UNIVERSITY OF AGDER

Modeling of an Underwater Robotic Manipulator With a Customized End-Effector for Removal of Derelict Pots

Tahiraj, Mergim
Børve, Åsmund

Supervisors
Filippo Sanfilippo
Damiano Padovani

This project is carried out as a part of the education at the University of Agder and is therefore approved as a part of this education. However, this does not imply that the University answers for the methods that are used or the conclusions that are drawn.

University of Agder, 2020
Faculty of Engineering and Science
Department of Engineering Sciences

Preface

This work is submitted as a master thesis at the Department of Engineering Sciences at the University of Agder (UiA). The thesis is a part of our Master of Science (MSc) degree. This project thesis was carried out during the spring semester of 2020, and is entirely innovative, which means it is not a continuation of any project thesis written in the fall of 2019.

This thesis is motivated by the demand for gathering marine litter, more specifically derelict pots, to avoid utilizing human divers, and a great desire is to increase the level of autonomy in subsea operations. This thesis is performed in conjunction with the Institute of Marine Research (Havforskningsinstituttet).

Acknowledgments

The past five months have been an exciting journey, dedicated to the last step in finishing a master's degree in Mechatronics Engineering at the University of Agder (UiA). The group members would like to express sincere gratitude to those who helped during this period.

A special thanks to the supervisors Filippo Sanfilippo and Damiano Padovani, for their guidance throughout the research project. Their feedback helped to structure and improve this thesis and to continue the project at tough points.

Our gratitude goes to Jan Christian Strandene as well, an employee at UiA, for contributing and helping with the manufacturing of unique parts.

Finally, we would like to express our gratitude to the people dearest to us. We want to thank parents and family for their love, support, and wisdom throughout the life.

Mergim & Åsmund

Abstract

This thesis investigates the opportunity of generating complementary technology devices for existing remotely operated vehicles (ROVs) so that abandoned material can be collected and retrieved, particularly from the bottom of the ocean.

According to World Wide Fund for Nature (WWF), over eight million tons of plastic end up in the sea each year [13]. Furthermore, they claim that based on a survey of Northern fulmars in the North Sea, as many as 9 out of 10 birds have plastic in their stomachs. However, plastic is not the only issue regarding marine litter. A survey from the Institute of Marine Research shows that accessories findings reported by divers in the bottom of the ocean contain a significant number of derelict pots in addition to many other objects, which is a result of ghost fishing [37]. This issue is given a high priority in this report.

The steps of this report were first to investigate existing solutions for deep-sea robotic manipulation. This paper will aim its focus on the development of an underwater robot arm, which is to be mounted on a ROV. The center of interest will be only on the arm and not the ROV. The robot arm's main task is to attach a custom-made end-effector (containing a carabiner and a balloon inside) unto derelict pots and then activate the balloon such that the derelict pots are lifted up to the sea surface. This prototype was intended to be actuated by seawater.
This project has been done in cooperation with Alf Ring Kleiven, a scientist from the Institute of Marine Research. Kleiven presented their issues on a meeting, and the project could finally be initiated with understandable and clear requirements. Based on their needs, the group could design and generate a proposed robot arm with a custom-made end-effector such that the prototype would satisfy their requirements. If their needs were able to be solved, the project would be considered a success.

Contents

.

List of Figures

List of Tables

Acronyms

ROV	Remotely Operated Vehicle
ROS	Robot Operating System
FOAM	Fluid-driven Origami-inspired Artificial Muscle
PAM	Pneumatic Artificial Muscle
DOF	Degrees of Freedom
HSE	Health, Safety and Environment
SOFA	Simulation Open Framework Architecture

Chapter 1

Introduction

1.1 Background and Motivation

As discussed in the meeting with the Institute of Marine Research, ghost fishing is a global problem regarding entrapment of sea life by human-made fishing equipment after they are lost or abandoned. Loss of such equipment causes trapping and eventually death of sea life. This process may be further continued since the trapped species may act as bait for further catch and therefore continue the cycle to cause a significant loss of possible marine resources [20]. An estimate for blue crabs lost due to derelict fishing equipment in Louisiana amounts to 12 million crabs annually with an approximated value of 4 million USD [2]. Current methods to find and remove such equipment are limited to search by divers or removal by ROVs with an attached manipulator arm. These searches are considered a shot in the dark due to the limited sight under the sea surface and the different depths of which the equipment may be located.

1.2 State of the Art

The current state of the art methods to clear ghost fishing equipment will be presented in this section.

Removal by Divers

Divers manually locate and attach a lifting bag which is inflated by the oxygen tank carried by the divers. The bag lifts the equipment to the surface for it to be collected. This method is not suitable for deep dives as diving is related to HSE issues such as decompression sickness, arterial air embolism, nitrogen narcosis, and of course, drowning [6]. Other constraints are legislative, which is governing the depths which the divers may legally dive. For depths up to $50\,m$, the diver is required to have a diver's license of class B while operating a lifting bag [21]. Diving beyond $50\,m$ requires other licenses and severely increases the risk of mentioned HSE issues. Removing equipment by divers is therefore constricted by available personnel and poses a significant health risk for the divers. Fig. 1.1 illustrates how this method is done in reality.

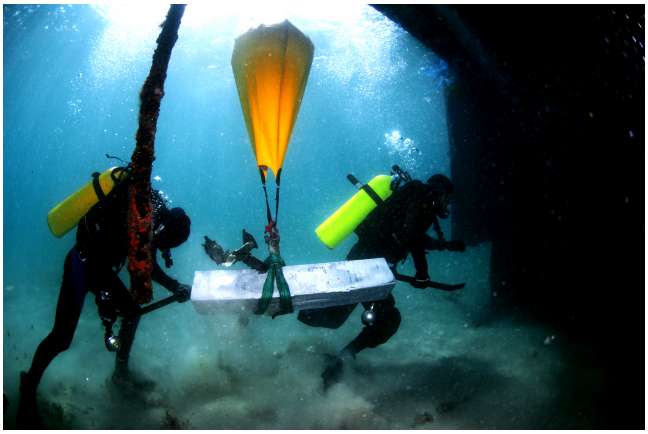


Figure 1.1: Divers using a lift bag to carry a payload [44]

Removal by ROV

The second method is performed by utilizing ROVs with a cutter and a gripping arm. Human operators manually operate them from an offshore vessel, with a low level of autonomy. This method is indeed related to a high cost, and the operations rely on both human performance/-experience and weather conditions. The operator maneuvering the ROV locates and brings the equipment to the surface by gripping and cutting (if necessary). A problem regarding this method is the available power since ROVs are powered by batteries with limited capacity. This issue reduces the number of dives between each search since the ROV must dive and ascend for each derelict equipment found. The ROV market is also expanding and becoming more available for amateur use (similar to airborne drones). This may open an opportunity for the removal of equipment by independent sources. Fig. 1.2 shows an illustration of this method.

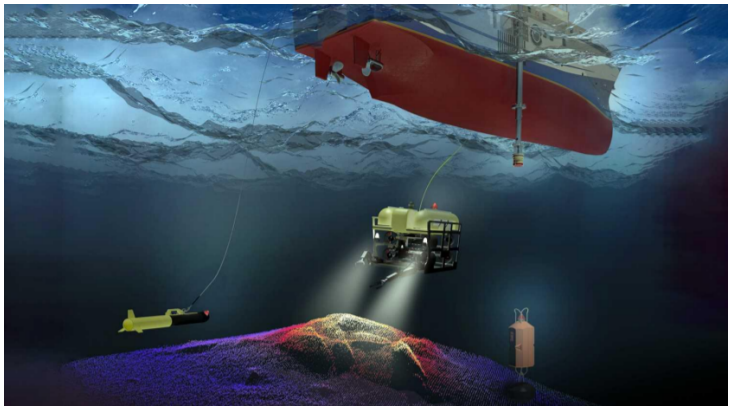


Figure 1.2: ROV detecting equipment [37]

The clearly defined requirements proposed by Kleiven, (the group's contact person and scientist from the Institute of Marine Research), aimed to utilize removal by ROV method with additional modifications. Their preference was to avoid lifting the ROV to the sea surface for each finding of marine litter. To maintain the ROV submerged at all times, elements were needed to be implemented on the ROV, which is the robot arm with a custom-made end-effector, and this will be introduced and explained more thoroughly later. Furthermore, it is needed to initiate movement of the derelict pots, so that only the derelict pots are ascending and not the ROV.

1.2.1 Design and Prototyping

A prototype is an early version of a new product that is presented for testing and demonstration of the product before further development. The new product can be a machine, instrument, or computer system. A prototype is often just a preliminary and simplified version of the new product, where the purpose is to demonstrate and test the features in realistic environments so that faults and weaknesses can be revealed. A prototype has a significant advantage in the design process, which is that it simulates the real and future product. It can aid in attracting customers to invest in the product before assigning any resources needed for implementation. Additionally, prototyping enables the ability to streamline the design development process, concentrating on essential interface factors, and at an early stage, unnecessary elements can be defined and indeed abandoned [23]. A prototype has economic advantages because it reduces the amount of work in improving the project, which contributes to saving, for instance, a customer's money. In this case, the customer is the Institute of Marine Research.

Simulation is an essential tool for fast prototyping, and a standard method to develop and function test a robotic system is to create a digital clone within a physics simulation. This is to ensure the functionality and design to be sufficient for the design case. Advantages by utilizing a digital clone are the ability to test and experiment with various algorithms/controllers in a safe environment. Simulation has a great advantage concerning economics since it offers the ability to develop new models at a lower cost compared to creating multiple physical prototypes. Damages can also be avoided with a digital version.

There are multiple platforms for simulating real-time applications. One aspect which may divide these into two main brackets is; Does it require a license? Examples that require a license are MATLAB and SimulationX. On the contrary are frameworks like SOFA and ROS. Both are open-source, which means they are maintained and developed by their respective communities. SOFA was initially built to simulate medical procedures but has also proved to be suited for soft robotics [41]. ROS is an open-source framework for designing and programming robots [29]. The field of robotics is a wide field of study, with many researchers working on their respective projects. The advantage of the ROS framework is that methods and shortcuts are all gathered in one framework to help aid robotics projects for all of its users.

Gazebo is also an open-source simulator of a physical environment and is widely used in conjunction with ROS due to its customizable environment and the ability to create customized plug-ins. A plug-in is a part of a software that serves a feature and can be added to an existing program. It was decided upon to use ROS with Gazebo and MATLAB in this project due to the growing use on the university and an available license provided by the University of Agder for MATLAB.Although ROS was chosen for this project, SOFA could prove to be better suited since it already has implemented fluid dynamics [41]. It is, however, difficult to know the strengths of each framework without experience.

1.2.2 Selection of Actuators

An important aspect of designing the manipulator is the means of actuation. Conventional methods are by electric rotational drives or by fluid-driven actuators, both linear and rotational. An article from Joseph Coleman et al. [34] reviews the commercialized manipulators and how they are actuated in great detail. However, since the budget of this project is in the lower end, soft actuators emerged as a potential candidate due to the low cost of production and excellent force-to-weight ratios. This led to the investigation of pneumatic artificial muscles (PAMs) as

actuators. PAMs are contractile or extensional devices operated by pressurized air filling a pneumatic bladder. Although the muscle is primarily pneumatically operated, there is nothing that prevents the technology from being hydraulically operated as well [5]. Since seawater was supposed to be utilized and not air, the group made two muscles for testing purposes; one with air and the other with water operation. The pneumatic muscle was tested with a compressor and functioned as intended. When the second was made, the group encountered issues due to low water pressure and a leaking connection. This problem was something the group thought could be fixed by utilizing a pump with a high-pressure source and low flow rate. Nevertheless, selecting a suitable pump for this muscle proved more troublesome than expected. The group discovered fluid-driven origami-inspired artificial muscles (FOAMs). Tab. 1.1 shows the different artificial muscles compared to each other in different fields, where FOAMs dominate in each aspect.

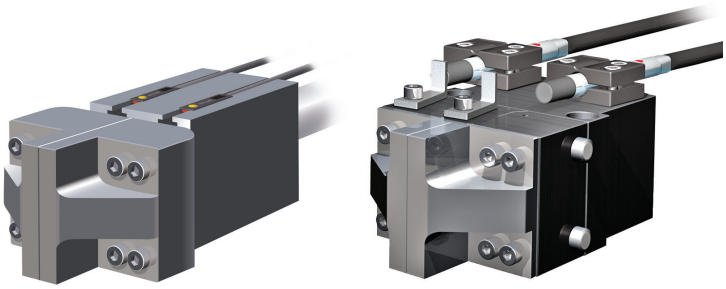
	FOAMs	**McKibben**	**PAMs**
Cost estimate	<$1	<$1	Unknown
Programmability	Multi-DoF	1D contraction	1D contraction
Max force-to-weight ratio	>10 kN/kg	8 kN/kg	>50 kN/kg
Operation pressure	Negative and positive	High>100 kPa	Low<10 kPa
Advantages	Safe, large-contraction	Simple, powerful	Powerful, large-contraction
Limitations	Buckling, skeleton failure	Small ratio	Bulky structure

Table 1.1: Comparison of different fluid-driven artificial muscles [38]

1.2.3 Selection of Sensors

Different sensors are required to be installed and implemented on the robot arm to design and generate a physical prototype. Such are **encoders**, which should be mounted in the revolute joints, and for each incrementation of the links, the encoder will provide feedback, and the pose of the arm can be estimated. However, it is not realistic that the location of the derelict pot is always known. For that reason, an idea might be to implement a **collision detection sensor**. This kind of sensor can have different forms. Some sensors can serve as a tactile recognition system, where if pressure is sensed on a surface, the robot will receive a signal to limit or stop its motions [4]. In this case, the sensor might be a button that sends binary numbers (0 & 1) to the robot's interface as feedback. Whenever the carabiner has opened, the number 1, for instance, should be displayed (meaning: the carabiner is now opened), and then, after a short time, the number 0 should be displayed, i.e., stop the motion because the end-effector is inserted and the carabiner is closed. The operation should not initiate before it is certain that the end-effector is completely attached to the derelict pot, and the carabiner has closed.

If a gripper is to be utilized, a sensor providing feedback whenever the jaws in the gripper are in contact would be of great importance. There are two options for this task: **magnetic field sensors** and **inductive proximity sensors**. One of the primary differences between these is the detection method that each option utilizes. The first mentioned uses an indirect method by observing the mechanism that moves the jaws, not the jaws themselves. The second one, however, uses a direct method that observes the jaws by detecting objects placed directly in the jaws. The two options mentioned can be seen in Fig. 1.3a, and Fig. 1.3b.



(a) Magnetic (b) Inductive

Figure 1.3: Robot gripper with magnetic field sensors and inductive proximity sensors [12]

Furthermore, each option has both advantages and disadvantages. If the magnetic option is utilized, it is possible to detect, for instance, remarkably small piston strokes if placed directly into extruded slots on the outside of an actuator. However, this option involves a magnet to be placed in the piston, which requires that the cylinder walls are nonmagnetic. The inductive proximity sensors enable the cylinder to be composed of any material without requiring magnets to be placed. Nevertheless, proximity sensors need more space for installation, longer setup time, and have other variables to consider. The basic sensors for robot grippers shown in Fig. 1.3a and Fig. 1.3b are employed whenever hard robotics is sufficient. This is not always desired. When operating with fragile objects, hard robotics may be an issue due to its material properties. This is where **soft robotics** plays a significant role, more of this in Sec. 7.1.

1.3 Objectives

This research aims to clarify and propose structure/suggestions for the marine litter market to achieve a product that the Institute of Marine Research can utilize in their work. This work will aim to aid the gathering of marine litter by proposing a low-cost manipulator system. A short presentation of FOAMs [38] is also presented as a possible candidate for a linear actuator to accommodate the low-cost system.

The group had its primary intention to develop a physical and compact prototype. However, due to the unexpected COVID-19 pandemic, many limitations occurred along with several changes concerning the project.

1.4 Research Questions

This thesis's objective is to investigate further and develop a complementary technology device for solving the marine litter problem. To achieve this, the research questions listed below have been identified:

1. Can we design and control a low-cost and easy-to-manufacture robotic arm, which is to be mounted on a ROV?
2. Can we design a custom-made end-effector that can be utilized for retrieving marine litter?
3. Can we create a digital clone that showcases the proposed design, and simulate its operation?

Chapter 2

Theory and Tools

This chapter will present the theory and components used in this project for the design of the end-effector, the overall system and the simulation. Furthermore, to design the manipulator successfully, requisite considerations and calculations will also be introduced.

2.1 Requirements

The prototype is required to:

- Pick derelict pot(s) at a predetermined and particular location inside a water tank.
- Have at least 1 DOF
- Contain actuator(s) driven by seawater

The requirements presented above were originally intended for the prototype, which was a robot arm mounted on a frame in a water tank (Sec. 2.6). Since a ROV would act as a free and movable base for the manipulator, it was decided that 1 DOF could prove sufficient. As a consequence, the arm was only thought of having 1 DOF. However, since the focus of the project turned to focus more on simulation, the group decided to add more DOFs. The location of the derelict pots is known in the simulation.

Again, the initial idea was to have the actuators driven by seawater. This could help reduce the actuator's weight since the surrounding seawater would act as a tank with the necessary fluid, unlike mineral oil-driven systems, which would need an extra tank to separate the working fluid from the environment. Another advantage would be if leakage occurred. Seawater leakage would have had no impact on the environment. However, after investigations and a phone call with Asker Hydraulikk, the group was told that they were not possible to get in Norway, and most hydraulic cylinders were not designed to be driven by seawater. To allow for seawater, they had to be customized in, for instance, England. Since the budget of the project is in the lower end, this ruled out readily available seawater actuators as a means of actuation. Thus leading to the investigation of the possibility to utilize a soft robotic actuator.

2.2 Product Development Theory

Product development methods are imperative, and applying these methods correctly is key for designing and fabricating a product in the best manner. Product development is an essential aspect of creating something innovative or improving something existing. This section will introduce the methods the group has found to be relevant.

2.2.1 Function-Means Tree

What it is

The function-means tree is an organizational and visual decomposition of a product's main function, divided into several layers of sub-functions. For each sub-function is the suggested means to fulfill these. It is organized as a folder structure with a top-down focus [18].

Why apply it

A function-means tree is a tool for concept generation with a top-down approach to the design. It is a creative process that can be compared to the approach of the morphological chart. A well-defined tree will have several means for each function and several concepts for the solution [18].

How to apply it

The product to be designed has a main function. This function will be the basis from which one will partition the sub-functions and corresponding means into sub-folders. For instance, a robot can be a means for the function of strawberry picking. This robot requires several functions to operate and to do its job. Functions such as locomotion, storage, power, and navigation are all means to be fulfilled and may further have sub-functions with their means. A summary of the function-means tree would be an enabler for creativity with a "top-down" focused design.

2.2.2 Morphological Chart

What it is

A morphological chart is a visual organization of ideas for product development in regards to solving a product's sub-functions. Included in the chart are the product's drawing of means paired with different ideas on which to realize these functions generated by the developers [24].

Why apply it

A morphological chart is an excellent tool for product development concerning concept generation. It is a way to give creativity a vessel on which it can unfold itself. Often may innovating ideas spring to life during a session with the morphological chart, thus to be prompted during the phase of concept generation [24].

How to apply it

First of all, the product needs to be specified. Furthermore, an evaluation on which sub-functions the product needs to fulfill its duties must be conducted. These functions are organized such that each and everyone will have a pool of ideas to choose from when solving the functionality. Brainstorming is a helpful addition to this method of organizing ideas and concept generation. Then, each concept is chosen with a combination of listed ideas to become ideas for a final

solution. It is crucial to try and keep the ideas within the same generality or means as the function it is supposed to solve. The last note would be to mention that this technique may be used further than one level of sub-functions. If, for example, the function of a product should be portable, then a new investigation by a morphological chart may be conducted to resolve its placement as a portable product.

2.2.3 Evaluation

What it is

Evaluation is the assessment of how the generated concepts solve a set of design dimension criteria [17].

Why apply it

The reason for applying this method is to organize several ideas concerning given weights and to decide which concept is the overall best [17].

How to apply it

When designing a product, one must consider different design dimensions such as; area usage, weight, simplicity, and so on in order to find suitable concepts. Each dimension is then given a factor on how important it is in relation to the finished concept. Each concept and its dimensions are evaluated by its factor on how it solved the design dimension. Finally, the concepts are evaluated and given a final score.
The relative weighting is due to some concepts that may solve some lower weighed dimensions exceptionally, meanwhile scoring poorly on higher valued design dimensions. The overall score will, therefore, suggest the best solution according to the weighted design dimensions [17].

2.3 Robotic Arm

The following subsections are related to the robot arm design and the types of equipment that are needed. Furthermore, what would be used to design the robot arm physically is also presented.

2.3.1 Mechanical Links

The function of the links is to attach the end-effector to the derelict pot from the ROV. Since the links are exposed to seawater, they must be made of a non-corrosive material. Besides, they are open to the underwater environment and, therefore, should be sturdy due to possible impacts with the environment. The third moment is the weight of the links, as they are the most significant contributor to the manipulator's overall weight. The last points should be costs as the budget for this project is limited and manufacturing. Tab. 2.1 shows the following materials that have been evaluated based on the considerations above.

Material	Weight	Cost	Mechanical	Manufacture
Plastic	+++	+++	- - -	+++
Aluminum	++	0	0	+
Stainless Steel	0	0	+++	+
Bronze	++	- -	0	++

Table 2.1: Material Selection

Thermoplastics scores highly in the evaluation due to the possibility of additive manufacturing (3D print), weight, and low costs. However, it is deemed unfit for the purpose, as the printers constrict the actual size of the parts. It might be an option for an early prototype but not as a demonstration prototype. The mechanical properties are not suited for a robust design, mainly due to the limited sight provided by the ROV camera. Collisions can be expected and should, therefore, be taken into consideration. The metals might, therefore, be a better fit for the links.

2.3.2 Hydraulic Design

Considerations and applications are made during the development of the manipulator system. When deciding to implement control of the hydraulic system, the weighing of the layout has been made to fit this purpose. The actuator on the arm is to be actuated by a working fluid and therefore needs to be controlled by a component. The two main layouts of controlling a hydraulic circuit are either by pump control or by servo valves.

Servo valves	
Pros	Easily controlled Varied flow
Cons	Expensive Hard to get by for seawater Increased complexity

Table 2.2: Evaluation of servo valves

Pumps	
Pros	Easily controlled
Cons	Expensive Hard to get by for seawater

Table 2.3: Evaluation of pumps

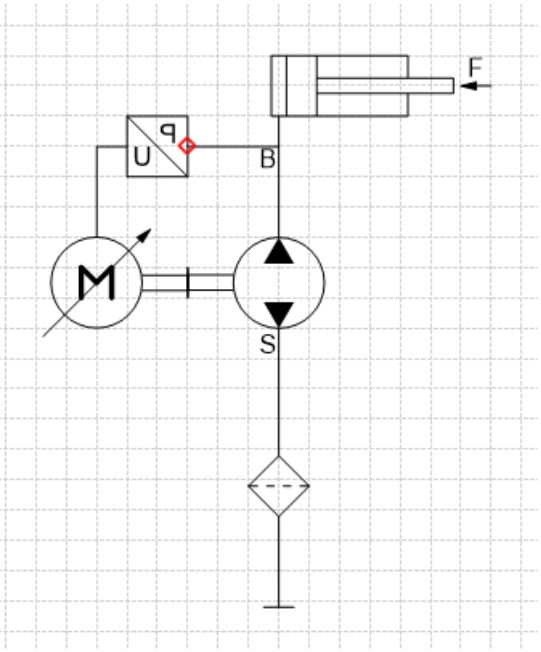


Figure 2.1: Proposed hydraulic layout for the actuator

After a great effort of investigating which layout to utilize, the group finally decided that a bi-directional pump with variable displacement, connected directly to the actuator was the best choice. Servo valves proved too expensive. Pumps, however, were within the budget.

2.3.3 Pump

The pump should ideally run on seawater, function underwater, and work under pressure conditions. However, off-the-shelf pumps with these properties are not easily found. Oil pumps might be an option for prototyping, although these pumps are usually in need of lubrication, which seawater does not provide. An oil pump may be subjected to abrasive corrosion, which is mechanical deformation due to mechanical contact between movable parts. Since these pumps are relatively cheap and readily available, they will serve as the pump of choice for prototyping.

2.3.4 Artificial Muscle

In the market today, there are four main types of actuators: Hydraulic, pneumatic, electric, and mechanical. Since it is predetermined that the actuator is to employ seawater as its working fluid, the principle of hydraulic actuators is to be used. A hydraulic actuator consists typically of a cylinder or fluid motor that utilizes hydraulic power to facilitate the mechanical procedure. The mechanical movement gives an output in terms of rotary, linear, or oscillatory motion [51]. Motors are often bi-rotational, which is defined by a triangle on the motor. This means that fluid is permitted to enter at either port. Some pumps can also be motors at the same time, and further still, be bi-rotational. In this project, no valves will be utilized, and the fluid will enter an artificial muscle directly instead of a cylinder. This artificial muscle is self-made and is assembled using a zig-zag skeleton composed of thin 316 steel plates, a TPU film, and a hose adapter made of plastic. These simple objects assembled correctly will be able to lift 1000x their weight, according to John A. Paulson et al. [38]

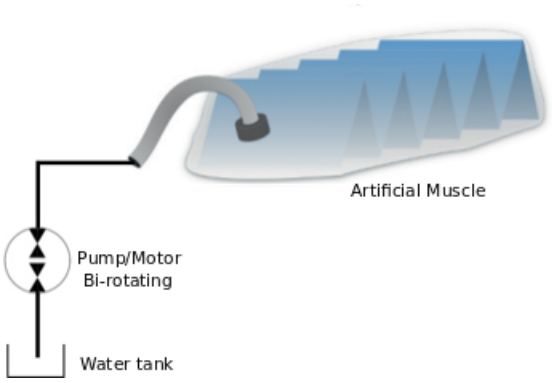


Figure 2.2: Artificial Muscle

Fig. 2.2 shows the working principle of the muscle. The pump pumps water from the ocean and enters the muscle, making the links move accordingly. In Fig. 2.3, the artificial muscle can be shown in operation.

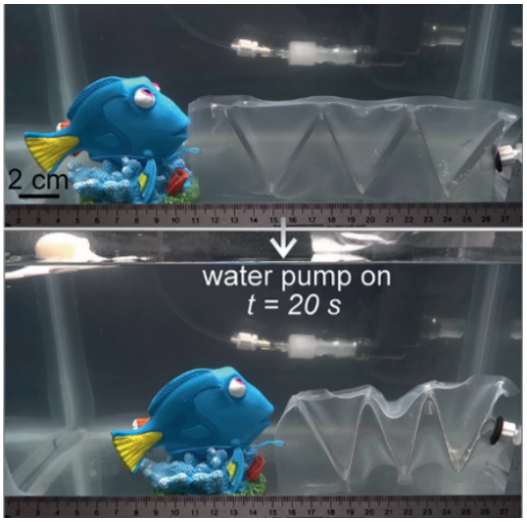


Figure 2.3: A water pump-driven hydraulic actuator pulls a fish for 3.5 cm in 20 s [38]

Springs

The steel links will be driven by a self-made actuator, which utilizes water as its fluid. This actuator can be made by utilizing a spring as the skeleton, or by zig-zag steel plates. For both cases, the spring constant, k, must be estimated in order to predict the undeformed length of the actuator.

$$F = kx \tag{2.1}$$

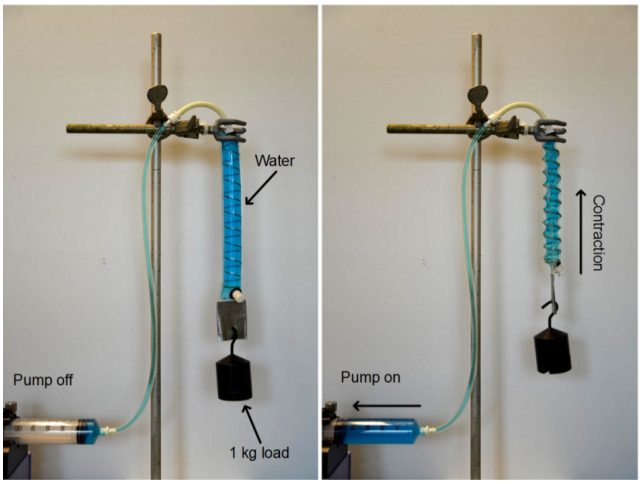


Figure 2.4: Water-driven load lifting test using a spring-based muscle [38].

Fig. 2.4 shows an origami-inspired artificial muscle utilizing a spring as a skeleton. In this case, the pump will create a suction that will contract the spring and then lift the payload.

2.3.5 Modeling Theory

Kinematics is the knowledge of the geometric movement of a system, in this case, the arm. The kinematics are necessary to control the arm in conjunction with the actuators. The kinematics of the arm, as well as the actuator control for the arm and the theory for modeling of the robotic arm, will be presented and described in this section. All formulas used in this subsection are gathered from MAS414: Mechanical Systems 2, unless otherwise specified. The course is held in Grimstad by Morten Kjeld Ebeesen, an associate professor at University of Agder.

Degrees of Freedom (DOF)

The number of DOF for a mechanical system is the same as the minimum number of coordinates required to fully define the configuration of the system [25]. The general equation for a planar system's number of DOF is given in Eq. 2.2.

$$DOF = 3 \cdot n_b - 2 \cdot n_{rj} - 2 \cdot n_{tj} - 1 \cdot n_d \tag{2.2}$$

Symbol	Description
n_b:	Number of bodies
n_{rj}:	Number of revolute joints
n_{tj}:	Number of translational joints
n_d:	Number of driven joints

Other constraints imposed by different joints may also be included in Eq. 2.2 like gears or revolute-revolute joints.

Joints and Links

In order to define a simplified model of a mechanical system, the links and joints of the system must be defined accordingly. The links define the mechanical bodies of the system and may have properties like mass, inertia, stiffness, and damping. A common practice is to align the local coordinate system for each link with the mass center of gravity for the body.
Joints may connect the bodies of a system. Two types of joints are commonly used, and those are; translational and revolute for planar motions. These are depicted respectively in Fig. 2.5 and Fig. 2.6.

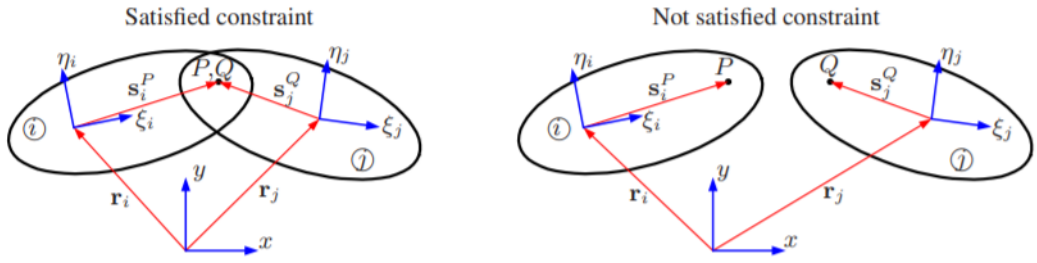


Figure 2.5: Revolute joint and how it constrains two bodies relative to a global coordinate system

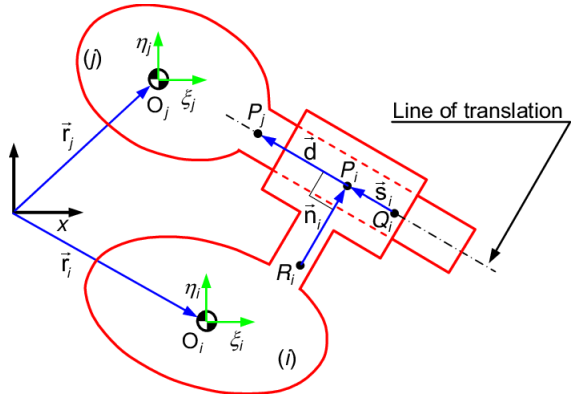


Figure 2.6: Translational joint and how it constrains two bodies relative to a global coordinate system

Denavit-Hartenberg parameters

Jacques Denavit and Richard Hartenberg created a standardized form for finding the forward kinematics of a robot system of succeeding joints and links. This method uses the length of the links, and the angles of the joints to find the position of the tooltip on the end of the last link [1]. By utilizing a very general mathematical notation, the pose of a n-joint robot can be described by Eq. 2.3.

$$\zeta_N \quad = \quad K(q; \theta, d, a, \alpha, \sigma) \tag{2.3}$$

where:

Symbol:		Description [unit]:
ζ_N	-	Pose
K	-	Kinematics (function)
q	-	Vector of the n-joint variables (angles, links[if sliding joint])
θ	-	Vector of joint angles
d	-	Vector of link offsets
a	-	Vector of link lengths
α	-	Vector of link twists
σ	-	Vector of joint types

$$\sigma_j = \begin{cases} R \rightarrow \theta_j = q_j \\ P \rightarrow d_j = q_j \end{cases} \tag{2.4}$$

σ is a vector that contains factors that are either R or P, and they indicate whether the joint is revolute or prismatic. In the case of a revolute joint, substitute the corresponding element of θ from the corresponding element of q. However, for prismatic joints, substitute the corresponding element of d from the corresponding element of q. All the other elements of d, θ, a and α are constant [1].

Rotation matrices

When conducting an analysis, some elements can be generalized and reused.
One of these elements is the rotational matrix for planar movement, which describes the rotation

of a body relative to a fixed reference frame.

$$A(\phi) = \begin{bmatrix} \cos(\phi) & -\sin(\phi) \\ \sin(\phi) & \cos(\phi) \end{bmatrix} \tag{2.5}$$

Where:
ϕ is the rotation relative to the reference frame in $[radians]$

The $A(\phi)$ matrix can be time differentiated to obtain another useful element for velocity and acceleration analysis:

$$B(\phi) = \begin{bmatrix} -\sin(\phi) & -\cos(\phi) \\ \cos(\phi) & -\sin(\phi) \end{bmatrix} \tag{2.6}$$

The matrices $A(\phi)$ and $B(\phi)$ are useful in describing both the joints, revolute and translational, and the constraints they impose, mathematically.

Jacobian and γ Matrices

The Jacobian matrix is the partially differentiated constraints matrix with the dependent coordinates. It is also possible to derive some general entries into the Jacobian depending on which joints are present. For revolute joints, the imposed constraints are:

$$\Phi = r_i + A(\phi_i) \cdot s_i - A(\phi_j) \cdot s_j - r_j = 0 \tag{2.7}$$

Where:
r_i and r_j are the distance from the reference frame to the relative frames of the bodies
s_i and s_j are the relative frames for each body

Eq. 2.7 is the mathematical formulation of the vectors depicted in Fig. 2.5. It is worth noting that if the revolute is connecting a body to the ground, then the last two terms will not be included in the constraint definition.
By differentiating the constraints in Eq. 2.7, the revolute inputs for the Jacobian can be acquired.

$$\dot{\Phi} = \dot{r_i} + \dot{\phi_i} B(\phi_i) \cdot s_i - \dot{\phi_j} B(\phi_j) \cdot s_j - \dot{r_j} = 0 \tag{2.8}$$

With all the elements thus far could a velocity analysis be conducted, although to conduct an acceleration analysis, the entries of the Jacobian must be further differentiated.

$$\ddot{\Phi} = \ddot{r_i} + \ddot{\phi_i} B(\phi_i) \cdot s_i - \dot{\phi_i^2} \cdot A(\phi_i) \cdot s_i - \ddot{\phi_j} B(\phi_j) \cdot s_j + \dot{\phi_j^2} \cdot A(\phi_j) \cdot s_j - \ddot{r_j} = 0 \tag{2.9}$$

As is evident from Eq. 2.9, two unique terms constitute the gamma (γ) matrix. These are $-\dot{\phi_i^2} \cdot A(\phi_i) \cdot s_i$ and $\dot{\phi_j^2} \cdot A(\phi_j) \cdot s_j$. By rearranging, the entries for the γ matrix can be obtained. A common practice is to put the γ matrix on the right-hand side of the equation, multiplying the terms by -1.

$$\gamma = \dot{\phi_i^2} \cdot A(\phi_i) \cdot s_i - \dot{\phi_j^2} \cdot A(\phi_j) \cdot s_j \tag{2.10}$$

The same procedure may be applied to a longitudinal driver, where the driver constrains an angle dependent on the driver's length. The following describes the constraints imposed on the

system by a longitudinal driver and its complimentary differentiated Jacobian entries as well as the gamma entries. It is also convenient to introduce the length as a vector in order to shorten the description of the constraints.

$$d = r_i + A(\phi_i) \cdot s_i - A(\phi_j) \cdot s_j - r_j \tag{2.11}$$

$$\dot{d} = \dot{r_i} + \dot{\phi_i} B(\phi_i) \cdot s_i - \dot{\phi_j} B(\phi_j) \cdot s_j - \dot{r_j} \tag{2.12}$$

$$\ddot{d} = \ddot{r_i} + \ddot{\phi_i} B(\phi_i) \cdot s_i - \dot{\phi_i}^2 A(\phi_i) \cdot s_i - \ddot{\phi_j} B(\phi_j) \cdot s_j + \dot{\phi_i}^2 A(\phi_i) \cdot s_i - \ddot{r_j} \tag{2.13}$$

Taking advantage of the notation from Eq. 2.11, 2.12, and 2.13, the constraints for a longitudinal driver can be written as:

$$\Phi = d'd - (l(t))^2 = 0 \tag{2.14}$$

Where:
d' is the transposed of the vector describing the driver
$l(t)$ is the function of extension for the longitudinal driver

As Eq. 2.14 states; the longitudinal driver only controls one degree of freedom. Further differentiation with the chain rule leads to the velocity and acceleration description of the constraint:

$$\dot{\Phi} = \dot{d}'d + d'\dot{d} - 2l(t)\dot{l(t)} = 0 \tag{2.15}$$

$$\ddot{\Phi} = 2\dot{d}'\dot{d} - 2d'\ddot{d} - 2\dot{l}(t)\dot{l(t)} - 2l(t)\ddot{l(t)} = 0 \tag{2.16}$$

The entries into the γ matrix might then be extracted:

$$\gamma = -2\dot{d}'\dot{d} - 2d' \cdot (-\dot{\phi_i}^2 \cdot A(\phi_i) \cdot s_i + \dot{\phi_j}^2 \cdot A(\phi_j) \cdot s_j) + 2l(t)\ddot{l(t)} \tag{2.17}$$

A System of Unconstrained Bodies - 2D

Equations of motion for a 2D-system consisting of b unconstrained bodies can be written as:

$$\begin{bmatrix} \mathbf{M}_1 & & & \\ & \mathbf{M}_2 & & \\ & & \ddots & \\ & & & \mathbf{M}_b \end{bmatrix} \begin{bmatrix} \ddot{\mathbf{r}}_1 \\ \ddot{\phi}_1 \\ \ddot{\mathbf{r}}_2 \\ \ddot{\phi}_2 \\ \vdots \\ \ddot{\mathbf{r}}_b \\ \ddot{\phi}_b \end{bmatrix} = \begin{bmatrix} \mathbf{f}_1 \\ n_1 \\ \mathbf{f}_2 \\ n_2 \\ \vdots \\ \mathbf{f}_b \\ n_b \end{bmatrix} \tag{2.18}$$

$$\mathbf{M} \qquad\qquad \ddot{\mathbf{q}} \quad = \quad \mathbf{g}$$

Eq. 2.18 can be written in compact form:

$$\mathbf{M}\ddot{\mathbf{q}} = \mathbf{g} \tag{2.19}$$

It might be of interest to examine how mechanical bodies behave when they are unconstrained. However, most of the time, we are interested in obtaining information about constrained bodies, which will be introduced below.

A System of Constrained Bodies - 2D

The equations of motion for a 2D-system consisting of b bodies interconnected by kinematic joints can be written as:

$$\mathbf{M}\ddot{\mathbf{q}} = \mathbf{g}^{ext} + \mathbf{g}^{c} \tag{2.20}$$

where $\mathbf{g}^c$ is a vector containing the reaction forces in the constraints. How this vector is calculated will be illustrated below.

The b bodies in the system must satisfy the equations of motion, yet at the same time, they must also satisfy a set of $m = 3 \cdot b$ independent constraint equations:

$$\mathbf{\Phi} = \mathbf{\Phi}(\mathbf{q}) = \mathbf{0} \tag{2.21}$$

The vector $\mathbf{q}$ contains m coordinates. The coordinate systems utilized in relation to $\mathbf{q}$ and $\mathbf{g}^c$ must match. The derivation below for the vector $\mathbf{g}^c$ containing the reaction forces in the constraints is based on an energy consideration.

The system is exposed to a virtual displacement $\delta\mathbf{q}$. It is assumed that the joints are friction less. The work done by the constraint forces during the displacement is zero as the forces on the two bodies in the joint are of equal magnitude but opposite in direction and the two bodies in the joint move the same distance:

$$\mathbf{g}^{c^T}\delta\mathbf{q} = 0 \tag{2.22}$$

The virtual displacement must be consistent with the constraints:

$$\mathbf{\Phi}_\mathbf{q}\delta\mathbf{q} = \mathbf{0} \tag{2.23}$$

This is based on a Taylor series:

$$\mathbf{\Phi}(\mathbf{q}+\delta\mathbf{q}) = \underbrace{\mathbf{\Phi}(\mathbf{q})}_{=0 \text{ from Eq. } 2.21} + \underbrace{\mathbf{\Phi}_\mathbf{q}\delta\mathbf{q}}_{=0} + \underbrace{\textbf{higher-order terms}}_{\text{To be neglected}} \tag{2.24}$$

Eq. 2.22 and Eq. 2.23 can be combined and merged into one equation:

$$\begin{aligned} \mathbf{g}^{c^T}\delta\mathbf{q} &= 0 \\ \mathbf{\Phi_q}\delta\mathbf{q} &= \mathbf{0} \\ &\Downarrow \\ \underbrace{\begin{bmatrix}\mathbf{g}^{c^T} \\ \mathbf{\Phi_q}\end{bmatrix}}_{(m+1)\times m}\delta\mathbf{q} &= \underbrace{\mathbf{0}}_{(m+1)\times 1} \end{aligned} \tag{2.25}$$

The matrix $\mathbf{\Phi_q}$ has the dimensions $m \times m$ and is nonsingular as the constraint equations are linear independent. Thereby $\mathbf{g}^{c^T}$ can be expressed as a linear combination of the rows in $\mathbf{\Phi}_q$, which can be written as:

$$\mathbf{g}^c = \mathbf{\Phi}_\mathbf{q}^T\boldsymbol{\lambda} \tag{2.26}$$

where $\boldsymbol{\lambda}$ is a vector with the dimension $m \times 1$ containing Lagrange multipliers, that are the coefficients in the linear combination. This means that the constraint reaction forces can be expressed by means of the Jacobian matrix and a set of Lagrangian multipliers.

System of Constrained Bodies

Equations of motion:

$$\mathbf{M}\ddot{\mathbf{q}} = \mathbf{g}^{ext} + \mathbf{g}^c = \mathbf{g}^{ext} + \mathbf{\Phi}_\mathbf{q}^T\boldsymbol{\lambda} \tag{2.27}$$

The constraint equations must also be satisfied:

$$\mathbf{\Phi}(\mathbf{q}) = \mathbf{0} \tag{2.28}$$

Eq. 2.28 is satisfied since it is a kinematic analysis and Eq. 2.27 is rearranged in order to solve for $\boldsymbol{\lambda}$:

$$\begin{aligned} \mathbf{M}\ddot{\mathbf{q}} &= \mathbf{g}^{ext} + \mathbf{\Phi}_\mathbf{q}^T\boldsymbol{\lambda} \Downarrow \\ \boldsymbol{\lambda} &= (\mathbf{\Phi}_\mathbf{q}^T)^{-1}(\mathbf{M}\ddot{\mathbf{q}} - \mathbf{g}^{ext}) \end{aligned} \tag{2.29}$$

2.3.6 Reusability

In this section, designing for the use of bolts according to proof load will be introduced. The advantage of this is that plastic deformation on bolts can be prevented. Proof load is the limit of the elastic range of a bolt. As long as a bolt is never exposed to radially or axially forces beyond its specified proof load, one can be assured that it has maintained its original size and shape and may be safely reused. Steel is an elastic material, and as tension is added, the bolt stretches a certain amount. After reaching the yield strength, a bolt has become so permanently deformed that it is generally accepted to no longer be safely reusable due to a loss of **ductility** (when a solid material deforms under tensile stress without breaking) [40].
Fig. 2.7 shows the region of reusability.

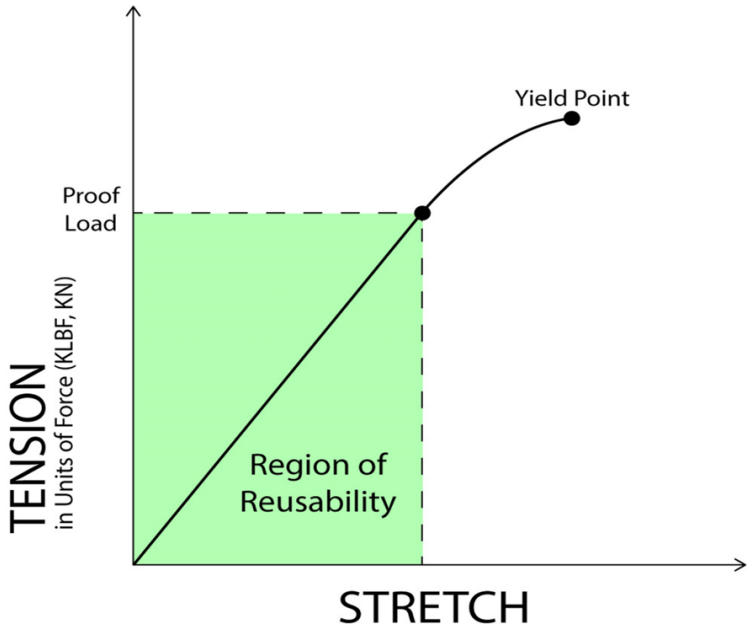


Figure 2.7: A simplified illustration that shows only up to the yield point [40]

Bolts can come loose. Therefore, a bolt with a high proof load is vital. There are multiple reasons for bolts to loosen, and some major reasons are [39]:

- **Vibration**. This can create relative transverse movement, which leads to self-loosening of the nut.
- **Relaxation**. This occurs after tightening due to embedment or gasket creep.
- **Temperature** fluctuation of objects.

2.3.7 PID Controllers

The PID controller is the most widely utilized controller type in the process industry and automation. It is a type of controller that calculates an output value based on the mathematical operations proportional gain (P), integral effect (I), and derivative effect (D) used on the input signal. The user can decide how to combine these three parts and how much is used for each part (tuning). Each part is summed to form the controller output signal $u(t)$. The difference between the setpoint (desired value), y_{set} and the measurement, $y(t)$ is called the regulation deviation $e(t)$, and this deviation should be as small as possible over time [15]. Fig. 2.8 shows a block diagram of a PID controller in a feedback loop.

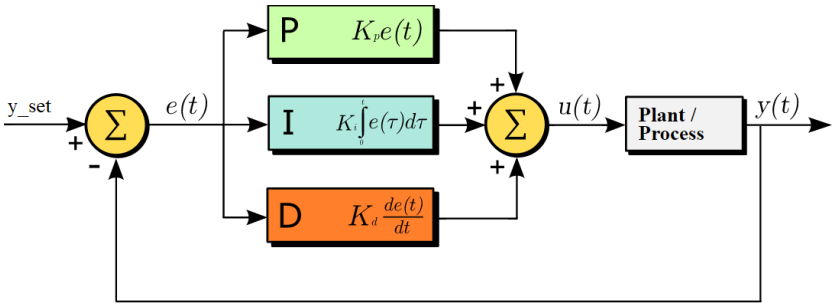


Figure 2.8: PID controller

P-part

In the P-part, a contribution that is proportional to the regulatory deviation we have at a particular time is calculated. The P-part reacts a lot to large deviations and less to small ones.

$$P_{out} = K_p e(t) \tag{2.30}$$

I-part

The I-part has to reset controllers (automatic reset). To integrate means to gather, and this is how the I-part of the controller works as well. The I contribution is a result of the accumulated average deviation; it only takes little account of the deviation at a certain time! The integrator stops integrating when the deviation is zero; Then $y_{set} = y(t)$.

$$I_{out} = \int_0^t e(\tau)d\tau \tag{2.31}$$

D-part

The D-effect in the controller looks at changes in the deviation (or measurement) and contributes to the output signal that is dependent on this change. If the change is zero (flat curve), then the contribution from the D-part is also zero. This is why the D-part is not used alone, but only in combination with the P and I parts.

$$D_{out} = K_d \frac{de(t)}{dt} \tag{2.32}$$

Now, summarizing P_{out}, I_{out}, and D_{out}, a mathematical form of the equation $u(t)$ can be obtained in Eq. 2.33. However, it is advantageous to use the standard form of the equation (Eq. 2.36), because it includes the reset time (T_i) and the derivative time (T_d). These parameters have some understandable physical meaning. The Eqs. 2.34 and 2.35 show the substitutions to convert from mathematical form to standard form.

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de(t)}{dt} \tag{2.33}$$

$$K_i = \frac{K_p}{T_i} \tag{2.34}$$

$$K_d = T_d \cdot K_p \tag{2.35}$$

As mentioned above, it is advantageous to use the standard form of the equation, Eq. 2.36.

$$u(t) = K_p \left(e(t) + \frac{1}{T_i} \int_0^t e(\tau) d\tau + T_d \frac{de(t)}{dt} \right) \tag{2.36}$$

2.4 Custom-Made End-Effector

At the edge of the robot arm, the custom-made end-effector is to be installed appropriately. As the word "custom-made" suggests, the end-effector is designed for the particular task, namely, attaching it to a derelict pot. The different equipment utilized will be presented here.

2.4.1 Inflator Body

A release mechanism that is used in life jackets will be utilized, and its function is to fill a lifejacket with air [49]. In this case, this will be used to fill the balloon with air and then lift the derelict pot to the sea surface. This section will introduce what components were used and how this approach has been made.

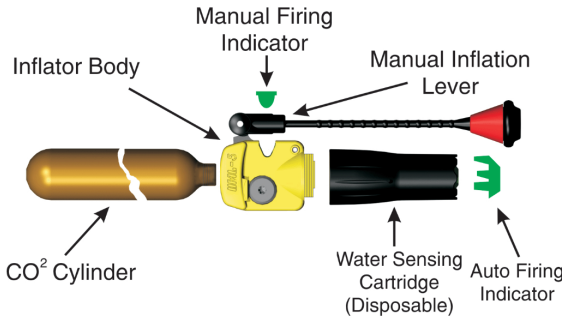


Figure 2.9: Lifejacket Rearming Kit for UML-5

Fig. 2.9 shows the mechanism which inflates a lifejacket. When the manual inflation lever is pulled, the CO_2 cylinder will be activated by a needle that is inside the inflator body, and then the lifejacket will be filled with air (CO_2). Another possibility to activate the CO_2 cylinder is to remove the water-sensing cartridge and the auto-firing indicator. If removed, a button formed like an "H" can be seen, and if this button is pushed, the needle will activate the CO_2 cylinder. The last-mentioned method of activating is the one that will be utilized. In addition to this mechanism, a carabiner will be included. The attaching of the box concept was selected along with the carabiner. More of this will be introduced in Sec. 3.4

2.4.2 Carabiner

Together with the inflator body, a carabiner is going to be used. This section will contain some information about the carabiner

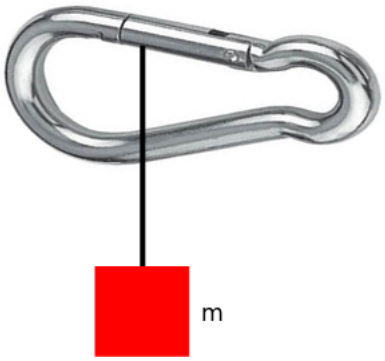


Figure 2.10: Carabiner

The amount of force required to open the carabiner is of high interest. The force is needed to inject it onto the derelict pot and is obtained by experiments, which can be shown in Fig. 2.10. After testing several masses, the one that opened the carabiner successfully was at approximately $3\,kg$. The pump chosen for this purpose needs to deliver enough pressure to the muscle such that the muscle opens the carabiner.

Carabiner Materials

- **Steel carabiners** are perfect for industrial use, as they have enormous strength. However, Croft [10] discourages to use these in seawater, because of their propensity to rust and degrade.
- **Aluminum carabiners** are what should be used for underwater applications, according to Croft [10]. They do not rust. However, if left in wet gear for an extended period, they can corrode. After use, they should be cleaned. If they start corroding, then they will eventually be weakened to the point of not being usable.

There might be other mechanisms that could have been used for the attaching of the end-effector. However, the group has done a considerable investigation and considers a carabiner to be sufficient for this project. The one shown in Fig. 2.10 is an aluminum carabiner, the sort of type intended to be used in the end-effector.

2.5 Buoyancy

An important contribution that will occur and needs to be taken into consideration is buoyancy. Imagine holding a slightly heavy object by hand. Holding that same object in a container full of water, it will feel lighter. See Fig. 2.11. That introduces the concept of buoyancy.

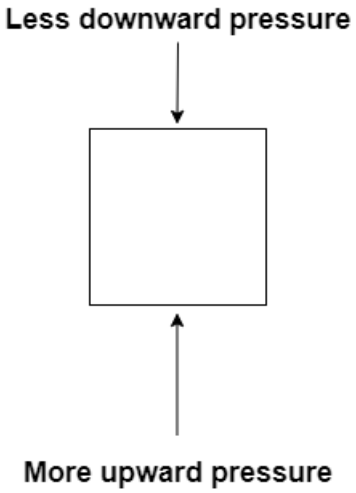


Figure 2.11: Illustration of an object in water

When an object is placed inside a fluid, the fluid pressure increases with depth. Since the pressure increases, the deeper we go, the force exerted upward due to the pressure will be greater than the force exerted on the top. This net upward force exerted on objects submerged in fluids is called the buoyant force.
Buoyancy is the sum of the forces acting against the surface of an object when fully or partially immersed in a liquid or gas. According to the Archimedes' principle, the buoyancy is directed vertically upwards and equal to the weight of the gas- or liquid amount displaced [27]:

$$B = \rho_f V g \tag{2.37}$$

The net force on an object must be zero if it is to be a situation of fluid statics such that the Archimedes principle is applicable. It is the sum of the buoyancy force and the object's weight:

$$F_{net} = 0 = mg - \rho_f V g \tag{2.38}$$

where:

Symbol:		Description [unit]:
B	-	Buoyant force $[N]$
ρ_f	-	Density of the fluid $[kg/m^3]$
V	-	Volume of the displaced body of liquid $[m^3]$
g	-	The gravitational acceleration at the location in question $[m/s^2]$
F_{net}	-	The net force on object $[N]$
m	-	Mass of object $[kg]$

2.6 Underwater Testing

Since the mission was to design a prototype suitable for working underwater, this had to be tested. Whenever reference is made to the water tank in the report, then it is referred to the one shown in Fig. 2.12. Only the robot arm was supposed to be submerged into water. However, the pump and other hardware were supposed to be placed outside the tank. Nevertheless, this was not used due to unexpected circumstances (more of that in Discussion)

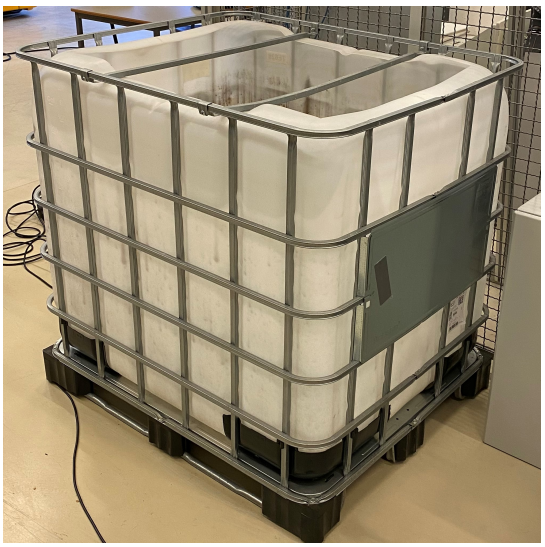


Figure 2.12: Water tank

2.7 Pressure Losses

As a first consideration, the plan was to place the pump and other hardware outside the water tank. This action will, indeed, lead to a distance from the pump to the actuator. Increased distance introduces pressure loss in lines/hoses. When fluid enters one end of a hose and leaves the other, pressure drop, or pressure loss, take place. During operation, fluids, solids, liquids, or gasses rubbing against the interior walls of the hose will cause friction. The pressure loss can be estimated with engineering models using fluid type, assembly specifications, flow rate, and more [9]. This subsection will contain the necessary operations done in order to minimize the losses and formulas used to calculate them. All formulas used in this section are gathered from MAS410: Hydraulic Components and Systems, unless otherwise specified.

Pressure Losses In Lines: Darcy-Weisbach equation

$$\Delta p = \lambda \cdot \frac{L}{d} \cdot \frac{1}{2} \cdot \rho \cdot u^2 \tag{2.39}$$

Pressure Losses In Fittings (bends, turns, crossings etc.)

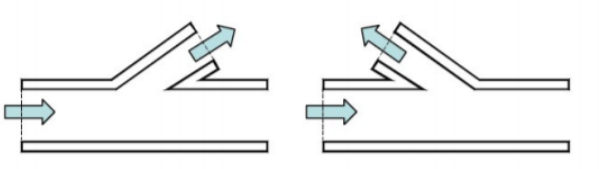


Figure 2.13: An illustration of fittings

If there are two different geometries, two different equations are necessary to utilize in order to find the pressure loss, which are Eq. 2.39 and Eq. 2.40. Fig. 2.13 illustrates what is meant by fittings.

$$\Delta p \quad = \quad \zeta \cdot \frac{1}{2} \cdot \rho \cdot u^2 \tag{2.40}$$

where:

Symbol:		Description [unit]:
Δp	-	Pressure difference $[bar]$
λ	-	Friction number $[-]$
L	-	Length of pipe/hose $[m]$
d	-	Diameter of pipe/hose $[m]$
ρ	-	Fluid density (seawater) $[kg/m^3]$
u	-	Fluid velocity $[m/s]$
ζ	-	Empirical friction factor

As a final statement, the group calculated the pressure losses for this specific scenario (i.e., referring to if the pump and the other hardware were placed outside the water tank), yet an actual and physical prototype would not have a considerable distance from the pump to the actuator. The pump would have been installed in a suitable place in the ROV, i.e., there would be a very short distance from the pump to the actuator. A short distance indicates very small to none pressure loss.

2.8 Simulation

ROS will in this project handle operations such as communication between the simulated environment in Gazebo amongst other tasks. In order to give the reader a better understanding of a typical ROS setup, the concepts: **Nodes**, **Topics**, **Messages**, **URDF** and **Package** will be further explained in short.

Nodes

A node is an independent running executable which may be defined according to its desired function of publishing, subscribing or both to a stream of data. An example of a node is the input command to a controller, which then would be publishing the data of commands.

Topics

A topic is data which is published by a node. To continue the aforementioned example; The topic is the command data from the node.

Message

A message is a definition of data which aids ROS to convert the data into different languages. The two main languages which is used within the ROS framework is Python and CPP. A message contains a field and a constant or a parameter.

URDF

URDF, or Unified Robot Description Format, is a XML based format for describing the robot. The description may contain geometry and properties of the robot, description of sensors and addition of custom plugins.

Package

A package is a structured directory in a catkin workspace containing the source code and instructions like dependencies towards other packages. The most common practice is to save packages in a catkin workspace similar to Fig. 2.14.

```
workspace_folder/        -- WORKSPACE
  src/                   -- SOURCE SPACE
    CMakeLists.txt       -- 'Toplevel' CMake file, provided by catkin
    package_1/
      CMakeLists.txt     -- CMakeLists.txt file for package_1
      package.xml        -- Package manifest for package_1
    ...
    package_n/
      CMakeLists.txt     -- CMakeLists.txt file for package_n
      package.xml        -- Package manifest for package_n
```

Figure 2.14: Description of a package and how it is organized within the workspace [31]

Gazebo

Gazebo is a simulation environment popular among ROS users due to the possibility of creating and integrating custom plugins. Gazebo-ROS packages are also relatively well developed and are useful for connecting the two interfaces.

Chapter 3

Methods

The procedure of how this project has been carried out is introduced in this chapter. During the beginning of the project, a physical prototype was intended to be designed and generated. However, later during the semester, some changes occurred, and the main focus on the project became now simulation, which meant to focus more on software and not hardware.

3.1 Project Approach: Scrum Development Methodology

This method is the most widely favored, agile software development approach. Essentially, this method is suitable for those development projects that are continually altering or too developing requirements. To give a general overview of how this method works before going in detail, it is merely a development model that initiates with transitory planning and conference and completes a concluding review.

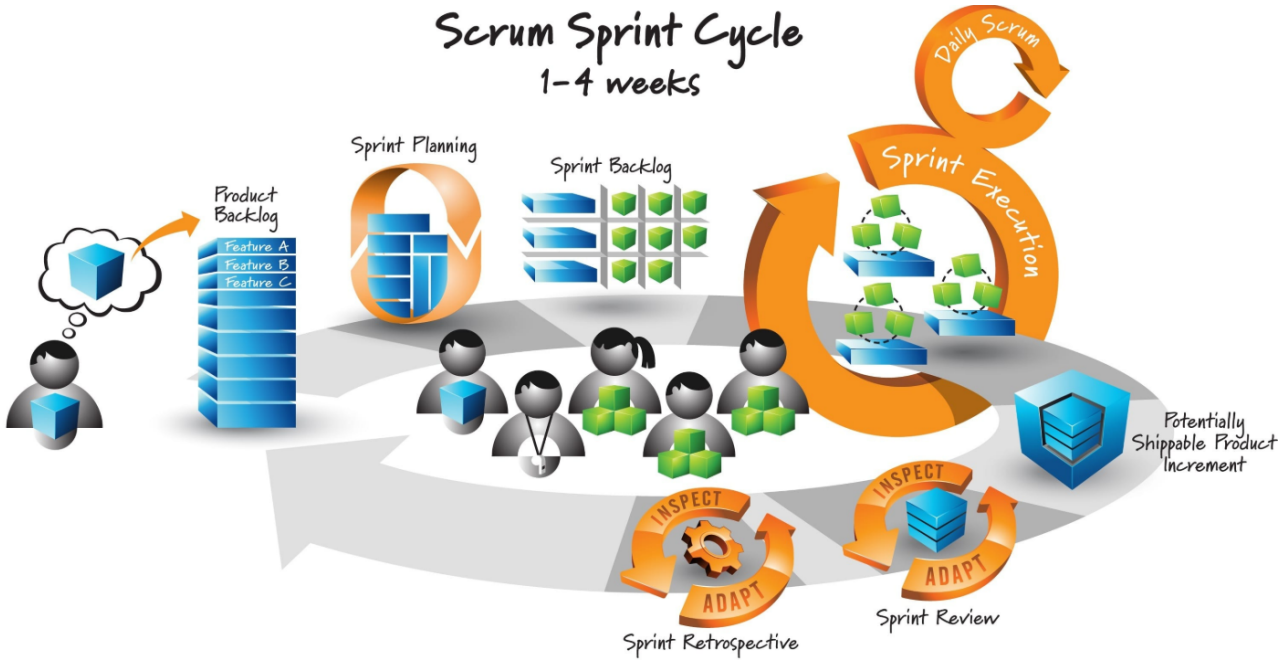


Figure 3.1: Scrum method [42]

Key Roles

Three key roles are needed for the framework to work well. First: The product owner, which is the person responsible for defining the features that are needed in the product. The product owner has bright ideas that turn into products. The scrum master is a servant leader to the team, responsible for protecting the team and the process, running the meetings, and keeping things going. The team can consist of developers, testers, and anyone else that helps in building the product [50]. In this case, the supervisors are the product owners and the scrum masters, yet the team is the students.

Product Backlog

This is where the product owner generates a list of the bright ideas and features, which are known as user stories, that could go into the product. Furthermore, the product owner prioritizes this generated list, and then the top items are brought to the team.

Sprint Planning

In this phase, the product owner, scrum master, and the team meet to discuss the user stories with top priority and estimate their relative sizes. What is approved to go into the next sprint will be decided here.

Sprint Backlog

User stories are a method of describing a feature set that follows a specific format that allows the product owner to specify the correct amount of detail for the team to approximate the size of the task. This is a list of user stories [22] that have been committed to for the next sprint. The team and the product owner have a great comprehension of what each of the user stories involves based on the discussions from the sprint planning meetings [16].

Sprint Execution

Sufficient planning is done to start with building the minimal feature set. What was planned is then build. Next, that small feature set is tested and reviewed, making it clear to "ship." When that cycle is complete, this results in a potentially shippable product, as seen in Fig. 3.2. This process typically occurs in 1-3 weeks, and this is repeated time and time, reducing the time from planning to development to testing. Each time through the planning process, only sufficient planning is done to complete the next incremental release. In the end, this results in several incremental releases (sprints). A sprint takes typically from 1-3 weeks, and the sprints are just repeated until the product is feature complete.

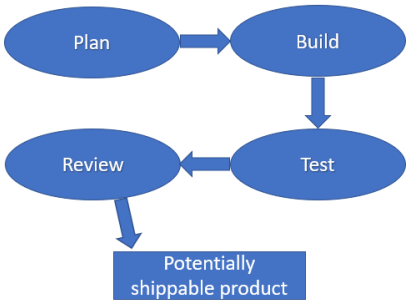


Figure 3.2: Illustration of sprints

During the sprint, the **Daily Scrum** occurs. This is a brief stand-up meeting, where the team discusses what they have completed since the previous meeting, what they are working on, and as well as any blocked items. The outcome is a **Potentially Shippable Product**, which means that the product owner can decide if it is ready to ship or any additional features needed before it finally ships.

Sprint Review & Sprint Retrospective

The **Sprint Review** and **Sprint Retrospective** meeting occurs at the end of the sprint. The first mentioned is where the team showcases their work to the product owner, and the last mentioned is where the team discusses what they can do to improve their process.

For each sprint, repeat the workflow shown in Fig. 3.1.

3.2 Product Development Approach

This section will contain different design approaches, which have been made by the aid of the product development methods. Besides, the advantages, disadvantages, and the application of the different concepts will be introduced. Afterward, they will be graded based upon simplicity, time consumption, efficiency, and other factors, which can be shown in Tab. 3.1. The one with the highest points will be chosen. All concepts are designed in SolidWorks.

3.2.1 Function-Means Tree

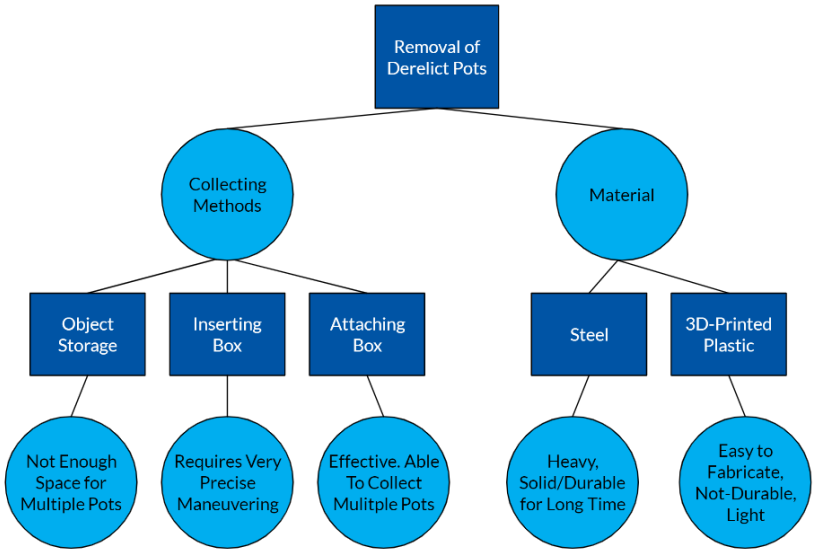


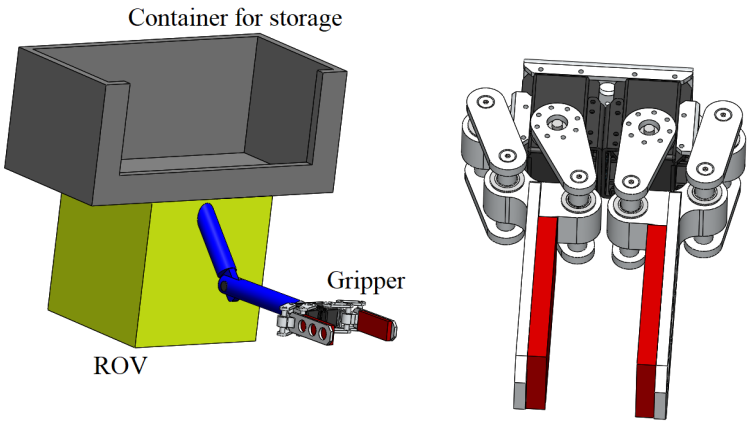
Figure 3.3: Function-Means Tree

Fig. 3.3 shows a function-means tree (FMT) and its purpose is to generate a graphical overview of the functions the end-effector should have to solve the main task of gathering derelict pots. Since the end-effector has a vital task, the group team members considered the material to be used, as well as production and storage, to mention a few.

3.2.2 Morphological Chart

The group did some thoroughly thinking of concepts for this assignment and took into consideration their functionality. Furthermore, some hand sketches were made to convey ideas easily and later SolidWorks was utilized to obtain more detailed and professional concepts. Based on the FMT, concepts were designed. These concepts are illustrated below.

Object Storage



(a) Complete figure (b) Close view of gripper

Figure 3.4: Object Storage

Fig. 3.4a shows a simple illustration of the first concept considered. This concept uses the principle that whenever a derelict pot (or any other object for that matter) is detected, the arm will grasp it by utilizing a self-made gripper and store it in a container attached to the ROV. The container should be mounted on the top of the ROV. Otherwise, the arm would require more DOF.

Inserting Box

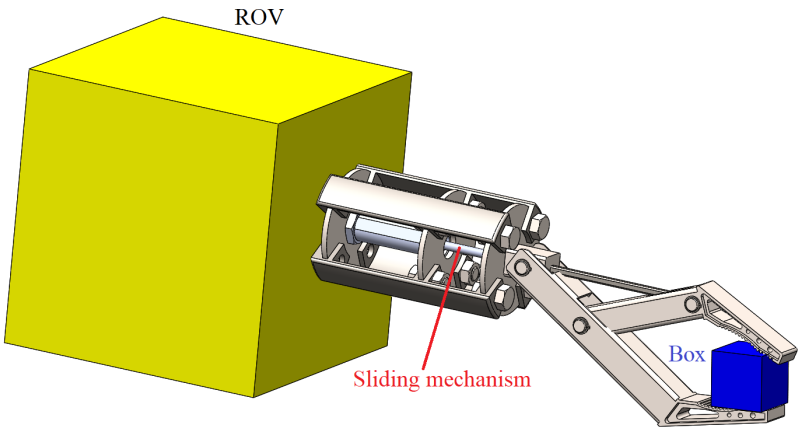


Figure 3.5: Inserting Box

Fig. 3.5 illustrates the second method considered. This concept works in such a manner that a customized arm will insert a self-made box (containing a balloon) inside the derelict pot. The ROV has an opening (not visible on the figure), which allows for the sliding mechanism to slide back and forth. The box functions as a hand grenade, i.e., it will be triggered after release. A possibility here is to have a detonator inside the box, which the user steering the ROV can activate. A camera mounted on the ROV is required to show the user that the box is inside the crab pot. The balloon will then start expanding and thus lifting the crab pot to the sea surface.

Attaching Box

This concept is a simplification of the previous concept, yet the principle is precisely the same. The mission is to lift the derelict pots to the sea surface. Instead of inserting the box inside the derelict pot, a carabiner will be used to initiate the box to activate and expand.

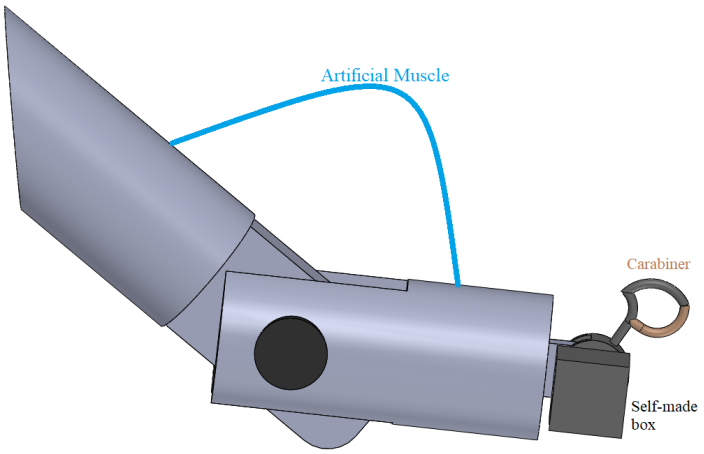


Figure 3.6: Attaching Box

Fig. 3.6 shows a simple illustration of how the assembly of the robot arm works. The figure shows neither the ROV nor the balloon inside the box. The artificial muscle will ensure that the links are expanded and contracted.

3.2.3 Evaluation of the Concepts

Since evaluation without system can be difficult, the method described in Sec. 2.2.3 was used. To determine the best suitable concept for the task, a table, weighting each concept up against each other on their ability to satisfy the criteria, is made.

Requirement	**Object Storage**	**Inserting Box**	**Attaching Box**
Ability to steer	+	-	+
Design	-	+	+
Assembly	0	+	+
Price	0	0	0
Mass	0	0	0
Safety	+	-	+
Efficiency	0	0	+
Load capacity	0	0	0
Total score	1	0	5

Table 3.1: Concept evaluation

The Attaching Box concept is chosen for further development, as this design seems to be best suited for the task. The Object Storage concept could defeat its purpose of saving battery time since the derelict pots are carried by the ROV. Besides, to be able to gather many derelict pots, a great amount of space is required. The Inserting Box concept could be a great solution, yet the insertion of the box would demand camera vision and very precise insertion, as one would need to get inside the potholes. Now that the concept is chosen, the derelict pot is assumed to be in a fixed and known distance from the arm.

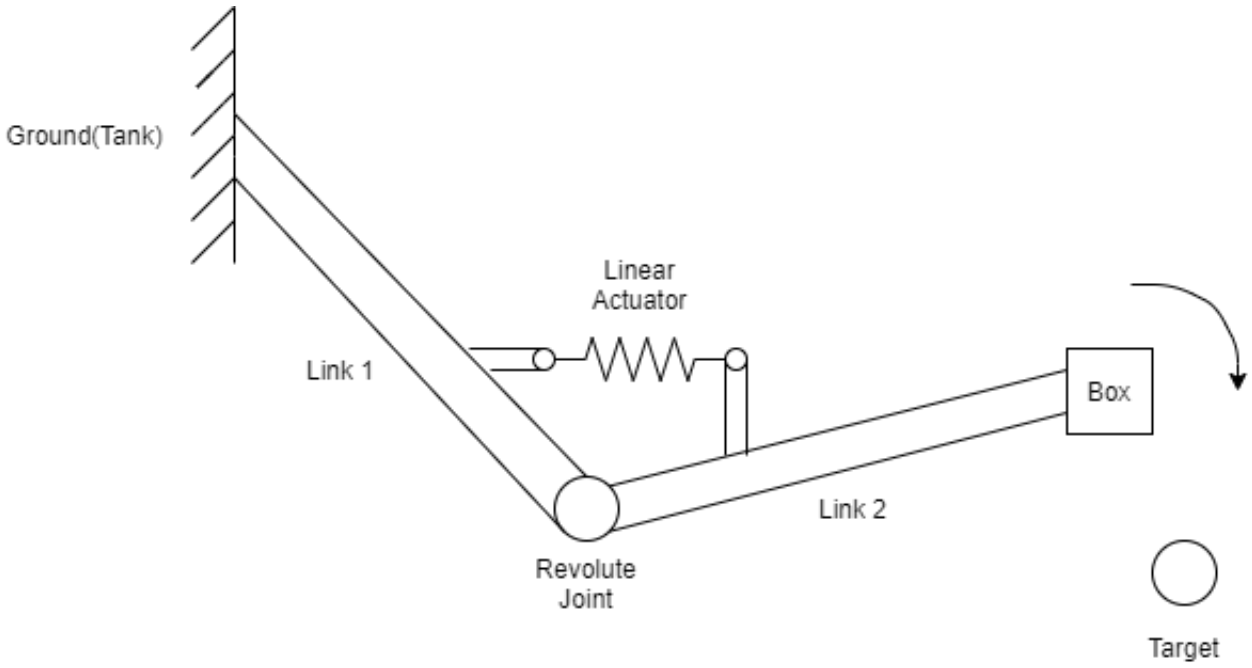


Figure 3.7: Simplified drawing of concept

Fig. 3.7 shows a simplified drawing of the concept for the manipulator with its components. The figure shows how the concept primarily was intended to look like, but later during the semester, the group added one more actuator and expanded the amount of DOF. Hence a revolute joint between the ROV and the first link (link 1).

3.3 The Robotic Arm

In this section, will the components of the manipulator, as well as the required theory to control the manipulator, be presented.

3.3.1 Encoders

An encoder is, in simple terms, an electromechanical feedback device that provides info about position, speed, count, and direction. They produce signals that are received by a control device in order to perform a specific function. Absolute and incremental are two main categories of encoders. The last-mentioned provide a steady stream of high and low pulses that indicate movement from one position to the next. It does not indicate its position, only the change in position. However, absolute encoders, indicate both that the position has changed and the location of that position with respect to shaft rotation. It provides a digital word (bit) for each increment of rotation. Absolute encoders are the best choice for applications where exact position needs to be known [8].

Encoders are to be utilized and mounted in the revolute joints. When the robot links move, the encoder will provide us information about how much it has turned. Then, it is possible to calculate the position of the end-effector. Fig. 3.8 shows a rotary encoder knob that can be rotated unlimited times. It has a flat sided shaft and has a built-in push button under the shaft. It is very lightweight ($6.43\,g$), so it will not affect the movement of the arms.

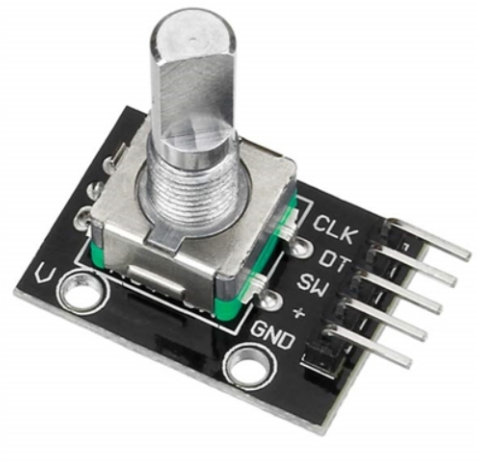


Figure 3.8: Encoder

3.3.2 Actuators

As mentioned in Sec. 1.2.2, the group fabricated two actuators. This section will introduce how they were constructed and an explanation of the muscles. The idea behind the muscles/actuators is that one end is attached to the first link, and the other end of the actuator is attached to the second link. Same for the second muscle, which is to be attached to the ROV and the first link.

Pneumatic Artificial Muscle

The reason why these muscles were created in the 1950s was because of orthotics [11]. They have several benefits, e.g., being lightweight, uncomplicated to manufacture, self-limiting (i.e., have a maximum contraction), and have load-length curves similar to the human muscle. The muscles are assembled of an inflatable inner tube/bladder inside a braided mesh, clamped at the ends. When the inner bladder is pressurized and expands, the geometry of the mesh acts like a scissor linkage and translates this radial expansion into linear contraction. In general terms, the artificial muscle contracts up to a maximum of typically 25% in a linear motion [28]. Furthermore, it is possible to achieve contractions around 40% though different materials and construction. Technically speaking, they can be designed to be lengthened as well, yet buckling may occur. Their operation stages can be shown respectively in Fig. 3.9a and Fig. 3.9b

(a) Relaxed (b) Pressurized, contraction

Figure 3.9: Pneumatic artificial muscle (PAM)

Hydraulic Artificial Muscle

When the pneumatic artificial muscle was made, the group was content with the results. Then, it was decided to manufacture another muscle actuated by water since this was one aspect of the master's thesis. However, this proved to be more demanding than expected, and the reason was, the group believed, the low water pressure from the tap combined with leakage from the water hose connection. Pressure from tap water is typically set to be between 2-6 bar. From Fig. 3.10a, and 3.10b, it can be seen that there is barely any contraction when the inner bladder is pressurized by water. In Fig. 3.10b, a significant amount of leakage from the water hose was present. The group discussed this issue and thought that if the hose was near leakage-free, the problem would be solved.

However, the pressure from the water hose was not high enough to create the contration, the group concluded that this artificial muscle would not fit this purpose and could not be used unless a pump capable of delivering enormous pressure was utilized. As mentioned earlier, finding a suitable pump for this purpose proved to be very challenging. After a great effort of researching and attempting to solve this issue, the FOAM was discovered. This was an excellent method since it did not require a lot of pressure nor flow rate, yet it was able to lift 1000x its own weight [38]. The group finally realized how the actuator was supposed to be designed, and now it was time to order the necessary elements to manufacture it.

(a) Relaxed (b) Pressurized, barely any contraction

Figure 3.10: Hydraulic artificial muscle

These actuators were fabricated before the school's lockdown, and before the items that the group ordered for the project arrived. When the items finally arrived, the group did not have access to the schools utilities nor the items.

3.3.3 System Analysis

In order to continue the mechanical design of the manipulator, a kinematic analysis must be conducted. The links are made of 316 steel, have a mass of 4 kg, and a diameter of 40 mm. The idea is to mount the first body (body 1) to the ROV with a revolute joint. Two linear actuators will ensure that the links are actuated according to the requirements of the end-effector.

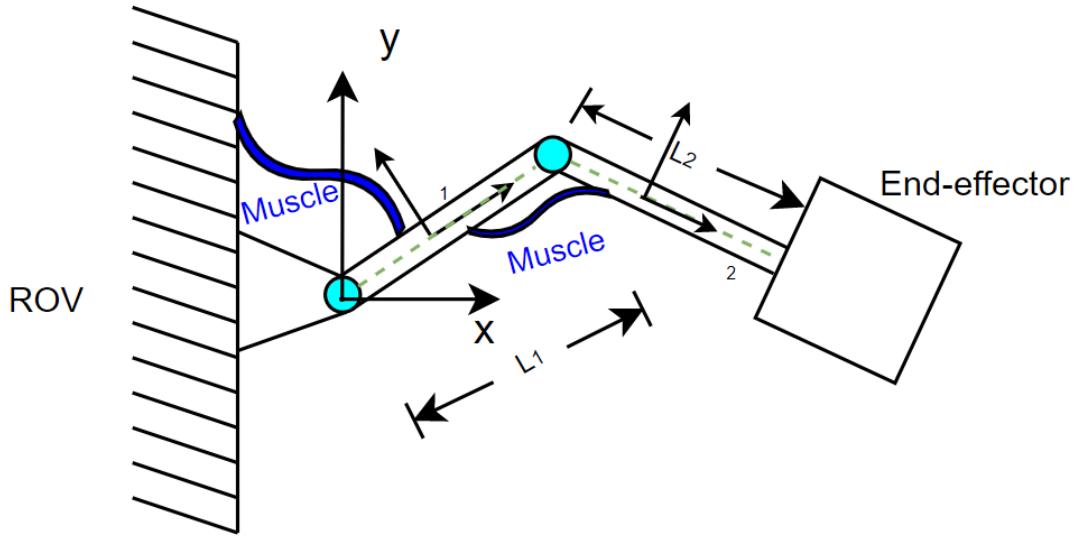


Figure 3.11: Simplified analysis of the arm

The following constraints are present:

- A revolute joint between the ROV (ground) and body 1 (= 2 constraints)
- A revolute joint between body 1 and body 2 (= 2 constraints)
- A longitudinal driver between the ROV and body 1 (=1 constraint)
- A longitudinal driver between body 1 and body 2 (=1 constraint)

Both longitudinal drivers are, in fact, artificial muscles (Sec 2.3.4), yet behave precisely as a hydraulic actuator, and are therefore modeled and simulated as such.

The longitudinal driver/artificial muscle between the ROV and body 1 is defined as:

1. $d_1(t) = 0.2\,m + 0.05\,\frac{m}{s} \cdot t$

while the one between body 1 and body 2 is defined as:

1. $d_2(t) = d_1(t)$

To summarize, there are in total six constraints (four kinematic and two drivers) that correspond to the six Cartesian coordinates assigned to the two bodies. The next pages will show the $\underline{\Phi}$, $\dot{\underline{\Phi}}$, and $\ddot{\underline{\Phi}}$ constraints, and later a more thorough explanation will be introduced by showing step-by-step calculations and how the Jacobian and γ entries are obtaining for the constraints.

The mathematical formulation of the constraint is:

$$\underline{\Phi} = \begin{bmatrix} \underline{\underline{A}}_1 \begin{bmatrix} -0.5 \cdot L_1 \\ 0 \end{bmatrix} + \underline{r}_1 \\ \underline{r}_1 + \underline{\underline{A}}_1 \begin{bmatrix} 0.5 \cdot L_1 \\ 0 \end{bmatrix} - \underline{\underline{A}}_2 \begin{bmatrix} -0.5 \cdot L_2 \\ 0 \end{bmatrix} - \underline{r}_2 \\ \left\{ \begin{bmatrix} 0 \\ 0.25 \end{bmatrix} - \underline{r}_1 - \underline{\underline{A}}_1 \begin{bmatrix} 0 \\ 0.08 \end{bmatrix} \right\}^T \left\{ \begin{bmatrix} 0 \\ 0.25 \end{bmatrix} - \underline{r}_1 - \underline{\underline{A}}_1 \begin{bmatrix} 0 \\ 0.08 \end{bmatrix} \right\} - (d_1(t))^2 \\ \left\{ \underline{r}_2 - \underline{\underline{A}}_2 \begin{bmatrix} 0 \\ 0.08 \end{bmatrix} - \underline{r}_1 + \underline{\underline{A}}_1 \begin{bmatrix} 0 \\ 0.08 \end{bmatrix} \right\}^T \left\{ \underline{r}_2 - \underline{\underline{A}}_2 \begin{bmatrix} 0 \\ 0.08 \end{bmatrix} - \underline{r}_1 + \underline{\underline{A}}_1 \begin{bmatrix} 0 \\ 0.08 \end{bmatrix} \right\} - (d_2(t))^2 \end{bmatrix} = \underline{0}$$

The time derivative of the constraint is:

$$\dot{\underline{\Phi}} = \begin{bmatrix} \dot{\phi}_1 \cdot \underline{\underline{B}}_1 \begin{bmatrix} -0.5 \cdot L_1 \\ 0 \end{bmatrix} + \dot{\underline{r}}_1 \\ \dot{\underline{r}}_1 + \dot{\phi}_1 \cdot \underline{\underline{B}}_1 \begin{bmatrix} 0.5 \cdot L_1 \\ 0 \end{bmatrix} - \dot{\phi}_2 \cdot \underline{\underline{B}}_2 \begin{bmatrix} -0.5 \cdot L_2 \\ 0 \end{bmatrix} - \dot{\underline{r}}_2 \\ 2 \left\{ \begin{bmatrix} 0 \\ 0.25 \end{bmatrix} - \underline{r}_1 - \underline{\underline{A}}_1 \begin{bmatrix} 0 \\ 0.08 \end{bmatrix} \right\}^T \left\{ -\dot{\underline{r}}_1 - \dot{\phi}_1 \cdot \underline{\underline{B}}_1 \begin{bmatrix} 0 \\ 0.08 \end{bmatrix} \right\} - 2 \cdot (d_1(t)) \cdot 0.05 \frac{m}{s} \\ 2 \left\{ \underline{r}_2 - \underline{\underline{A}}_2 \begin{bmatrix} 0 \\ 0.08 \end{bmatrix} - \underline{r}_1 + \underline{\underline{A}}_1 \begin{bmatrix} 0 \\ 0.08 \end{bmatrix} \right\}^T \left\{ \dot{\underline{r}}_2 - \dot{\phi}_2 \cdot \underline{\underline{B}}_2 \begin{bmatrix} 0 \\ 0.08 \end{bmatrix} - \dot{\underline{r}}_1 + \dot{\phi}_1 \cdot \underline{\underline{B}}_1 \begin{bmatrix} 0 \\ 0.08 \end{bmatrix} \right\} \cdots \\ \cdots - 2 \cdot (d_2(t)) \cdot 0.05 \frac{m}{s} \end{bmatrix} = \underline{0}$$

The second time derivative of the constraint is:

$$\ddot{\underline{\Phi}} = \begin{bmatrix} \ddot{\phi}_1 \cdot \underline{\underline{B}}_1 \begin{bmatrix} -0.5 \cdot L_1 \\ 0 \end{bmatrix} - \dot{\phi}_1^2 \cdot \underline{\underline{A}}_1 \begin{bmatrix} -0.5 \cdot L_1 \\ 0 \end{bmatrix} + \ddot{\underline{r}}_1 \\ \ddot{\underline{r}}_1 + \ddot{\phi}_1 \cdot \underline{\underline{B}}_1 \begin{bmatrix} 0.5 \cdot L_1 \\ 0 \end{bmatrix} - \dot{\phi}_1^2 \cdot \underline{\underline{A}}_1 \begin{bmatrix} 0.5 \cdot L_1 \\ 0 \end{bmatrix} - \ddot{\phi}_2 \cdot \underline{\underline{B}}_2 \begin{bmatrix} -0.5 \cdot L_2 \\ 0 \end{bmatrix} + \dot{\phi}_2^2 \cdot \underline{\underline{A}}_2 \begin{bmatrix} -0.5 \cdot L_2 \\ 0 \end{bmatrix} - \ddot{\underline{r}}_2 \\ 2 \left\{ \begin{bmatrix} 0 \\ 0.25 \end{bmatrix} - \underline{r}_1 - \underline{\underline{A}}_1 \begin{bmatrix} 0 \\ 0.08 \end{bmatrix} \right\}^T \left\{ -\ddot{\underline{r}}_1 - \ddot{\phi}_1 \cdot \underline{\underline{B}}_1 \begin{bmatrix} 0 \\ 0.08 \end{bmatrix} + \dot{\phi}_1^2 \cdot \underline{\underline{A}}_1 \begin{bmatrix} 0 \\ 0.08 \end{bmatrix} \right\} + \cdots \\ \cdots + 2 \left\{ -\dot{\underline{r}}_1 - \dot{\phi}_1 \cdot \underline{\underline{B}}_1 \begin{bmatrix} 0 \\ 0.08 \end{bmatrix} \right\}^T \left\{ -\dot{\underline{r}}_1 - \dot{\phi}_1 \cdot \underline{\underline{B}}_1 \begin{bmatrix} 0 \\ 0.08 \end{bmatrix} \right\} - 2 \cdot (0.05 \tfrac{m}{s})^2 \\ 2 \left\{ \underline{r}_2 - \underline{\underline{A}}_2 \begin{bmatrix} 0 \\ 0.08 \end{bmatrix} - \underline{r}_1 + \underline{\underline{A}}_1 \begin{bmatrix} 0 \\ 0.08 \end{bmatrix} \right\}^T \left\{ \ddot{\underline{r}}_2 - \ddot{\phi}_2 \cdot \underline{\underline{B}}_2 \begin{bmatrix} 0 \\ 0.08 \end{bmatrix} + \dot{\phi}_2^2 \cdot \underline{\underline{A}}_2 \begin{bmatrix} 0 \\ 0.08 \end{bmatrix} - \ddot{\underline{r}}_1 + \cdots \right. \\ \left. \cdots + \ddot{\phi}_1 \cdot \underline{\underline{B}}_1 \begin{bmatrix} 0 \\ 0.08 \end{bmatrix} - \dot{\phi}_1^2 \cdot \underline{\underline{A}}_1 \begin{bmatrix} 0 \\ 0.08 \end{bmatrix} + 2 \left\{ \dot{\underline{r}}_2 - \dot{\phi}_2 \cdot \underline{\underline{B}}_2 \begin{bmatrix} 0 \\ 0.08 \end{bmatrix} - \dot{\underline{r}}_1 + \dot{\phi}_1 \cdot \underline{\underline{B}}_1 \begin{bmatrix} 0 \\ 0.08 \end{bmatrix} \right\}^T \cdots \right. \\ \left. \cdots \left\{ \dot{\underline{r}}_2 - \dot{\phi}_2 \cdot \underline{\underline{B}}_2 \begin{bmatrix} 0 \\ 0.08 \end{bmatrix} - \dot{\underline{r}}_1 + \dot{\phi}_1 \cdot \underline{\underline{B}}_1 \begin{bmatrix} 0 \\ 0.08 \end{bmatrix} \right\} - 2 \cdot (0.05 \tfrac{m}{s})^2 \right. \end{bmatrix} = \underline{0}$$

A **longitudinal driver** controls the distance between two points located on two individual bodies. The length is specified in the function $l(t)$ and a longitudinal driver controls one DOF. In order to calculate the kinematic constraints, the driver needs to be replaced by a vector d, illustrated in Fig. 3.12.

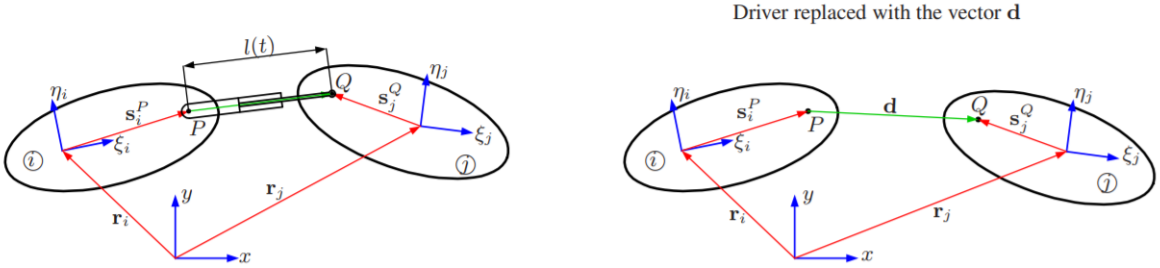


Figure 3.12: Longitudinal driver

The following vectors are defined and substituted for respectively driver 1 (from ROV to body 1) and driver 2 (the driver between body 1 and body 2):

$$\underline{d}_1 = \underline{r}_1 + \underline{\underline{A}}_1 \begin{bmatrix} 0 \\ 0.08 \end{bmatrix} - \begin{bmatrix} 0 \\ 0.25 \end{bmatrix} \quad \&$$

$$\underline{d}_2 = \underline{r}_2 - \underline{\underline{A}}_2 \begin{bmatrix} 0 \\ 0.08 \end{bmatrix} - \underline{r}_1 + \underline{\underline{A}}_1 \begin{bmatrix} 0 \\ 0.08 \end{bmatrix}$$

Furthermore, the vector variables are differentiated with respect to time.

$$\dot{\underline{d}}_1 = \dot{\underline{r}}_1 + \dot{\phi}_1 \cdot \underline{\underline{B}}_1 \begin{bmatrix} 0 \\ 0.08 \end{bmatrix} \tag{3.1}$$

$$\dot{\underline{d}}_2 = \dot{\underline{r}}_2 - \dot{\phi}_2 \cdot \underline{\underline{B}}_2 \begin{bmatrix} 0 \\ 0.08 \end{bmatrix} - \dot{\underline{r}}_1 + \dot{\phi}_1 \cdot \underline{\underline{B}}_1 \begin{bmatrix} 0 \\ 0.08 \end{bmatrix} \tag{3.2}$$

Now that the vectors have been differentiated with respect to time, the $\underline{\Phi}$ constraint for the first longitudinal driver is:

$$\underline{\Phi} = (\underline{d}_1)^T \underline{d}_1 - (d_1(t))^2 = 0 \tag{3.3}$$

$$\begin{aligned} \dot{\underline{\Phi}} &= (\dot{\underline{d}}_1)^T \underline{d}_1 + (\underline{d}_1)^T (\dot{\underline{d}}_1) - 2d_1(t)\dot{d}_1(t) \\ &= 2(\underline{d}_1)^T (\dot{\underline{d}}_1) - 2d_1(t)\dot{d}_1(t) = 0 \\ &= 2(\underline{d}_1)^T \underbrace{\left(\dot{\underline{r}}_1 + \dot{\phi}_1 \cdot \underline{\underline{B}}_1 \begin{bmatrix} 0 \\ 0.08 \end{bmatrix} \right)}_{\text{Substituting with Eq. 3.1}} - 2d_1(t)\dot{d}_1(t) = 0 \Downarrow \end{aligned}$$

$$\underbrace{\begin{bmatrix} 2(\underline{d}_1)^T & 2(\underline{d}_1)^T \underline{\underline{B}}_1 \begin{bmatrix} 0 \\ 0.08 \end{bmatrix} \end{bmatrix}}_{\text{Entries in Jacobian matrix}} \begin{bmatrix} \dot{\underline{r}}_1 \\ \dot{\phi}_1 \end{bmatrix} = 2d_1(t)\dot{d}_1(t) \tag{3.4}$$

Eq. 3.4 shows the entries that define the Jacobian matrix for the first driver. It is of interest to obtain the gamma matrix as well. In order to so, the $\dot{\underline{\Phi}}$ and Eqs. 3.1 and 3.2 need to be differentiated with respect to time to obtain the acceleration matrix.

$$\ddot{\underline{d}}_1 = \ddot{\underline{r}}_1 + \ddot{\phi}_1 \cdot \underline{\underline{B}}_1 \begin{bmatrix} 0 \\ 0.08 \end{bmatrix} - \dot{\phi}_1^2 \cdot \underline{\underline{A}}_1 \begin{bmatrix} 0 \\ 0.08 \end{bmatrix} \tag{3.5}$$

$$\ddot{\underline{d}}_2 = \ddot{\underline{r}}_2 - \ddot{\phi}_2 \cdot \underline{\underline{B}}_2 \begin{bmatrix} 0 \\ 0.08 \end{bmatrix} + \dot{\phi}_2^2 \cdot \underline{\underline{A}}_2 \begin{bmatrix} 0 \\ 0.08 \end{bmatrix} - \ddot{\underline{r}}_1 + \ddot{\phi}_1 \cdot \underline{\underline{B}}_1 \begin{bmatrix} 0 \\ 0.08 \end{bmatrix} - \dot{\phi}_1^2 \cdot \underline{\underline{A}}_1 \begin{bmatrix} 0 \\ 0.08 \end{bmatrix} \tag{3.6}$$

$$\begin{aligned} \ddot{\underline{\Phi}} &= 2(\dot{\underline{d}}_1)^T (\dot{\underline{d}}_1) + 2(\underline{d}_1)^T (\ddot{\underline{d}}_1) - 2\dot{d}_1(t)\dot{d}_1(t) - 2d_1(t)\ddot{d}_1(t) \\ &= 2(\dot{\underline{d}}_1)^T (\dot{\underline{d}}_1) + 2(\underline{d}_1)^T \underbrace{\left(\ddot{\underline{r}}_1 + \ddot{\phi}_1 \cdot \underline{\underline{B}}_1 \begin{bmatrix} 0 \\ 0.08 \end{bmatrix} - \dot{\phi}_1^2 \cdot \underline{\underline{A}}_1 \begin{bmatrix} 0 \\ 0.08 \end{bmatrix} \right)}_{\text{Substituting with Eq. 3.5}} \end{aligned}$$

$$-2\left(\dot{d}_1(t)\right)^2 - 2d_1(t)\ddot{d}_1(t) = 0 \Downarrow$$

$$\underbrace{\begin{bmatrix} 2(\underline{d}_1)^T & 2(\underline{d}_1)^T \underline{\underline{B}}_1 \begin{bmatrix} 0 \\ 0.08 \end{bmatrix} \end{bmatrix}}_{\text{Entries in Jacobian matrix}} \begin{bmatrix} \ddot{\underline{r}}_1 \\ \ddot{\phi}_1 \end{bmatrix} =$$

$$2\left(\dot{d}_1(t)\right)^2 + \underbrace{\cancel{2d_1(t)\ddot{d}_1(t)}^{0}}_{\ddot{d}_1(t)=0} + \underbrace{2(\underline{d}_1)^T \left(\dot{\phi}_1^2 \cdot \underline{\underline{A}}_1 \begin{bmatrix} 0 \\ 0.08 \end{bmatrix}\right) - 2(\dot{\underline{d}}_1)^T(\dot{\underline{d}}_1)}_{\text{Entries in } \gamma \text{ matrix}} \tag{3.7}$$

Now for driver 2, the operation is done in the same procedure.

$$\begin{aligned} \underline{\Phi} &= (\underline{d}_2)^T \underline{d}_2 - (d_2(t))^2 = 0 \qquad (3.8) \\ \dot{\underline{\Phi}} &= (\dot{\underline{d}}_2)^T \underline{d}_2 + (\underline{d}_2)^T(\dot{\underline{d}}_2) - 2d_2(t)\dot{d}_2(t) \\ &= 2(\underline{d}_2)^T(\dot{\underline{d}}_2) - 2d_2(t)\dot{d}_2(t) = 0 \\ &= 2(\underline{d}_2)^T \underbrace{\left(\dot{\underline{r}}_2 - \dot{\phi}_2 \cdot \underline{\underline{B}}_2 \begin{bmatrix} 0 \\ 0.08 \end{bmatrix} - \dot{\underline{r}}_1 + \dot{\phi}_1 \cdot \underline{\underline{B}}_1 \begin{bmatrix} 0 \\ 0.08 \end{bmatrix}\right)}_{\text{Substituting with Eq. 3.2}} - 2d_2(t)\dot{d}_2(t) = 0 \Downarrow \end{aligned}$$

$$\underbrace{\begin{bmatrix} -2(\underline{d}_2)^T & 2(\underline{d}_2)^T \underline{\underline{B}}_1 \begin{bmatrix} 0 \\ 0.08 \end{bmatrix} & 2(\underline{d}_2)^T & -2(\underline{d}_2)^T \underline{\underline{B}}_2 \begin{bmatrix} 0 \\ 0.08 \end{bmatrix} \end{bmatrix}}_{\text{Entries in Jacobian matrix}} \begin{bmatrix} \dot{\underline{r}}_1 \\ \dot{\phi}_1 \\ \dot{\underline{r}}_2 \\ \dot{\phi}_2 \end{bmatrix} = 2d_2(t)\dot{d}_2(t) \tag{3.9}$$

$$\begin{aligned} \ddot{\underline{\Phi}} &= 2(\dot{\underline{d}}_2)^T(\dot{\underline{d}}_2) + 2(\underline{d}_2)^T(\ddot{\underline{d}}_2) - 2\dot{d}_2(t)\dot{d}_2(t) - 2d_2(t)\ddot{d}_2(t) \\ &= 2(\dot{\underline{d}}_2)^T(\dot{\underline{d}}_2) + 2(\underline{d}_2)^T \underbrace{\left(\ddot{\underline{r}}_2 - \ddot{\phi}_2 \cdot \underline{\underline{B}}_2 \begin{bmatrix} 0 \\ 0.08 \end{bmatrix} + \dot{\phi}_2^2 \cdot \underline{\underline{A}}_2 \begin{bmatrix} 0 \\ 0.08 \end{bmatrix} - \ddot{\underline{r}}_1 + \ddot{\phi}_1 \cdot \underline{\underline{B}}_1 \begin{bmatrix} 0 \\ 0.08 \end{bmatrix} - \dot{\phi}_1^2 \cdot \underline{\underline{A}}_1 \begin{bmatrix} 0 \\ 0.08 \end{bmatrix}\right)}_{\text{Substituting with Eq. 3.6}} \end{aligned}$$

$$-2\left(\dot{d}_2(t)\right)^2 - \underbrace{\cancel{2d_2(t)\ddot{d}_2(t)}^{0}}_{\ddot{d}_2(t)=0} = 0 \Downarrow$$

$$\underbrace{\begin{bmatrix} -2(\underline{d}_2)^T & 2(\underline{d}_2)^T \underline{\underline{B}}_1 \begin{bmatrix} 0 \\ 0.08 \end{bmatrix} & 2(\underline{d}_2)^T & -2(\underline{d}_2)^T \underline{\underline{B}}_2 \begin{bmatrix} 0 \\ 0.08 \end{bmatrix} \end{bmatrix}}_{\text{Entries in Jacobian matrix}} \begin{bmatrix} \dot{\underline{r}}_1 \\ \dot{\phi}_1 \\ \dot{\underline{r}}_2 \\ \dot{\phi}_2 \end{bmatrix} =$$

$$2\left(\dot{d}_2(t)\right)^2 + \underbrace{2(\underline{d}_2)^T\left(\dot{\phi}_1^2 \cdot \underline{\underline{A}}_1 \begin{bmatrix} 0 \\ 0.08 \end{bmatrix}\right) - 2(\underline{d}_2)^T\left(\dot{\phi}_2^2 \cdot \underline{\underline{A}}_2 \begin{bmatrix} 0 \\ 0.08 \end{bmatrix}\right) - 2(\underline{\dot{d}}_2)^T(\underline{\dot{d}}_2)}_{\text{Entries in } \gamma \text{ matrix}} \tag{3.10}$$

Now, the revolute joints are addressed. First the one between ROV and body 1, then the one between body 1 and body 2. A revolute joint, which is shown in Fig. 3.13, is a hinge between two bodies. The points P and Q must have the same position, velocity, and acceleration for the joint to be satisfied.

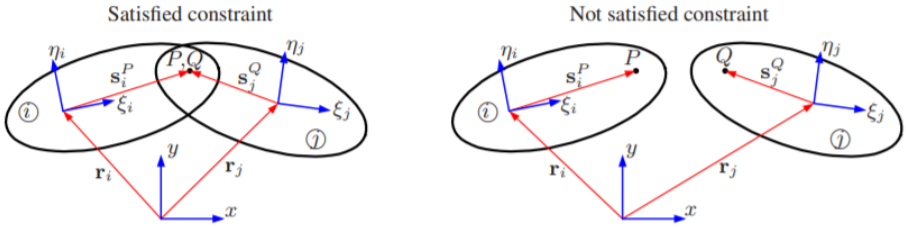


Figure 3.13: Revolute joint

$$\underline{\Phi} = \underline{r}_1 + \underline{\underline{A}}_1 \begin{bmatrix} -0.5 \cdot L_1 \\ 0 \end{bmatrix} = \underline{0} \tag{3.11}$$

$$\underline{\dot{\Phi}} = \underline{\dot{r}}_1 + \dot{\phi}_1 \cdot \underline{\underline{B}}_1 \begin{bmatrix} -0.5 \cdot L_1 \\ 0 \end{bmatrix} = \underline{0} \Downarrow$$

$$\underbrace{\begin{bmatrix} \underbrace{I}_{\text{Identity matrix}} & \underline{\underline{B}}_1 \begin{bmatrix} -0.5 \cdot L_1 \\ 0 \end{bmatrix} \end{bmatrix}}_{\text{Entries in Jacobian matrix}} \begin{bmatrix} \underline{\dot{r}}_1 \\ \dot{\phi}_1 \end{bmatrix} = \underline{0} \tag{3.12}$$

$$\underline{\ddot{\Phi}} = \underline{\ddot{r}}_1 + \ddot{\phi}_1 \cdot \underline{\underline{B}}_1 \begin{bmatrix} -0.5 \cdot L_1 \\ 0 \end{bmatrix} - \dot{\phi}_1^2 \cdot \underline{\underline{A}}_1 \begin{bmatrix} -0.5 \cdot L_1 \\ 0 \end{bmatrix} = \underline{0} \Downarrow$$

$$\underbrace{\begin{bmatrix} I & \underline{\underline{B}}_1 \begin{bmatrix} -0.5 \cdot L_1 \\ 0 \end{bmatrix} \end{bmatrix}}_{\text{Entries in Jacobian matrix}} \begin{bmatrix} \underline{\ddot{r}}_1 \\ \ddot{\phi}_1 \end{bmatrix} = \underbrace{\dot{\phi}_1^2 \cdot \underline{\underline{A}}_1 \begin{bmatrix} -0.5 \cdot L_1 \\ 0 \end{bmatrix}}_{\text{Entries in } \gamma \text{ matrix}} \tag{3.13}$$

Now for the second revolute joint, the operation is executed in the same procedure.

$$\underline{\Phi} = \underline{r}_1 + \underline{\underline{A}}_1 \begin{bmatrix} 0.5 \cdot L_1 \\ 0 \end{bmatrix} - \underline{\underline{A}}_2 \begin{bmatrix} -0.5 \cdot L_2 \\ 0 \end{bmatrix} - \underline{r}_2 = \underline{0} \tag{3.14}$$

$$\underline{\dot{\Phi}} = \underline{\dot{r}}_1 + \dot{\phi}_1 \cdot \underline{\underline{B}}_1 \begin{bmatrix} 0.5 \cdot L_1 \\ 0 \end{bmatrix} - \dot{\phi}_2 \cdot \underline{\underline{B}}_2 \begin{bmatrix} -0.5 \cdot L_2 \\ 0 \end{bmatrix} - \underline{\dot{r}}_2 = \underline{0} \Downarrow$$

$$\underbrace{\begin{bmatrix} I & \underline{\underline{B}}_1 \begin{bmatrix} 0.5 \cdot L_1 \\ 0 \end{bmatrix} & -I & -\underline{\underline{B}}_2 \begin{bmatrix} -0.5 \cdot L_1 \\ 0 \end{bmatrix} \end{bmatrix}}_{\text{Entries in Jacobian matrix}} \begin{bmatrix} \dot{\underline{r}}_1 \\ \dot{\phi}_1 \\ \dot{\underline{r}}_2 \\ \dot{\phi}_2 \end{bmatrix} = \underline{0} \tag{3.15}$$

$$\ddot{\underline{\Phi}} = \ddot{\underline{r}}_1 + \ddot{\phi}_1 \cdot \underline{\underline{B}}_1 \begin{bmatrix} 0.5 \cdot L_1 \\ 0 \end{bmatrix} - \dot{\phi}_1^2 \cdot \underline{\underline{A}}_1 \begin{bmatrix} 0.5 \cdot L_1 \\ 0 \end{bmatrix} - \ddot{\phi}_2 \cdot \underline{\underline{B}}_2 \begin{bmatrix} -0.5 \cdot L_2 \\ 0 \end{bmatrix}$$

$$+\dot{\phi}_2^2 \cdot \underline{\underline{A}}_2 \begin{bmatrix} -0.5 \cdot L_2 \\ 0 \end{bmatrix} - \ddot{\underline{r}}_2 = \underline{0} \Downarrow$$

$$\underbrace{\begin{bmatrix} I & \underline{\underline{B}}_1 \begin{bmatrix} 0.5 \cdot L_1 \\ 0 \end{bmatrix} & -I & -\underline{\underline{B}}_2 \begin{bmatrix} -0.5 \cdot L_1 \\ 0 \end{bmatrix} \end{bmatrix}}_{\text{Entries in Jacobian matrix}} \begin{bmatrix} \ddot{\underline{r}}_1 \\ \ddot{\phi}_1 \\ \ddot{\underline{r}}_2 \\ \ddot{\phi}_2 \end{bmatrix} =$$

$$\underbrace{\dot{\phi}_1^2 \cdot \underline{\underline{A}}_1 \begin{bmatrix} 0.5 \cdot L_1 \\ 0 \end{bmatrix} - \dot{\phi}_2^2 \cdot \underline{\underline{A}}_2 \begin{bmatrix} -0.5 \cdot L_2 \\ 0 \end{bmatrix}}_{\text{Entries in } \gamma \text{ matrix}} \tag{3.16}$$

Now that all the necessary constraints have been addressed, the solving of these equations and the reaction forces is done in MATLAB, and the scripts can be found in Appendix B.10. The reaction forces are solved with the method presented in Sec. 2.3.5: **A System of Constrained Bodies - 2D**. Fig. 3.14 illustrates the points of action, which are needed for understanding the figures showing the reaction forces. The skeletons for these scripts were provided by Morten Kjeld Ebbesen as a part of the course MAS414: Mechanical Systems 2.

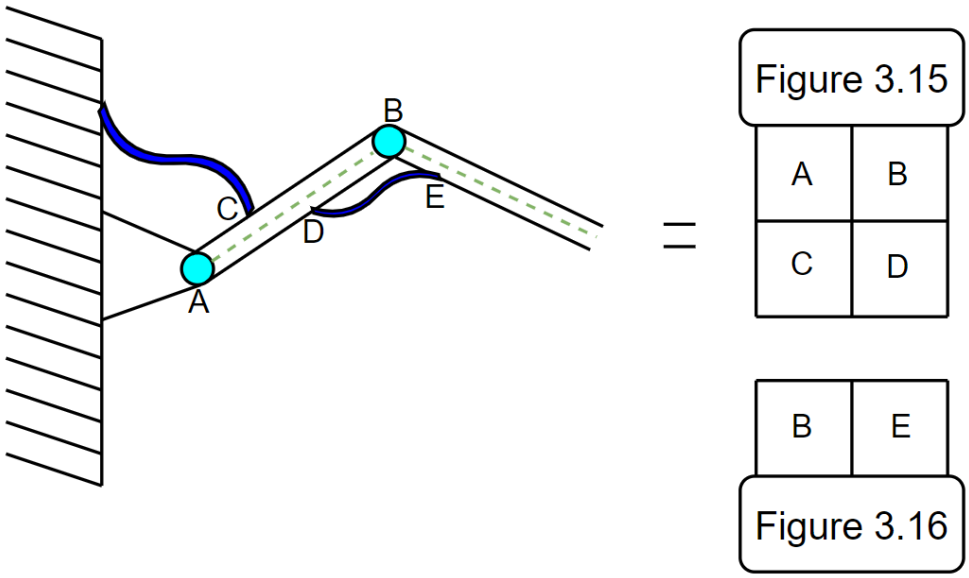


Figure 3.14: Explanation

The figures below show how the reaction forces behave during a timespan of 4 seconds.

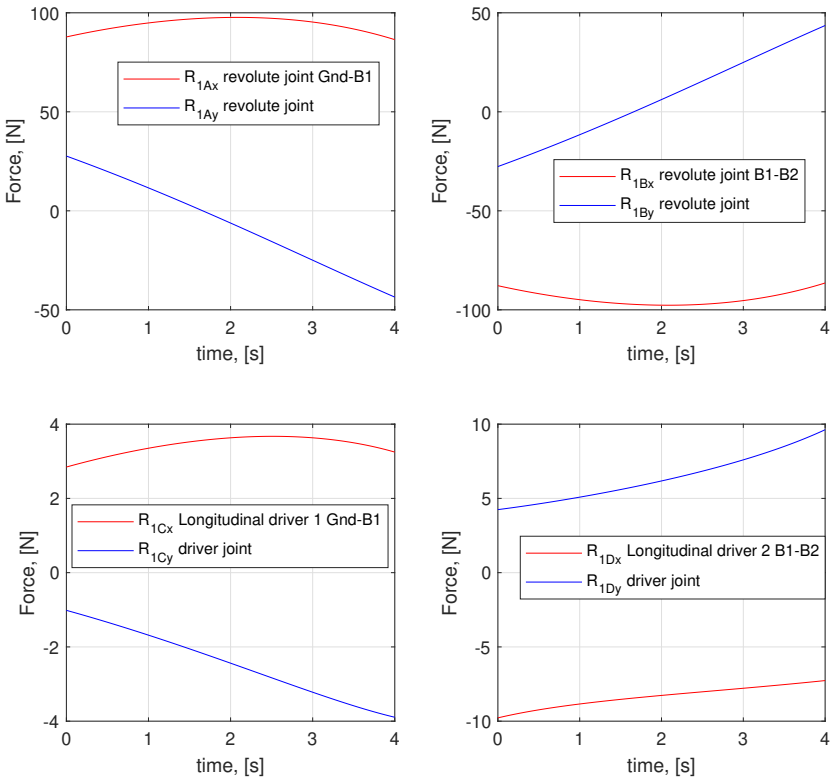


Figure 3.15: Reaction forces on the first link (body 1)

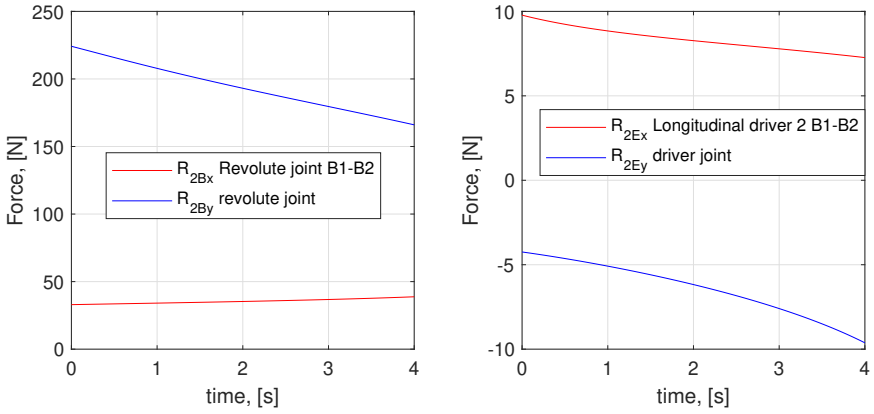


Figure 3.16: Reaction forces on the second link (body 2)

Position, velocity and acceleration

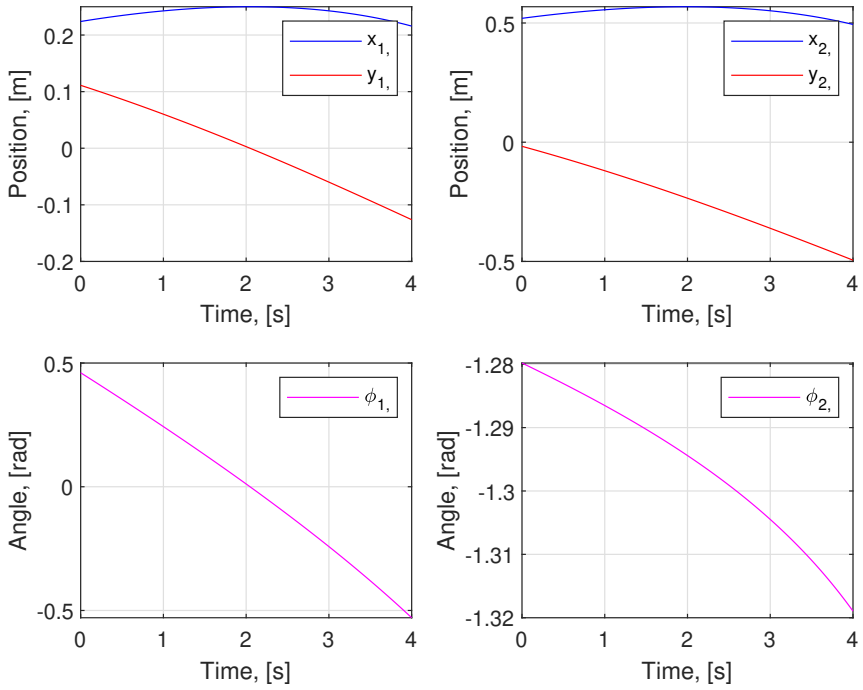


Figure 3.17: Pose for the two links.

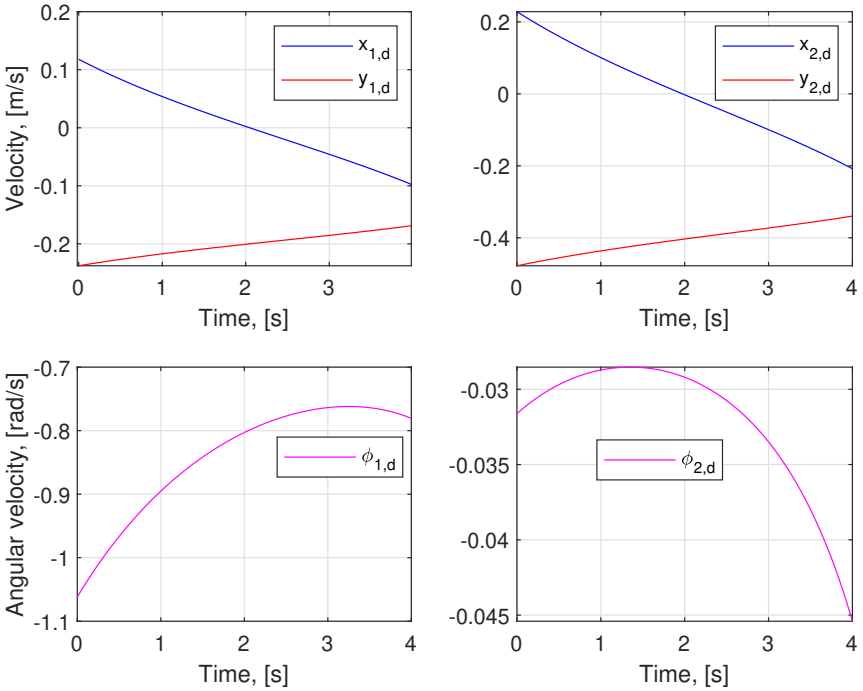


Figure 3.18: Velocities for the two links

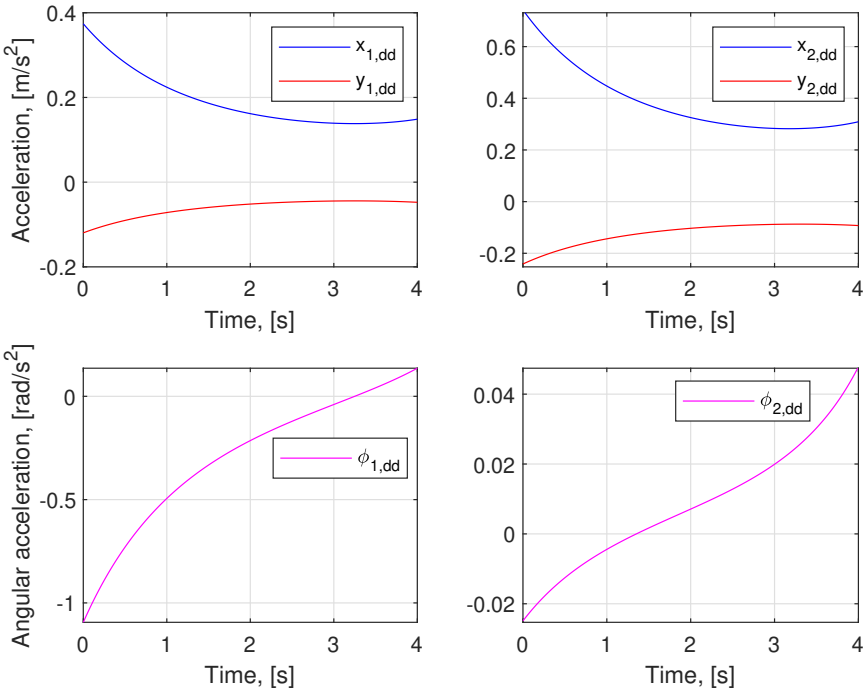


Figure 3.19: Accelerations for the two links

3.3.4 Actuator and Pump Design

This subsection will illustrate what calculations need to be done for the actuator to work adequately. In addition, equations to be utilized for selecting a sufficient pump will also be shown. For many applications used in industry, an open-loop (OL) hydraulic circuit is not adequate, because it is impossible to control the flow going into an actuator. Closed-loop systems (CL) are utilized in such cases. The pump can define the main difference. The OL draws all of its flow from a reservoir and delivers the majority of it back to the reservoir, after executing a specific function. However, the CL draws most of the flow directly from the outlet of an actuator(s). When changing the circuit from OL to CL, this implies that the command signal for the servo valve(s) and the variable pump(s) will be changed from a command signal to a position signal. This leads to another behavior compared to using an open-loop because now the position will be controlled and not the signal [35]. However, servo valves are not used for this project. The only variable to control is the pump. The maximum forces, i.e., the worst-case scenario, the actuators require are computed in MATLAB and are respectively:

- $F_{driver1} = 7.56\,N$ &
- $F_{driver2} = 12.06\,N$

According to John A. Paulson et al. [38], a $1\,kg$ weight can be effortlessly lifted in the air by a cylindrical muscle using water as the internal fluid (flow rate: $80\,mL/min$). Furthermore, they claim that air is the most reachable fluid for making a lightweight artificial muscle, and the surrounding water can be directly used for actuation in an underwater environment.

The artificial muscles are going to be controlled by Pump Control, and the method is obtained from MAS410: Hydraulic Components and Systems. This is based on controlling the flow into the actuator. This is a suitable method because it has:

- No power losses ($p_P = p_A$ and $Q_P = Q_A$)
- Constant velocity

The pump which is to be used is variable and bi-rotational. The process has two operations:

1. Moving downwards to insert the end-effector to the derelict pot
2. Returning to a desired position, depending on the operator maneuvering the ROV and the arm (the overall system)

In this case, only point 1 is considered. Depending on how much force the actuator requires, a suitable pump and cylinder can be selected. To clearly illustrate and clarify how the cylinder is modeled, an ISO schematic of the hydraulic circuit is shown in Fig. 3.20.

In order to increase the flexibility of the system by having the opportunity to adjust the piston velocity during extension easily, the use of flow control valves (FCVs) should be considered. However, the group finds it of higher importance to complete the given task, instead of controlling the velocity.

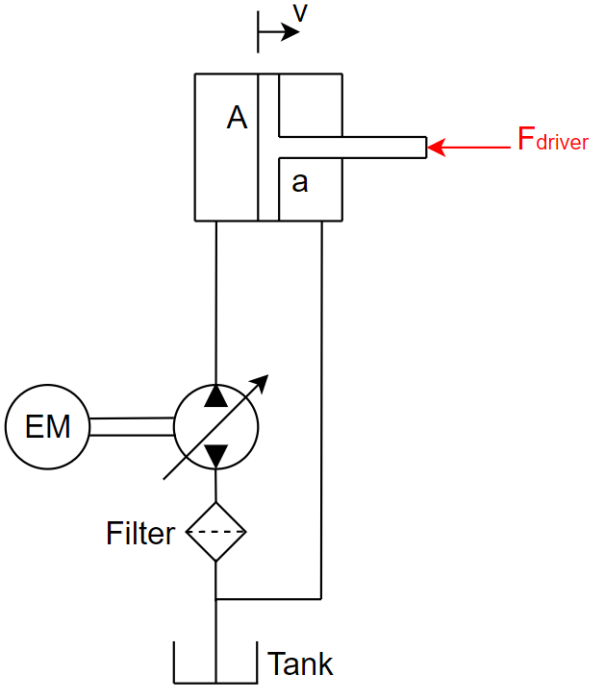


Figure 3.20: Circuit

Fig. 3.20 shows the hydraulic circuit. The following equations are utilized:

$$A = \frac{\pi}{4} \cdot (d_{bore})^2 \tag{3.17}$$

$$A_{rod} = \frac{\pi}{4} \cdot (d_{rod})^2 \tag{3.18}$$

$$a = A - A_{rod} \tag{3.19}$$

The table below lists the known parameters of the system.

Symbol:		Description [unit]:	Value:
d_{bore}	-	Bore side diameter $[m]$	30e-3
d_{rod}	-	Cylinder rod diameter $[m]$	15e-3
v	-	Piston velocity $[m/s]$	0.05
A	-	Area of bore side $[m^2]$	7.069e-4
A_{rod}	-	Area of rod $[m^2]$	1.767e-4
a	-	Area of rod chamber $[m^2]$	5.301e-4

By using Newton's second law, the following can be obtained:

$$\sum F = \underbrace{m \cdot \cancelto{0}{\ddot{x}}}_{\ddot{d}_1(t) = \ddot{d}_2(t) = 0} \quad \Downarrow$$

Since the only operating condition under investigation is the piston extension with resistant load, then the pressure due to the external load is defined as:

$$\begin{aligned} p_P \cdot A - \underbrace{\cancelto{0}{p_a} \cdot a}_{p_a = 0} &= F_{driver} \Downarrow \\ p_P &= \frac{F_{driver}}{A} \end{aligned} \tag{3.20}$$

Furthermore, Eq. 3.21 − Eq. 3.25 are necessary equations for finding a suitable pump:

$$\alpha_P = \frac{D_P}{D_{P_{Max}}} \tag{3.21}$$

$$Q_{th,P} = (\alpha_P \cdot D_{P_{Max}}) \cdot n_P \tag{3.22}$$

$$T_{th,P} = \frac{(\alpha_P \cdot D_{P_{Max}}) \cdot \Delta p_P}{2\pi} \tag{3.23}$$

$$Q_P = Q_{th,P} \cdot \eta_{vP} \tag{3.24}$$

$$T_P = \frac{T_{th,P}}{\eta_{hmP}} \tag{3.25}$$

where:

Symbol:		Description [unit]:	Value
α_P	-	Displacement setting. For "standard" units $\rightarrow 0.25 \leq \alpha \leq 1$ $[-]$	
$Q_{th,P}$	-	Max theoretical flow. Obtained from datasheet $[L/min]$	
n_P	-	Max speed. Obtained from datasheet $[RPM]$	
$T_{th,P}$	-	Theoretical torque. Obtained from datasheet $[Nm]$	
Δp_P	-	Pressure pump. Obtained from datasheet $[bar]$	
Q_P	-	Max required flow by pump $[L/min]$	
η_{vP}	-	Volumetric efficiency $[-]$	
T_P	-	Pump torque $[Nm]$	
η_{hmP}	-	Hydro mechanical efficiency $[-]$	

From Eq. 3.20, the amount of pressure needed from the pump can be calculated. After this value is obtained, a suitable pump that delivers this pressure can be found by using catalogs. Even if the pump delivers more flow than required by the cylinders, it can be adjusted by the displacement setting. The displacement setting, α, is adjusted in order to change both the direction of the flow and the amount.

Since the procedure is only going to be simulated, selecting specific pumps and actuators is not presented here, only the necessary calculations. The flow needed from the pump can be calculated based on the actuator's speed. With this information, it can be concluded that for the first actuator, a pump with the following characteristics can be used:

- $Q_{P1} = v \cdot A = 3.534 \cdot 10^{-5}\, m^3/s = 2.12\, L/min$
- $p_{P1} = F_{driver1}/A = 10693.7\, Pa = 0.10\, bar$

while for the other actuator, the pump need to have (same procedure):

- $Q_{P2} = 2.12\, L/min$
- $p_{P2} = 0.17\, bar$

The plan was to design, fabricate, and utilize an artificial muscle as presented in Sec. 3.3.2. However, due to the difficulties of access to the university, the group needed to make a simulation of the gathering procedure, more of this in the Discussion. Consequently, the soft artificial muscle was not implemented in the simulation. For that reason, a traditional hydraulic cylinder needed to be employed. So, to summarize this subsection, the following has been done:

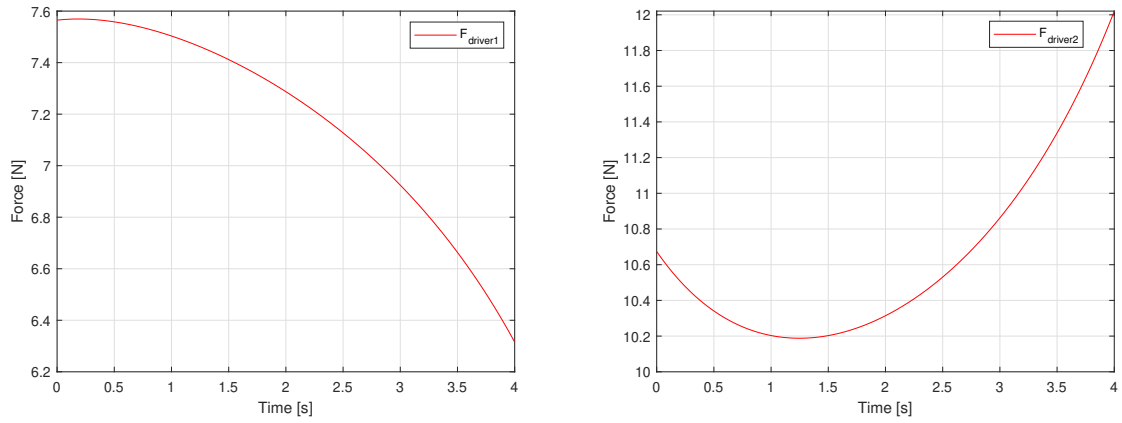
First, an actuator with a bore side diameter of $30\, mm$ and a rod diameter of $15\, mm$ was determined to be used. Then, calculations whether this was sufficient or not were done. To be certain that the selection of the actuator is correct, the following equation is used:

$$A > \frac{F_{driver,x}}{p_{Max}} \tag{3.26}$$

where:

Symbol:		Description [unit]:	Value
A	-	Piston-side area $[m^2]$	30e-3
$F_{driver1}$	-	Max required force from 1st actuator $[N]$	7.559
$F_{driver2}$	-	Max required force from 2nd actuator $[N]$	12.06
p_{Max}	-	Max pressure delivered from pump. Using relative small pumps $[Pa]$	6e5

From Eq. 3.26, it can be obtained that the higher the maximum pressure delivered by the pump is, the smaller the overall value in the fraction. This means that the actuator with the chosen specifics is acceptable.



(a) First actuator, ROV-body 1

(b) Second actuator, body 1-body 2

Figure 3.21: The behavior of both actuators

The behaviour of the actuators can be seen in respectively Fig. 3.21a and Fig. 3.21b.

3.3.5 Joint Examination for Reusability

It might be of interest to examine whether the revolute joint between the two links is in the danger zone of being no longer reusable after a short while. A calculation will be performed on the area where the bolts experience the highest forces. The mechanical links are 40 mm in diameter, so choosing a M20 8.8 bolt, with the purpose to find out how much this bolt can handle. The bolt functions as a revolute joint. Fig. 3.22 shows the link with the joint that has the worst-case scenario, i.e. highest forces occur in revolute joint between body 1 and body 2. This is where the calculation will be performed.

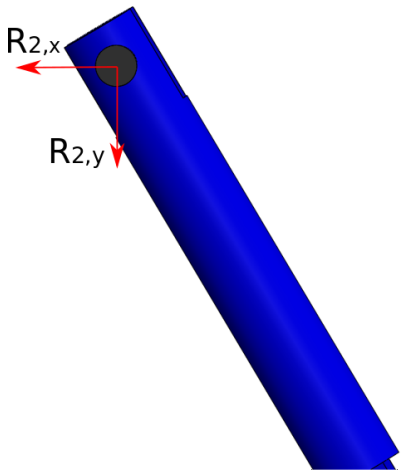


Figure 3.22: Second link of the robot arm

Applying the Pythagorean theorem:

$$R_2 = \sqrt{(R_{2,x})^2 + (R_{2,y})^2} \tag{3.27}$$

The only forces acting on the bolt is R_2 and now it is possible to examine if the bolt will be overloaded or if it will handle this force. The bolt is as mentioned 20 mm.

$$\tau = \frac{R_2}{2A} \tag{3.28}$$

$$\tau = \frac{R_2}{2\pi r^2}$$

where:

Symbol:		Description [unit]:	Value:
r	-	Radius of the bolt $[mm]$	10
$R_{2,x}$	-	Force along the X-axis in hole $[N]$	38.78
$R_{2,y}$	-	Force along the Y-axis in hole $[N]$	224.20
R_2	-	Diagonal force, the force acting on the bolt $[N]$	227.53
τ	-	Shear stress on the bolt $[MPa]$	0.36

Since a 8.8 bolt is used, the yield strength is given from the last number $\Rightarrow \sigma_F = 800 \cdot 0.8 = 640\,MPa$. The shear stress is far below the yield strength. In order to investigate if the bolt is in the region of usability, a section from an ISO Metric table is shown below:

Nominal Diameter Of Thread d $[mm]$	**Pitch P** $[mm]$	**Nominal Stress Area As** $[mm^2]$	**Proof Load** $[N]$
20	2.50	245	147000

Table 3.2: An ISO Metric table which illustrates information about bolts [14]

With the information gathered from Tab. 3.2, it is certain that the bolt will handle the force $R_2 = 227.53\,N$. The proof load is $147000\,N$ for a 8.8 bolt and this means that the bolt can handle a force up to this value. If this limit is exceeded, the bolt can not be reused.

3.3.6 Buoyancy Investigation

For the reason that everything will occur underwater, buoyancy should be investigated, (see Sec. 2.5 for theory). It is of interest to examine whether the links will struggle to perform the necessary operations due to buoyant force.

$$F_{buo} = \rho_f V g \tag{3.29}$$

$$m_{tot} = m_{ee} + m_{steel} \tag{3.30}$$

$$F_{net} = m_{tot} g \tag{3.31}$$

where:

Symbol:		Description [unit]:	Value:
F_{buo}	-	Buoyant force $[N]$	5.07
ρ_f	-	Density of seawater $[kg/m^3]$	1029
V	-	Volume of the displaced body of liquid $[m^3]$	5.0265e-4
g	-	The gravitational acceleration $[m/s^2]$	9.81
m_{tot}	-	Total mass $[kg]$	6
m_{ee}	-	Mass of end-effector $[kg]$	2
m_{steel}	-	Mass of steel $[kg]$	4
F_{net}	-	The net force $[N]$	58.86

Since $F_{net} > F_{buo}$, buoyancy will not be an issue. However, in the simulation part, buoyancy is not taken into consideration. Although buoyancy will not be an issue, it is undoubtedly a factor contributing to creating deviations in the simulation. More of this in Discussion

3.4 End-Effector Model Description

The push-button will have one DOF, allowing for movement inwards and outwards. Same for the carabiner, which is rotating around a pivot point. All the other elements in the end-effector are to be considered passive.

3.4.1 Assembly

The end-effector is composed of the inflator body assembled with the carabiner, a CO_2 cylinder and a self-made box. In Sec. 3.2.2, the box was not visually shown in a detailed perspective. Designing the end-effector in such a manner that when the carabiner opens, activation of the balloon is one of the main objectives in this project, referring to the second research question *"Can we design a custom-made end-effector that can be utilized for retrieving marine litter?"* Several methods were designed and investigated its functionality, and those are introduced here. All concepts are designed in SolidWorks.

Method 1

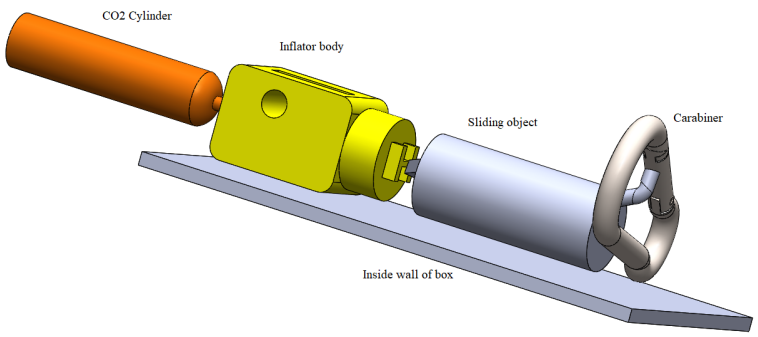
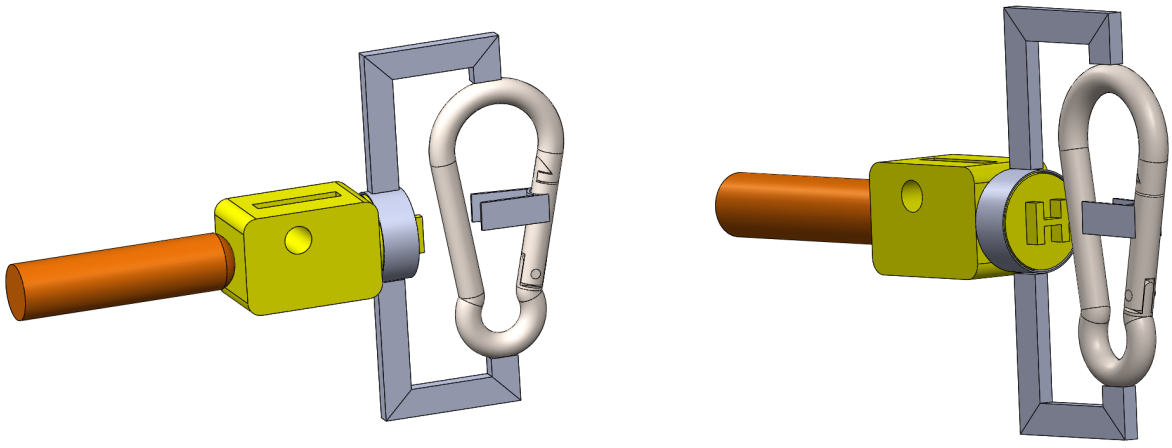


Figure 3.23: Method 1

Fig. 3.23 shows how the first method was considered. The sliding object will have the ability to slide back and forth since one end is attached to the carabiner. When the carabiner opens, the sliding object will be moved to the left, and the tip of the sliding object will push the button, which will activate the CO_2 cylinder. Then the balloon (not shown in the figure) will expand, and the derelict pot will be lifted to the sea surface.



(a) Back view

(b) Front view

Figure 3.24: Method 2

Method 2

Fig. 3.24 shows both the back and front view of the second method. Neither the box nor the balloon is displayed. The principle behind this method is that an "A-form" object is welded on the carabiner. When the carabiner opens, the button will be pushed by the welded object. Another object (also welded) is present to ensure that the carabiner is fastened and will not slide. Fig. 3.25 shows the complete release mechanism.

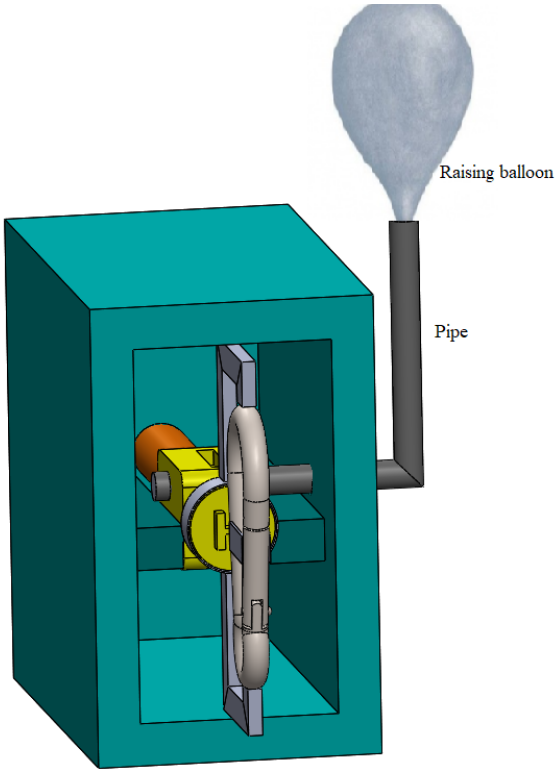


Figure 3.25: Method 2 complete

Method 3

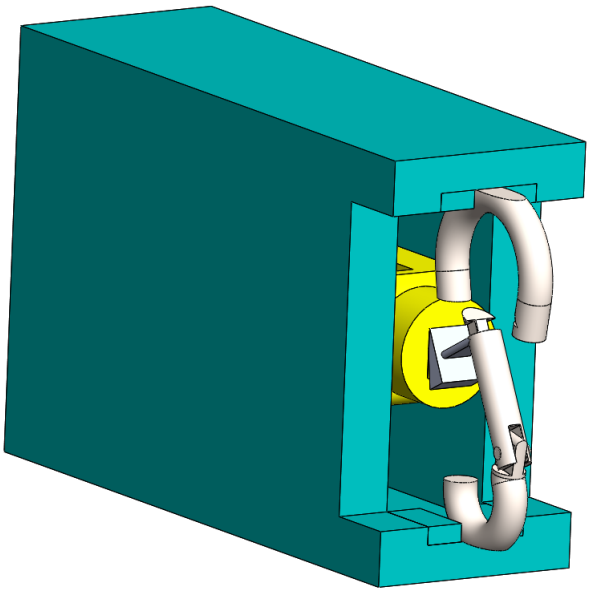


Figure 3.26: Method 3

Fig. 3.26 shows method 3 without the pipe and the raising balloon. It is quite similar to method 2, except for some modifications that have been done. The carabiner has been cut in order to avoid using the C-shaped profile. Here the carabiner is pressed onto the box. The complexity of method 1 and the unnecessary component of method 2 led to the selection of method 3.

Problems occurred when method one was tested. The sliding object used to get stuck at times and did not work very well. An additional pipe has been added, and its function is to send the air from the CO_2 cylinder through the inflator body to the party balloon (not shown in Fig. 3.26). However, the balloon must have a discharge valve/release valve so that air is released when pressure is reduced on the way up. A simple party balloon does not have this accessory. The idea of this selected method is to ensure that the carabiner is connected to the box via "press-fit." The carabiner will not slide sideways due to the tracks. Besides, to prevent the carabiner from sliding, the box has been cut in such a manner that no object will collide with it.

3.4.2 Collision Detection Sensor

As mentioned in Sec. 1.2.3, there are different types of this kind of sensor. The one shown in Fig. 3.27 is microcontroller compatible and can be used for robot collision detection or touch collision detection. This is a simple, yet effective solution that may be utilized. Some kind of mechanism/object needs to be designed in such a manner that when the carabiner opens, the red button will be pushed by this mechanism, and hence provide feedback. This detection sensor functions in such a manner that when a collision is detected, output=0; when the switch is released, output=1. The information can then further be interpreted whether the end-effector has been attached or not.

Figure 3.27: Collision crash sensor detector

3.5 Overall Control of the System

When considering the system as a whole, it has been assumed that the ROV can be controlled by itself; therefore, the focus will be on the manipulator and how to control it. The ROV is consequently considered as ground, as has been done in the kinematics section where the reference frame is set at the center of the revolute joining the ROV and the first link together. The idea for the actuation of the two revolute joints was to be actuated by elongation of a hydraulic cylinder. With these considerations in mind, the goal is to control the position of the end-effector. This can be done following the conversion in Fig. 3.28 where 'Y' and 'Z' are the desired position in the plane of operation, 'q_0' and 'q_1' are the desired joint angles to accommodate this position and 'L_1' and L_2' are the cylinder lengths required to achieve these angles. Fig. 3.28 shows the conversion from desired Cartesian coordinates to end-effector position by cylinder length control.

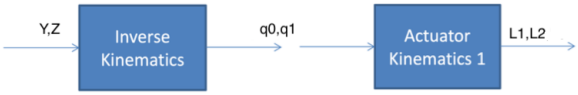


Figure 3.28: Conversion

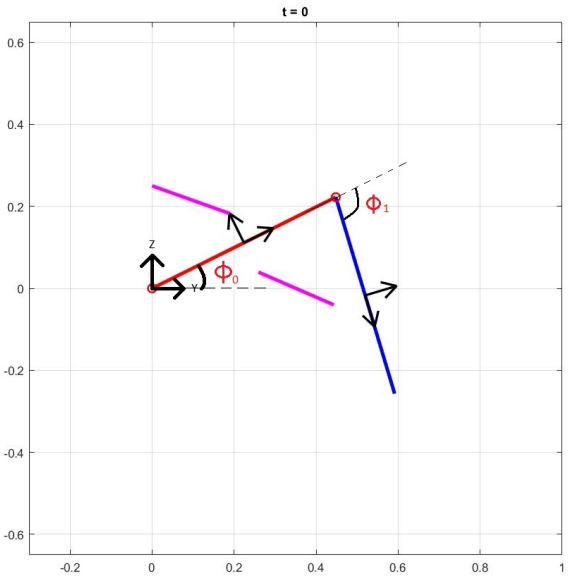


Figure 3.29: Simplified geometry

The method, shown in Fig. 3.28, would be applied on the manipulator geometry in Fig. 3.29. Fig. 3.29 shows a simplified geometry where the red line represents the first link, the blue line the second link, the purple lines represent each driver, the labeled coordinate system represents the reference frame and the two unlabeled represent each links' local coordinate system. The units on both axes are in *meters*.

3.5.1 Inverse Kinematics

Following the structure in Fig. 3.28 is the conversion from the end-effector position to joint angles, the first to be conducted. Take note that the end-effector local coordinate frame has not been added to Fig. 3.29, although it is supposed located at the end of link 2. Taking this into consideration, the position of the end-effector can be represented by the configuration of the two angles 'ϕ_0' and 'ϕ_1',

$$Y_{ee} = L_1 \cdot \cos(\phi_0) + L_2 \cdot \cos(\phi_0 + \phi_1) \tag{3.32}$$

$$Z_{ee} = L_1 \cdot \sin(\phi_0) + L_2 \cdot \sin(\phi_0 + \phi_1) \tag{3.33}$$

Where:

Y_{ee} is the y-coordinate of the end-effector	$[m]$
Z_{ee} is the z-coordinate of the end-effector	$[m]$
ϕ_0 is the angle between the reference frame and the first link	$[rad]$
ϕ_1 is the angle between the frame of the first link and the second link	$[rad]$
L_1 is the length of the first link	$[m]$
L_2 is the length of the second link	$[m]$

Having two equations with two unknowns and applying the identity of addition for two angles, can the configuration of the joint angles in order to reach the desired end-effector position, be calculated.

$$\phi_1 = \cos^{-1}\left(\frac{Y_{ee}^2 + Z_{ee}^2 - L_1^2 - L_2^2}{2 \cdot L_1 \cdot L_2}\right) \tag{3.34}$$

$$\phi_0 = \tan^{-1}\left(\frac{Z_{ee}^2}{Y_{ee}^2}\right) - \tan^{-1}\left(\frac{L_2 \cdot \sin(\phi_1)}{L_1 + L_2 \cdot \cos(\phi_1)}\right) \tag{3.35}$$

3.5.2 Actuator Kinematics

From the inverse kinematics are the joint angles required to reach the desired Cartesian point defined. Since the two driven angles are actuated by linear actuators, the conversion from joint angles to actuator lengths must be conducted. The proposed geometries are the lengths defining the simplified geometry in Fig. 3.29. The vectors derived in Sec. 3.3.3, which represent the length of each actuator, may be described solely by ϕ_0 and ϕ_1 by substituting the r_1 and r_2 vectors by the first and second row of the Φ matrix.

$$d_1 = A(\phi_0)\begin{bmatrix}\frac{L_1}{2}\\0\end{bmatrix} + A(\phi_0)\begin{bmatrix}0\\0.08\end{bmatrix} - \begin{bmatrix}0\\0.25\end{bmatrix} \tag{3.36}$$

$$d_2 = A(\phi_0)\begin{bmatrix} L_1 \\ 0 \end{bmatrix} + A(\phi_1)\begin{bmatrix} \frac{L_2}{2} \\ 0 \end{bmatrix} - A(\phi_1)\begin{bmatrix} 0 \\ 0.08 \end{bmatrix} - A(\phi_0)\begin{bmatrix} \frac{L_1}{2} \\ 0 \end{bmatrix} + A(\phi_0)\begin{bmatrix} 0 \\ 0.08 \end{bmatrix} \tag{3.37}$$

Both Eq. 3.36 and Eq. 3.37 are the vector description of the cylinders. It is therefore required to convert these vectors into absolute lengths. The simplified description of the absolute lengths are the following:

$$l_1(t) = \sqrt{d_1' d1} \tag{3.38}$$

$$l_2(t) = \sqrt{d_2' d_2} \tag{3.39}$$

Where:
d_1' and d_2' are the transpose of each respective vector
$l_1(t)'$ and $l_2(t)'$ are the command signal for each cylinder

The command signals to each cylinder can now be calculated using Eq. 3.38 and 3.39 from the inverse kinematics according to the desired Cartesian coordinates, and therefore the arm's tooltip can be controlled. However, it is important to keep in mind that this analysis and the forward kinematics are considering the end of link 2, not including the end-effector. For the idea of a fixed position of the end-effector relative to the end of link 2, the calculations and methods can be modified to control, e.g., the position of the midpoint on the movable hook by adding the constant displacement from the relative coordinate frame of link 2.

3.6 Modeling and Simulation

This section describes the applied methods in ROS and the structure of the simulation. It is highly recommended to investigate the resources and tutorials provided by the ROS community in order to understand ROS's capabilities. The workspace in which the package development has to be set up correctly according to the conventions for a catkin workspace (click here for further information).

3.6.1 Packages

All the packages which the simulated arm depends on are listed in Tab. 3.3. The amount can be significantly reduced, as many of them are convenience tools. How they depend on each other can be seen by running the command:

```
rqt_dep
```

when the corresponding path to the package of interest is sourced. However, with this many dependencies, the graphical overview can be quite complex. Running the command:

```
rosdep install --from-paths src --ignore-src -r -y
```

will install all the packages which a package depends upon [32].

media_export	rosservice	rosout
geometry_msgs	message_generation	genlisp
image_transport	roswtf	rosparam
sensor_msgs	rosbuild	gennodejs
visualization_msgs	actionlib_msgs	rosunit
joint_state_publisher	rosconsole	geneus
interactive_markers	rostest	roslaunch
orocos_kdl	std_msgs	gencpp
angles	actionlib	topic_tools
rosconsole_bridge	rosgraph_msgs	message_runtime
laser_geometry	tf2_msgs	roslz4
class_loader	xmlrpcpp	roscpp_serialization
nav_msgs	tf2	rosbag_storage
pluginlib	roscpp	roscpp_traits
map_msgs	tf2_py	std_srvs
urdf	rosgraph	rostime
python_qt_binding	tf2_ros	rosbag
kdl_parser	ros_environment	cpp_common
resource_retriever	tf	rostopic
message_filters	rospack	genpy
rviz	tf2_kdl	rosnode
rosclean	roslib	genmsg
xacro	robot_state_publisher	rosmsg
rosmaster	rospy	catkin

Table 3.3: Package dependencies

The software required to follow this method is listed in Tab. 3.4.

Software	
OS	Ubuntu LTS 18.04.4 LTS (Bionic Beaver)
Physics Engine	Gazebo
Modeling	ROS Melodic
Communication / interface	ROS Melodic

Table 3.4: Software required

3.6.2 Unified Robot Description Format

After configuring the workspace and installing all the relevant packages, the process of creating a simulated robot may commence. The URDF is where the model is defined. The term model includes all the links and simulated hardware which constitutes the robot. The URDF is written in XML language, a markup language which can define a format for how an element should be described [45].

Listing 3.1: Link example

```
1  <!--First_link-->
2          <link name="first_link">
3                  <visual>
4                          <geometry>
5                                  <cylinder length="${len}" radius="${
                                       radius}"/>
6                          </geometry>
7                          <origin xyz="0 ${len/2 + radius} 0" rpy="${
                               pi/2} 0 0"/>
8                          <material name="grey"/>
9                  </visual>
10                 <collision>
11                         <geometry>
12                                 <cylinder length="${len}" radius="${
                                       radius}"/>
13                         </geometry>
14                         <origin xyz="0 ${len/2 + radius} 0" rpy="${
                               pi/2} 0 0"/>
15                 </collision>
16                 <xacro:default_inertia_rod masse="3" />
17         </link>
```

From the code snippet in Listing 3.1, it can be seen how rigid links are defined. Line 2 states that we are to define a link in the URDF. This leads to all the subtags of modifiable parameters within the link initiation tag and the end of link tag. The three main elements of a link definition are: visuals, collision and physical properties. The visual definition of the link is self-explanatory. The collision tag defines the space of the link and how it should interact if in collision with an other link. The final tag specifies the physical properties of the link such as weight and inertia. As can be seen in line 16, is a macro used in order to assign the elements within the physics tag for a massive rod. By constructing macros, the amount of written code can be reduced and thus make the URDF more clean.

Listing 3.2: Xacro example

```
<!--Inertia and mass of massive rod-->
        <xacro:macro name="default_inertia_rod" params="masse">
                <inertial>
                        <mass value="${masse}" />
                        <inertia ixx="${inertiaH}" ixy="${inertiaH}"
                            ixz="${inertiaH}"
                            iyy="${inertiaH}" iyz="${inertiaH}"
                            izz="${inertiaL}" />
                </inertial>
        </xacro:macro>
```

In order to create a macro like the one in Listing. 3.2, the package 'xacro' must be added to the workspace of the package [33]. This macro takes the parameter of *masse* and fills in the tags for the inertial definition of the link. The parameters in the inertia tag is calculated based on the inertia of a massive cylinder with another xacro feature; the property function.

Listing 3.3: Property example

```
<!--Inertia along 'hard' axis, massive cylinder-->
        <xacro:property name="inertiaH" value="${mass*radius*radius
            /2}"/>
```

The xacro property feature is used to define a single data type unlike the macro feature which can assign multiple. Listing. 3.3 depicts the calculation of the inertia for the 'hard' axis of a massive cylinder.

Listing 3.4: Joint example

```
        <!--Revolute joint Gnd to Link 1-->
        <joint name="rev_to_first_link" type="revolute">
                <parent link="base_link"/>
                <child link="first_link"/>
                <origin xyz="0 0.27 0"/>
                <axis xyz="1 0 0"/>
                <limit effort="1000.0" lower="-0.548" upper="0.548"
                    velocity="0.5"/>
        </joint>
```

The next building block of the model is the addition of joints. In Listing. 3.4 is the definition of the revolute joint connecting the first link to the ROV (or ground), as the ROV is defined. First of all is the type of joint defined. This specifies the elements which have to be defined in order to create the proper joint. The parent link is the reference frame for which the joint will place the child link frame and is done in the origin tag in the joint definition. The frames are the origins defined for each of the respective links. The axis tag specifies which axis is unconstrained in the child link. As for the revolute in Listing. 3.4, the unconstrained axis is the x-axis. Limits can also be imposed to effort (force/torque), movement (in radians) and velocity.

Listing 3.5: Actuation example

```
1          <!--First Revolute-->
2    <transmission name="rev_first_link">
3         <type>transmission_interface/SimpleTransmissions</type>
4         <actuator name="$motor_rev_joint">
5              <mechanicalReduction>1</mechanicalReduction>
6         </actuator>
7         <joint name="rev_to_first_link">
8              <hardwareInterface>hardware_interface/
                   EffortJointInterface</hardwareInterface>
9         </joint>
10     </transmission>
```

The final building block of the URDF file is the actuation of relevant joints. The actuators are defined in the URDF as is shown in Listing. 3.5. Line 3 specifies a transmission type. For simple one joint and one actuator is 'SimpleTransmission' sufficient [48]. This type can represent actuators by reduction or amplification of the transmission, or as it is defined here; a 1:1 [48]. In line 7 and 8 is the joint to be actuated specified and the means of which it is controlled stated, respectively. Position, velocity and effort control are supported. The scheme in Fig. 3.30 illustrates the hardware_interface where the simulated actuators are defined in the loop.

3.6.3 ROS Control

ROS control is a cluster of packages that provide generalized PID controllers and ensures communication from external nodes all the way to the simulation, be it custom made or pre-built controllers [47].

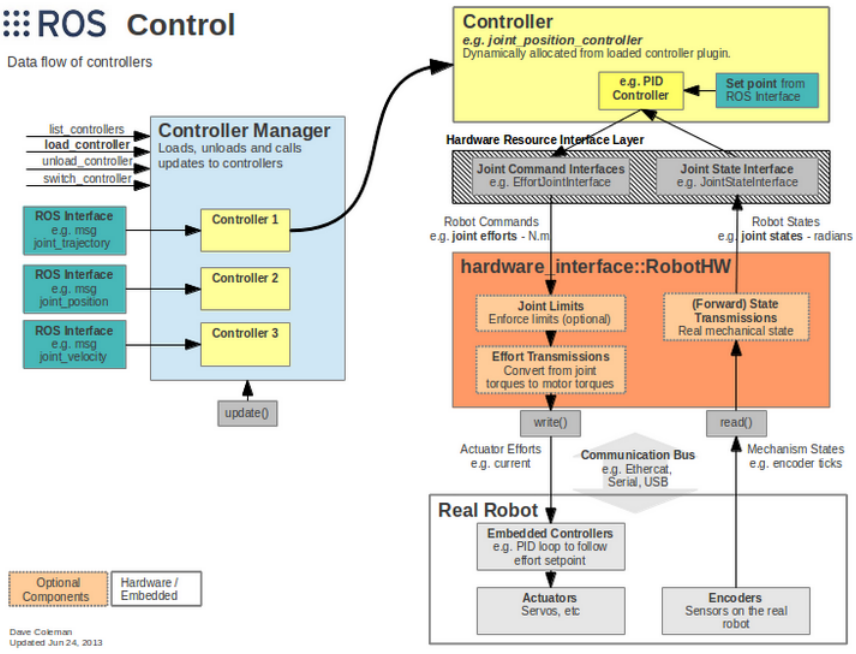


Figure 3.30: Data flow for the package ros_control [7].

The description of control setup will follow the structure in Fig. 3.30. In order for this setup to work, the custom plugin gazebo_ros_control must be added in the URDF description within the robot tag.

Listing 3.6: Gazebo-ROS Control Plugin example

```
1 <!--Gazebo-ROS plugin-->
2         <gazebo>
3                 <plugin name="gazebo_ros_control" filename="
                      libgazebo_ros_control.so">
4                       <robotNamespace>/uw_arm</robotNamespace>
5                 </plugin>
6         </gazebo>
```

A simplified scheme of the data flow for each controller is depicted in Fig. 3.31. The actuator definition is included in the Gazebo model, although to give a more conventional description of the data flow, it is separated from the model. The joints are actuated by torque as is defined in line 8 of Listing 3.5

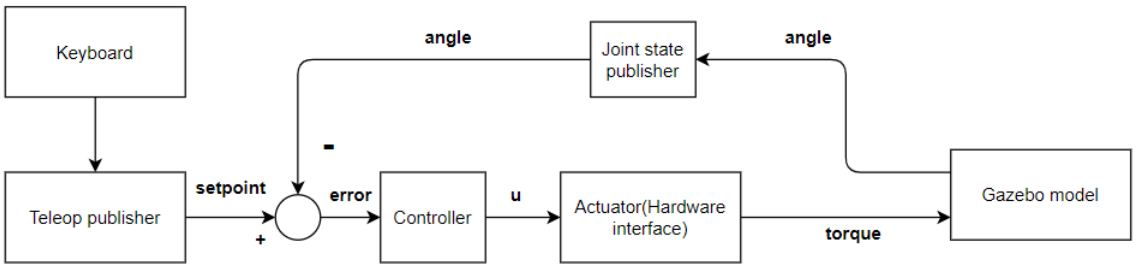


Figure 3.31: Control scheme for the controllers.

Controller Manager and Controller

The controllers are initialized by the Controller Manager package [30]. This must be specified in a launch file.

Listing 3.7: Controller_Launch example

```
1   <!-- Load joint controller configurations from YAML file to
        parameter server -->
2  <rosparam file="$(find uw_arm_control)/config/
       controller_instructions.yaml" command="load"/>
3
4  <!-- load the controllers -->
5  <node name="controller_spawner" pkg="controller_manager" type="
       spawner" respawn="false"
6    output="screen" ns="/uw_arm" args="joint1_position_controller
       joint2_position_controller joint_state_controller"/>
```

In Listing 3.7, the initialization of the controllers and the call to the 'controller_manager' can be seen. In line 2 are parameters such as K_p, K_i, K_d and update_rate for each respective controller and the 'joint_state_publisher', uploaded to a ros parameter server. Those parameters are then acquired by the spawned controllers which are initiated in line 5 and 6. The usage of the YAML file is to store parameters in order to avoid setting them manually for each run of a simulation. After calling the controller_manager and loading the controllers, the PID controllers are ready to control each of the driven joints and the joint_state_publisher is ready to publish the joint states for the joints (acting as a sensor).

Set Point

The set points should be given remotely in order to control the manipulator by an operator. The package 'teleop_twist_keyboard' from Bence Magyar et al. [3] has been used as a basis for the tele operation node in this project. Joint1 is rotated by keyboard inputs '1' and '2' while Joint2 is rotated with the inputs 'q' and 'w'. The pairs of keys represent a given amount of rotation in clockwise and counter clockwise direction. The key inputs may be chosen arbitrarily, as has been done with the amount of rotation for each press of the bound keys. Increments of approximately $5\,deg$ has been chosen as suitable.

Chapter 4

Results

This chapter features the resulting design of the robotic manipulator as well as the resulting behaviour of the Gazebo Model with control system implemented. The behaviour of the Gazebo Model considering how well it responds to the corresponding input values are illustrated in waveform charts.

4.1 The Complete Manipulator

The end-effector can be considered a passive object, but the carabiner and the button are the components with one DOF. To maneuver the robot arm so that the end-effector is inserted onto the derelict pots, the actuators need to extend or contract the arm's links. The amount of extension or contraction depends on the flow that the pumps send to the actuators. Depending on the forces the actuator requires, the amount of pressure and consequently the amount of flow can be determined.

From the video simulation, it can be seen that the links are moving to the derelict pot and are performing an insertion. The cylinder's operation can be more clearly illustrated in the MATLAB simulation. The simulation in MATLAB simulates the first operation, which is to move downwards and insert the end-effector to the derelict pot. Here, the extension of the cylinders can be seen moving with a constant velocity. This is because the group chose to utilize the Pump Control method described in Sec. 3.3.4. Since the movement of the cylinders is constant, this indicates that the method has been applied correctly.

Referring to the first research question, which is: "*Can we design and control a low-cost and easy-to-manufacture robotic arm, which is to be mounted on a ROV?*", the group considers it appropriate to say that this has been accomplished to a certain extent. There is nothing that suggests that the robot links can not be designed in the way introduced in this report. However, if the group had managed to make the prototype work well with hard robotics, further development would be conducted. The developed prototype would then be designed and fabricated to be categorized as a soft robotic. The second question, "*Can we design a custom-made end-effector that can be utilized for retrieving marine litter?*" can also be said to have been finished, since the design allows for relatively easy fabrication of the end-effector. The parts are off-the-shelf products, and nothing suggests that the end-effector will not work. The third question "*Can we create a digital clone that showcases the proposed design, and simulate its operation?*" makes it slightly difficult to determine whether it has been accomplished or not. This is because, in the simulation, the actual ascending is not visualized. Therefore, there remain two options left. One is to build the complete prototype and perform underwater physical experiments. The

second is to develop further the simulating part, which then includes water impacts and other necessary sensors. To further ensure that this would work, the group would focus on developing a physical prototype of what has been introduced in this report. The reason is that simulation and real-life tend to deviate. In other words, if something works perfectly in the simulation, it is not granted that it will work similarly in a real-life scenario.

Controller Performance

Two PID controllers are implemented for the two driven joints of the manipulator. 'Joint1' is the revolute joint between the yellow ROV and the first link. Consequently is 'Joint2' the joint between the first link and the second link.

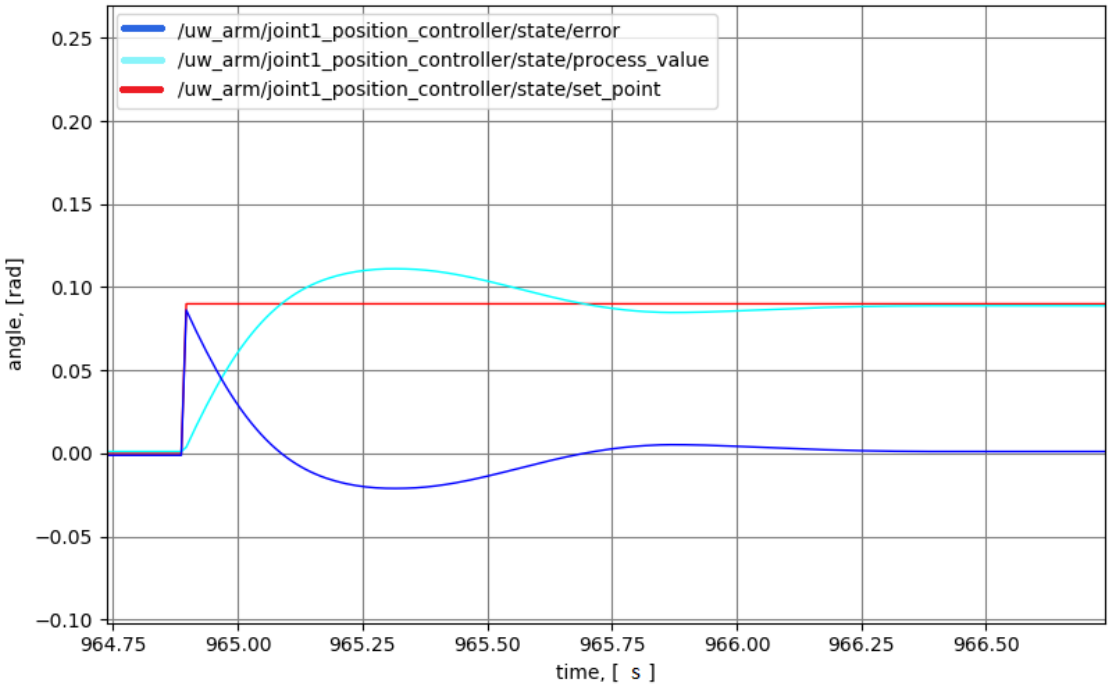


Figure 4.1: The step response for PID, joint1

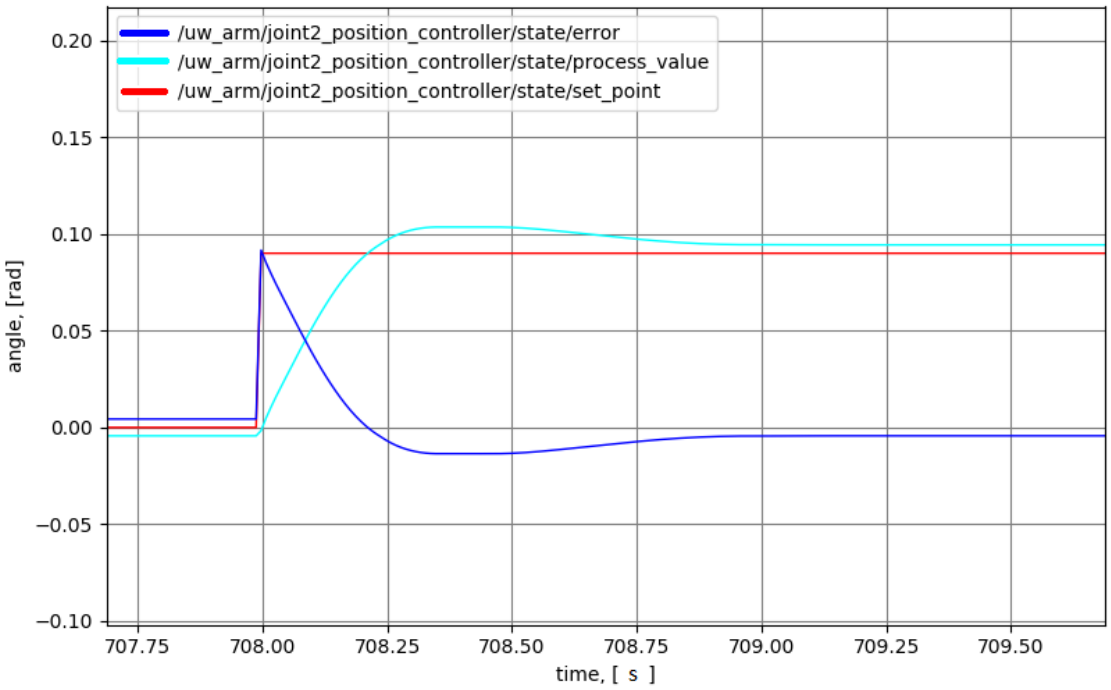


Figure 4.2: The step response for PID, joint2

The performance of the two manually tuned controllers can be seen in Fig. 4.1 and Fig. 4.2. In Fig. 4.3 and Fig. 4.4, the propagated disturbance when moving the other joint can be seen. A step input of $0.09\,rad$ has been used. Characteristics of the two controllers are listed in Tab. 4.1.

Parameters	Joint1	Joint2
Rise time $[s]$	0.2	0.2
Settling time $[s]$	0.48	0.95
Overshoot $[\%]$	22.2	16.6

Table 4.1: Characteristics of the final controllers

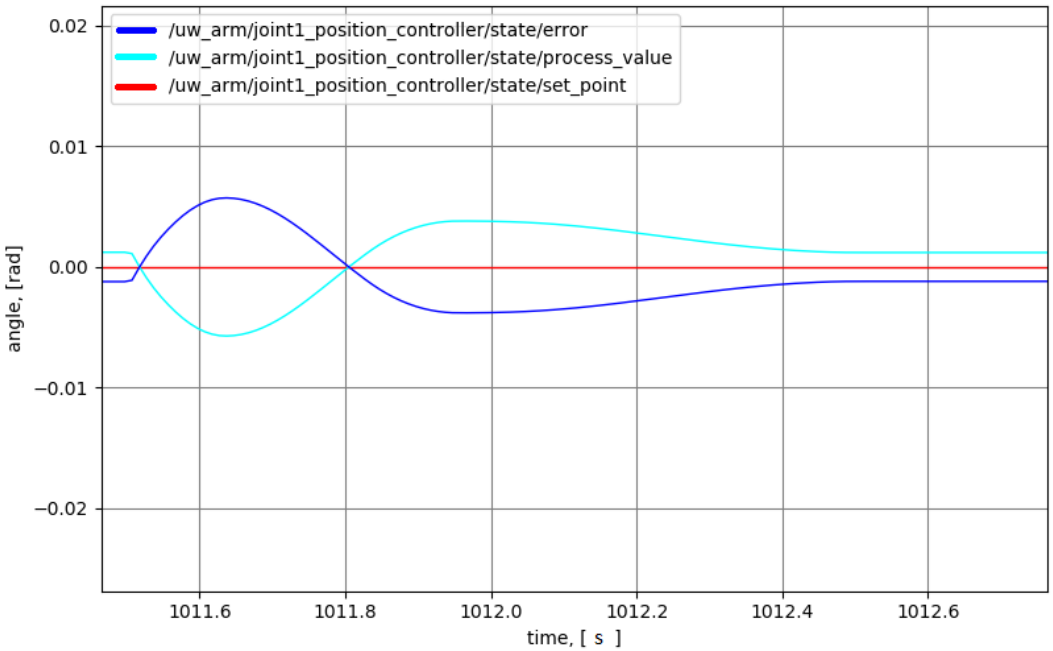


Figure 4.3: The propagated disturbance on joint1 from moving joint2

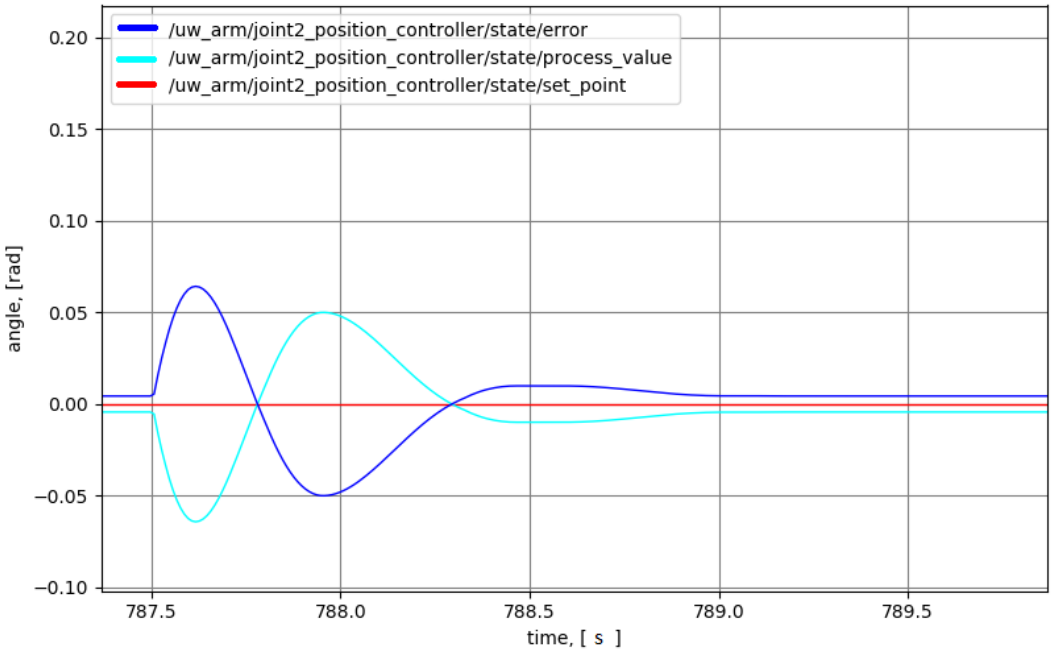


Figure 4.4: The propagated disturbance on joint2 from moving joint1

4.2 Custom End-Effector

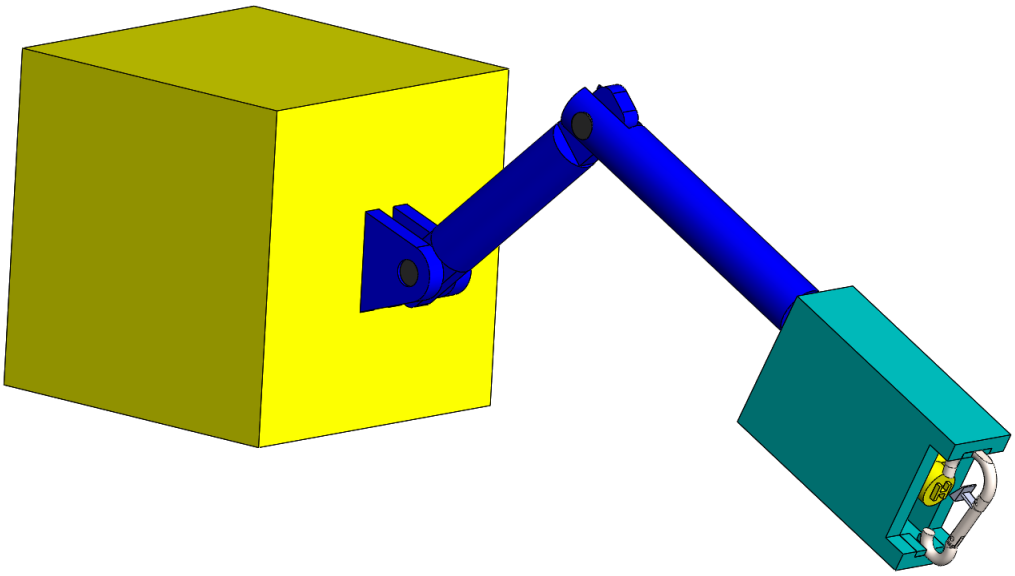


Figure 4.5: Isometric view

Fig. 4.5 shows an isometric view of the prototype designed in SolidWorks, without the artificial muscle nor the other electrical components. The end-effector is designed to be attached to the rigid bars of the derelict pots, and inflate a balloon to ascend the pot to the surface.
It is a combination of the two already existing methods; lifting bags attached by divers and ROV operation.

4.3 The Complete Simulation

A short video of the robotic manipulator in action can be seen here.
Furthermore, shown below are screenshots of the sequence of events. The simulation and the screenshots from it is showing the principle of operation.

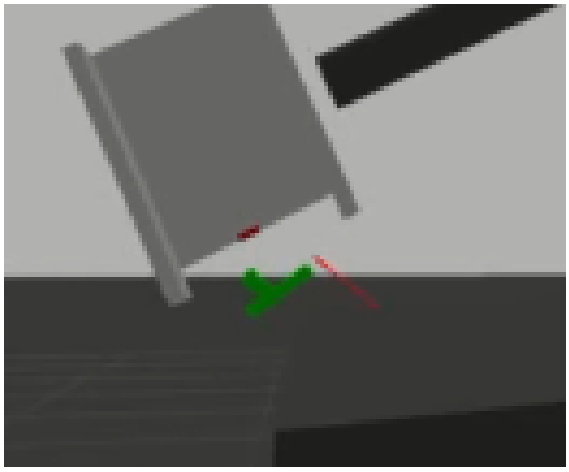


Figure 4.6: Close-up of the end-effector

In Fig. 4.6, the main components of the end-effector are visible with the light, grey box representing the exploded view of the end-effector. The green component is the hook attached by a passive revolute joint, the red square above it is the button. The button is also passive, although modeled with a translational joint.

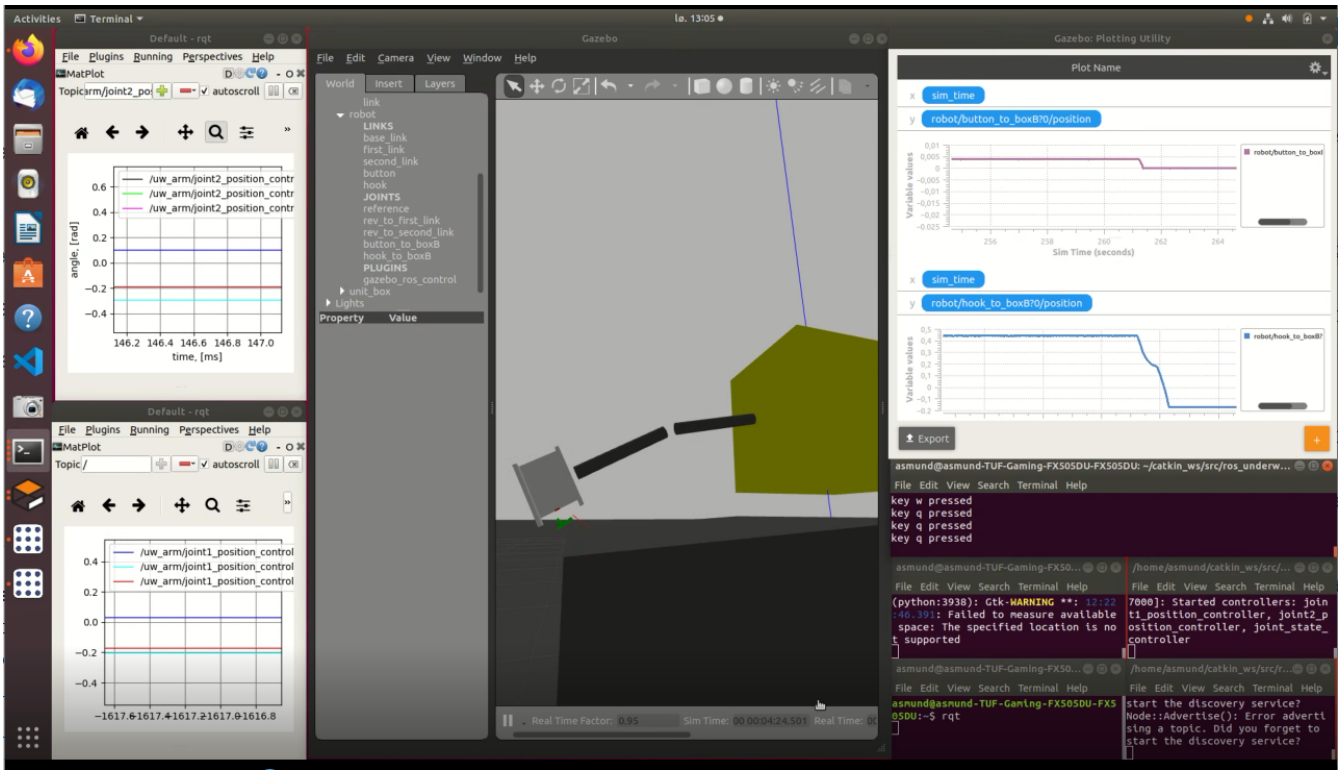


Figure 4.7: Object placed beneath the arm in its workspace

In Fig. 4.7, the arm loaded together with a rigid box, which is placed within reach of the manipulator can be seen. The two windows on the left-hand side represent the joint positions, while the upper right-hand side graphs represent the hook and the button. In the lower right corner, the required nodes are running in each respective terminal;

1. teleop
2. 2x RQT
3. ros_underwater_simulator
4. uw_arm_control

The 'teleop' node is the executable which reads the users key inputs, the two 'RQT' nodes run each of the graphical representation of the joint positions, 'ros_underwater_simulator' is the launch file loading the URDF into the Gazebo world, while 'uw_arm_control' loads the controllers along with the 'joint_state_publisher'.

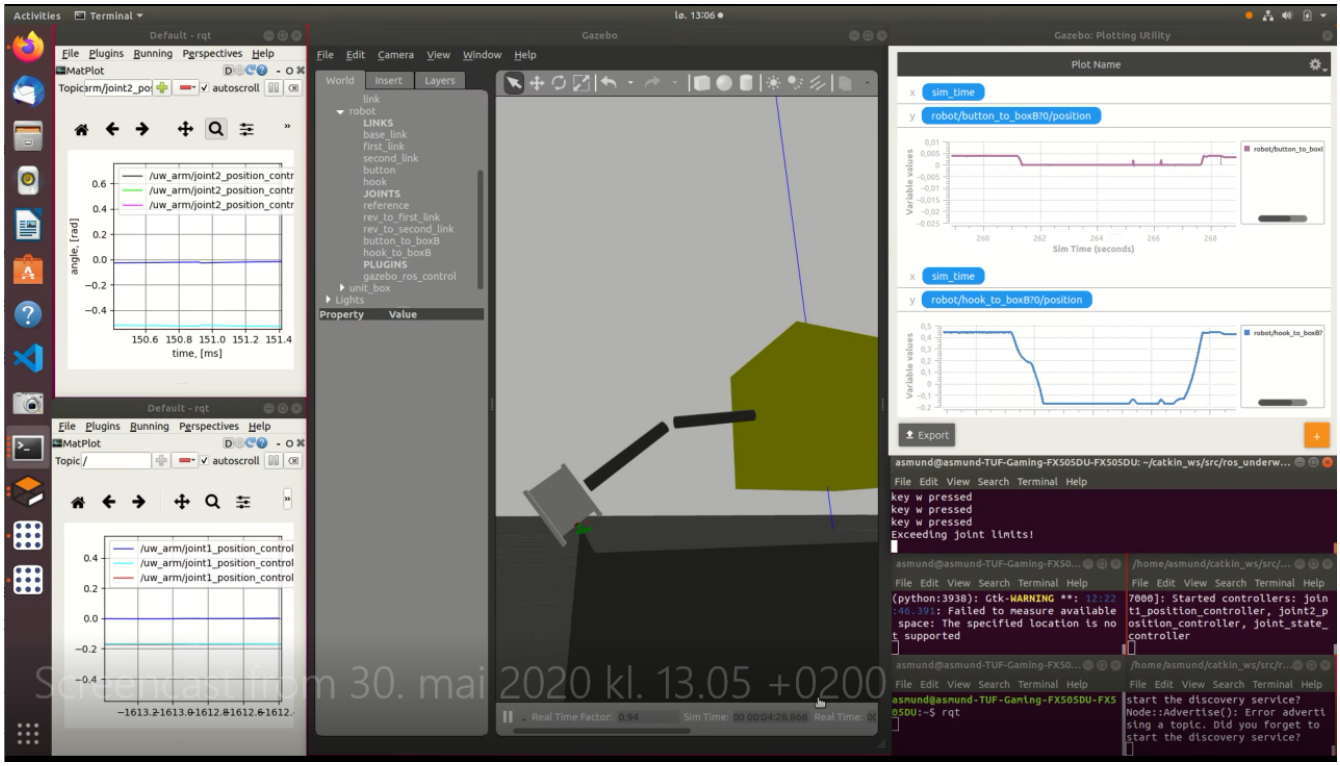


Figure 4.8: The arm interacting with the rigid object

In Fig. 4.8, the end-effector is interacting with the rigid box. This is further visualized by the graphs representing the hook and the button in the upper right corner. The red graph displays the button displacement within its line of translation, while the blue graph displays the amount of rotation in radians for the revolute joint at the hook.

Chapter 5

Discussion

The Institute of Marine Research desired that the group should make a physical prototype, yet the challenges of the COVID-19 pandemic made it difficult for the group to access necessary equipment and other elements due to the lockdown of the schools. Therefore, it was decided to change perspective and focus on simulation. However, a simplification in this project was done, which was that the impact of water was not accounted for. In a real scenario, this would occur and cause more disturbances and complications in the simulation. Regardless, further improvements for transferring the knowledge between different domains are necessary for the simulated model to be feasible in industrial applications.

5.1 Simulation

The ROS-Gazebo framework was decided upon at a late stage of the project due to the ongoing circumstances, and the resulting simulation reflects this. A proper simulation would have real models of sensors included in the manipulator model. Adding simplified water dynamics would also be a major improvement. The barrier for increasing the quality of the simulation occurred due to interfacing between Gazebo and ROS. In order to implement actuator kinematics and end-effector position control, then data must be read from the model in Gazebo by a ROS node. This is done in the Gazebo-ROS control plugin, albeit the collective effort of its contributors. An experienced Gazebo-ROS/Cpp/Python user would pass this barrier with ease. It is, however, an act of balance since the intended simulation should run in real-time. These boundaries are far from tested and therefore is this version a mere starting point for further development.

5.2 End-Effector

The custom end-effector is the result of a combination of the current methods investigated in section 1.2. The problem the Institute of Marine Research addressed was the inefficiency of the multiple ascends and descends during search missions. This end-effector is a first draft idea with a simplistic but presumed mode of operation. However, there is no way to confirm that it will function correctly if not tested on an actual mechanical model. Consequently, a physical model would play a crucial part. However, the main problem with this design might be inflation without proper 'hooking' of the derelict pots' rigid bar since it inflates on a certain amount of opening. A major improvement would be to add some sort of spring-loaded mechanism which would load the spring during the opening and push the button on the closing of the hook. Two other mechanisms are neglected during the design process; the release from the manipulator and a magazine holding and loading multiple of the proposed end-effector.

Chapter 6

Conclusion

As a primary stage, the idea was to design and generate a physical prototype. Due to the unexpected circumstances (the COVID-19 pandemic), which led to a lockdown of pretty much everything in the society, the focus was shifted to simulation. The research questions stated whether the group was able to design and create a low-cost and easy-to-manufacture robotic arm together with a custom-made end-effector to be mounted on the ROV, which was to be utilized as an ocean cleaner for marine litter. Furthermore, simulating this operation was to be executed. The ROV was not in the scope of interest. A model of the prototype and the other components was built in SolidWorks. Besides, a simplified model of the prototype was built in ROS/Gazebo, and, for testing purposes, it was verified through simulations before testing the SolidWorks components on the ROS/Gazebo interface.

The movement of the robotic arm, along with the end-effector, proved to work adequately fine in the simulation. It is fairly acceptable to say that the model built is correctly designed; however, water buoyancy has not been accounted for and is something that should be added in a simulation. The group is certain that simulating with buoyancy would result in small deviations such as delays. Objects submerged in water can not move as fast as they do on land. However, the priority was to ensure that the simulation worked, and the group focused on implementing all other necessities, like mechanical properties of the components, limitations, and control theory. Furthermore, the group is content with the tuning of the controller parameters, shown in Fig. 4.1 and Fig. 4.2. In general, tuning of parameters can always be improved, yet the amount of time required makes it impractical to improve more than necessary.

As a final statement, the group is partially satisfied with the overall achievements because the group members can conclude with high confidence that more could have been achieved if access to the University was possible during this semester. The work has been conducted with a practical point of view, i.e., the multiple types of equipment were intended to be simple yet effective. It can be concluded that the research is essential outside academia because ghost fishing is a global issue, and this paper aims to solve this exact issue. It is vital to mention that simulation and in practice does not always agree. Consequently, the group intended to employ an artificial muscle, as mentioned in Sec. 2.3.4, yet was obliged to utilize a standard hydraulic cylinder. Implementing the artificial muscle on a real robotic arm consisting of steel links will result in deviation when compared with the results obtained when calculating for a standard cylinder.

Chapter 7

Future Work

There would be enough amount for improvements if further work regarding this project was conducted. If the group was given more time, a focus would be aimed at implementing soft robotics instead of their counterparts, which is hard robotics.

7.1 Soft Robotics vs Hard Robotics

This section will introduce the fundamental differences between soft and hard robotics, and some areas where the different robotic end-effectors are utilized in the industry. Furthermore, the advantages and disadvantages of them will be presented.

Competitions in the industry is a well-defined subject and are something that will continue as long as technology develops. Chris Rahn et al. [46] state that modern researchers have been profoundly inspired by the marvelous capabilities of soft animal and plant structures. This inspiration has led to the development of hard robots that imitate soft structures and, besides, soft robots that utilize electroactive polymer (EAP) and pneumatic artificial muscle (PAM) actuators. The majority of soft robots developed are based on these two technologies. However, recent developments have been done by substituting air (pneumatic) with seawater. The advantages of utilizing seawater instead of air is that while the robot is underwater, it has access to virtually unlimited seawater.

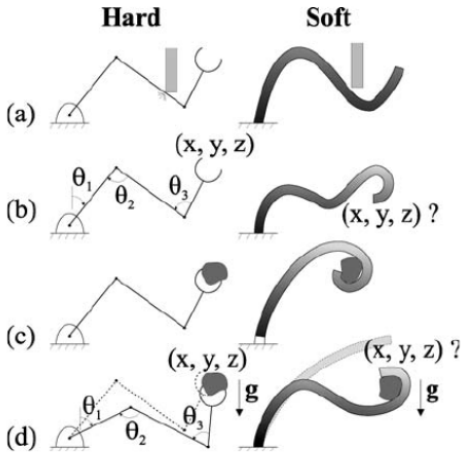


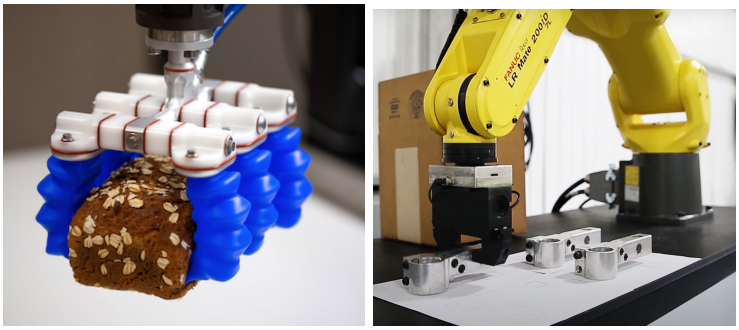
Figure 7.1: Capabilities of hard and soft robots: (a) dexterous, (b) able to monitor and control position, (c) able to manipulate and (d) to load objects. [46]

Fig. 7.1(a) illustrates how soft robots and standard hard robots utilize distinct mechanisms to allow for delicate mobility. Furthermore, it is normally desired to have the ability to sense and control the contour of a soft robot. However, this is demanding, since their contour is continuous, so the accurate measurement of the shape and tip position is tricky. Determining what needs to be measured and how to utilize the measurements to control mobility is difficult. However, hard robots operate differently. They measure the position of each joint with a high-resolution encoder (where θ_1, θ_2 and θ_3 are the joint angles respectively), as shown in Fig. 7.1(b) [46]. If, for instance, a hard robot is used, the joint positions can be processed by utilizing forward kinematics to precisely determine the contour and tip position of the robot. Likewise, by using the inverse kinematics, the joint positions that provide the desired tip position can be calculated. This is done by measuring the joint positions with the encoders and then comparing those values with the desired positions. The desired positions are computed with the inverse kinematics, and then the artificial muscles ensure that the errors converge to zero.

Soft robots act reciprocally with the surroundings differently in comparison to hard robots. The contour is exposed to loads by the surroundings, which can be either distributed loading, for instance, gravity or by direct contact. It can be seen from Fig. 7.1(d) that loading causes the soft joints in a hard robot to change position. However, the rigid links remain straight. This change in position is, as mentioned, measured by encoders. A controller is utilized to either compensate for the loading or recognize that the robot has come in contact with the surroundings [46]. The contour and the tip position can be precisely defined in both cases. In soft robots, however, gravity and contact loading cause continuous deformation. This may not be observable or controllable from the limited sensors or actuators. So in this scenario, it should be utilized more advanced sensors. Soft robotics grasp and handle objects of varying sizes by using whole arm manipulation and this can be seen in Fig. 7.1(c). The arm wraps itself around the object, and a tight grasp and a high-friction contact allow the arm to lift the object. Hard robots, however, grasp and handle objects with a specialized end-effector that is generally designed for a particular size and type of object [46].

7.1.1 Grippers

An end-effector, to put in simple terms, is a device/tool connected to the end of a robot arm. The nature of the end-effector depends on the intended task.



(a) Soft gripper [43] (b) Hard gripper

Figure 7.2: Hard robot gripper and soft robot gripper

Fig. 7.2a shows a pneumatic mGrip soft robotic gripper tailored for the food and beverage and packaging industries. As can be seen from the figure, the gripper is soft at such a level that the bread is not "crushed". According to the company, which has designed and generated this gripper, this gripping system conforms to an object and picks it without the need for sensors, machine vision, or numerical computation [43]. In Fig. 7.2b, a typical hard gripper can be seen. This is a spectacular gripper to utilize when not dealing with fragile objects, and as such, it can be seen from the figure that this gripper is grasping steel profiles. In this project, derelict pots are not to be considered fragile objects, that is why hard robotics were first utilized. However, to add flexibility and other benefits, employing soft robotics is on the safe side, since there is a possibility that there can be trapped animals inside. In addition to employing soft robotics in future work, the end-effector is also an object that would require modifications, as mentioned in Sec. 5.2. There are limitations in a project, and it is impossible to solve everything.

7.2 Autonomous Operation

Technological evolution is developing with an exponential speed [19]. As a result, the demand for precise localization systems in subsea operations is continuously rising. To increase the level of autonomy in subsea operations, the group would invest a great effort of applying machine learning. This section describes and introduces the reader to the necessary software to work with machine learning, which may be utilized to detect derelict pots autonomously.

7.2.1 Pose Estimation with Machine Learning

The current procedure of gathering derelict pots at the bottom of the ocean is to have an operator on a ship steering and controlling a ROV. A camera will be necessary to be mounted on the ROV such that the operator can obtain visualization. How about if the ROV was able to detect the derelict pots almost all by itself? However, completely autonomous operations are not necessarily an aim in itself, yet an increased level of autonomy from what is seen today is wanted. Schjølberg et al. [36] propose shared control, a control regime where certain modes are performed autonomously, and others are performed by the operator. The human operator needs to remain in the loop and can interrupt any actions initiated by the autonomous system at any given time. To increase the level of autonomy in ROV operations, localization systems are vital. Localization refers to an object's understanding of its position and attitude relative to the surroundings.

ROVs can be utilized in many areas, e.g., executing manipulation tasks on subsea installments, which requires exceptionally accurate steering of the ROV, to avoid collisions and damages on subsea equipment. Such failures can lead to huge expenses, yet even worse, they can cause leakages of petroleum to the surrounding environment. Consequently, very accurate localization systems are crucial for an increased level of autonomy in inspection, maintenance, and repair (IMR) operations. This leads to the need for a real-time algorithm with great accuracy.

There are several ways to handle the localization problem. **Map-based localization** is achievable in static surroundings if a precise map is available. However, this is rarely the case in underwater environments. Another approach is to localize relative to a particular coordinate or a particular object. Prior knowledge of the structure where the ROV is operating is imposed by this method, as one or more reference points for relative localization is necessary. A third approach to the localization problem is simultaneously localization and mapping (**SLAM**), which simultaneously localizes a vehicle with respect to a map while updating the map on the go [26].

Bibliography

[1] QUT Robot Academy. *Denavit-Hartenberg notation*. Retrieved 04.03.20, from `https://robotacademy.net.au/lesson/denavit-hartenberg-notation/`.

[2] Alford A Anderson, J. *Ghost fishing activity in derelict blue crab traps in Louisiana*. Marine Pollution Bulletin, 79(1-2), 261-267., 2014.

[3] Enrique Fernandez Perdomo Bence Magyar and Mike Purvis. *ROS-teleop*. retrieved 02.06.20, from `https://github.com/ros-teleop/teleop_twist_keyboard`, 2020.

[4] Samuel Bouchard. *7 Types of Industrial Robot Sensors*. Retrieved 02.06.20, from `https://blog.robotiq.com/bid/72633/7-Types-of-Industrial-Robot-Sensors`, 2014.

[5] Vrije Universiteit Brussel. *Pneumatic artificial muscles (PAMs)*. Retrieved 25.03.20, from `http://mech.vub.ac.be/multibody/topics/pam.htm`, 2012.

[6] Khao Lak Explorer Dive Center. *What are the Risks of Scuba Diving?* Retrieved 19.02.20, from `https://www.khaolakexplorer.com/similan-guide/risks-of-scuba-diving/`, 2020.

[7] Sachin Chitta, Eitan Marder-Eppstein, Wim Meeussen, Vijay Pradeep, Adolfo Rodríguez Tsouroukdissian, Jonathan Bohren, David Coleman, Bence Magyar, Gennaro Raiola, Mathias Lüdtke, and Enrique Fernández Perdomo. ros_control: A generic and simple control framework for ros. *The Journal of Open Source Software*, 2017.

[8] Encoder Products Company. *What IS an encoder?* Retrieved 01.05.20, from `http://encoder.com/blog/company-news/what-is-an-encoder/`, 2016.

[9] Gates Corporation. *Pressure Drop*. Retrieved 02.03.20, from `https://www.gates.com/us/en/knowledge-center/engineering-applications/pressure-drop.html`.

[10] Trevor Croft. *A Boater's Guide to Carabiners*. Retrieved 06.06.20, from `https://www.raftingmag.com/rafting-magazine/boaters-guide-to-carabiners`.

[11] Frank Daerden and Dirk Lefeber. Pneumatic artificial muscles: actuators for robotics and automation. *European Journal of Mechanical and Environmental Engineering*, 47, 03 2002.

[12] Shawn Day. *Basic Sensors for Robot Grippers*. Retrieved 27.05.20, from `https://automation-insights.blog/2016/04/13/basic-sensors-for-robot-grippers/`, 2020.

[13] World Wide Fund for Nature. *FORSØPLING AV HAVET*. Retrieved 29.01.20, from `https://www.wwf.no/dyr-og-natur/hav-og-fiske/plast-i-havet`, 2018.

[14] Fullerfasteners. *PROOF LOADS FOR FASTENERS*. Retrieved 08.05.20, from `https://www.fullerfasteners.com/tech/proof-loads-for-fasteners/`, 2020.

[15] Jan Tommy Gravdahl. *PID-regulator*. Retrieved 08.10.19, from `https://snl.no/PID-regulator`, 2018.

[16] Harshall. *What is Scrum Framework in 5 minutes*. Retrieved 15.05.20, from `https://www.lonare.com/what-is-scrum-framework-in-5-minutes/`, 2018.

[17] Mette Mo Jakobsen. *Produktutvikling - Verktoykasse for utvikling av konkurransedyktige produkter*, side 127. Fortuna Forlag AS, 1997.

[18] Mette Mo Jakobsen. *Produktutvikling - Verktoykasse for utvikling av konkurransedyktige produkter*, side 128. Fortuna Forlag AS, 1997.

[19] Alison E. Berman Jason Dorrier. *Technology Feels Like It's Accelerating - Because It Actually Is*. Retrieved 04.06.20, from `https://singularityhub.com/2016/03/22/technology-feels-like-its-accelerating-because-it-actually-is/`, 2016.

[20] Alf Ring Kleiven. *Spøkelsesfiske*. Retrieved 19.02.20, from `https://snl.no/sp%C3%B8kelsesfiske`, 2020.

[21] Lovdata. *Forskrift om endring i forskrift om utførelse av arbeid, bruk av arbeidsutstyr og tilhørende tekniske krav (forskrift om utførelse av arbeid)*. Retrieved 24.02.20, from `https://lovdata.no/dokument/LTI/forskrift/2018-12-14-1976`, 2020.

[22] Project Management. *Scrum Boards: Why Are They Important and How Do They Work?* Retrieved 15.05.20, from `https://kissflow.com/project-management/scrum-board-overview/`, 2020.

[23] Shanuj Mishra. *The Importance of Prototyping in Designing*. Retrieved 01.06.20, from `https://uxdesign.cc/importance-of-prototyping-in-designing-7287c7035a0d`, 2019.

[24] J. Moultrie. *Design management*. Retrived 19.09.19, from `https://www.ifm.eng.cam.ac.uk/research/dmg/tools-and-techniques/morphological-charts/`, 2019.

[25] Parviz E. Nikravesh. *Computer-Aided Analysis of Mechanical Systems, 1. edition*. New Jersey, USA: Prentice-Hall inc., 1988.

[26] N. Palomeras, Sharad Nagappa, David Romagós, Nuno Gracias, and Marc Carreras. Vision-based localization and mapping system for auv intervention. pages 1–7, 06 2013.

[27] Bjørn Pedersen. *oppdrift - fysikk*. Retrieved 20.04.20, from `https://snl.no/oppdrift_-_fysikk`.

[28] Ellen Roche. *Pneumatic Artificial Muscles*. Retrieved 20.02.20, from `https://softroboticstoolkit.com/book/pneumatic-artificial-muscles`.

[29] ROS.org. *About ROS*. Retrieved 10.03.20, from `https://www.ros.org/about-ros/`, 2020.

[30] ROS.org. *controller_manager*. retrieved 24.05.20, from `http://wiki.ros.org/controller_manager`, 2020.

[31] ROS.org. *Creating a ROS package*. Retrieved 23.04.20, from `http://wiki.ros.org/ROS/Tutorials/CreatingPackage`, 2020.

[32] ROS.org. *rosdep*. retrieved 18.05.20, from `http://wiki.ros.org/rosdep`, 2020.

[33] ROS.org. *xacro*. retrieved 20.05.20, from `http://wiki.ros.org/xacro`, 2020.

[34] Edin Omerdić Gerard Dooly Daniel Toal Satja Sivčev, Joseph Coleman. *Underwater manipulators: A review*. Ocean Engineering, vol. 163, september 2018, Pages 431-450, 2018.

[35] David Saunders. *Closed Loop VS Open Loop*. Retrieved 15.03.19, from `https://www.crossco.com/blog/closed-loop-vs-open-loop-hydraulic-systems`, 2017.

[36] Ingrid SchjÃ¸lberg, Tor Gjersvik, Aksel Transeth, and Ingrid Utne. Next generation subsea inspection, maintenance and repair operations. *IFAC-PapersOnLine*, 49:434–439, 12 2016.

[37] Ingrid Løset. *Tiltak mot spøkelsesfiske i Raet Nasjonalpark*. Havforskningsinstituttet, Green Bay, Handelens Miljøfond.

[38] Daniela Rus Shuguang Li, Daniel M. Vogt and Robert J. Wood. *FOAMs: Soft Robotic Artificial Muscles*. Retrieved 19.02.20, from `https://wyss.harvard.edu/technology/foams-soft-robotic-artificial-muscles/`.

[39] SmartBolts. *Fundamentals of Basic Bolting*. Retrieved 05.06.20, from `http://www.smartbolts.com/fundamentals/`, 2020.

[40] SmartBolts. *What is Proof Load of a Bolt and How is it Different from Yield Strength?* Retrieved 05.05.20, from `http://www.smartbolts.com/insights/proof-load-bolt-different-yield-strength/`, 2020.

[41] sofaframeworks.org. *Story of SOFA*. retrieved 03.06.20, from `https://www.sofa-framework.org/about/story/`, 2020.

[42] Springtimesoft. *Agile software development through Scrum*. Retrieved 15.05.20, from `https://springtimesoft.co.nz/agile-software-development-scrum/`, 2020.

[43] DE Staff. *FANUC to distribute Soft Robotics grippers*. Retrieved 28.05.20, from `https://www.design-engineering.com/fanuc-to-distribute-soft-robotics-grippers-1004034173/`, 2019.

[44] Wikipedia the free encyclopedia. *Lift Bags*. Retrieved 19.02.20, from `https://en.wikipedia.org/wiki/Lifting_bag`, 2020.

[45] Wikipedia the free encyclopedia. *XML*. retrieved 20.05.20, from `https://en.wikipedia.org/wiki/XML`, 2020.

[46] Deepak Trivedi, Chris Rahn, William Kier, and Ian Walker. Soft robotics: Biological inspiration, state of the art, and future research. *Applied Bionics and Biomechanics*, 5:99–117, 10 2008.

[47] Adolfo Rodríguez Tsouroukdissian. *ROS control, an overview*. presented at ROSCon 2014, September 12, 2014.

[48] Adolfo Rodriguez Tsouroukdissian. *transmission_interface::SimpleTransmission Class Reference*. retrieved 24.05.20, from `http://docs.ros.org/hydro/api/transmission_interface/html/c++/classtransmission__interface_1_1SimpleTransmission.html`, 2020.

[49] uml.co.uk. *MK5 Automatic Inflator*. retrieved 04.06.20, from `https://www.uml.co.uk/mk5_automatic_inflator.html`, 2020.

[50] 5D Vision. *The Scrum Master is a servant leader*. Retrieved 15.05.20, from `https://www.5dvision.com/post/the-scrum-master-is-a-servant-leader/`.

[51] Western Process Controls (WPC). *Hydraulic Actuators*. Retrieved 21.01.20, from `http://www.wpc.com.au/actuation-controls/hydraulic-actuators`.

Appendices

Appendix A

Text files

A.1 - Read Me for Gazebo-ROS sim

A.1 Read Me for Gazebo-ROS sim

Figure/ICT/README.txt

```
                                    How to run the simulation
-------------------------------------------------------------------------------------------------------
Setup:
        The simulation runs on the following software: Ubuntu 18.0.4, ROS Melodic and Gazebo 9.0.
The mentioned programs can be downloaded and installed by following the instructions in these urls:

-Ubuntu 18.0.4        https://www.linuxtechi.com/ubuntu-18-04-lts-desktop-installation-guide-screenshots/
-ROS Melodic        http://wiki.ros.org/melodic/Installation/Ubuntu
-Gazebo 9.0        http://gazebosim.org/tutorials?tut=install_from_source&cat=install

Alternatively could a virtual machine be setup to run Ubuntu on another OS, but the simulation is
not tested for this case.

Note: Before starting this simulation should the user either have some experience with ROS and Gazebo
or atleast have a look at the beginner tutorial for ROShttp://wiki.ros.org/ROS/Tutorials and then
 move on to more advanced tutorials including Gazebohttp://gazebosim.org/tutorials

First create a catkin workspaceSee ROS beginner tutorials and source the ROS environment using the
command in a terminalCTRL+ALT+t:

        $ source /devel/setup.bash

This should ensure that ROS commands can be used. If everything got installed successfully, copy
the two packages from this repository:

        https://github.com/aasmub14/ROS_simulation.git

into the /src/ folder in the catkin workspace. Then navigate to the /src/ folder in the terminal
and run the command:

        $ rosdep install PACKAGE_NAME

for each of the packages found in the repository. This will then find and install the necessary
dependencies, other packages, needed to run the simulation.
-------------------------------------------------------------------------------------------------------
Start up:
        Open a terminal and find the location of /ros_underwater_simulatorroscd and
run the command:

        $ roslaunch ros_underwater_simulator controllable.launch model:=urdf/uw_arm_control_ee.urdf

A Gazebo window should pop up with the model of the robot.

Next open another terminal and find the same package, but change directory to /nodes/. Then
run the command:

        $ python teleop_final.py

to launch the command node. This node takes the inputs from keybindings; '1','2','Q' and 'W'
to control the two driven joints.

Then open a third terminal and find the package /uw_arm_control/. This package contains the
controller instructions and is started by running the command:

        $ roslaunch uw_arm_control controller.launch
```

Now should the manipulator be controllable by the aforementioned key inputs.

Running:

In order to monitor the controllers and the joints can a terminal be opened and enter the command:

```
$ rqt
```

This opens a GUI where ROS topics can be plotted by selecting 'plugins->visualization->plot' and then select the topic of interest.
The same can be done in Gazebo by selecting 'window->plot' and select the Gazebo topic of interest.

Disclaimer:

We are by no means experts with this framework and the model is far from representing the idea of an underwater manipulatorNo water physics would tip this off, but feel free to send questions and improve on it.

Appendix B

MATLAB-scripts

B.1 Phi

```
function Fi = Phi(q,t)

% Geometry
l1 = 0.5;
l2 = 0.5;
rev = 0.08;
%Driver dummy values
init_length1 = 0.2;
cmd1 = 0.05*t;
init_length2 = 0.2;
cmd2 = 0.05*t;
%Constant vectors
b1 = [l1/2; 0];
b2 = [l2/2; 0];
rev1 = [0; rev];
rev2 = [0; rev];
rov = [0; 0.25];
% Cartesian coordinates
r1 = q(1:2,1);
phi1 = q(3,1);
r2 = q(4:5,1);
phi2 = q(6,1);
% Constraint vector as a function of q and t

%Rev GND - Body 1
Fi(1:2,1) = A(phi1)*b1 - r1;
%Rev Body1 - Body2
Fi(3:4,1) = r1 + A(phi1)*b1 + A(phi2)*b2 - r2;
%Longitudinal driver ROV-B1
d1 = r1 + A(phi1)*rev1 - rov;
Fi(5,1) = d1'*d1 - (init_length1 + cmd1)^2;
%Longitudinal driver B1-B2
d2 = r2 - A(phi2)*rev2 - r1 + A(phi1)*rev1;
Fi(6,1) = d2'*d2 - (init_length2 + cmd2)^2;
```

B.2 Jacobi

```
function J = Jacobi(q)
% Proper notation for the coordinates for each body
r1 = q(1:2,1);
phi1 = q(3,1);
r2 = q(4:5,1);
phi2 = q(6,1);

% Geometry
l1 = 0.5;
l2 = 0.5;
rev = 0.08;
%Constant vectors
b1 = [l1/2; 0];
b2 = [l2/2; 0];
rev1 = [0; rev];
rev2 = [0; rev];
rov = [0; 0.25];

% Initialization of the jacobian matrix
J = zeros(6,6);
% Revolute joint between body 1 and ground
J(1:2,1:3) = [-eye(2), Brot(phi1)*b1];
% Revolute joint between body 1 and 2
J(3:4,1:6) = [eye(2), Brot(phi1)*b1, -eye(2), Brot(phi2)*b2];
% Longitudinal driver between ROV and Body 1
d1 = r1 + A(phi1)*rev1 - rov;
J(5,1:3) = [2*d1', 2*d1'*Brot(phi1)*rev1];
% Longitudinal driver between body 2 and body 1
d2 = r2 - A(phi2)*rev2 - r1 + A(phi1)*rev1;
J(6,1:6) = [-2*d2', 2*d2'*Brot(phi1)*rev1, 2*d2', -2*d2'*Brot(phi2)*
    rev2];
```

B.3 γ

```
function g = Gamma(q,qd,t)
% Proper notation for the coordinates for each body
r1 = q(1:2,1);
phi1 = q(3,1);
r2 = q(4:5,1);
phi2 = q(6,1);

r1d   = qd(1:2,1);
phi1d = qd(3,1);
r2d   = qd(4:5,1);
phi2d = qd(6,1);

% Geometry
l1 = 0.5;
l2 = 0.5;
rev = 0.08;
%Driver dummy values
init_length1 = 0.2;
cmd1dd = 0;
init_length2 = 0.2;
cmd2dd = 0;
%Constant vectors
b1 = [l1/2; 0];
b2 = [l2/2; 0];
rev1 = [0; rev];
rev2 = [0; rev];
rov = [0; 0.25];

% Initialization of the gamma-vector
g = zeros(6,1);
% Revolute joint between body 1 and ground
g(1:2,1) = - phi1d^2*A(phi1)*b1;
% Revolute joint between body 1 and 2
g(3:4,1) = - phi1d^2*A(phi1)*b1 - phi2d^2*A(phi2)*b2;
% Longitudinal driver between body 1 and ground
d1 = r1 + A(phi1)*rev1 - rov;
d1d = r1d + phi1d*Brot(phi1)*rev1;
g(5,1) = -2*(d1'*(phi1d^2*A(phi1)*rev1) - d1d'*d1d + cmd1dd);
% Longitudinal driver between body 2 and ground
d2 = r2 - A(phi2)*rev2 - r1 + A(phi1)*rev1;
d2d = r2d - phi2d*Brot(phi2)*rev2 - r1d + phi1d*Brot(phi1)*rev1;
g(6,1) = 2*(d2'*(phi2d^2*A(phi2)*rev2 - phi1d^2*A(phi1)*rev1) + d2d
    '*d2d - cmd2dd);
```

B.4 Newton Raphson Solver

```
function b = NewtonRaphson(FunFcn,x,p,tol,relax)
% NewtonRaphson(FunFcn,x,p,tol,relax)
%
% NewtonRaphson Solves a set of functions (linear or non linear) by means of
% Newton-Raphson iteration.
% FunFcn is a string holding name of function
% x      is initial guess on variables
% p      is a set of fixed parameters
% tol    is the tolerance to be met in iteration
% relax  is the relaxation factor for the change in the coordinate vector
%
% Michael Rygaard Hansen
% Aalborg University, 1999
%
% modified by
% Niels L. Pedersen 2003
%
% Explanation added by
% Morten Kjeld Ebbesen 2005

% Initialization
if nargin < 4 | isempty(tol), tol = 1e-10; end
if nargin < 5 | isempty(relax), relax = 1.0; end

% Get rank of problem
n = length(x);

% Set perturbation
pert = 1e-10;

% Initialize counter
i = 0;

% Initial residual vector computation
b = feval(FunFcn, x, p);
n2 = length(b);

% Do Newton-Raphson iteration procedure. Stop when counter = 100 or
% tolerance met.
while norm(b)>tol & i<100

  % Increment counter
  i = i+1;

  % Generate jacobian
```

```
46    for j=1:n
47      temp=x(j);
48      if x(j)==0
49        x(j) = x(j)+pert;
50        delta=pert;
51      else
52        x(j) = x(j)*(1.0+pert);
53        delta = pert*temp;
54      end
55      bPert = feval(FunFcn, x, p);
56      x(j) = temp;
57      for k=1:n
58        jacob(k,j)=(bPert(k)-b(k))/delta;
59      end
60    end
61
62    % Do correction
63    x = x - relax*(jacob\b);
64    b = feval(FunFcn, x, p);
65    c = norm(b);
66  end
67  b = x;
```

B.5 NewtonR

```
function b = NewtonR(Phi,Jacobi,x,t,tol,relax)
% function b = NewtonR(Phi,Jacobi,x,t,eps,k1,k2)
%
% Phi er en streng indeholdende navnet pÃ funktionen
%     indeholdende bindingsligningerne Phi(x,p)
% Jacobi er en streng indeholdende navnet pÃ funktionen
%     indeholdende jacobimatrice Jacobi(x,p)
% x er en vektor med startgÃttet for de variable koordinater
% p er en vektor med mulige konstanter. Hvis ingen konstanter
%     eksisterer sÃttes p til et 0 i funktionskaldet
% relax er en relaxationsfaktor for Ãndringen af koordinatvektoren
%     i Newton-Raphson iterationen. Den bÃ¸r som standard sÃttes
%     til 1 i funktionskaldet.
%
% Output fra NewtonR er en vektor med koordinater, der
   tilfredsstiller
% det opstillede ligningssystem
%
%
% Morten Ebbesen, Aalborg University, 2005

if nargin < 5 | isempty(tol), tol = 1e-10; end
if nargin < 6 | isempty(relax), relax = 1.0; end

nmax = 150;
% error = norm(feval(Phi,x,t,k1,k2));
error = norm(feval(Phi,x,t));
i = 0;
while (error>tol & i<nmax)
    i = i + 1;
%     deltax = feval(Jacobi,x,k1)\feval(Phi,x,t,k1,k2);
    J = feval(Jacobi,x);
    P = feval(Phi,x,t);
    deltax = J\P;
%     deltax = feval(Jacobi,x)\feval(Phi,x,t);
    error = norm(deltax);
    x = x - relax*deltax;
%     x = x - deltax;
end
b = x;
```

B.6 FZEROS

```
% function b = Newton(FunFcn,x,p)
function b = fzerom_par(FunFcn,x,p)
%FZEROS  Solves set of functions (linear or non linear) by means of
% Newton-Raphson iteration.
% FunFcn is a string holding name of function
% x is initial guess on variables
% p is a set of fixed parameters
% tol is the tolerance to be met in iteration
%
% Michael Rygaard Hansen
% Aalborg University, 1999
%
% Initialization
if nargin < 4 | isempty(tol), tol = 1e-10; end
if nargin < 5 | isempty(relax), relax = 1.0; end

%% Set tolerance
% tol = 1e-5;
%
%%set relaxation
% relax = 1.0;

% Get rank of problem
n = length(x);

% Set perturbation
pert = 1e-10;

% Initialize counter
i = 0;

% Initial residual vector computation
b = feval(FunFcn, x, p);
n2 = length(b);

% Do Newton-Raphson iteration procedure. Stop when counter = 100 or
% tolerance met.
while norm(b)>tol & i<100

  % Increment counter
  i = i+1;

  % Generate jacobian
  for j=1:n
    x(j) = x(j)+pert;
    bPert = feval(FunFcn, x, p);
    x(j) = x(j)-pert;
```

```
48        for k=1:n
49          jacob(k,j)=(bPert(k)-b(k))/pert;
50        end
51     end
52
53     % Do correction
54     x = x - relax*(jacob\b);
55     b = feval(FunFcn, x, p);
56     c = norm(b);
57   end
58   b = x;
```

B.7 Coordinate System 2D

```
function M = CoordSys2D(la,lt,va)

M = [[la+lt*cos(pi-va),lt*sin(pi-va)]',...
[la,0]',...
[la+lt*cos(pi+va),lt*sin(pi+va)]',...
[la,0]',...
[0,0]',...
[0,la]',...
[lt*cos(3*pi/2-va),la+lt*sin(3*pi/2-va)]',...
[0,la]',...
[lt*cos(3*pi/2+va),la+lt*sin(3*pi/2+va)]'];
```

B.8 Body Plot

```
function BodyPlot(t,D,n,tt,h)

figure(h)
for i=1:n
    subplot(2,n,i)
    hold on
        plot(t,D(:,i*3-2),'b-')
        plot(t,D(:,i*3-1),'r-')
    hold off
    grid on
    box on
    legend(['x_{',num2str(i),',',tt,'}'], ['y_{',num2str(i),',',tt,'
        }'])

    subplot(2,n,i+n)
    hold on
        plot(t,D(:,i*3),'m-')
    hold off
    grid on
    box on
    legend(['\phi_{',num2str(i),',',tt,'}'])
end
```

B.9 RHvel

```
function a = RHvel(q,t)

%Driver dummy values
cmd1 = 0.05;
cmd2 = 0.05;
% Initialization of the right hand side
a = zeros(6,1);
% Longitudinal driver Gnd Body 1
a(5,1) = 2*cmd1;
% Longitudinal driver Body2 Body 1
a(6,1) = 2*cmd2;
```

B.10 Main

```
clear all
close all
clc

% Geometry
l1 = 0.5;
l2 = 0.5;
rev = 0.08;

%Physical properties
g = -9.81;        % gravity acceleration
m1 = 3;           % mass of body 1
J1 = 0.0625;      % mass moment of inertia for body 1
m2 = 3;           % mass of body 2
J2 = 0.0625;      % mass moment of inertia for body 2
M = diag([m1,m1,J1,m2,m2,J2],0); % mass matrix for the system
Fgrav = M*[0,g,0,0,g,0]'; % external forces

% Initial guess for the cartesian coordinates for the bodies in the
% mechanism at T = 0
T = 0;
q = [l1/2 0 0 (l1 +l2/2) 0 0]';

solution = NewtonRaphson('Phi',q,T) %NewtonRaphson(FunFcn,x,p,tol,
    relax)

% Time for the analysis
tmin = 0;
tmax = 4;
N = 500;
t = linspace(tmin,tmax,N)';

% Initialization of the data arrays
Y = zeros(N,6);              % Position matrix
Yd = zeros(N,6);             % Velocity matrix
Ydd = zeros(N,6);            % Acceleration matrix
Reac1 = zeros(N,8);          % Reaction forces on body 1
Reac2 = zeros(N,4);          % Reaction forces on body 2
Fdriver1 = zeros(N,1);       % The longitudinal driver Gnd-Body1
Fdriver2 = zeros(N,1);       % The longitudinal driver Body1-Body2
q = solution;

for i=1:N
%     q = NewtonRaphson('Phi',q,t(i,1));
    q = NewtonR('Phi','Jacobi',q,t(i,1)); % Position analysis using
        the analytic jacobian matrix
    J = Jacobi(q); % Computing the jacobian matrix
```

```
46      Jtrans = J'; % !!Computation of the inverse of the Jacobian
            matrix
47      qd = J\RHvel(q,t(i,1)); % Velocity analysis
48      qdd = J\Gamma(q,qd,t(i,1)); % Acceleration analysis
49      lambda = J'\(M*qdd - Fgrav); % Conmputation of the lagrangian
            multipliers for the reaction forces
50      Y(i,:) = q'; % Storing position data
51  %     Y1(i,1:6) = q1';
52      Yd(i,:) = qd'; % Storing velocity data
53      Ydd(i,:) = qdd'; % Storing acceleration data
54      % Reaction forces
55      %                 revolute Gnd-B1                 revolute B1-B2
56      Reac1(i,1:8) = [Jtrans(1:2,1:2)*lambda(1:2,1); Jtrans(1:2,3:4)*
            lambda(1:2,1) ; Jtrans(1:2,5)*lambda(3,1) ; Jtrans(1:2,6)*
            lambda(6,1)]';
57      %                 revolute B1-B2                longitudinal
            driver 2 (extension)
58      Reac2(i,1:4) = [Jtrans(4:5,3:4)*lambda(4:5,1);     Jtrans(4:5,6)*
            lambda(6,1)]';
59      %                 longitudinal driver 1
60      Fdriver1(i,1) = sign(lambda(3,1))*norm(Jtrans(1:2,3)*lambda(3,1)
            );
61      %                 longitudinal driver 2
62      Fdriver2(i,1) = sign(lambda(6,1))*norm(Jtrans(4:5,6)*lambda(6,1)
            );
63  end
64
65
66  % Numerical derivation
67  dt = t(2,1)-t(1,1);
68  Ydnum = zeros(N-2,6);
69  Yddnum = zeros(N-2,6);
70  for i=1:N-2 % Numerical derivation for velocity analysis
71      Ydnum(i,:) = (Y(i+2,:)-Y(i,:))/(2*dt);
72  end
73  for i=1:N-2 % Numerical derivation for acceleration analysis
74      Yddnum(i,:) = (Yd(i+2,:)-Yd(i,:))/(2*dt);
75  end
76
77  %Plotting data
78  figure(1)
79  subplot(2,2,1)
80  hold on
81      plot(t,Reac1(:,1),'r-')
82      plot(t,Reac1(:,2),'b-')
83  hold off
84  grid on
85  box on
86  legend('R_{1Ax} revolute joint Gnd-B1','R_{1Ay} revolute joint')
```

```
87
88  subplot(2,2,2)
89  hold on
90      plot(t,Reac1(:,3),'r-')
91      plot(t,Reac1(:,4),'b-')
92  hold off
93  grid on
94  box on
95  legend('R_{1Bx} revolute joint B1-B2','R_{1By} revolute joint')
96
97  subplot(2,2,3)
98  hold on
99      plot(t,Reac1(:,5),'r-')
100     plot(t,Reac1(:,6),'b-')
101 hold off
102 grid on
103 box on
104 legend('R_{1Bx} Longitudinal driver 1 Gnd-B1','R_{1By} driver joint'
        )
105
106 subplot(2,2,4)
107 hold on
108     plot(t,Reac1(:,7),'r-')
109     plot(t,Reac1(:,8),'b-')
110 hold off
111 grid on
112 box on
113 legend('R_{1Bx} Longitudinal driver 2 B1-B2','R_{1By} driver joint')
114
115 figure(2)
116 subplot(1,2,1)
117 hold on
118     plot(t,Reac2(:,1),'r-')
119     plot(t,Reac2(:,2),'b-')
120 hold off
121 grid on
122 box on
123 legend('R_{2Bx} Revolute joint B1-B2','R_{2By} revolute joint')
124
125 subplot(1,2,2)
126 hold on
127     plot(t,Reac2(:,3),'r-')
128     plot(t,Reac2(:,4),'b-')
129 hold off
130 grid on
131 box on
132 legend('R_{2Bx} Longitudinal driver 2 B1-B2','R_{2By} driver joint')
133
134 figure(3)
```

```
135 hold on
136     plot(t,Fdriver1(:,1),'r-')
137 hold off
138 grid on
139 box on
140 legend('F_{driver1}')
141
142 figure(4)
143 hold on
144     plot(t,Fdriver2(:,1),'r-')
145 hold off
146 grid on
147 box on
148 legend('F_{driver2}')
149
150
151 BodyPlot(t,Y,2,'',50)
152 BodyPlot(t,Yd,2,'d',51)
153 BodyPlot(t,Ydd,2,'dd',52)
154
155 BodyPlot(t(2:N-1,1),Ydnum,2,'d',61)
156 BodyPlot(t(2:N-1,1),Yddnum,2,'dd',62)
157 % Animation
158 la = 0.08;
159 lt = 0.04;
160 va = 30/180*pi;
161 NC = 100; %Number of points on the circles
162 tc2 = linspace(0,2*pi,NC);%Draws a circle with NC as input
163 RC1 = 0.02; %Radius circle 1
164 LW1 = 3;
165 LW2 = 2;
166 % FS1 = 16;
167 Coord = CoordSys2D(la,lt,va);
168 body1 = [[-l1/2;0], [l1/2;0]];
169 body2 = [[-l2/2;0], [l2/2;0]];
170 circle1 = [-l1/2+RC1/2*cos(tc2);RC1/2*sin(tc2)];
171 circle2 = [l1/2+RC1/2*cos(tc2);RC1/2*sin(tc2)];
172 circle3 = [RC1/2*cos(tc2);RC1/2*sin(tc2)];
173 figure(100)
174 set(100,'Units','Centimeter')          % SÃtter arbejdsenheden til
      cm
175 % set(100,'Units','Normalized')
176 % set(100,'PaperPosition',[0 0 bplot hplot]);
177 % set(100,'Position',[0 0 1 1]); %5 5 14 9]);
178 set(100,'Position',[1 1 18 18]);
179
180 for i=1:1:N
181     cla
182     A1 = A(Y(i,3));
```

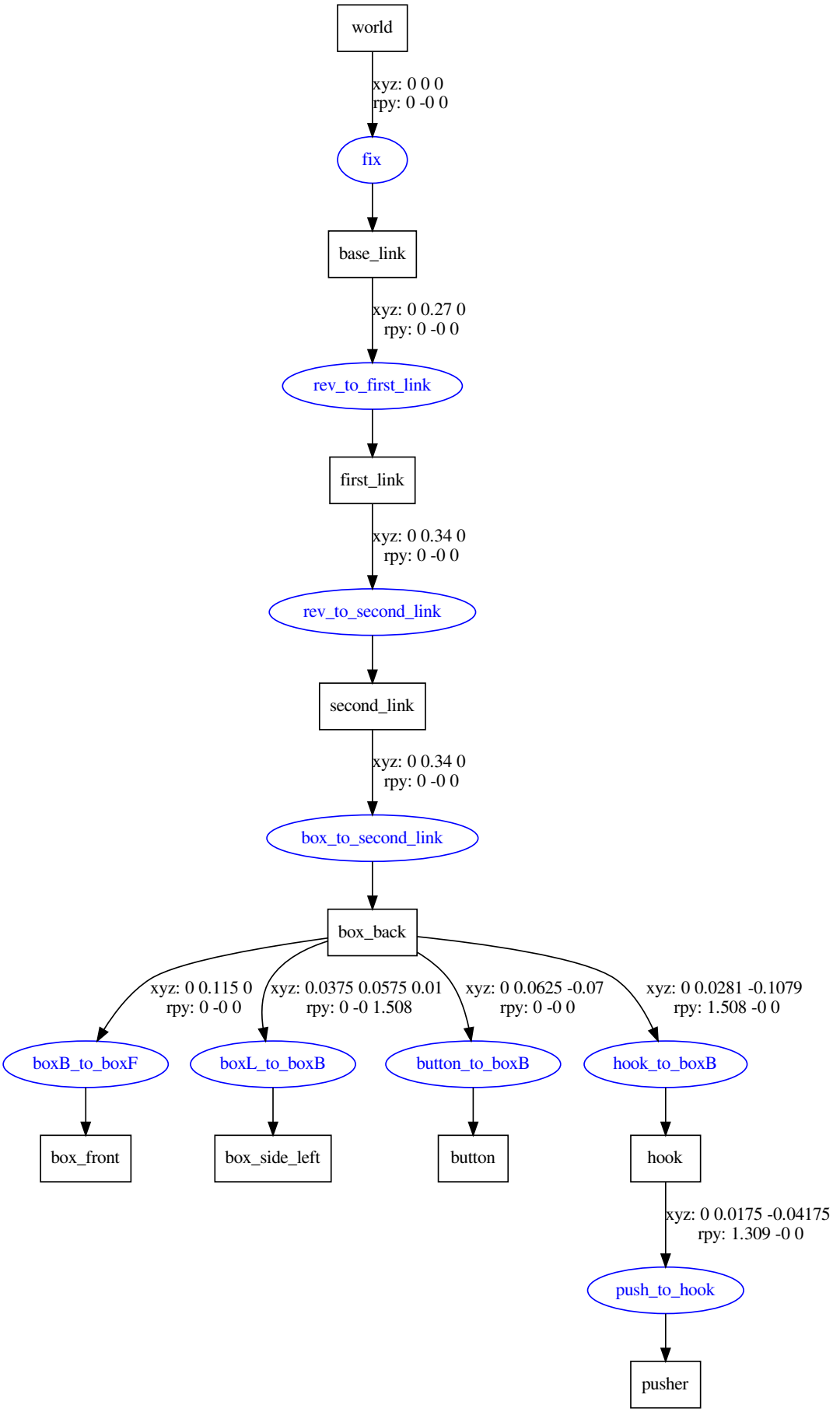
```
183     r1 = Y(i,1:2)';
184     A2 = A(Y(i,6));
185     r2 = Y(i,4:5)';
186     %Plotting Body 1
187     body1Plot = A1*body1;
188     body1Plot(1,:) = r1(1,1) + body1Plot(1,:);
189     body1Plot(2,:) = r1(2,1) + body1Plot(2,:);
190     %Plotting rev gnd-b1
191     circle1Plot = A1*circle1;
192     circle1Plot = [r1(1,1) + circle1Plot(1,:); r1(2,1) + circle1Plot
            (2,:)];
193     %Plotting Body 2
194     body2Plot = A2*body2;
195     body2Plot(1,:) = r2(1,1) + body2Plot(1,:);
196     body2Plot(2,:) = r2(2,1) + body2Plot(2,:);
197     %Plotting rev Body1-Body2
198     circle2Plot = A1*circle2;
199     circle2Plot = [r1(1,1) + circle2Plot(1,:); r1(2,1) + circle2Plot
            (2,:)];
200     C1 = A1*Coord;
201     C1(1:2,:) = [r1(1,1) + C1(1,:); r1(2,1) + C1(2,:)];
202     %Plotting drivers
203     Line1 = [r1 + A1*[0;rev], [0;0.25]];
204     Line2 = [r1 + A1*[0;-rev], r2 - A2*[0;rev]];
205     C2 = A2*Coord;
206     C2(1:2,:) = [r2(1,1) + C2(1,:); r2(2,1) + C2(2,:)];
207
208
209     hold on
210         plot(body1Plot(1,:),body1Plot(2,:),'r-','LineWidth',LW1)
211         plot(circle1Plot(1,:),circle1Plot(2,:),'r-','LineWidth',LW1
                /2)
212         plot(body2Plot(1,:),body2Plot(2,:),'b-','LineWidth',LW1)
213         plot(circle2Plot(1,:),circle2Plot(2,:),'r-','LineWidth',LW1
                /2)
214         plot(C1(1,:),C1(2,:),'k-','LineWidth',LW2)
215         plot(C2(1,:),C2(2,:),'k-','LineWidth',LW2)
216         plot(Line1(1,:), Line1(2,:), 'm-','LineWidth',LW1)
217         plot(Line2(1,:), Line2(2,:), 'm-','LineWidth',LW1)
218     hold off
219     set(gca,'Units','Centimeter','Position',[1,1,16,16])
220 %     axis equal
221     box on
222     grid on
223     axis([-0.3 1 -0.65 0.65])
224     title(['t = ',num2str(t(i))])
225
226     if (i==1)
227         pause
```

```
228         end
229         pause(0.001)
230 end
```

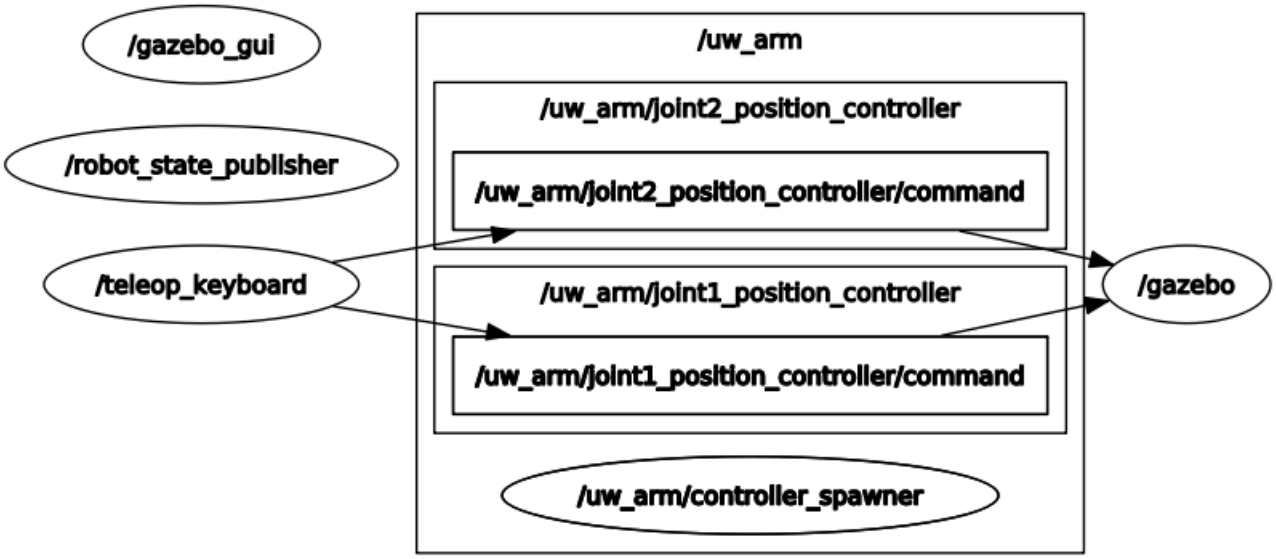
Appendix C

ROS

C.1 URDF, links and joints


world
xyz: 0 0 0
rpy: 0 -0 0
fix
base_link
xyz: 0 0.27 0
rpy: 0 -0 0
rev_to_first_link
first_link
xyz: 0 0.34 0
rpy: 0 -0 0
rev_to_second_link
second_link
xyz: 0 0.34 0
rpy: 0 -0 0
box_to_second_link
box_back
xyz: 0 0.115 0
rpy: 0 -0 0
xyz: 0.0375 0.0575 0.01
rpy: 0 -0 1.508
xyz: 0 0.0625 -0.07
rpy: 0 -0 0
xyz: 0 0.0281 -0.1079
rpy: 1.508 -0 0
boxB_to_boxF
boxL_to_boxB
button_to_boxB
hook_to_boxB
box_front
box_side_left
button
hook
xyz: 0 0.0175 -0.04175
rpy: 1.309 -0 0
push_to_hook
pusher

C.2 Node setup



C.3 Teleop_keyboard

Listing C.1: tele_op

```
#!/usr/bin/env python

import rospy

from std_msgs.msg import Float64

import sys, select, termios, tty

msg = """
Joint Control
---------------------------
Revolute 1:
Rotation by 5 degrees in positive direction: press '1'
Rotation by 5 degrees in negative direction: press '2'
Revolute 2:
Rotation by 5 degrees in positive direction: press 'q'
Rotation by 5 degrees in negative direction: press 'w'
CTRL-C to quit
"""
#Read key press function
def getKey():
    tty.setraw(sys.stdin.fileno())
    rlist, _, _ = select.select([sys.stdin], [], [], 0.1)
    if rlist:
        key = sys.stdin.read(1)
    else:
        key = ''

    termios.tcsetattr(sys.stdin, termios.TCSADRAIN, settings)
    return key



if __name__=="__main__":
    settings = termios.tcgetattr(sys.stdin)

    #Creating the node for teleop and the two publishes
    rospy.init_node('teleop_keyboard')
    pub = rospy.Publisher('/uw_arm/joint1_position_controller/
        command', Float64, queue_size=1) #This topic should match the
         topic of the model which one wishes to control
    pub2 = rospy.Publisher('/uw_arm/joint2_position_controller/
        command', Float64, queue_size=1) #This topic should match the
         topic of the model which one wishes to control
    float_msg = Float64()

```

```
43      accumulated_first = 0
44      accumulated_second = 0
45      direction_first = 0
46      direction_second = 0
47
48      try:
49          print(msg)
50          while(1):
51              accumulated_first = accumulated_first
52              accumulated_second = accumulated_second
53              key = getKey()
54              if key == '1':
55                  print('key 1 pressed')
56                  direction_first = 0.09 #This ensures that key input
                        '1' sends a command to rotate the joint approx 5
                        deg in positive direction
57              elif key == '2':
58                  print('key 2 pressed')
59                  direction_first = -0.09 #This ensures that key input
                        '2' sends a command to rotate the joint approx 5
                        deg in negative direction
60              elif key == 'q':
61                  print('key q pressed')
62                  direction_second = 0.09 #This ensures that key input
                        '1' sends a command to rotate the joint approx 5
                        deg in positive direction
63              elif key == 'w':
64                  print('key w pressed')
65                  direction_second = -0.09 #This ensures that key
                        input '2' sends a command to rotate the joint
                        approx 5deg in negative direction
66              else:
67                  direction_first = 0
68                  direction_second = 0
69                  if (key == '\x03'):
70                      break
71
72              #Calculation of position for the first revolute
73              accumulated_first = direction_first + accumulated_first
74              if accumulated_first > 0.548:
75                  accumulated_first = 0.548
76                  float_msg.data = accumulated_first
77                  pub.publish(float_msg)
78                  print('Exceeding joint limits!')
79              elif accumulated_first < -0.548:
80                  accumulated_first = -0.548
81                  float_msg.data = accumulated_first
82                  pub.publish(float_msg)
83                  print('Exceeding joint limits!')
```

```
84                 else:
85                     float_msg.data = accumulated_first
86                     pub.publish(float_msg)
87 
88                 # float_msg.data = accumulated_first
89                 # pub.publish(float_msg)
90                 #Calculation of position for the second revolute
91                 accumulated_second = direction_second +
                       accumulated_second
92                 if accumulated_second > 0.548:
93                     accumulated_second = 0.548
94                     float_msg.data = accumulated_second
95                     pub2.publish(float_msg)
96                     print('Exceeding joint limits!')
97                 elif accumulated_second < -0.548:
98                     accumulated_second = -0.548
99                     float_msg.data = accumulated_second
100                    pub2.publish(float_msg)
101                    print('Exceeding joint limits!')
102                else:
103                    float_msg.data = accumulated_second
104                    pub2.publish(float_msg)
105 
106        except Exception as e:
107            print(e)
108 
109        finally:
110            pub.publish(float_msg)
111            pub2.publish(float_msg)
112 
113        termios.tcsetattr(sys.stdin, termios.TCSADRAIN, settings)
```

C.4 URDF/XACRO

Listing C.2: robot_description

```
<?xml version="1.0"?>
<robot name="twoDOF_manipulator" xmlns:xacro="www.ros.org/wiki/xacro
    ">

        <!--Xacro constants-->

        <!--Numerical-->
        <xacro:property name="mass" value="3.0"/>
        <xacro:property name="len" value="0.3"/>
        <xacro:property name="boxDim" value="0.5"/>
        <xacro:property name="pi" value="3.1416"/>
        <xacro:property name="damping" value="0.2"/>
        <xacro:property name="friction" value="0.1"/>
        <xacro:property name="radius" value="0.02"/>
        <!--Box properties-->
        <xacro:property name="boxB" value="0.06 0.01 0.18"/>
        <xacro:property name="boxS" value="0.115 0.01 0.18"/>
        <xacro:property name="boxMass" value="0.18"/>
        <!--Strings-->

        <!--Xacro math-->

        <!--Inertia along 'easy' axis, massive cylinder-->
        <xacro:property name="inertiaL" value="${(mass*radius*radius
            /4) + (mass*len*len/12)}"/>
        <!--Inertia along 'hard' axis, massive cylinder-->
        <xacro:property name="inertiaH" value="${mass*radius*radius
            /2}"/>

        <!--Xacro macros-->

        <!--Inertia and mass of massive rod-->
        <xacro:macro name="default_inertia_rod" params="masse">
                <inertial>
                        <mass value="${masse}" />
                        <inertia ixx="${inertiaH}" ixy="${inertiaH}"
                             ixz="${inertiaH}"
                                iyy="${inertiaH}" iyz="${inertiaH}"
                                izz="${inertiaL}" />
                </inertial>
        </xacro:macro>
        <!--Default link creation-->
        <xacro:macro name="linkage" params="enumeration append">
                <link name="${enumeration}_link">
                        <visual>
                                <geometry>
```

```
43                                      <cylinder length="${len}"
                                           radius="${radius}"/>
44                              </geometry>
45                              <origin xyz="0 0 ${len/2}" rpy="0 ${
                                    pi/2} 0"/>
46                              <material name="grey"/>
47                      </visual>
48                      <collision>
49                              <geometry>
50                                      <cylinder length="${len}"
                                           radius="${radius}"/>
51                              </geometry>
52                              <origin xyz="0 0 ${len/2}" rpy="0 ${
                                    pi/2} 0"/>
53                      </collision>
54                      <xacro:default_inertia_rod masse="3" />
55              </link>
56      <!--Joint creation for the macroed link-->
57              <joint name="rev_to_${enumeration}_link" type="
                    revolute">
58                      <parent link="base_link"/>
59                      <child link="${enumeration}_link"/>
60                      <origin xyz="${boxDim/2 + radius +.01 +
                            append*(len+radius)} 0 0}"/>
61                      <axis xyz="1 0 0"/>
62              </joint>
63      </xacro:macro>
64
65      <!--Color definitions-->
66      <material name="yellow">
67              <color rgba="1 1 0 1"/>
68      </material>
69
70      <material name="black">
71              <color rgba="0 0 0 1"/>
72      </material>
73
74      <material name="grey">
75              <color rgba="0.5 0.5 0.5 1"/>
76      </material>
77      <link name="world"/>
78      <!--ROV/base link-->
79      <link name="base_link"><!--The base link is where the
            reference frame is set-->
80    <visual>
81      <geometry>
82        <box size="${boxDim} ${boxDim} ${boxDim}"/>
83      </geometry>
84                      <material name="yellow"/>
```

```
85      </visual>
86                      <collision>
87                              <geometry>
88                                      <box size="${boxDim} ${boxDim} ${
                                           boxDim}"/>
89                              </geometry>
90                      </collision>
91                      <inertial>
92                              <mass value="${100.00}"/>
93                              <inertia ixx="${inertiaH}" ixy="${inertiaH}"
                                     ixz="${inertiaH}" iyy="${inertiaH}" iyz=
                                    "${inertiaH}" izz="${inertiaL}"/>
94                      </inertial>
95    </link>
96
97    <joint name="reference" type="fixed">
98      <origin xyz="0.0 0.0 1"/>
99      <parent link="world"/>
100     <child link="base_link"/>
101   </joint>
102         <!--First_link-->
103         <link name="first_link">
104                     <visual>
105                             <geometry>
106                                     <cylinder length="${len}" radius="${
                                           radius}"/>
107                             </geometry>
108                             <origin xyz="0 ${len/2 + radius} 0" rpy="${
                                   pi/2} 0 0"/>
109                             <material name="grey"/>
110                     </visual>
111                     <collision>
112                             <geometry>
113                                     <cylinder length="${len}" radius="${
                                           radius}"/>
114                             </geometry>
115                             <origin xyz="0 ${len/2 + radius} 0" rpy="${
                                   pi/2} 0 0"/>
116                     </collision>
117                     <xacro:default_inertia_rod masse="3" />
118         </link>
119         <!--Joint creation for the macroed link-->
120         <joint name="rev_to_first_link" type="revolute">
121                     <parent link="base_link"/>
122                     <child link="first_link"/>
123                     <origin xyz="0 ${(boxDim/2) + radius} 0"/>
124                     <axis xyz="1 0 0"/>
125                     <limit effort="1000.0" lower="-0.548" upper="0.548"
                            velocity="0.5"/>
```

```
126                 <dynamics damping="0.2" friction="0.2"/>
127         </joint>
128 
129         <!--second_link-->
130         <link name="second_link">
131                 <visual>
132                         <geometry>
133                                 <cylinder length="${len}" radius="${
                                        radius}"/>
134                         </geometry>
135                         <origin xyz="0 ${len/2 + radius} 0" rpy="${
                                pi/2} 0 0"/>
136                         <material name="grey"/>
137                 </visual>
138                 <collision>
139                         <geometry>
140                                 <cylinder length="${len}" radius="${
                                        radius}"/>
141                         </geometry>
142                         <origin xyz="0 ${len/2 + radius} 0" rpy="${
                                pi/2} 0 0"/>
143                 </collision>
144                 <xacro:default_inertia_rod masse="3.0" />
145         </link>
146         <!--Joint creation for the macroed link-->
147         <joint name="rev_to_second_link" type="revolute">
148                 <parent link="first_link"/>
149                 <child link="second_link"/>
150                 <origin xyz="0 ${len + 2*radius} 0"/>
151                 <axis xyz="1 0 0"/>
152                 <limit effort="1000.0" lower="-0.548" upper="0.548"
                        velocity="0.5"/>
153                 <dynamics damping="0.1" friction="0.1"/>
154         </joint>
155 
156         <!--Box end effector-->
157 
158         <!--Back-->
159         <link name="box_back"><!--The base link is where the
                reference frame is set-->
160     <visual>
161       <geometry>
162         <box size="0.06 0.01 0.18"/>
163       </geometry>
164                         <material name="yellow"/>
165     </visual>
166                 <collision>
167                         <geometry>
168                                 <box size="0.06 0.01 0.18"/>
```

```
169                     </geometry>
170             </collision>
171             <inertial>
172                     <mass value="${boxMass}"/>
173                     <inertia ixx="${0.01}" ixy="${0.0}" ixz="
                            ${0.0}" iyy="${0.01}" iyz="${0.0}" izz="
                            ${0.01}"/>
174             </inertial>
175     </link>
176
177     <!--Fix to second link-->
178     <joint name="box_to_second_link" type="fixed">
179             <parent link="second_link"/>
180             <child link="box_back"/>
181             <origin xyz="0 ${len + 2*radius} 0"/>
182     </joint>
183
184     <!--Front-->
185     <link name="box_front"><!--The base link is where the
            reference frame is set-->
186   <visual>
187     <geometry>
188       <box size="0.06 0.01 0.18"/>
189     </geometry>
190                     <material name="yellow"/>
191   </visual>
192             <collision>
193                     <geometry>
194                             <box size="0.06 0.01 0.18"/>
195                     </geometry>
196             </collision>
197             <inertial>
198                     <mass value="${boxMass}"/>
199                     <inertia ixx="${0.01}" ixy="${0.0}" ixz="
                            ${0.0}" iyy="${0.01}" iyz="${0.0}" izz="
                            ${0.01}"/>
200             </inertial>
201     </link>
202
203     <!--Fix to back part-->
204     <joint name="boxB_to_boxF" type="fixed">
205             <parent link="box_back"/>
206             <child link="box_front"/>
207             <origin xyz="0 0.115 0"/>
208     </joint>
209
210     <!--Side-->
211     <link name="box_side_left">
212     <visual>
```

```
213         <geometry>
214           <box size="0.125 0.01 0.16"/>
215         </geometry>
216                         <material name="yellow"/>
217       </visual>
218                 <collision>
219                         <geometry>
220                                 <box size="0.125 0.01 0.16"/>
221                         </geometry>
222                 </collision>
223                 <inertial>
224                         <mass value="${boxMass}"/>
225                         <inertia ixx="${0.01}" ixy="${0.0}" ixz="
                                ${0.0}" iyy="${0.01}" iyz="${0.0}" izz="
                                ${0.01}"/>
226                 </inertial>
227         </link>
228
229         <!--Fix to back part-->
230         <joint name="boxL_to_boxB" type="fixed">
231                 <parent link="box_back"/>
232                 <child link="box_side_left"/>
233                 <origin xyz="0.0375 0.0575 0.01" rpy="0 0 ${pi/2}"/>
234         </joint>
235
236         <!--Hook with pusher-->
237         <link name="hook">
238       <visual>
239         <geometry>
240           <cylinder length="0.045" radius="0.0045"/>
241         </geometry>
242                         <origin xyz="0 0 ${-0.0225 - 0.009}" rpy="0
                                0 0"/>
243                         <material name="yellow"/>
244       </visual>
245                 <collision>
246                         <geometry>
247                                 <cylinder length="0.045" radius="
                                        0.0045"/>
248                         </geometry>
249                         <origin xyz="0 0 ${-0.0225 - 0.009}" rpy="0
                                0 0"/>
250                 </collision>
251                 <inertial>
252                         <mass value="${boxMass}"/>
253                         <inertia ixx="${0.01}" ixy="${0.0}" ixz="
                                ${0.0}" iyy="${0.01}" iyz="${0.0}" izz="
                                ${0.01}"/>
254                 </inertial>
```

```
255         </link>
256
257         <link name="pusher">
258     <visual>
259       <geometry>
260         <cylinder length="0.027" radius="0.005"/>
261       </geometry>
262                         <origin xyz="0 0 0.0085" rpy="0 0 0"/>
263                         <material name="yellow"/>
264     </visual>
265                 <collision>
266                         <geometry>
267                                 <cylinder length="0.027" radius="
                                        0.007"/>
268                         </geometry>
269                         <origin xyz="0 0 0.0085" rpy="0 0 0"/>
270                 </collision>
271                 <inertial>
272                         <mass value="${boxMass}"/>
273                         <inertia ixx="${0.01}" ixy="${0.0}" ixz="
                                ${0.0}" iyy="${0.01}" iyz="${0.0}" izz="
                                ${0.01}"/>
274                 </inertial>
275         </link>
276
277         <!--Fixed to hook-->
278         <joint name="push_to_hook" type="fixed">
279                 <parent link="hook"/>
280                 <child link="pusher"/>
281                 <origin xyz="0 0.0175 -0.04175" rpy="${pi/2 - pi/12}
                        0 0"/>
282         </joint>
283
284         <!--Revolute at box-->
285         <joint name="hook_to_boxB" type="revolute">
286                 <parent link="box_back"/>
287                 <child link="hook"/>
288                 <origin xyz="0 0.0281 -0.1079" rpy="${pi/2} 0 0"/>
289                 <axis xyz="1 0 0"/>
290                 <dynamics damping="0.01" friction="0.01"/>
291                 <limit effort="1000.0" lower="-0.17" upper="${pi/4}"
                        velocity="0.5"/>
292         </joint>
293
294         <!--Button-->
295         <link name="button">
296     <visual>
297       <geometry>
298         <box size="0.013 0.013 0.004"/>
```

```
299       </geometry>
300                         <origin xyz="0 0 0" rpy="0 0 0"/>
301                         <material name="yellow"/>
302     </visual>
303                 <collision>
304                         <geometry>
305                                 <box size="0.013 0.013 0.004"/>
306                         </geometry>
307                         <origin xyz="0 0 0" rpy="0 0 0"/>
308                 </collision>
309                 <inertial>
310                         <mass value="${boxMass}"/>
311                         <inertia ixx="${0.01}" ixy="${0.0}" ixz="
                               ${0.0}" iyy="${0.01}" iyz="${0.0}" izz="
                               ${0.01}"/>
312                 </inertial>
313         </link>
314
315         <!--Prismatic at box-->
316         <joint name="button_to_boxB" type="prismatic">
317                 <parent link="box_back"/>
318                 <child link="button"/>
319                 <origin xyz="0 0.0625 -0.07" rpy="0 0 0"/>
320                 <axis xyz="0 0 1"/>
321          <limit effort="1000.0" lower="0.0" upper="0.004" velocity="
               100.0"/>
322         </joint>
323
324
325         <!--Gazebo-ROS plugin-->
326         <gazebo>
327                 <plugin name="gazebo_ros_control" filename="
                       libgazebo_ros_control.so">
328                         <robotNamespace>/uw_arm</robotNamespace>
329                 </plugin>
330         </gazebo>
331
332         <!--Transmissions-->
333         <!--First revolute-->
334     <transmission name="rev_first_link">
335         <type>transmission_interface/SimpleTransmissions</type>
336         <actuator name="$motor_rev_joint">
337             <mechanicalReduction>1</mechanicalReduction>
338         </actuator>
339         <joint name="rev_to_first_link">
340             <hardwareInterface>hardware_interface/
                    EffortJointInterface</hardwareInterface>
341         </joint>
342     </transmission>
```

```
343
344         <!--Second revolute-->
345     <transmission name="rev_second_link">
346         <type>transmission_interface/SimpleTransmissions</type>
347         <actuator name="$motor_rev_two_joint">
348             <mechanicalReduction>1</mechanicalReduction>
349         </actuator>
350         <joint name="rev_to_second_link">
351             <hardwareInterface>hardware_interface/
                    EffortJointInterface</hardwareInterface>
352         </joint>
353     </transmission>
354
355         <!--Adding gazebo attributes-->
356         <gazebo reference="button">
357                 <gravity>false</gravity>
358                 <selfCollide>true</selfCollide>
359                 <material>Gazebo/Red</material>
360                 <mu1>0.2</mu1>
361         <mu2>0.2</mu2>
362         </gazebo>
363
364         <gazebo reference="pusher">
365                 <gravity>false</gravity>
366                 <selfCollide>true</selfCollide>
367                 <material>Gazebo/Green</material>
368                 <mu1>0.2</mu1>
369         <mu2>0.2</mu2>
370         </gazebo>
371
372         <gazebo reference="hook">
373                 <gravity>false</gravity>
374                 <selfCollide>true</selfCollide>
375                 <material>Gazebo/Green</material>
376                 <mu1>0.2</mu1>
377         <mu2>0.2</mu2>
378         </gazebo>
379
380         <gazebo reference="second_link">
381                 <gravity>false</gravity>
382                 <selfCollide>true</selfCollide>
383                 <material>Gazebo/Grey</material>
384                 <mu1>0.2</mu1>
385         <mu2>0.2</mu2>
386         </gazebo>
387
388         <gazebo reference="first_link">
389                 <gravity>false</gravity>
390                 <selfCollide>true</selfCollide>
```

```
391                 <material>Gazebo/Grey</material>
392                 <mu1>0.2</mu1>
393         <mu2>0.2</mu2>
394         </gazebo>
395
396         <gazebo reference="base_link">
397                 <gravity>false</gravity>
398                 <material>Gazebo/Yellow</material>
399         </gazebo>
400 </robot>
```

C.5 Launch file URDF

Listing C.3: Launch file URDF

```
<launch>
<!--Sets global constants(or strings) for script-->
  <arg name="model" default="$(find ros_underwater_simulator)/urdf/
      uw_arm_control_ee.urdf"/><!--Path to urdf model-->
  <arg name="gui" default="true" /><!--GUI toggle-->
  <arg name="headless" default="false"/>
  <arg name="debug" default="false"/>
  <arg name="paused" default="false"/>
  <arg name="use_sim_time" default="true"/>

  <param name="robot_description" command="$(find xacro)/xacro $(arg
       model)" /><!--model is linked to the specified path and
      executed both as xacro and urdf-->
  <param name="use_gui" value="$(arg gui)"/>

<!--Loads Gazebo related elements-->
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="debug" value="$(arg debug)" />
    <arg name="gui" value="$(arg gui)" />
    <arg name="paused" value="$(arg paused)"/>
    <arg name="use_sim_time" value="$(arg use_sim_time)"/>
    <arg name="headless" value="$(arg headless)"/>
  </include>

<!--Loads the URDF with the arg containing the path to specified
     URDF-->
  <node name="urdf_spawner" pkg="gazebo_ros" type="spawn_model" args
      ="-z 0 -unpause -urdf -model robot -param robot_description"
      respawn="false" output="screen" />

<!--Rviz loading(Optional)-->
  <!--<node name="rviz" pkg="rviz" type="rviz" args="-d $(arg
      rvizconfig)" />-->

<!--Loads controllers-->
  <!-- <rosparam file="$(find ros_underwater_simulator)/controller/
      controller_instructions.yaml" command="load" ns="
      first_rev_controller"/>
  <node name="controller_spawner_first_rev" pkg="controller_manager"
       type="spawner" args="first_rev_controller" /> -->

<!--Loads controllers-->
  <!-- <rosparam file="$(find ros_underwater_simulator)/controller/
      controller_instructions_second_rev.yaml" command="load" ns="
      second_rev_controller"/>
  <node name="controller_spawner_second_rev" pkg="controller_manager
```

```
      " type="spawner" args="second_rev_controller" /> -->
35
36 <!--Calls and loads the joint state controller which publishes the
     joint states-->
37   <!-- <rosparam file="$(find ros_underwater_simulator)/controller/
       jointStates.yaml" command="load" ns="joint_state_controller" />
38   <node name="joint_state_controller_spawner" pkg="
       controller_manager" type="spawner" args="joint_state_controller
       " /> -->
39
40 <!--Calls and loads the robot state publisher-->
41   <!-- <param name="my_robot_description" textfile="$(find
       ros_underwater_simulator)/urdf/uw_arm_control_ee.urdf"/> -->
42   <!-- <node pkg="robot_state_publisher" type="robot_state_publisher
       " name="rob_st_pub" >
43       <remap from="robot_description" to="robot_description" />
44       <remap from="joint_states" to="different_joint_states" />
45     </node> -->
46 </launch>
```

C.6 Launch Controllers

Listing C.4: Launch fiel for controllers

```
<launch>

  <!-- Load joint controller configurations from YAML file to
      parameter server -->
  <rosparam file="$(find uw_arm_control)/config/
      controller_instructions.yaml" command="load"/>

  <!-- load the controllers -->
  <node name="controller_spawner" pkg="controller_manager" type="
      spawner" respawn="false"
    output="screen" ns="/uw_arm" args="joint1_position_controller
       joint2_position_controller joint_state_controller"/>

  <!-- convert joint states to TF transforms for rviz, etc -->
  <node name="robot_state_publisher" pkg="robot_state_publisher"
      type="robot_state_publisher"
    respawn="false" output="screen">
    <remap from="/joint_states" to="/robot/joint_states" />
  </node>

</launch>
```

C.7 YAML controllers

Listing C.5: YAML

```
uw_arm:
  # 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: rev_to_first_link
    pid: {p: 100.0, i: 0.01, d: 10.0}
  joint2_position_controller:
    type: effort_controllers/JointPositionController
    joint: rev_to_second_link
    pid: {p: 15.0, i: 0.1, d: 1.0}
```