
```

Listing A.8: Fixing base link to world frame to avoid robot from falling.

```
1 <?xml version = "1.0" encoding="UTF-8"?>
2    <launch>
3       <!--Overwriting arguments to put in the world
           creation-->
4       <arg name="debug" default="false"/>
5       <arg name="gui" default="false"/>
6       <arg name="pause" default="true"/>
7       <arg name="world" default="$(find my_simulations)/
           worlds/my_world.world"/>
8
9       <!--including gazebo_ros launcher-->
10      <include file="$(find gazebo_ros)/launch/empty_world.
           launch">
11         <arg name="world_name" value="$(arg world)"/>
12         <arg name="debug" value="$(arg debug)"/>
13         <arg name="gui" value="$(arg gui)"/>
14         <arg name="paused" value="$(arg pause)"/>
15         <arg name="use_sim_time" value="true"/>
16      </include>
```

```
    </launch>
```

Listing A.9: Launch file for a arbitrary created world. Parameters are chosen for the behaviour of the simulation. These are then fed into the normal "empty world" launching where the world is overrun by the sel-created world.

```
<?xml version = "1.0" ?>
<sdf version="1.5">
   <world name="myworld">
      <include>
         <uri>model://sun</uri>
      </include>

      <include>
         <uri>model://ground_plane</uri>
      </include>

      <include>
         <uri>model://cylindrical_tank_D28</uri>
         <pose>-14.5 -14.5 0 0 0 0</pose>
         <static>true</static>
      </include>

      <model name="ceiling_cylinder">
      <static>true</static>
      <pose>0 0 6 0 0 0</pose>
      <link name="link">
         <visual name="visual_cylinder">
         <pose>0 0 -3 0 0 0</pose>
         <cast_shadows>false</cast_shadows>
         <geometry>
            <cylinder>
               <radius>14</radius>
               <length>6</length>
            </cylinder>
         </geometry>
         <material>
            <script>
             <uri>file://media/materials/scripts/gazebo.
                material</uri>
            <name>Gazebo/BlueTransparent</name>
            </script>
         </material>
         </visual>
      </link>
      </model>
```

```
    </world>
</sdf>
```

Listing A.10: Description of a created world. All models are included and different characteristics are described for these.

```
<?xml version="1.0"?>
<model>
    <name>Cylindrical tank</name>
    <version>1.0</version>
    <sdf version="1.5">model.sdf</sdf>

    <author>
        <name>Paal HS</name>
        <email>palhske@stud.ntnu.no</email>
    </author>

    <description>
        A simple cylindrical tank
    </description>
</model>
```

Listing A.11: Model configuration is a necessary step when creating a model which only described what is created and by who.

```
<?xml version="1.0" ?>
<sdf version="1.5">
 <model name="cylindrical_tank_D28">
  <static>true</static>
  <link name="link">
   <collision name="collision">
    <geometry>
     <mesh>
      <uri>model://cylindrical_tank_D28/meshes/
          tank_D28mesh.STL</uri>
      <scale>0.001 0.001 0.001</scale>
     </mesh>
    </geometry>
    <surface>
     <contact>
      <collide_bitmask>0xffff</collide_bitmask>
     </contact>
     <friction>
      <ode>
       <mu>100</mu>
```

```
          <mu2>50</mu2>
        </ode>
      </friction>
    </surface>
  </collision>
  <visual name="visual">
    <cast_shadows>false</cast_shadows>
    <geometry>
      <mesh>
        <uri>model://cylindrical_tank_D28/meshes/
            tank_D28mesh.STL</uri>
        <scale>0.001 0.001 0.001</scale>
      </mesh>
    </geometry>
    <material>
      <script>
        <uri>file://media/materials/scripts/gazebo.material
            </uri>
        <name>Gazebo/Grey</name>
      </script>
    </material>
  </visual>
 </link>
</model>
</sdf>
```

Listing A.12: This is an SDF description file for a created model from SolidWorks. The STL of the model is scaled to appropriate size and in much similar fashion as for links, collision and visual properties are defined including a friction parameter for the interaction with other objects in a simulation.

```
# Publish all joint states
joint_state_controller:
  type: joint_state_controller/JointStateController
  publish_rate: 50

# Trajectory controller
joint_trajectory_controller:
  type: effort_controllers/JointTrajectoryController
  joints:
    - Joint01
    - Joint12
    - Joint23
    - Joint34
    - Joint45
    - Joint56
```

```
  constraints:
    goal_time: 0.5
    stopped_velocity_tolerance: 0.5

  gains:
    Joint01: {p: 5000, d: 0.5, i: 5, i_clamp: 1}
    Joint12: {p: 2000, d: 0.2, i: 2, i_clamp: 1}
    Joint23: {p: 2000, d: 0.2, i: 2, i_clamp: 1}
    Joint34: {p: 2000, d: 0.2, i: 2, i_clamp: 1}
    Joint45: {p: 2000, d: 0.2, i: 2, i_clamp: 1}
    Joint56: {p: 2000, d: 0.2, i: 2, i_clamp: 1}

  stop_trajectory_duration: 0.5
  state_publish_rate: 50
  action_monitor_rate: 20
```

Listing A.13: This is the code excerpt for the joint trajectory controller that is the main building block for controls using MoveIt. In a much similar for as for simple position controllers a trajectory controller is defined by a chosen type. Further, all joints to be controlled by the overall JTC is defined with its corresponding PID gains. Other constraints can be set such as the goal time of stopped velocity tolerance which is parameters for how the trajectory controller should complete the gives tasks.

```
controller_list:
  - name: r7/joint_trajectory_controller
    action_ns: "follow_joint_trajectory"
    type: FollowJointTrajectory
    joints: [Joint01, Joint12, Joint23, Joint34, Joint45,
        Joint56]
```

Listing A.14: Follow Joint Trajectory controller type. This is the connected type of control action connected with the trajectory controller for doing the "action" that is compatible with the MoveIt commands.

```
<launch>
    <!--Load ros-controllers to the param server-->
    <rosparam file="$(find r7_moveit_config)/config/
        controllers.yaml"/>
    <param name="use_controller_manager" value ="false"/>
    <param name="trajectory_execution/
        execution_duration_monitoring" value="false"/>
    <param name="moveit_controller_manager" value="
        moveit_simple_controller_manager/
        MoveItSimpleControllerManager"/>
</launch>
```

Listing A.15: Spawning of controller manager for executing trajectory commands.

```
<launch>
   <!--Loading the joint names-->
   <rosparam command="load" file="$(find r7_moveit_config)/
      config/joint_names.yaml"/>

   <include file="$(find r7_moveit_config)/launch/
      planning_context.launch">
      <arg name="load_robot_description" value="false"/>
   </include>

   <node name="joint_state_publisher" pkg="
      joint_state_publisher" type="joint_state_publisher">
      <param name="/use_gui" value="false"/>
      <rosparam param="/source_list">[/r7/joint_states]</
         rosparam>
   </node>

   <include file="$(find r7_moveit_config)/launch/
      move_group.launch">
      <arg name="publish_monitored_planning_scene" value="
         true"/>
   </include>

   <include file="$(find r7_moveit_config)/launch/
      moveit_rviz.launch">
      <arg name="config" value="true"/>
   </include>
</launch>
```

Listing A.16: Implementation of MoveIt. First, the joint names are loaded into a parameter list such that MoveIt can locate the necessary joints to control the simulation. Further, the MoveIt planning context is initiated (that can be seen in the RViz visualization that is also spawned), joint state publisher is initiated for feedback (here remapped) and the MoveIt core (or MoveGroup node) is initialized making it possible to use MoveIt for controlling the simulated robot.

Appendix B

Matlab code

This part of the appendix presents code excerpts from the Matlab code used for multi goal path planning that is mentioned in the thesis. Some self defined functions are described in addition to code for generating of a fuzzy workspace. For full code, please see attached zip file.

```
nr = 1; %Number of points in between each waypoint that
    is set.
height = 0.1;
figure('Name','Test');
axis([-1 1 -1 1 -1 1]);
title('Trajectory test');
xlabel('X');
ylabel('Y');
zlabel('Z');
hold on;
grid on;

tTimes = linspace(0,1,nr);
tInterval = [0 5];
[s,sd,sdd] = trapveltraj([0 1],numel(tTimes));

T0 = trvec2tform([0.0 0.3 0.3])*eul2tform([-pi 0 -pi],'
    XYZ');
Tf = trvec2tform([0.0 0.4 height])*eul2tform([-pi 0 -pi
    ],'XYZ');
[T,dT,ddT] = transformtraj(T0,Tf,tInterval,tTimes,'
    TimeScaling',[s;sd;sdd]);
plotTransforms(tform2trvec(T),tform2quat(T));
traj = [tform2trvec(T) tform2quat(T)];
```

```
21
22      rl = 0.45; %radius
23      theta = pi/2:-pi/10:-pi/2;
24
25      for i = 1:length(theta)
26            Tfi = trvec2tform([rl*cos(theta(i)) rl*sin(theta(i
                  )) height])*eul2tform([-pi 0 -pi],'XYZ');
27            [Ti,dTi,ddTi] = transformtraj(Tf,Tfi,tInterval,
                  tTimes,'TimeScaling',[s;sd;sdd]);
28            plotTransforms(tform2trvec(Ti),tform2quat(Ti));
29            traj = [traj ; tform2trvec(Ti) tform2quat(Ti)];
30
31            Tf = Tfi;
32      end
33      hold off;
```

Listing B.1: This code presents the generation of a multi goal path of a simple half circle. First, a trapezoidal velocity profile is described for use in the orientational transformations. Further, two initial transformation matrices (or poses) are defined for starting position. Using the number of points, the transformtraj function is used to create transformations in between the two defined matrices. These transformations are then plotted and added to a variable storing the transformation matrices in poses (containing position and quaternion). Using the same approach in a loop through the defined vector of angles, the total path consisting of poses can be exported for further use.

```
1  function T = cleanDups(T)
2      rows=size(T,1);
3      toRemove = [];
4
5      for i=1:rows-1
6          if round(T(i,1:3),5) == round(T(i+1,1:3),5)
7              toRemove = [toRemove,i];
8          end
9      end
10     for tR=toRemove
11         T(tR,:)=[];
12     end
13 end
```

Listing B.2: Created matlab function for removing poses that were too close to each other. The function loops through the input poses (rows), tags if two consecutive poses are similar within 5 decimals and finally deletes these poses (rows).

```
1      %Joint limits
2      j1_min = -3;
3      j1_max = 3;
4      j2_min = -1.4;
```

```
    j2_max = 3;
    j3_min = -1;
    j3_max = 2.5;
    j4_min = -3;
    j4_max = 3;
    j5_min = -1;
    j5_max = 2.5;
    j6_min = -3.1;
    j6_max = 3.1;

    % Number of angles
    N = 10000;

    j1 = j1_min + (j1_max-j1_min)*rand(N,1);
    j2 = j2_min + (j2_max-j2_min)*rand(N,1);
    j3 = j3_min + (j3_max-j3_min)*rand(N,1);
    j4 = j4_min + (j4_max-j4_min)*rand(N,1);
    j5 = j5_min + (j5_max-j5_min)*rand(N,1);
    j6 = j6_min + (j6_max-j6_min)*rand(N,1);

    % Home position from Join01 to Joint 56 (base and up to
        joint 1 added as
    % constant to plotting) Since these are kind of the
    M =[[1, 0, 0, -0.001];
        [0, 1, 0, 0];
        [0, 0, 1, 0.70653];
        [0, 0, 0, 1]] ;

    % Screw axes for the joints
    Slist = [[0; 0; 1; 0; 0; 0], ...
            [0; 1; 0; -0.114; 0; 0.045], ...
            [0; 1; 0; -0.41153; 0; 0.045], ...
            [0; 0; 1; 0; 0; 0], ...
            [0; 1; 0; -0.63153; 0; 0.035], ...
            [0; 0; 1; 0; 0.01; 0]];

    height_ground = 0.041;

    %% Plotting points
    figure('Name','Workspace Generic Points')
    title('Workspace generic [points]');
    xlabel('X');
    ylabel('Y');
    zlabel('Z');
    hold on;
```

```
    grid on;

    for i = 1:N
        thetalist = [j1(i);j2(i);j3(i);j4(i);j5(i);j6(i)];
        T = FKinSpace(M,Slist,thetalist);
        X=T(1,4);
        Y=T(2,4);
        Z=T(3,4)+height_ground; %Added height from ground
        plot3(X,Y,Z,'.')
    end
    hold off;
```

Listing B.3: Script for creating a fuzzy workspace using defined joint limits, uniform Monte Carlo method and screw theory (with its available functions) for generating possible transformation matrices (poses). Here, the positional components are plotted. For plotting of orientational components or further utilisation, see attachments.

```
    nr = 1; %Number of points in between each waypoint that
        is set.
    height = 0.2;
    figure('Name','Cleaning');
    axis([-1 1 -1 1 -1 1]);
    title('Trajectory cleaning');
    xlabel('X');
    ylabel('Y');
    zlabel('Z');
    hold on;
    grid on;

    tTimes = linspace(0,1,nr); %Number of points/frames in
        between.
    tInterval = [0 5]; %Time interval
    [s,sd,sdd] = trapveltraj([0 1],numel(tTimes));

    T0 = trvec2tform([0.0 0.3 0.3])*eul2tform([-pi 0 -pi],'
        XYZ'); %Start matrix
    Tf = trvec2tform([0.0 0.4 height])*eul2tform([-pi 0 -pi
        ],'XYZ');
    [T,dT,ddT] = transformtraj(T0,Tf,tInterval,tTimes,'
        TimeScaling',[s;sd;sdd]);

    plotTransforms(tform2trvec(T),tform2quat(T));
    trajc = [tform2trvec(T) tform2quat(T)];

    rs = 0.3; %inner radius
    rl = 0.5; %outer radius
```

```
theta = pi/2:-pi/10:-pi/2;
theta = [theta flip(theta)]; %All angle values inside
    the defined trajectory

for i = 1:length(theta)
    if i < length(theta)/2
        Tfi = trvec2tform([rl*cos(theta(i)) rl*sin(theta(i
            )) height])*eul2tform([-pi 0 -pi],'XYZ');
        [Ti,dTi,ddTi] = transformtraj(Tf,Tfi,tInterval,
            tTimes,'TimeScaling',[s;sd;sdd]);
        plotTransforms(tform2trvec(Ti),tform2quat(Ti));
        trajc = [trajc ; tform2trvec(Ti) tform2quat(Ti)];

        Tf = Tfi;
    else
        Tfi = trvec2tform([rs*cos(theta(i)) rs*sin(theta(i
            )) height])*eul2tform([-pi 0 -pi],'XYZ');
        [Ti,dTi,ddTi] = transformtraj(Tf,Tfi,tInterval,
            tTimes,'TimeScaling',[s;sd;sdd]);
        plotTransforms(tform2trvec(Ti),tform2quat(Ti));
        trajc = [trajc ; tform2trvec(Ti) tform2quat(Ti)];

        Tf = Tfi;
    end
end
hold off;
```

Listing B.4: Path planning for cleaning operation. See B.1 for more details

```
nr = 1;
oR = 0.45; %outer radius
a = 0.3; %inner radius
maxH = 0.60;

% Inward motion spiral
n = 2.5; % Choose number of revolutions
b = (oR-a)/n;
numb = n*20;
theta = flip(linspace(0,n,numb));

figure('Name','TrajectoryFeed Part 1');
axis([-1 1 -1 1 -1 1]);
title('Waypoints for feeding w constant z');
xlabel('X');
ylabel('Y');
```

```
17    zlabel('Z');
18    hold on;
19    grid on;
20
21    tTimes = linspace(0,1,nr);
22    tInterval = [0 5];
23    [s,sd,sdd] = trapveltraj([0 1],numel(tTimes)); %Simple
          trapezoidal velocity profile for configuring
          orientations.
24
25    % Setting initial points (start ca startpoint of the
          spiral motion
26    initX = round((a+b*theta(1))* cos(2*pi*theta(1)),1);
27    initY = round((a+b*theta(1))* sin(2*pi*theta(1)),1);
28
29    %Creating a simple "Starting point"
30    T0 = trvec2tform([0 0 0.4])*eul2tform([0 0 0],'ZYX');
31    Tf = trvec2tform([initX initY maxH])*eul2tform([atan2(
          initY,initX) atan2(sqrt(initY^2+initX^2),maxH)/2 0],'
          ZYX');
32    [T,dT,ddT] = transformtraj(T0,Tf,tInterval,tTimes,'
          TimeScaling',[s;sd;sdd]);
33
34    plotTransforms(tform2trvec(T),tform2quat(T));
35    trajf = [tform2trvec(T) tform2quat(T)];
36
37    for i = 1:length(theta)
38       xi = (a+b*theta(i))*cos(2*pi*theta(i));
39       yi = (a+b*theta(i))*sin(2*pi*theta(i));
40       Tfi = trvec2tform([xi yi maxH])*eul2tform([atan2(yi,
             xi) atan2(sqrt(yi^2+xi^2),maxH)/2 0],'ZYX'); %atan
             (sqrt(yi^2+xi^2)/maxH)
41       [Ti,dTi,ddTi] = transformtraj(Tf,Tfi,tInterval,tTimes
             ,'TimeScaling',[s;sd;sdd]);
42       plotTransforms(tform2trvec(Ti),tform2quat(Ti));
43       trajf = [trajf ; tform2trvec(Ti) tform2quat(Ti)];
44       Tf = Tfi;
45    end
46    hold off;
```

Listing B.5: Path planning for feeding operations with constant z height. See B.1 for more details

```
1     % Same parameters as in other feed listing
2
3     figure('Name','TrajectoryFeed Part 2');
```

```
    axis([-1 1 -1 1 -1 1]);
    title('Waypoints for feeding w z-constraint');
    xlabel('X');
    ylabel('Y');
    zlabel('Z');
    hold on;
    grid on;

    tTimes = linspace(0,1,nr);
    tInterval = [0 5];
    [s,sd,sdd] = trapveltraj([0 1],numel(tTimes)); %Simple
        trapezoidal velocity profile for configuring
        orientations.

    initX = round((a+b*theta(1))* cos(2*pi*theta(1)),1);
    initY = round((a+b*theta(1))* sin(2*pi*theta(1)),1);

    T0 = trvec2tform([0 0 0.4])*eul2tform([0 0 0],'ZYX');
    Tf = trvec2tform([initX initY maxH])*eul2tform([atan2(
        initY,initX) atan2(sqrt(initY^2+initX^2),maxH)/2 0],'
        ZYX');
    [T,dT,ddT] = transformtraj(T0,Tf,tInterval,tTimes,'
        TimeScaling',[s;sd;sdd]);

    plotTransforms(tform2trvec(T),tform2quat(T));
    trajf1 = [tform2trvec(T) tform2quat(T)];

    % Loading workspace definition
    workspace = load('BravoWorkspace.mat'); %Loading the
        struct
    WSAShape = workspace.shp; %Loading the shape (sphere)
    radius = workspace.radius; %Loading the shapes radius
    center = workspace.center; %Loading the shape center

    for i = 1:length(theta)
        xi = (a+b*theta(i))*cos(2*pi*theta(i));
        yi = (a+b*theta(i))*sin(2*pi*theta(i));
        Tfi = trvec2tform([xi yi maxH])*eul2tform([atan2(yi,
            xi) atan2(sqrt(yi^2+xi^2),maxH)/2 0],'ZYX'); %atan
            (sqrt(yi^2+xi^2)/maxH)
        [Ti,dTi,ddTi] = transformtraj(Tf,Tfi,tInterval,tTimes
            ,'TimeScaling',[s;sd;sdd]);

        % Checking if point is inside workspace and handling
            exceptions
```

```
39        for j = 1:size(Ti,3)
40            [nearIDX, tf] = checkPosTrans(WSAShape,Ti(:,:,j));
41            if tf == false % If point is NOT inside workspace
42                zProj = sqrt(radius^2 - (Ti(1,4,j)-center(1))^2
                      - (Ti(2,4,j)-center(2))^2) + center(3); %
                    Projecting point down onto the workspace
43                if isreal(zProj) %If projection is imaginary, x
                     and y is outside workspace
44                    Ti(3,4,j) = zProj;
45                else %Using the nearest point from the AShape
46                    Ti(1,4,j) = WSAShape.Points(nearIDX,1);
47                    Ti(2,4,j) = WSAShape.Points(nearIDX,2);
48                    Ti(3,4,j) = WSAShape.Points(nearIDX,3);
49                end
50            end
51        end
52        plotTransforms(tform2trvec(Ti),tform2quat(Ti));
53        trajf1 = [trajf1 ; tform2trvec(Ti) tform2quat(Ti)];
54        Tf = Tfi;
55    end
56    hold off;
```

Listing B.6: Path planning for feeding operations with projected z height. See B.1 for more details

```
1     % Same parameters as in other feed listing
2
3     figure('Name','TrajectoryFeed');
4     axis([-1 1 -1 1 -1 1]);
5     title('Waypoints for feeding w nearestNeighbor');
6     xlabel('X');
7     ylabel('Y');
8     zlabel('Z');
9     hold on;
10    grid on;
11
12    tTimes = linspace(0,1,nr);
13    tInterval = [0 5];
14    [s,sd,sdd] = trapveltraj([0 1],numel(tTimes));
15
16    initX = round((a+b*theta(1))* cos(2*pi*theta(1)),1);
17    initY = round((a+b*theta(1))* sin(2*pi*theta(1)),1);
18
19    T0 = trvec2tform([0 0 0.4])*eul2tform([0 0 0],'ZYX');
20    Tf = trvec2tform([initX initY maxH])*eul2tform([atan2(
          initY,initX) atan2(sqrt(initY^2+initX^2),maxH)/2 0],'
```

```
    ZYX');
[T,dT,ddT] = transformtraj(T0,Tf,tInterval,tTimes,'
    TimeScaling',[s;sd;sdd]);

plotTransforms(tform2trvec(T),tform2quat(T));
trajf2 = [tform2trvec(T) tform2quat(T)];

% Loading workspace definition
workspace = load('BravoWorkspace.mat'); %Loading the
    struct
WSAShape = workspace.shp; %Loading the shape

for i = 1:length(theta)
    xi = (a+b*theta(i))*cos(2*pi*theta(i));
    yi = (a+b*theta(i))*sin(2*pi*theta(i));
    Tfi = trvec2tform([xi yi maxH])*eul2tform([atan2(yi,
        xi) atan2(sqrt(yi^2+xi^2),maxH)/2 0],'ZYX');
    [Ti,dTi,ddTi] = transformtraj(Tf,Tfi,tInterval,tTimes
        ,'TimeScaling',[s;sd;sdd]);

    % Checking if point is inside workspace, moving them
        to closest
    % point if not.
    for j = 1:size(Ti,3)
        [nearIDX, tf] = checkPosTrans(WSAShape,Ti(:,:,j));
        if tf == false
            Ti(1,4,j) = WSAShape.Points(nearIDX,1);
            Ti(2,4,j) = WSAShape.Points(nearIDX,2);
            Ti(3,4,j) = WSAShape.Points(nearIDX,3);
        end
    end
    plotTransforms(tform2trvec(Ti),tform2quat(Ti));
    trajf2 = [trajf2 ; tform2trvec(Ti) tform2quat(Ti)];
    Tf = Tfi;
end
hold off;
```

Listing B.7: Path planning for feeding operations using *NearestNeighbor*. See B.1 for more details

```
nr = 1; %Number of points in between each waypoint that
    is set.
maxH = 0.3; %Max height possible for robot standing
yStart = 0.25;
xFin = 0.4;
theta = linspace(0,pi/2,15);
```

```
6
7     figure('Name','Trajectory Grab');
8     axis([-1 1 -1 1 -1 1]);
9     title('Waypoints for grabbing');
10    xlabel('X');
11    ylabel('Y');
12    zlabel('Z');
13    hold on;
14    grid on;
15
16    tTimes = linspace(0,1,nr);
17    tInterval = [0 5]; %Time interval
18    [s,sd,sdd] = trapveltraj([0 1],numel(tTimes));
19
20    initX = (xFin-0.1)*sin(theta(1));
21    initY = (yStart-0.1)*cos(theta(1));
22    initZ = (maxH)*cos(theta(1))+0.2;
23
24    T0 = trvec2tform([0 0 0.4])*eul2tform([0 0 0],'ZYX');
25    Tf = trvec2tform([initX initY initZ])*eul2tform([0 0 0],
          'ZYX');
26    [T,dT,ddT] = transformtraj(T0,Tf,tInterval,tTimes,'
          TimeScaling',[s;sd;sdd]);
27
28    plotTransforms(tform2trvec(T),tform2quat(T));
29    trajg = [tform2trvec(T) tform2quat(T)]; waypoints
30
31    xt = xFin.*sin(theta);
32    yt = yStart.*cos(theta);
33    zt = (maxH).*cos(theta)+0.2;
34
35    for i = 1:length(theta)
36          xi = xFin*sin(theta(i));
37          yi = yStart*cos(theta(i));
38          zi = (maxH)*cos(theta(i))+0.2;
39          Tfi = trvec2tform([xi yi zi])*eul2tform([0 theta(i
                ) 0],'ZYX');
40          [Ti,dTi,ddTi] = transformtraj(Tf,Tfi,tInterval,
                tTimes,'TimeScaling',[s;sd;sdd]);
41
42          for j = 1:size(Ti,3)
43             [nearIDX, tf] = checkPosTrans(WSAShape,Ti(:,:,j
                   ));
44             if tf == false
45                Ti(1,4,j) = WSAShape.Points(nearIDX,1);
```

```
                Ti(2,4,j) = WSAShape.Points(nearIDX,2);
                Ti(3,4,j) = WSAShape.Points(nearIDX,3);
            end
        end
        plotTransforms(tform2trvec(Ti),tform2quat(Ti));
        trajg = [trajg ; tform2trvec(Ti) tform2quat(Ti)];
        Tf = Tfi;
    end
    hold off;
```

Listing B.8: Path planning for grasping operation. See B.1 for more details

Appendix C

C++ code

This part of the appendix presents code excerpts from the created C++ scripts that is used for programmatic control of the MoveGroup Node to control the robot during runtime. This includes description of header, implementation and run files. For full code, please see attached zip file.

```
#include "cstring"
#include "geometry_msgs/Pose.h"
#include "moveit/move_group_interface/move_group_interface.
    h"
#include "moveit/planning_scene_interface/
    planning_scene_interface.h"
#include "moveit_msgs/MoveItErrorCodes.h"
#include "string"
#include "vector"
#include "ros/duration.h"
#include "ros/ros.h"
#include "ros/time.h"

namespace r7_planning
{
class PlanningClass
    {
    public:
      PlanningClass() : move_group(planningGroup)
      {
        move_group.allowReplanning(true);
        move_group.setNumPlanningAttempts(10);
        move_group.setPoseReferenceFrame("base_link");
      }
```

```
23      void goToPoseGoal(geometry_msgs::Pose &pose);
24      void cartesianPath();
25    private:
26      const std::string planningGroup = "arm";
27      moveit::planning_interface::MoveGroupInterface
            move_group;
28      moveit::planning_interface::PlanningSceneInterface
            virtual_world;
29      moveit::planning_interface::MoveGroupInterface::Plan
            my_plan;
30    };
31  }
```

Listing C.1: Initial header file for the generic manipulator to test programmatic control concepts. This file describes the initial created class for controlling a spawned MoveGroup Node. Here, a planning group is initiated for cummincation with the active MoveGroup Node and some parameters for the planner is set. Further, a couple of methods (or functions) are defined for programmatically moving to a given Pose pointer or to run a pre-uploaded cartesian path from a csv file. The private variables are used for initiation of the MoveGroup Node, or representing the node itself, the planning interface or the plan inside the MoveGroup Node.

```
1   #include "planning.hpp"
2   #include "iostream"
3
4   using namespace r7_planning;
5
6   void PlanningClass::goToPoseGoal(geometry_msgs::Pose &pose)
7   {
8    move_group.setPoseTarget(pose);
9    bool success =
10       (move_group.plan(my_plan) == moveit_msgs::
             MoveItErrorCodes::SUCCESS);
11   if (!success)
12     throw std::runtime_error("No plan found");
13   move_group.execute();
14  }
15
16  void PlanningClass::cartesianPath()
17  {
18   std::vector<geometry_msgs::Pose> waypoints;
19   move_group.setGoalOrientationTolerance(0.5);
20   move_group.setGoalPositionTolerance(0.2);
21   move_group.setMaxVelocityScalingFactor(0.5);
22
23   geometry_msgs::Pose target_pose;
24   target_pose.position.x = 0.3;
```

```
25  target_pose.position.y = 0.3;
26  target_pose.position.z = 0.1;
27  target_pose.orientation.w = 0.001;
28  target_pose.orientation.x = 0.0;
29  target_pose.orientation.y = 1.0;
30  target_pose.orientation.z = 0.0;
31  waypoints.push_back(target_pose);
32
33  // Reading information from csv file
34  std::ifstream myFile;
35  myFile.open("/home/user/catkin_ws/src/motion_planning
36  /src/trajectory_gen.csv");
37  std::string px, py, pz, ow, ox, oy, oz;
38
39  if (!myFile.is_open())
40  {
41    ROS_WARN("ERROR: File Open");
42  }
43
44  while (myFile.good())
45  {
46    geometry_msgs::Pose wp;
47    std::getline(myFile, px, ',');
48    std::getline(myFile, py, ',');
49    std::getline(myFile, pz, ',');
50    std::getline(myFile, ow, ',');
51    std::getline(myFile, ox, ',');
52    std::getline(myFile, oy, ',');
53    std::getline(myFile, oz, '\n');
54
55    wp.position.x = atof(px.c_str());
56    wp.position.y = atof(py.c_str());
57    wp.position.z = atof(pz.c_str());
58    wp.orientation.w = atof(ow.c_str());
59    wp.orientation.x = atof(ox.c_str());
60    wp.orientation.y = atof(oy.c_str());
61    wp.orientation.z = atof(oz.c_str());
62    waypoints.push_back(wp);
63  }
64
65  moveit_msgs::RobotTrajectory trajectory;
66  const double jump_threshold = 0.0;
67  const double eef_step = 0.05;
68  double fraction = move_group.computeCartesianPath(
      waypoints, eef_step,jump_threshold, trajectory);
```

```
  ROS_INFO_STREAM("Percentage of path followed: " << 100 *
      fraction);
  sleep(5);
  my_plan.trajectory_ = trajectory;

  if (fraction == 1.0) {
    move_group.execute(my_plan);
    ROS_INFO("All good, 100 percent of path was created");
  } else {
    ROS_WARN("Could not compute the cartesian path :( ");}
  ros::shutdown();
}
```

Listing C.2: This is the implementation file corresponding to the header file in C.1. Here, the functionalities of the presented methods in the header file is implemented. For the goToPoseGoal function, this involves setting the pose as a target, plan a trajectory to obtain this pose if possible and finally execute this motion if fulfilled. The next function uses the *computeCartesianPath* function of the MoveGroup to calculate a trajectory fulfilling all waypoints that are read from an input csv file. Finally, this trajectory, if completed, is executed.

```
#include "planning.hpp"

int main(int argc, char **argv) {
  using namespace r7_planning;

  ros::init(argc, argv, "custom_interfacing");
  ros::NodeHandle node_handle;
  ros::AsyncSpinner spinner(2);
  spinner.start();

  if (argc != 2) {
    ROS_INFO(" ");
    ROS_INFO("\tUsage:");
    ROS_INFO(" ");
    ROS_INFO("\trosrun planning run n");
    return 1;
  }

    r7_planning::PlanningClass my_plannings;

    geometry_msgs::Pose P1;
    P1.position.x = 0.0;
    P1.position.y = 0.4;
    P1.position.z = 0.2;
    P1.orientation.w = 0.001;
    P1.orientation.x = 0;
```

```
    P1.orientation.y = 1.0;
    P1.orientation.z = 0;

    int selection = atoi(argv[1]);
    switch (selection)
    {
      case 1:
        my_plannings.goToPoseGoal(P1);
        break;
      case 2:
        my_plannings.cartesianPath();
        break;
     }
      spinner.stop();
      return 0;
}
```

Listing C.3: This is the main file for running the initial code for the generic manipulator. An asynchronous spinner is initiated for handling multiple topics at the same time. The input arguments from the used shell is then used to choose which functions to be run in the switch such that multiple tests can be done without recompilation of the motion planning package is unnecessary.

```

namespace bravo_planning {
class PlanningClass {
   public:
     PlanningClass() : move_group(planningGroup) {

       move_group.allowReplanning(true);
       move_group.setNumPlanningAttempts(3);
       move_group.setPlanningTime(3);
       move_group.setPoseReferenceFrame("base_link");
       move_group.setGoalOrientationTolerance(0.05);
       move_group.setGoalPositionTolerance(0.05);
       move_group.setMaxVelocityScalingFactor(0.8);
       move_group.setPlannerId("RRTkConfigDefault");
      }

      void goToPoseGoal(geometry_msgs::Pose &pose);
      void cartesianPath();
      void cartesianPathCsv(std::string &fil);
      void goToNamedGoal(std::string &name);
      void runPosesCsv(std::string &fil, int interval);
      void runPoseCsv(std::string &fil, int which);
      void jointsCSV(std::string &fil);
      void getPose();
```

```
25
26      private:
27        const std::string planningGroup = "arm";
28
29        moveit::planning_interface::MoveGroupInterface
              move_group;
30
31        moveit::planning_interface::PlanningSceneInterface
              virtual_world;
32
33        moveit::planning_interface::MoveGroupInterface::Plan
              my_plan;
34
35        moveit_msgs::RobotTrajectory trajectory;
36
37        // Other parameters used in Cartesian path planner etc.
38        const double jump_threshold = 0.0;
39        const double eef_step = 0.01; //10cm
40        bool avoid_collisions = false;
41      };
42  }
```

Listing C.4: Final header file that follows the same approach as for the generic manipulator only increased usability by making the functions more generic for waypoint validation. Some more parameters for the MoveGroup Node is also set initially for better performance of the planner interface. For implementation files, please the the attached zip file.

Appendix D

Contents of digital attachment

The zipped digital attachment contains the following materials:

- CAD files for generic robot and URDF assemblies including STL files of the Bravo and generic manipulator.
- Dimensions and other illustrations of the two manipulators
- The ROSject for each of the two manipulators including described parts such as ROS setup, MoveIt package and C++ files.
- All Matlab code.

Appendix E

Interview with Artec Aqua

Interview conducted with representatives from Artec Aqua at their main office in Ålesund, 27^{th} of May 2020. Artec Aqua is a large and fast-growing contractor within land-based fish farming and is today the contractor of the facility of the Autosmolt2025 partner, Salmon Evolution at Harøya that is under construction. Artec Aqua has since 2010 been a total contractor, hence leading projects through the use of suppliers within their specialised fields such as production, construction, assembly, automation and so on. Therefore, they are in a unique position for mapping the supplier industry of its most novel solutions. The interview was conducted using discussion as an approach since it was aimed at giving an overall view of the general land-based fish farming facilities and operations, and not in a survey manner. The interview has been transcribed into relevant questions and answers and has been filtered such that no sensitive information will be published. The answers should be interpreted as general meaning that there will undoubtedly be some deviations in both operating or future planned facilities.

Interview objects:
Magnus Seth, Projectleader process
Inge Finnes Saunes, Project procurement

1. Q: How is the general land-based fish farming facility designed? What sizes are the tanks dedicated to the different stages in the life cycle of the fish?

Most of our new facilities are centrally controlled, not always a control room but centralised control from, i.e. an Ipad, hence all control and monitoring of temperature, oxygen, light, water speed, feed etc. can be controlled even remotely. Each tank also has often its own control panel or pad of sorts where more local controls are done. As of facility design, most facilities are composed of separated sections based on the stage in life cycle such as hatching, most juvenile stage and the last grow-out stage, but the exact composition may vary. The sections typically consist of large and clean industrial areas (hall) with tanks, pipes, hoses, and other essential equipment placed reasonably. If possible, it is tried to keep the path from tank to ceiling clean of objects to mitigate unnecessary threats to

fish welfare such as falling object or leakages.

When it comes to tank and tank design, these vary much based on application and customer desire/needs even though most of our projects are using circular-shaped tanks. For a smolt production facility growing fish from hatching to about 200 grams the following is typical: hatching in cabinets, Ø5 meters tank at start feeding, Ø8 meters for 3-20 grams, Ø10-15 meters for further growth phase up until 2-300 grams. The tanks can, of course, be bigger such as Ø28 meters for post-smolt up to 1.5 kg. Height and shape vary based on customer desires.

2. Q: How is feeding typically done today? What is the level of automation? Could automation improve it?

Feeding solutions can vary between facilities, but most new facilities are operating with a centralised feeding system. These systems are using a central silo connected to a feeding station through ducts where a funnel/dispenser is used together with one or multiple feeding screws for reaching out to a large cross-section of the tanks. The number of feeding screws connected typically depends on tank size and the feeding screws are typically feeding at $2/3$ of the radius towards the tank wall. The actual filling of the funnel/dispenser are some places done manually to date, but the centralised systems work pretty well. As for improvement, it is unlikely that any new robotic solution could improve this dramatically since it is already working quite well. Some state of the art facilities have connected the feeding to a centralised computer system that determines the feed based on size, biomass, water, desired growth rate and so on.

3. Q: How is dead fish removal done today? What is the level of automation? Could automation improve it?

Dead fish is typically descending in water as long as it has not occurred a mass death which happens rarely but is a great disaster. The mortality rate is generally low in most new operating facilities. However, most of these are FTS or FTS-R (Flow-Through System with partial Recycling) due to costly investing related to full RAS facilities. Nevertheless, new facilities have typically installed an automatic collection of dead fish in a centre placed strainer that transports them into a container placed at the tank side for easy inspection and removal. Artec Aqua's solution is called *FishTrap* and transports the trapped fish, dead or alive to the topside container using pumps where the alive fish can easily find their way back to the tank of origin by swimming. The removal and inspection of this container is done manually by the fish farmers but is usually conducted as part of their quality checking and inspection routine, which is implemented for monitoring fish welfare. It is not likely that this operation could be automated easily since a lot of the daily tasks of the fish farmers include extensive inspection of the dead fish and not just removal.

4. Q: How is the cleaning of tanks done today? What is the level of automation? Could automation improve it?

Cleaning of tanks is typically done manually by one or several operators using pressurised water, detergent and disinfection in between cycles/batches. This could happen 3-4 times a year based on the number of batches, transportation in between tanks, water problems or other reasons. The cleaning is an important task and is mainly being done to reduce

biological risks regarding strains or pathogens that the previous batch might be the origin of, hence reduce potential infection. If the tank hydraulics are designed and controlled as intended, there should be no need for cleaning while there is biomass inside the tanks even though there might stick some waste/sludge to the tank walls at the water level. This task is not a fun task for any operator, working close to detergents and disinfectants so surely room for automation in this regard. However, some industrial solutions are using ceiling-mounted nozzles, such as solutions from HL. Skjong but these are often viewed as too expensive of an investment by most fish farmers. A critical note on the tanks cleaning part is the criticality of not scratching or harming the tank surfaces. One minor scratch in the coating layer can result in fatal results such as tear and wear on the fish if it becomes a sharp edge, and most critically such scratches can quickly become the home for strains or pathogens that will be hard to get rid off. Cleaning of pipes is also a task where autonomous systems might be the right solution.

5. Q: What is a possible bottleneck for implementation of new and novel technology?

RAS technology is not new, but there are still considerable challenges with the actual implementation with regards to biological control resulting in a need for costly expertise within the field, often requiring PhDs. Also, fish farmers are conscious of their spending, especially within uncertain novel technologies that have yet to be proved as worthy investments.

6. Q: What is the most state-of-the-art facility you have been part of?

That has to be Salmobreed Salten which was constructed in 2018. The facility is a high tech broodstock facility delivering roe to customers in Norway and worldwide. The facility has implemented all the latest industrial solutions within its respective fields and utilises, among other things FTS, FTS-R and RAS for fresh- brackish- and saltwater.

7. Q: What do you think will be the most significant technological enhancements happening within the industry the coming year?

Improved solutions (hopefully one complete solution) for biological control using machine learning and expertise to automatically monitor water quality and all necessary parameters at different places in the water cycle in real-time, and remotely operate and analyse the data remotely, hence removing the complex and costly work from the shoulders of the fish farmers and facilitate for increased RAS use and fish welfare. The combination of expertise and continuous learning through the use of machine learning might make it possible to detect and regulate for problems early on, and ultimately predict future problems that might occur. Blue unit has an impressive data platform for water quality monitoring that is used in the industry, but it is not flawless.

8. Q: General discussion about fish farming.

Fish farming, or especially salmon farming is a vulnerable industry that can easily be affected by global trends such as falling consumption in China, Japan, EU, US etc. due to a variety of reasons such as political challenges, environmentalist campaigns or other reasons beyond the producers' control. When it comes to the increasing efforts within land-based full-scale salmon farming using exclusively RAS facilities, this is not seen necessarily as a great problem or challenge in the industry due to the high seal of approval

worldwide for the atlantic salmon farmed in its right habitat, namely in the nutrient-rich ocean of the Northern Hemisphere. Either way it has been an increasing trend in recent years that the MAB in different municipalities has been decreased due to biological risks of yellow or red declared areas (as of the traffic light regime). This has led to a lot of dissatisfaction, overstepping and fines in the industry and is one of the reasons for increased activity/investments for both offshore and land-based salmon farming.

Appendix F

Converting CAD to URDF using SolidWorks plug-in

The use of the SW2URDF is quite straight forward but is a critical part for further simulations and will therefore be explained in detail. After the plug-in has been installed, its functionality can be found right beneath the save button found under the file link in the top menu of SW. When the function is started, the model is re-built and the menu as seen in fig. F.1 spawns. The menu is intuitive and as long as the model has been created and mated in a sufficient manner, the creation of the URDF file can be done fast using this tool. An important note before delving into the setup is to move the assembly into a "known" position, a home configuration, since the exported robot configuration will ultimately be the spawning state of the robot in simulations. To start off, the base has to be defined. This is the part of the manipulator that is mounted to either the ground or other mobile robots if desired. Make sure to choose the parts as found in the model tree and not choose features by clicking directly on the manipulator in the visualisation window. The chosen parts will be highlighted in the model window. Further, it is possible to define child parts of this link, that is the next link. When this dependency has been defined, a joint in between these links are automatically created. The link names and joint names can be described as desired. In this thesis the following has been used as standard; base_link for the base of the manipulator, link X for the numbered link starting from 1, and Joint(X-1)X describing the joint in between the previous link and the next. It is important to notice that it is possible to define multiple parts for each of the links. This results in the use of all available parts to describe the full robotic system by the complete links and joints in between them. This can be seen in fig. F.2 where the three first parts of the assembly has been chosen in the model tree to define the base link. In addition, a child link has been defined as link1.

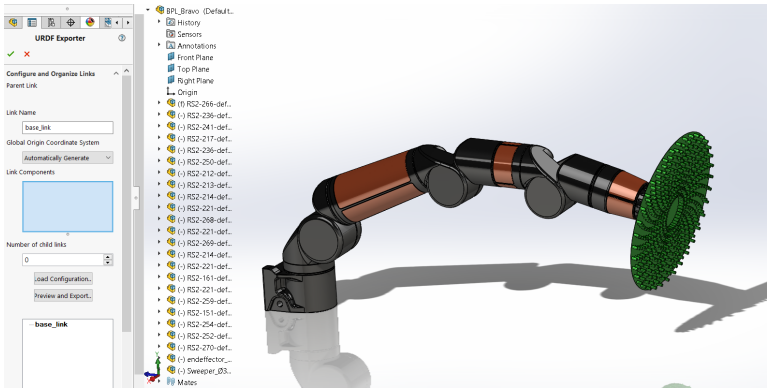


Figure F.1: Initial setup assistant for SolidWorks to URDF plug-in for the Bravo manipulator

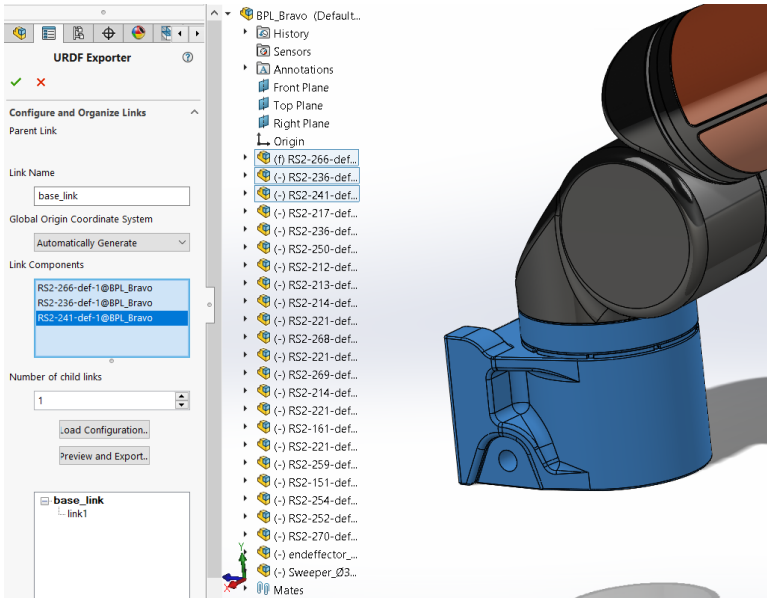


Figure F.2: SW2URDF definition of base for the Bravo manipulator

Moving on to the link1 in the "link tree" of the SW2URDF plug-in, more choices appear. For this and the remaining links there must be defined and desribed a joint in between the parent and child link. The plug-in uses the mates of the parts from the assembly in SW to automatically detect and generate the reference coordinate systems, axes and joint types. These can be directly specified by the user themselves, but the plug-in works sufficiently to do this by itself. It is also possible to define this further down the road of the plug-in or after the software has exported it into URDF descriptions. This auto generation should always be reviewed, especially with regards to the generated coordinate systems that is auto generated. If specific descriptions such as using screw theory (i.e. rotations around z

axis) is desired, then this should be changed at this step. As seen in fig. F.3, a joint name has been defined, the link components has been chosen in the model tree, a new child link (link2) has been defined and the rest of the configuration is set to automatic. Due to the mating defined in the assembly, that is, the circular parts has been concentric constrained and a coincident constrain defined such that the parts are connected there exist one degree of freedom in the existing mating between the chosen parts. This is used used by the plug-in to create a continuous joint as default. A continuous joint is a revolute joint without limits. This can easily be changed to be defined as a revolute joint in the final step of the plug-in, or redefining it in the configuration. There exist 6 different joint configurations in the URDF "language" [105] but for robotic manipulators it is normal to define the system based on only revolute joints as mentioned in 2.1 and therefore these other types will not be further mentioned. The same procedure as defined here applies for the rest of the links including the end-effector. Control of specific end-effector tools are more complex and is not a part of this thesis. It was therefore decided to use controls from the base link and up until link 6 and attach "dummy" end effectors for visualising different operations.

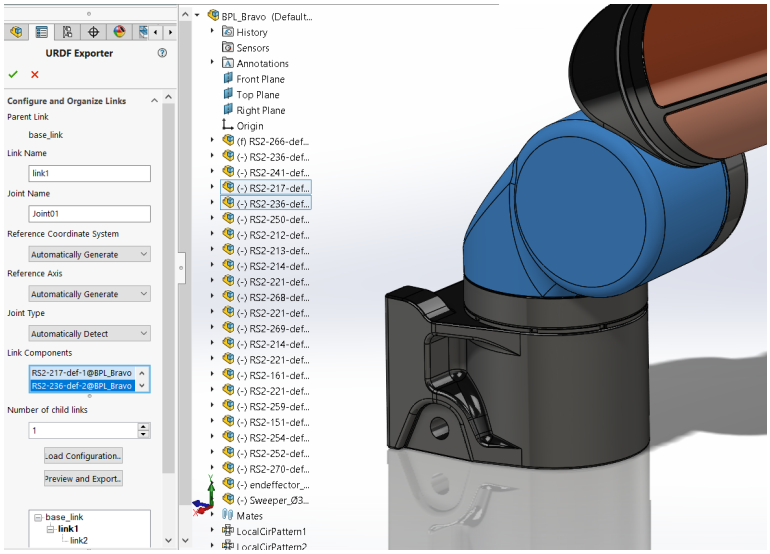


Figure F.3: Configuration of joints

Finishing up with the last definition of the link (i.e. the end-effector), clicking the preview and export will automatically create the origin and centre axis of the joints and using the configurator for joint properties as seen in fig. F.4, each joint can be more specifically defined characteristics. Even though the user does not know the exact dynamics and limits of each joint it is recommended to assign numbers to these variables. This is an advantage since the definition of these properties are strictly necessary for the conversion between URDF and SDF file to be able to place the robotic system in a Gazebo simulation environment and not just the visualisation of the robotic system directly using the basic URDF's. These values can therefore be set equal to 1 or similar just to automatically create the properties in the xml files that the plug-in is configuring. The joint configurations can be

completed by defining the joint types, and setting initial limits and dynamics. The last step of the plug-in is to redefine the link properties if they are clearly wrong or the user wants to specify these as seen in fig. F.5. Since the software is using the assembly to generate all of the properties, they might be quite wrong based on possible errors of material, volume and position of planes. Nevertheless, the auto generated properties are sufficient for simple simulation purposes.

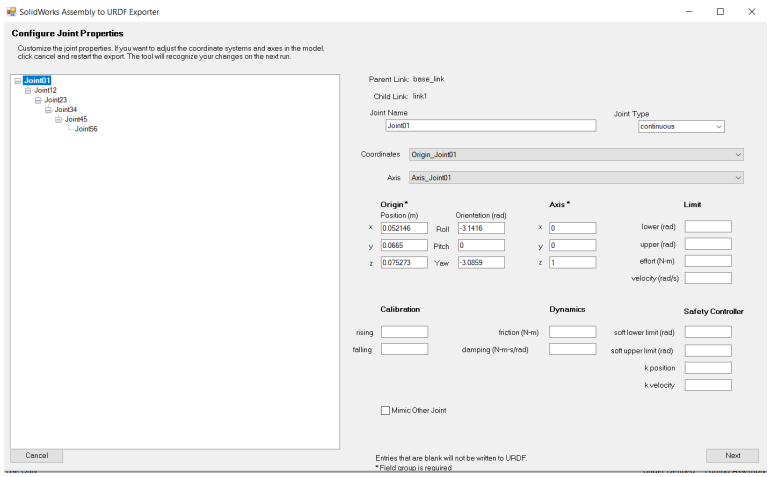


Figure F.4: SW2URDF joints configuration

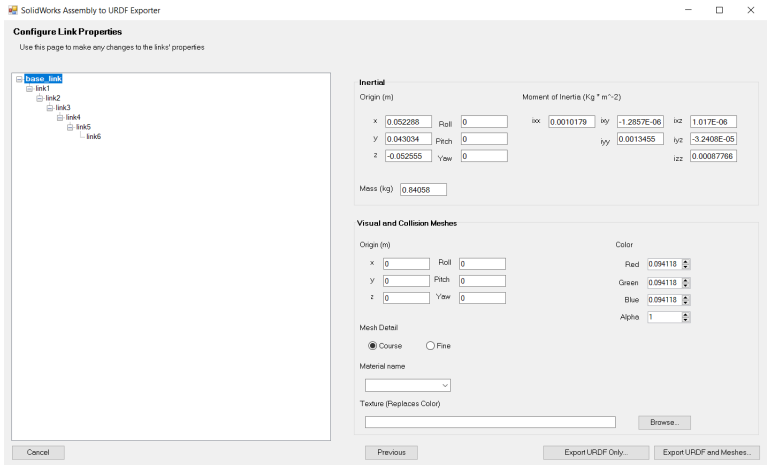


Figure F.5: SW2URDF links configuration

Finally, the URDF can be exported to desired location on the computer. The plug-in then creates folders for all necessary URDF files to be able to implement the created robotic manipulator in a simulation environment based on ROS, either the RViz for simple visu-

alisation of the robot or Gazebo for simulating the manipulator in an environment. The folders consist of a config folder for the definition of the joints, a launch folder for creating and spawning model in simulation environment, a folder for the meshes of the defined links (STL files), a folder for textures (pictures to be wrapped around the parts) which is seldom used, the most important URDF folder and file where the whole robotic system is defined (XML format) and finally two xml files for defining dependencies etc. which are named CMakeLists and export. When importing the manipulator into the desired simulation software (here the ROSDS), it is mainly the meshe files and the URDF description that is recommended to use directly. This is due to the rapid patches that is being done in the ROS simulation community which can lead to deprecated code in the remaining folders or just the fact that different simulation environments uses different approaches for configuring and implementing created robotic systems. The final configuration of the Bravo manipulator in SW can be seen in fig. F.6.

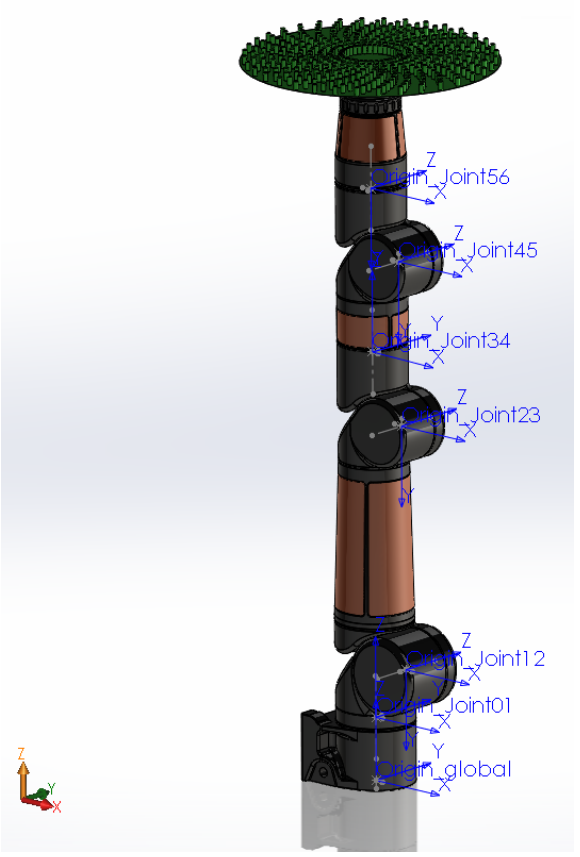


Figure F.6: Completed SW2URDF setup with the respective defined joint coordinate systems for the Bravo manipulator