University of Agder

Intelligent Robotic Rehabilitation Through a Human-Robot Interaction

A starting point for a robotic infrastructure which can be an aid for physiotherapists by issuing and adjusting assistive or resistive exercises

Lars Bleie Andersen

SUPERVISORS
Filippo Sanfilippo, Associate Professor
Mohammad Poursina, Associate Professor

University of Agder, 2021
Faculty of Engineering and Science
Department of Engineering and Sciences

Master

Acknowledgements

It has been a pleasure to work with the EVEr3 robot and the people that I have been in contact with during this project. I would like to acknowledge:

Alexandra Christine Hott and Anders M. Östling for the consultations regarding medical considerations and the cuff.

Filippo Sanfilippo and Muhammad Ali Poursina for their contributions.

Jesper Smith from Halodi Robotics for answering questions regarding the EVEr3 robot.

Arminas Gronskis for the collaboration during the initial C++ and ROS2 programming and for being a great guy when working with the EVEr3 robot in parallel projects.

Abstract

This thesis contains a starting point for a robotic infrastructure which can be an aid for physiotherapists by issuing and adjusting assistive or resistive exercises. With added equipment for specialized operations, the Halodi EVEr3 Robot could be utilized as a basis for this kind of infrastructure. When a physiotherapist is performing a lower extremity rehabilitative movement on a patient, the EVEr3 robot could record this behaviour and mimic it for a desired number of repetitions. The design of a prototype is thought to be done safely as a collaboration with healthcare professionals.

Contents

List of Figures

List of Tables

List of Abbreviations

ARM	Assistive Robotic Manipulator
BASh	Bourne Again Shell
D	Derivative
DAE	Digital Asset Exchange
DDS	Data Distribution Service
DH	Denavit-Hartenberg
DoF	Degrees of Freedom
ERF	Electro-Rheological Fluid
I	Integral
MPC	Model Predictive Control
OSRF	Open Source Robotics Foundation
P	Proportional
QoS	Quality of Service
ROS	Robot Operating System
RTPS	Real-Time Publish Subscribe protocol
SDF	Simulation Description Format
SEA	Series Elastic Actuator
UML	Unified Modeling Language
URDF	Unified Robotic Description Format
XML	eXtensible Markup Language
YAML	YAML Ain't Markup Language (a recursive acronym)

Nomenclature

$\dot{\theta}_i$	The angular velocity of the ith joint
ω	The imaginary term in a complex frequency
σ	The real term in a complex frequency
$\mathbf{A}_i$	The homogeneous transformation matrix for the ith joint
$\mathbf{J}$	The Jacobian matrix which transforms velocities
$\mathbf{T}_i^0$	The total transformation matrix from the base frame to the ith frame
$e(t)$	Feedback error value
I_{yy}	Moment of inertia about the y axis
K_d	The derivative gain
K_i	The integral gain
K_p	The proportional gain
K_u	The ultimate gain
$L1$	Length of the thigh link from the start joint to the end joint
$L2$	Distance from the joint of the crus to the center of the handle
m	Mass
o_i	The origin of the ith coordinate frame
s	The complex frequency
t	Time variable
T_d	The derivative time
T_i	The integral time
T_u	The ultimate time period
$u(t)$	Correction value (the input to the process from the controller)
x	Length on x-axis
$x(t)$	Reference value
x_i	The x-axis of the ith coordinate frame
x_{ee}	The x-coordinate of the end-effector, measured from the base frame
y	Length on y-axis
$y(t)$	Process value

y_i The y-axis of the ith coordinate frame

z Length on z-axis

z_i The z-axis of the ith coordinate frame

z_{ee} The y-coordinate of the end-effector, measured from the base frame

Chapter 1

Introduction

1.1 Motivation

Robotic systems are believed to be used as standard rehabilitation tools in the near future, many inspired by the cost of the labour intensive care that is required to cover the number of patients in need for such treatment. This is due to the fact that physical rehabilitation can take several weeks or even months until full range of motion and joint flexibility are regained [2, 12]. Utilizing robotics for rehabilitation can increase the number of training sessions and reduce personnel cost by assigning one therapist to train several robots. Robotic tools can also implement a variety of mechanical manipulations that are impossible for physical therapists to execute due to various current human limitations (e.g. sensing, strength, speed and repeatability) [11, 25]

When deciding a useful contribution to the area of rehabilitation robots, it was discussed with healthcare professionals that patients with decreased voluntary lower extremity function could be suitable as the target group. This decreased function is often a result of neurological injuries (e.g. spinal cord injuries, head injury or stroke). These are patients in which physiotherapists passively move the leg, because they can not actively move it themselves in the beginning, and then as they improve they can progressively participate in active movement of their leg.

The knee was selected as the target joint. This is a joint with problems for many different patient groups with illnesses as gait, diabetes and stroke. It is also a common target joint for rehabiliation after surgery [25, 15].

EVEr3 is a humanoid robot with a friendly appearance which can contribute to a more positive relationship with the patient during rehabilitation in comparison to a specialized end effector robot. The end goal for the project was therefore set to create a system that can induce a capability in EVEr3 to mimic assistive movements for knee recovery. Active assistive exercises is the type of exercises where the robot gives aid to an active movement by the patient. This type of rehabilitation has a positive effect if the robotic device is adaptive to the needs of the patient. Passive assistive exercises is the type of exercises where a physical therapist performs a desired motion and the robotic device mimics this motion on the patient without any intended interfering movement from muscle activation by the patient [20].

1.2 Rehabilitation Robots

There are two types of rehabilitation robots: One type is assistive robots that aid people with lost limbs by using telemanipulation, which is the transmitting of a desired movement by the use of a device. The other type is called rehabilitators or therapy robots. Therapy robots are machines or tools for rehabilitation therapists that allow patients to perform specific movements that improve recovery and minimize functional decline [20].

Robotic systems used in the field of neurorehabilitation from brain injuries can be categorized into exoskeletons and end effector type robots, where an exoskeleton is a wearable robot with joints and links [12]. Today, powered exoskeletons are being produced by companies such as Ekso Bionics, ReWalk Robotics, Rehab Robotics Hocoma and Lockheed Martin.

Rehabilitation robots should always provide targeted physical support adapted to the functional abilities of the patient in a way to enable functional movements. They should also be able to adapt their output impedance and physical support to the need of the patient without disrupting functional movement patterns. Most active rehabilitation devices contain an actuation system and a degree of intelligence [15]. did not favorably or negatively affect the gains in motor control or strength associated with this training

1.3 State of the Art in Therapy Robots

The first modern therapy robot was designed in 1992 at Massachusetts Institute of Technology, called MIT-MANUS (MANUS = "Hand" in Latin language) [19]. It has the ability to not only record the hand movement of a therapist and then perform it on a patient, but also execute the movement with varying degrees of firmness [19]. A 2004 study on the effect of MIT-MANUS on 46 subjects found no significant improvements for stroke patients that received assistive or resistive therapy [43]. Kahn et.al. (2014) suggests that a possible explanation for this, is that the form of robotic forces (assistive or resistive) did not matter as much as the patients themselves tried to move [22]. Another study in 2004 showed that an adaptive control strategy where the robot adjusted the interference based on the capability of the patient was better for moderate impaired stroke patients [20].

In 2006, an intelligent robotic system was designed by Aktogan et. al. [2]. This was meant to be an answer to the limitations of the therapy robots at the time. These limitations included the lack of motion freedom and active control, meaning that the robots could not perform complicated exercises. Their system is able to interpret patient reactions, storing the information received, acting according to the available data and learning from the previous experiences.

A portable Active Knee Rehabilitation Orthotic Device intended to guide and facilitate the recovery of gait was designed by Weiberg et.al. in 2007 [47]. The skeleton consists of a brake, brace, gear and sensors. The knee brace includes resistive and variable ERF based damping which is controlled in ways that promotes motor recovery in stroke patients.

ARMin is a therapy robot that was developed by Nef et. al. in 2009 for patient-cooperative arm therapy [32]. It is a semi-exoskeleton robot equipped with position, force and torque sensors. The therapy robot takes into account the activity of the patient and provides only the required support. It allows precise joint actuation and 3D movement of the arm, and includes a audiovisual user interface.

In 2011, a powered exoskeleton for robot-assisted rehabiliation was developed by Beyt et.al. [3]. This skeleton was made to improve physical human–robot interaction by using pleated pneumatic artificial muscles as high-torque actuators for the skeleton and a PSMC for sliding mode control in order to achieve safe and adaptable guidance.

Chen et.al. wrote a paper in 2016 which presented a knee–ankle–foot robot that is portable to carry out training at outpatient and home settings [8]. It includes a SEA made up by two springs in series with different stiffness values as the basis for the robot-human interaction, and it records the movements of the skeletal muscles via electromyography.

1.4 Safety in Human-Robot Interactions

A rehabilitative procedure involving robots can have severe consequences, so patient safety must be considered. As Pan et.al. (2016) states regarding a robotic rehabilitative exercise: The patient should be taken as a “cooperator” of the training activity, and the movement speed and range of the training movement should be dynamically regulated according to the internal or external state of the subject, just as what the therapist does in clinical therapy [35].

Vasic and Billard (2013) identified four elements that must be considered during a human-robot interaction [46]: Where is the biggest danger, who is the most endangered person in the interaction with a robot, what are the consequences of potential injuries, and which factors have the greatest impact on safety. They cite Ogorodnikova (2008) [33] when stating that accidents caused by robots can be grouped into three main categories: Engineering errors, human mistakes and poor environmental conditions.

Mohebbi (2020) made a review of means for safety when utilizing assistive robotic manipulators (Fig. 1.1). One notable feature is the need for task adaptation by using measurements of the force or changes in the joint positions due to the motion of the musculoskeletal system, and feed them back as control inputs to the robot. In that way minimum interaction forces can be achieved while eliminating task tracking errors. This is often done by implementing impedance or admittance control [31]. An article by Neville Hogan (1984) also generally states that the control of any robot manipulator in contact with its environment should not only be concerned with the trajectory control of the manipulator alone, but be combined with impedance control [18]. The mechanical impedance of an object gives an indication of its ability to resist movement when a force/torque is applied to it, and impedance control is used in rehabilitation robots to compensate with a torque for a position deviation created by external movement from the patient. For linear systems, the inverse of impedance is the admittance which gives an indication of the ability to resist a force/torque when a movement is applied to it. Admittance control is used in rehabilitation robots to compensate with a position deviation from the planned trajectory when an external torque from the patient is registered by the robot [42].

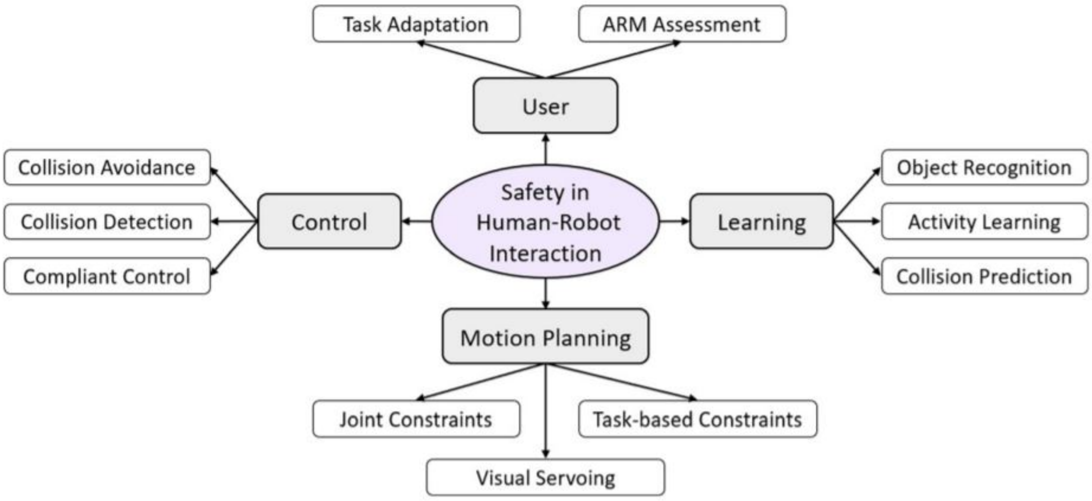


Figure 1.1: Safety actions during interaction with ARMs [31]

1.5 Project Objectives

In order to realize a functional prototype, five objectives were set at the start of this project. The ULM activity diagram in Fig. 1.2 contains the steps that were set for the desired procedure.

1) Create a code that can manipulate the joints and hands of EVEr3
   a) Simulate the transfer of data to digital twin
   b) Test on real EVEr3 robot

2) Create a leg model for a Gazebo simulation

3) Design a cuff for the human-robot interaction

4) Collaborate with healthcare professionals
   a) Input on the created cuff
   b) Input for a rehabilitative movement

5) Create an interaction between the leg model and the digital twin
   a) Constrain one hand of the digital twin onto a limb of the leg model
   b) Move the limb of the human model
   c) Make the digital twin follow along

d) Detect and store the movement of digital twin into an array
e) Make the digital twin perform the movement a set number of times

6) Implement on EVEr3
   a) Perform the procedure
   b) Adjust the control parameters

1.5.1 UML Activity Diagram

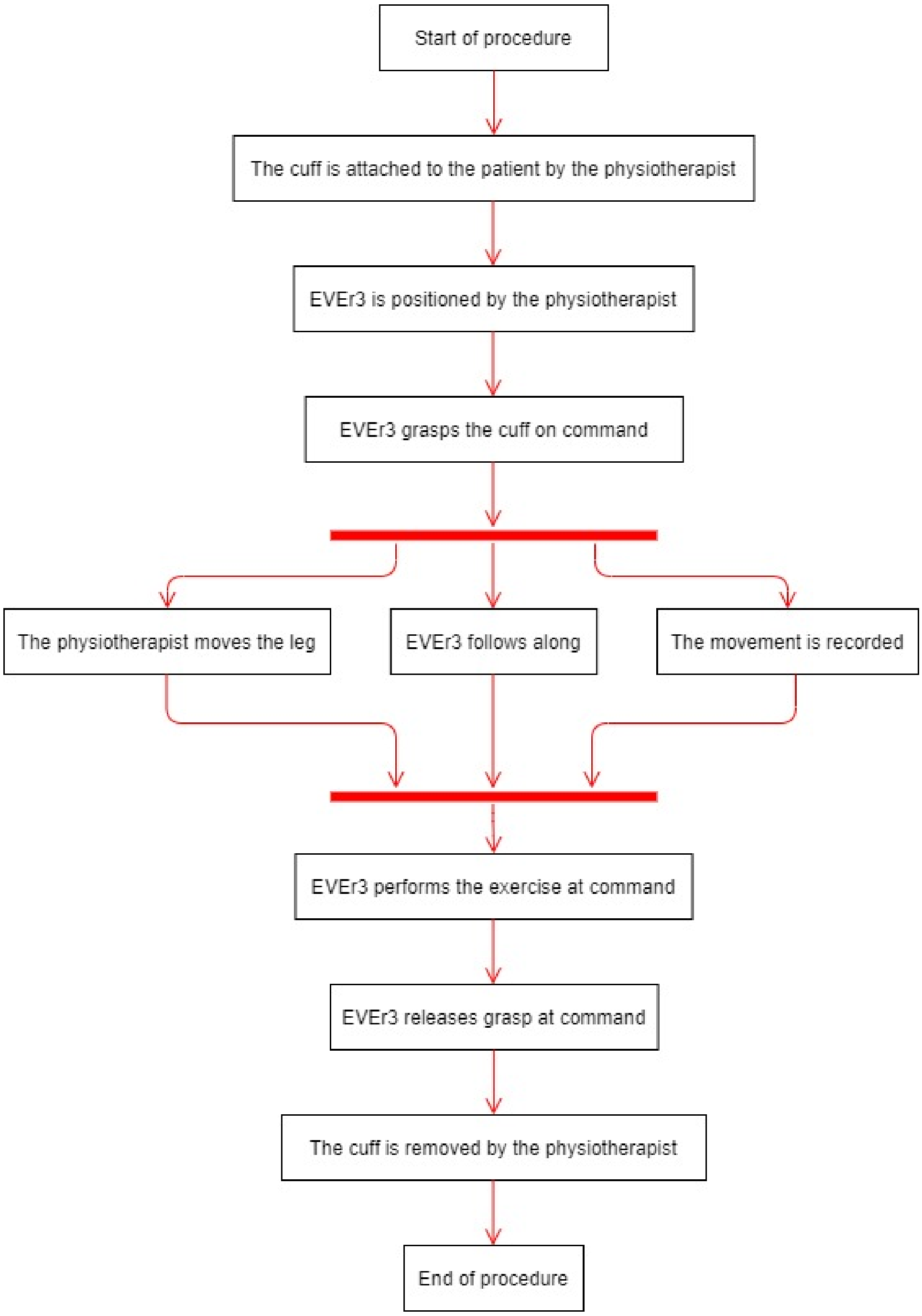


Figure 1.2: ULM activity diagram for the wanted steps in the rehabilitation procedure

Chapter 2

Theory

2.1 The EVEr3 Robot

The EVEr3 (s/n A0-20-04-002) (Fig. 2.1) is the robot that was used as the basis for this project. It is a human-sized (183 [cm], 76[kg]) robot with 24 Revo1 motors that was made by Halodi. The battery is 54V 20Ah with a 1.1 kWh output to the robot. The battery and charging system is designed to allow operation of the robot while charging. It uses a synthetic rope-based transmission system for each joint that makes it possible to control the joint torques directly using current control on the motors. It has two computers; one for balance and motion control (Intel i7-8650U with 16[GB] RAM) and one for computing and connectivity with an Intel Core i7-8850H processor, a RTX 2080 GPU, 32[GB] RAM, a 1[TB] Solid State Storage and a 867[Mbps] 802.11ac wifi card.

EVEr3 uses open loop torque control for the joints, open loop Cartesian force control of the arms and has a MPC whole body balance system with push recovery. The robot can interface to various hands and grippers through mechanical adapters. It can handle a total payload of 15 [kg] and 6[kg] per straight arm, excluding the hands/grippers. The maximum velocity for wheel movement is $12[\frac{km}{h}]$ [13]. It comes with two emergency stop-buttons; one that is wired and placed on its right shoulder and another which is wireless and portable up to 20[m] away from the robot.

Figure 2.1: The EVEr3 Robot made by Halodi Robotics

2.2 Software

2.2.1 ROS2

ROS is a framework for writing robot software. It contains libaries and tools that are intended to make robot programming easier and more collaborative by having a standardized programming language for everyone who wants to program robots. ROS became the standard for OSRF in 2013 [37].

ROS2 Foxy was the lastest version of ROS at the time this project got started, and the biggest difference between ROS1 and ROS2, except from all the syntax changes, is the use of DDS and RTPS middelware for communication, and the QoS which allows users to specialize communication between nodes [34].

ROS2 has libraries which supports the programming languages C++ and Python. The ROS2 files provided by Halodi were all written in C++, so all ROS2 contributions in this project were also written in the C++ language. The most used ROS2 filesystem concepts are Workspaces, Packages and Nodes. There are three types of communication interfaces: Topics, Services and Actions.

Workspaces

A ROS2 workspace is a directory which one creates with the intention of using it as a workspace for one or more ROS2 packages. One then has to create a subfolder called *src* which contains all packages in that workspace. It is necessary to source the workspace in the workspace directory every time one wants to use a package. Sourcing involves loading the files in all subfolders into the current shell script and make the files available for use in that particular shell. If one uses a package often, like the ROS2 installation package, one can add them to the Bash Shell Script (.bashrc) which contain various commands to be initialized for every shell.

Packages

A ROS2 Package is the base for everything that a ROS2 Program needs to function. When one types the standard command for CMake, which is the C++ version for ROS2, it will create a package folder with the *include* and *src* subdirectories, in addition to the CMakeLists.txt file and the package.xml file which together must contain the maintainers, dependencies and directories that either are included in a script or which are required in order to run the package.

There are six types of dependencies that a package can have: *Build* dependencies specify which packages are needed to build the package. This is the case when any file from these packages is required at build time. *Build export* dependencies specify which packages are needed to build libraries against this package. *Execution* dependencies specify which packages are needed to run code in the package. This is the case when the code depends on shared libraries. *Test* dependencies specify only additional dependencies for unit tests. *Build tool* dependencies specify build system tools which this package needs to build itself. *Documentation tool* dependencies specify documentation tools which the package needs to generate documentation. In addition, there is a *Depend* tag, which specifies three dependencies; *Build*, *Build export*, and *Execution*.

Nodes

A ROS node is an independent executable file which performs computation.. They are divided into publisher/provider nodes that generate data and subscriber/client nodes that are interested in data. Nodes can communicate with each other using messages delivered through topics. These messages contains the data which has been computed in the publisher node and is useful for subscriber nodes.

Topics

A topic is a communication line between nodes that wants to exchange messages. The nodes have no idea who they are exchanging messages with, only that they publish or subscribe to a particular

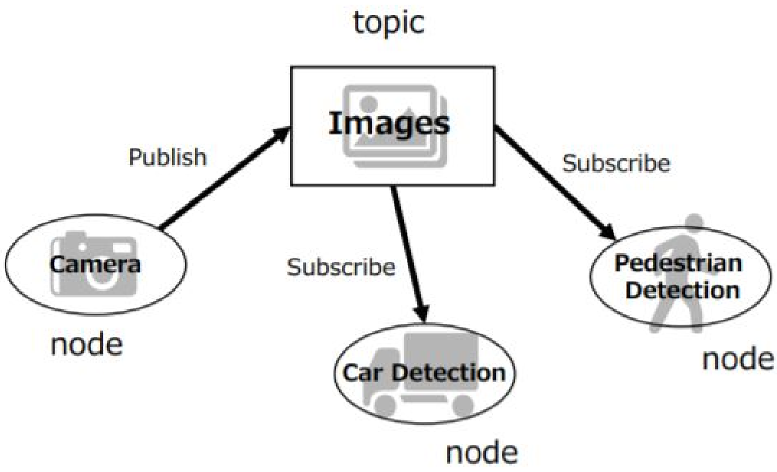


Figure 2.2: An example of communication between ROS2 nodes through a topic [29]

topic. There can be multiple publishers and subscribers to a topic, and topics are intended for one-way communication either way the data flows through the topic.

Services

A service uses a pair of messages; one for a request and one for a reply. A providing node offers a service, and a client node calls the service by sending the request message and awaiting the reply, but there is no information about the progress of the transfer.

Actions

An action is the third form of communication within ROS. Action clients send a request to an action provider and will get a feedback of the transfer progress while receiving the wanted data. Actions also allows the client to cancel the transfer before it completes.

2.2.2 Gazebo

Gazebo is a 3D dynamic simulator which can handle multiple robot models in complex environments. It supports testing of algorithms and designing robot models. The model of the robot is inserted in a Gazebo World environment which can include robots, sensors, objects and global parameters including light to see the model and physics properties of the world.

Plugins can be used to communicate with a model in Gazebo. They examine the model tags and loads the hardware interfaces, and have direct access to all the functionality of the simulator through standard Gazebo-made C++ classes. Gazebo includes inbuilt plugins from which one can build customised plugins. There are six types of plugins: Model, sensor, system, visual and GUI.

Models

Models can be made as a URDF model or as a SDF model. Both URDF and SDF are made as XMLs, but the difference is that URDF models does only contain the information about the robot model itself, i.e. kinematic and dynamical properties, while SDF models contains both a description of the model and the Gazebo world it is presented in. URDF is the standard format for ROS models, while SDF was created as a part of the Gazebo simulator. The URDF files are made up of elements, such as <robot>, <link> and <joint>, arranged in hierarchical structures called XML trees.

2.2.3 RViz

RViz is a 3D vizualisation tool for testing the behaviour of robot models. One can publish ROS2 messages to the model and see if it behaves correctly when publishing messages (e.g. altering the joints of a robot model). RViz uses models created with the URDF format. By adding the dependencies for wanted behaviour in the package.xml file, RViz can visualize the behaviour of the robot and what it is perceiving.

2.2.4 The Digital Twin

The digital twin was given as a model from Halodi. All the implemented Gazebo simulation files from Halodi were imported using Git. Halodi Robotics has made their GitHub repository available for the public so that anyone can try to download and use the model of the EVEr3 Robot. All files created by Halodi which includes messages, API, model etc. can be found in this repository. Halodi provides a Gazebo world with a model of the robot and the following examples of manipulating the model via a controller:

- Wave the right hand
- Return to default pose
- Move the left hand in a 5 point trajectory
- Drive in a circle
- Move the head up and down

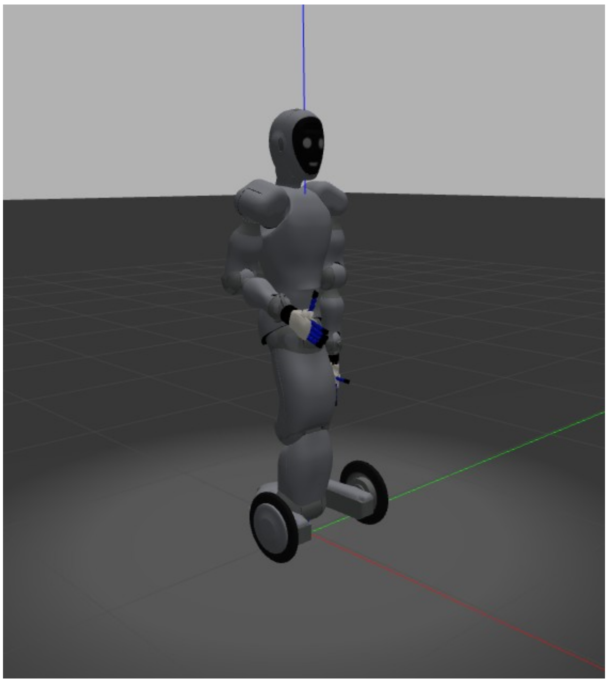


Figure 2.3: The digital twin in its Gazebo world

When comparing the modelling in Gazebo with the real EVEr3, there will be a Real-time factor which is an indicator of how much slower Gazebo executes the desired movements than intended by the script. In the appended video "RTF_example.avi", there is an example of this effect. Even with a RTF of 0.85, there is a significant delay. An extract from this video is presented in Fig. 2.4.

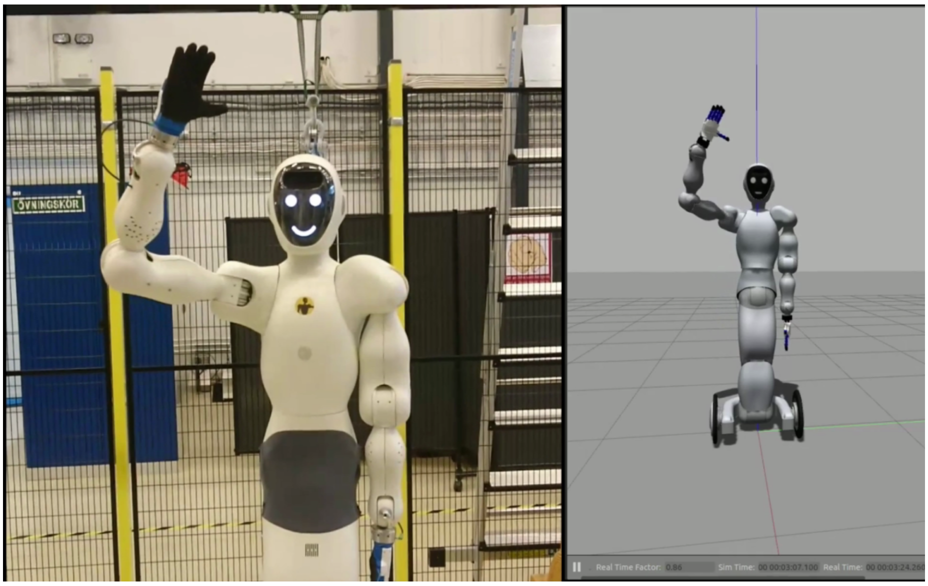


Figure 2.4: Executing a ROS2 command for both the real EVEr3 and the digital twin

2.3 Kinematics

2.3.1 Forward Kinematics

In order to manipulate a model, the kinematic formulas can be used for the motion planning. Kinematics is concerned with positions, velocities and accelerations, but not the forces that cause the movement to happen. All robots are considered to be made up of bodies called links which are connected by joints to form a kinematic chain as a multibody system with a rigidly attached coordinate frame for each link. The first coordinate frame (o_0, x_0, y_0, z_0) is the base frame of reference and the end point is called the end effector.

Forward kinematics is performed to calculate the resultant motion of the end effector from joint movement in the kinematic chain. The movement of the end effector and its derivatives is called the cartesian or task space and the joint movement with its derivatives is called the joint space. The Denavit-Hartenberg convention is often used to describe forward robot kinematics for robots with more than one DoF, where the DoF is equal to the total number of independent joint displacements. The DH convention consist of using four parameters to describe the geometrical relationships between the links and homogeneous transformation matrices to describe how the relationships are altered with regards to translation and rotation [28].

The four parameters are:

- a_i: Link length. The distance between the axes z_0 and z_1, and is measured along the axis x_1
- d_i: Link offset. The perpendicular distance from the origin o_0 to the intersection of the x_1 axis with z_0 measured along the z_0 axis
- α_i: Link twist. The angle between the axes z_0 and z_1, measured in a plane normal to x_1. The positive sense for α is determined from z_0 to z_1 by the right-handed rule
- θ_i: Joint angle. The angle between x_0 and x_1 measured in a plane normal to z_0

Each homogeneous transformation matrix A_i is the product of four transformations:
$A_i = Rot_{z,\theta_i} Trans_{z,d_i} Trans_{x,a_i} Rot_{x,\alpha_i}$ (Eq. (2.1, 2.2)).
There are two conditions that must be fulfilled in order to express the transformations in this form [28] (also see Fig. 2.5):

- The axis x_i is perpendicular to the axis z_{i-1} and z_i
- The axis x_i intersects the axis z_{i-1}

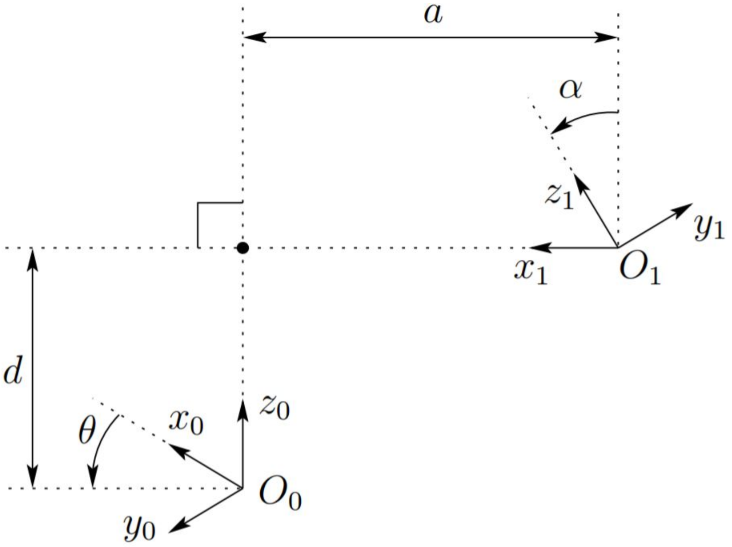


Figure 2.5: Coordinate frames satisfying the two DH conditions [28]

$$A_i = \begin{bmatrix} cos(\theta_i) & -sin(\theta_i) & 0 & 0 \\ sin(\theta_i) & cos(\theta_i) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\alpha_i) & -sin(\alpha_i) & 0 \\ 0 & sin(\alpha_i) & cos(\alpha_i) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.1}$$

$$= \begin{bmatrix} cos(\theta_i) & -sin(\theta_i)cos(\alpha_i) & sin(\theta_i)sin(\alpha_i) & a_i cos(\theta_i) \\ sin(\theta_i) & cos(\theta_i)cos(\alpha_i) & -cos(\theta_i)sin(\alpha_i) & a_i sin(\theta_i) \\ 0 & sin(\alpha_i) & cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

For a planar arm with only revolute joints, the only non-zero variables are a_i and θ_i, so the matrices will be as in Eq. (2.3, 2.4).

$$A_i = \begin{bmatrix} cos(\theta_i) & -sin(\theta_i) & 0 & 0 \\ sin(\theta_i) & cos(\theta_i) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

$$= \begin{bmatrix} cos(\theta_i) & -sin(\theta_i) & 0 & a_i cos(\theta_i) \\ sin(\theta_i) & cos(\theta_i) & 0 & a_i sin(\theta_i) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

In order to get from the base frame (o_0, x_0, y_0, z_0) to frame i (o_i, x_i, y_i, z_i), one multiplies the homogeneous transformation matrices as in Eq. (2.5).

$$\mathbf{T}_i^0 = A_0(...)A_i \tag{2.5}$$

2.3.2 Inverse Kinematics

Inverse kinematics is performed to calculate the joint motion in the kinematic chain which is required for a movement of the end effector. Solving the inverse kinematics of a robotic manipulator is often harder than solving the forward kinematics due to multiple solutions which is dependant on the configuration of the chain [16].

A numerical approach is common for solving the inverse kinematics. For 3D systems it is often computationally demanding and takes a long time to perform inverse kinematics [26]. The complexity of inverse kinematics decreases with the number of links, so for simpler planar systems, the inverse kinematics can be calculated by a geometric approach: Using the reference frame of the base to find the inverse relations to the position of the end effector [28].

2.4 PID Control

A control system is needed to ensure steady-state accuracy when the desired joint values are reached. This is done by the Halodi PID controller for both the digital twin and the EVEr3 robot. The EVEr3 robot has PID controllers for every joint, and also the control used in the YAML scripts for ROS control uses a PID controller. This is a controller that is used in many systems where there is no offset and the process requires a fast response time. The proportional control of the system counteracts the reaction to a small change in the error value ($e(t) = x(t) - y(t)$). The integral control is added because the P-controller alone will never reach the desired steady state value. The integral control always attempts to make $e(t) = 0$, resulting in an overshoot of $u(t)$. The derivative control is therefore used to make the system converge to steady state even if the system is oscillating. It does this by slowing down the correction [27]. The mathematical expression for a PID controller is shown in Eq. (2.6) and the equivalent Laplacian form of the expression is given in Eq. (2.7). There are several methods that has been made with the purpose of calculating the loop tuning parameters: Ziegler–Nichols, Cohen–Coon, Internal model control, Gain-phase margin and Optimum integral error for load disturbance [44].

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de(t)}{dt} \tag{2.6}$$

$$U(s) = K_p + \frac{K_i}{s} + K_d s \tag{2.7}$$

$$s = \sigma + j\omega \tag{2.8}$$

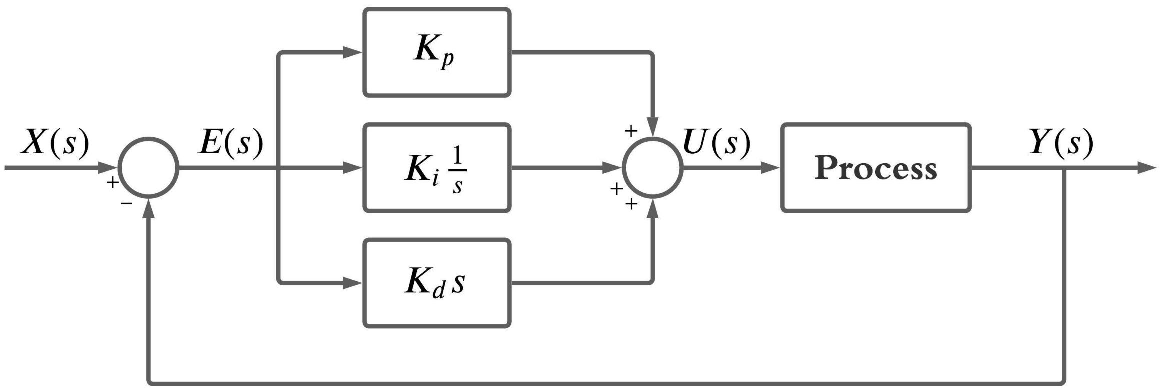


Figure 2.6: Block diagram of a PID controller represented in the frequency domain

Ziegler-Nichols: Ultimate Sensitivity tuning method

The Ziegler-Nichols Ultimate Sensitivity tuning method has been used in the field of robotics and is a robust, simple method [9]. It is performed by first setting the integral and derivative gains to zero. The proportional gain is then increased until one gets a stable oscillation from the output. This gain value is recorded as K_u and the period of this stable oscillation is recorded as T_u. K_u and T_u are used to set K_p, T_i and T_d according to Tab. 2.1 depending on the desired behaviour. The K_i value is calculated by Eq. (2.9) and the K_d value by Eq. (2.10) [45].

Rule name	K_p	T_i	T_d
Classic Ziegler-Nichols	$0.6K_u$	$0.5T_u$	$0.125T_u$
No overshoot	$0.2K_u$	$0.5T_u$	$0.33T_u$

Table 2.1: Standard Ziegler-Nichols values for ultimate sensitivity tuning method [45]

$$K_i = \frac{K_p}{T_i} \tag{2.9}$$

$$K_d = K_p T_i \tag{2.10}$$

2.5 Medical Considerations

For our rehabilitation robot, it is important that the movements do not induce or contribute to disability, which is defined as instability that interferes with the required function of the knee [1]. A second consideration for therapy robots is the avoidance of abrasion. Abrasion is a superficial graze, which is a damage to the skin caused by external scraping. Patients with neurological problems in the lower extremity can have reduced sensation due to the neurological injury, which makes them more susceptible to developing abrasions and sores. Another vulnerable group of patients is people with diabetes for whom the damage is due to a series of multiple mechanisms, including decreased cell and growth factor response, which lead to diminished peripheral blood flow and decreased creation of new blood vessels in the body [7].

2.5.1 Allergic Contact Dermatitis

Allergic contact dermatitis is a common health issue that must be avoided when creating the cuff for the human-robot interaction. It consists of two phases, where the initial phase consists of repeated exposure, followed by an inflammatory reaction [24].

A lot of metals like nickel, gold, palladium, mercury and cobalt can cause allergic contact dermatitis [14]. For our project, the relevant allergic contact dermatitis is textile contact dermatitis which affects people who are allergic to certain fabrics. Notable fabrics that commonly cause allergic reactions are latex and polyester.

Hypoallergenic fabrics are then required. Hypoallergenic fabrics are woven tightly and made of natural fibers. Common hypoallergenic fabrics are sheepskin, cotton, linen and silk.

2.5.2 Knee Anatomy

The knee is one of the largest and the most complex synovial joint in the body [21]. It is called synovial because the bone-to-bone connection is parted by a synovial fluid (Fig. 2.7). Synovial fluid is a plasma that also contains substances like hyaluronic acid which is secreted by the joint tissues around the fluid. Temperature has an effect of the viscosity of the fluid; The viscosity of the fluid decreases in inflammatory conditions, and in decreasing temperature the viscosity of the synovial fluid increases, which may explain why joint stiffness increases in colder weather [6].
The knee joint consists of four bones which include the femur, fibula, tibia and patella (Fig. 2.8). The thin fibula bone is fixed to the back of the tibia by very short tendons, connected to the femur with tendons and to the hamstrings with ligaments. Tendons connect muscles to bone and ligaments connect bone to bone. The end of the femur has two rounded shapes that fits into the top of the tibia, and this connection is joined partly together by a fibrocartilagous meniscus that provides a soft joint connection, and also by several bursae (cushioning sacks of fluid) and ligaments that cover this connection [4].

The patella "hangs" in the front of this joint connection and is covered by a tendon which is connected to the quadriceps and a ligament that connects to the femur. The patella also has other structures connected to it, like the fat pad beneath the ligament and a bursa on the end of the ligament and beneath the tendon. Each bursa are prone to inflammation caused by tramua and

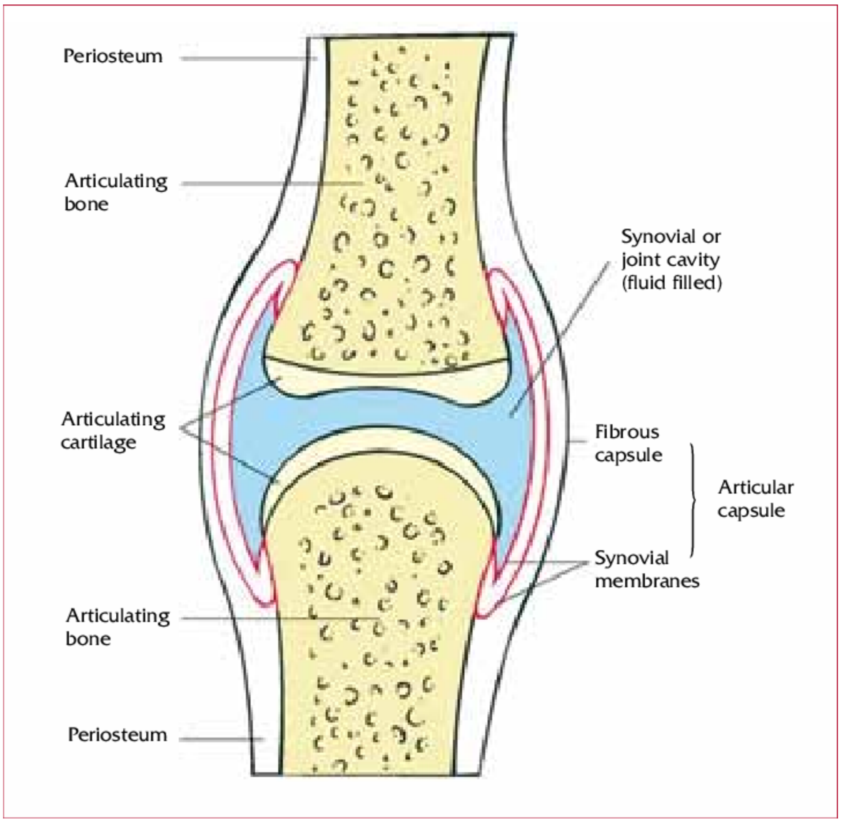


Figure 2.7: A general synovial joint [10]

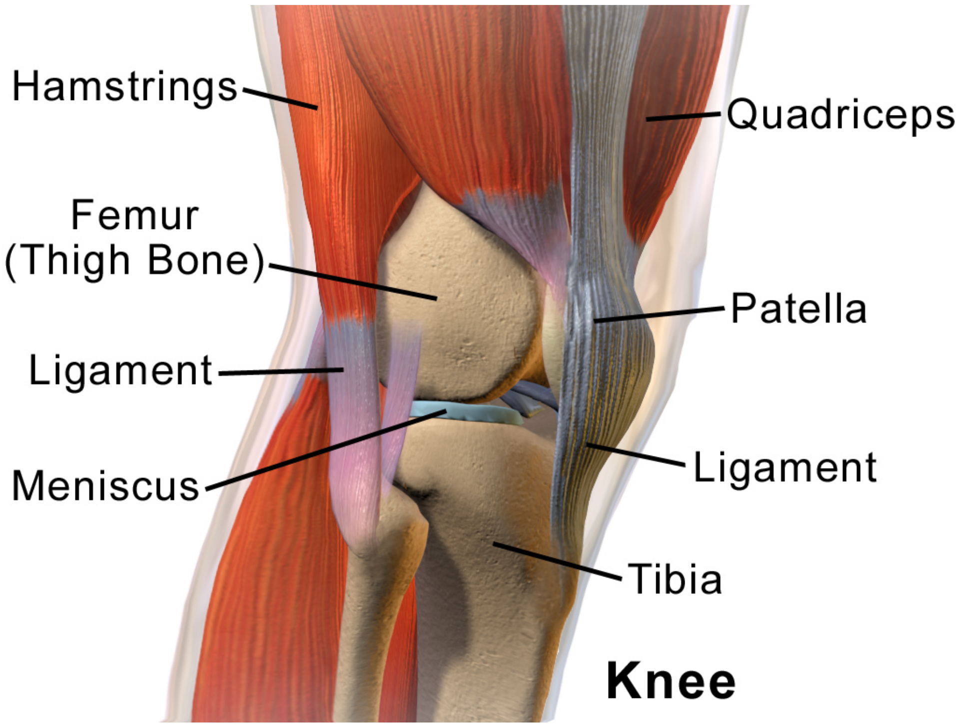


Figure 2.8: The ligaments and osseous structure in a knee [30]

overuse. The patella allows a greater extension of the knee, and the main extensor of the knee whose contraction extends the tibia is the quadriceps femoris which divides itself into four muscles and is comprised of six muscles in total [4].

For the flexion of the knee joint, the hamstrings are the main muscles. The hamstrings are located at the back of the thigh and consists of four muscles: The semimembranosus, the semitendinosus, and the long and short heads of the biceps femoris [39]. During flexion, the semimembranosus and its attachment to the meniscus pulls the meniscus backwards in order to prevent the crushing of meniscus by the femur and tibia [21].

2.5.3 Range of Motion

Studies on the normal range of motion are uncommon today because such values are well established, but a study on the knee was done by Kumar et.al. in India (2012) [41]. Both passive and active range of motion was investigated, where active range of motion is when opposing muscles contract and relax to move the limbs, and passive range of motion is when an external force moves the limbs. They state that the range of motion is greater in young subjects and decreases gradually with age. When the subjects were positioned on their back in a prone position, the average active range of motion for the knee flexion was 131.7° and passive range of motion was 141.9° [41]. These findings, both range of motion of the knee and with respect to age are in accordance with an older study called NHANES 1 which was conducted between 1971 and 1975 where they investigated the hip and knee flexion of 1892 subjects. They found that the average range of motion for the hip flexion was 120° [38].

Chapter 3

Methods

3.1 Leg Model

A 2-DoF robot model of a human leg with 2 revolute joints (Fig.3.1) was made by using the SolidWorks software. The parameters were set as shown in table 3.1. A radius of 20[mm] was set for the revolute joints. All parameters can be found in App.A.2.2. The leg model consists of two movable links of length L1 and L2 that move within the (x,z) plane. L1 was set to 610 [mm] between joint centers with a thigh length of 500[mm]. L2 was set from the center of knee joint to the center of the handle. The links are connected by revolute joints whose joint axes are all perpendicular to the plane. A radius of 20[mm] was set for the revolute joints.

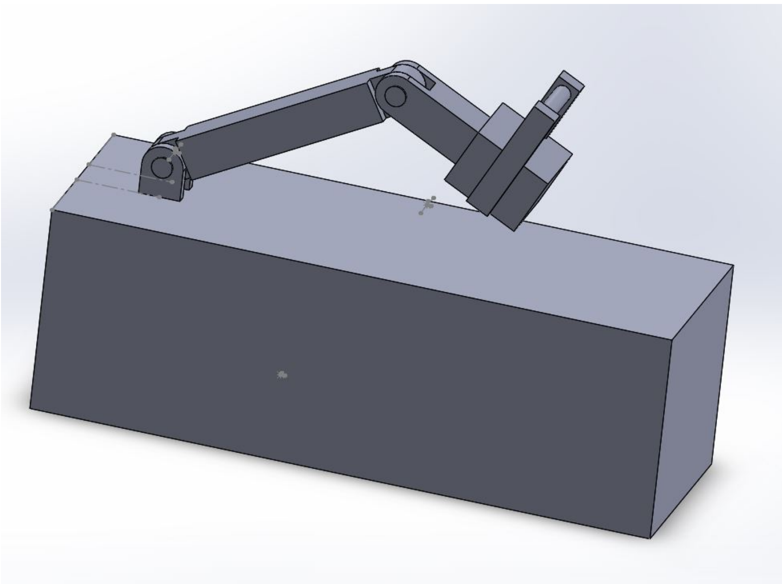


Figure 3.1: SolidWorks model of the leg model in bent knee position

The mass of the bed was set arbitrarily to 100[kg]. The mass of each link was approximated from an article by Plaegenhoef et.al. where cadavers obtained on 135 living subjects were used to estimate an average weight of the segmented limbs [36]. The moments of inertia for rotation about the mid end of each limb was set by the formula in Eq. (3.1), where the center of mass was set at the centre of each link and the calculation was simplified by considering each link as a cylinder structure. A mass of 0.5[kg] was added for the brace and handle, but their length parameters was discarded for further simplification. The results are shown in Tab. 3.1. The only relevant moment of inertia for the simulation is the I_{yy}, so all other moments of inertia were set to zero.

Part	x $[mm]$	y $[mm]$	z $[mm]$	m $[kg]$	I_{yy} $[kgm^2]$
Bed	1000	500	500	100	34.8958
Thigh	610	100	100	10	0.8396
Braced Crus w/Handle	305	100 to 150	195	3.5	0.1411

Table 3.1: Parameters for the leg model

$$I_{yy} = m\left(\frac{\frac{y^2}{2}}{4} + \frac{x^2}{3}\right) = m\left(\frac{y^2}{16} + \frac{x^2}{3}\right) \tag{3.1}$$

3.1.1 Kinematics

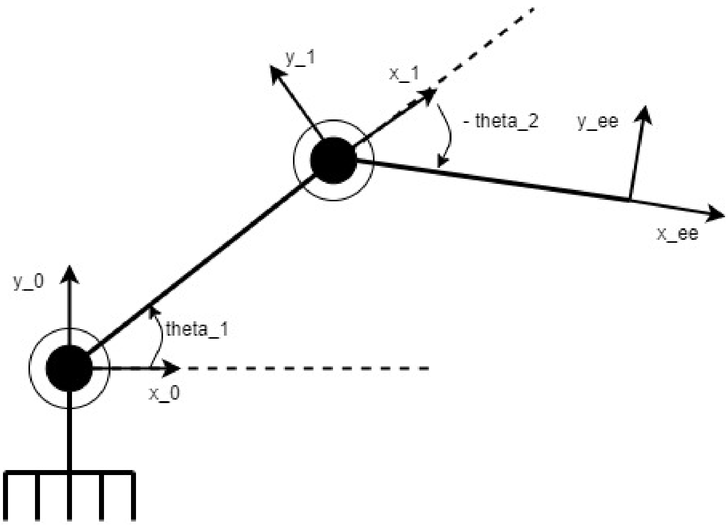


Figure 3.2: Kinematic schematic of the leg model

Forward Kinematics

The forward kinematics of the leg model was calculated by using the DH convention and setting a rigid frame at the middle of each joint and at the end effector position which was set at the center of the handle. The base frame was set at the bed hinge and a coordinate system was attached to each link as seen in Fig. 3.2. The Denavit-Hartenberg parameters were measured and put in table 3.2. According to the DH convention, the motion should take place about the z axes of the rigid body frames for revolute joints, so the z-axis in Gazebo is called the y-axis when using this convention.

Link	a_i $[m]$	α $[°]$	d_i $[m]$	θ_i $[°]$
L1	0.610	0	0	θ_1*
L2	0.362	0	0	θ_2*

Table 3.2: Denavit-Hartenberg parameters for the leg model

The homogeneous transformation matrices were set in Eq. (3.2 ,3.3) before they were multiplied in order to find the total transformation matrix in Eq. (3.4).

$$\mathbf{A}_1 = \begin{bmatrix} cos(\theta_1) & -sin(\theta_1) & 0 & L1cos(\theta_1) \\ sin(\theta_1) & cos(\theta_1) & 0 & L1sin(\theta_1) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.2}$$

$$\mathbf{A}_2 = \begin{bmatrix} cos(\theta_2) & -sin(\theta_2) & 0 & L2cos(\theta_2) \\ sin(\theta_2) & cos(\theta_2) & 0 & L2sin(\theta_2) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.3}$$

$$\mathbf{T}_2^0 = A_1 A_2 = \begin{bmatrix} cos(\theta_1+\theta_2) & (-sin(\theta_1+\theta_2)) & 0 & (L1cos(\theta_1) + L2cos(\theta_1+\theta_2)) \\ sin(\theta_1+\theta_2) & cos(\theta_1+\theta_2) & 0 & (L1sin(\theta_1) + L2sin(\theta_1+\theta_2)) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.4}$$

The x and y components of the end effector can be found in the two first entries in the last column

[28]. They were extracted from the transformation matrix and inserted into Eq. (3.5) and Eq. (3.6).

$$x_{ee}(\theta_1, \theta_2) = L1cos(\theta_1) + L2cos(\theta_1 + \theta_2) \tag{3.5}$$

$$z_{ee}(\theta_1, \theta_2) = L1sin(\theta_1) + L2sin(\theta_1 + \theta_2) \tag{3.6}$$

In order to find the relation between the joint velocities and the end-effector velocities, the partial derivatives for the x_{ee} and z_{ee} components were calculated before arranging them into a 2x2 Jacobian matrix and obtaining the wanted relations given in Eq. (3.11). The Jacobian matrix then represents the first order linear behaviour of the system.

$$\frac{\partial(x_{ee})}{\partial\theta_1} = -L1sin(\theta_1) - L2sin(\theta_1 + \theta_2) \tag{3.7}$$

$$\frac{\partial(x_{ee})}{\partial\theta_2} = -L2sin(\theta_1 + \theta_2) \tag{3.8}$$

$$\frac{\partial(z_{ee})}{\partial\theta_1} = L1cos(\theta_1) + L2cos(\theta_1 + \theta_2) \tag{3.9}$$

$$\frac{\partial(z_{ee})}{\partial\theta_2} = L2cos(\theta_1 + \theta_2) \tag{3.10}$$

$$\begin{bmatrix} \dot{x_{ee}} \\ \dot{z_{ee}} \end{bmatrix} = \mathbf{J} \begin{bmatrix} \dot{\theta_1} \\ \dot{\theta_2} \end{bmatrix} = \begin{bmatrix} \frac{\partial(x_{ee})}{\partial\theta_1} & \frac{\partial(x_{ee})}{\partial\theta_2} \\ \frac{\partial(z_{ee})}{\partial\theta_1} & \frac{\partial(z_{ee})}{\partial\theta_2} \end{bmatrix} \begin{bmatrix} \dot{\theta_1} \\ \dot{\theta_2} \end{bmatrix} \tag{3.11}$$

Inverse Kinematics

First, the radius from the base frame to the end effector position can be calculated by Pythagoras' theorem (Eq. (3.12)). The cosine rule can be used to express a relationship between the links and the base radius to the end effector (Eq. (3.13)), where ψ is the angle between the links at the knee joint.

A positive θ_2 angle can then be expressed as $\pi - \psi$. By using the property of the shape of the cosine function, the relationship in Eq. (3.14) can be set. Inserting Eq. (3.14) into Eq. (3.13) yields the equation in Eq. (3.15) and subsequently the θ_2 angle in Eq. (3.16). Due to the fact that the knee joint is constrained to only negative angles, the unique solution is found by inserting a negative sign before the equation in Eq. (3.17).

A new triangle can be set with L2 as the hypotenuse, $L2cos(\theta_2)$ as the adjacent side and $L2sin(\theta_2)$ as the opposite side. From the base frame, another new triangle can then be established with an angle β between the radius and the length $(L1 + L2cos(\theta_2))$ in Eq. (3.18).

Going back to the first triangle with the relationship in Eq. (3.12), a final angle, γ can be set from the base frame (Eq. 3.19). By using the fact that θ_1 can be expressed as $\gamma + \beta$, θ_1 can be obtained by using the equation in Eq. (3.20). Due to the fact that θ_2 always will be negative, the sign before the second term becomes negative and the unique solution in Eq. (3.21) can be set.

$$r^2 = x_{ee}^2 + z_{ee}^2 \tag{3.12}$$

$$r^2 = L1^2 + L2^2 - 2L1L2cos(\psi) \iff cos(\psi) = \frac{L1^2 + L2^2 - r^2}{2L1L2} = \frac{L1^2 + L2^2 - x_{ee}^2 - z_{ee}^2}{2L1L2} \tag{3.13}$$

$$cos(\theta_2) = -cos(\psi) \tag{3.14}$$

$$cos(\theta_2) = -\frac{L1^2 + L2^2 - x_{ee}^2 - z_{ee}^2}{2L1L2} \tag{3.15}$$

$$\theta_2(x_{ee}, z_{ee})_{prelim} = arccos\left(\frac{x_{ee}^2 + z_{ee}^2 - L1^2 - L2^2}{2L1L2}\right) \tag{3.16}$$

$$\theta_2(x_{ee}, z_{ee}) = -arccos\left(\frac{x_{ee}^2 + z_{ee}^2 - L1^2 - L2^2}{2L1L2}\right) \tag{3.17}$$

$$\beta = arctan\left(\frac{L2sin(\theta_2)}{L1 + L2cos(\theta_2)}\right) \tag{3.18}$$

$$\gamma = arctan\left(\frac{z_{ee}}{x_{ee}}\right) \tag{3.19}$$

$$\theta_1(x_{ee}, z_{ee})_{prelim} = \gamma + \beta = arctan\left(\frac{z_{ee}}{x_{ee}}\right) + arctan\left(\frac{L2sin(\theta_2)}{L1 + L2cos(\theta_2)}\right) \tag{3.20}$$

$$\theta_1(x_{ee}, z_{ee}) = arctan\left(\frac{z_{ee}}{x_{ee}}\right) - arctan\left(\frac{L2sin(\theta_2)}{L1 + L2cos(\theta_2)}\right) \tag{3.21}$$

The initial position of the end effector is given in Eq. (3.22, 3.23). The formulas for the inverse kinematics were verified by using the formulas for the forward kinematics. Inserting the arbitrary angles: $\theta_1 = 15°$, $\theta_2 = -10°$ in Eq. (3.5) and Eq. (3.6) yields the cartesian coordinates in Eq. (3.24, 3.25). Inserting these into Eq. (3.17) and subsequently into Eq. (3.21) along with the θ_2 angle from Eq. (3.26), gives the results in Eq. (3.26, 3.27).

$$x_{ee}(0°, 0°) = 0.610cos(0°) + 0.362cos(0°) = 0.972[m] \tag{3.22}$$

$$z_{ee}(0°, 0°) = 0.610sin(0°) + 0.362sin(0°) = 0[m] \tag{3.23}$$

$$x_{ee}(15°, -10°) = 0.610cos(15°) + 0.362cos(15° - 10°) = 0.9498[m] \tag{3.24}$$

$$z_{ee}(15°, -10°) = 0.610sin(15°) + 0.362sin(15° - 10°) = 0.1894[m] \tag{3.25}$$

$$\theta_2(0.9498, 0.1894) = -arccos\left(\frac{0.94984^2 + 0.18943^2 - 0.610^2 - 0.362^2}{2 \cdot 0.610 \cdot 0.362}\right) = -10[°] \tag{3.26}$$

$$\theta_1(0.9498, 0.1894) = arctan\left(\frac{0.1894}{0.9498}\right) - arctan\left(\frac{0.362sin(-10)}{0.610 + 0.362cos(-10)}\right) = 15[°] \tag{3.27}$$

3.1.2 Creating the Leg Model File

The SolidWorks model was exported as a URDF file using an online converter. It had was modified to be compatible with ROS2 and first inspected in the Graphiz tool (Fig. 3.3) to visualize the model tree.

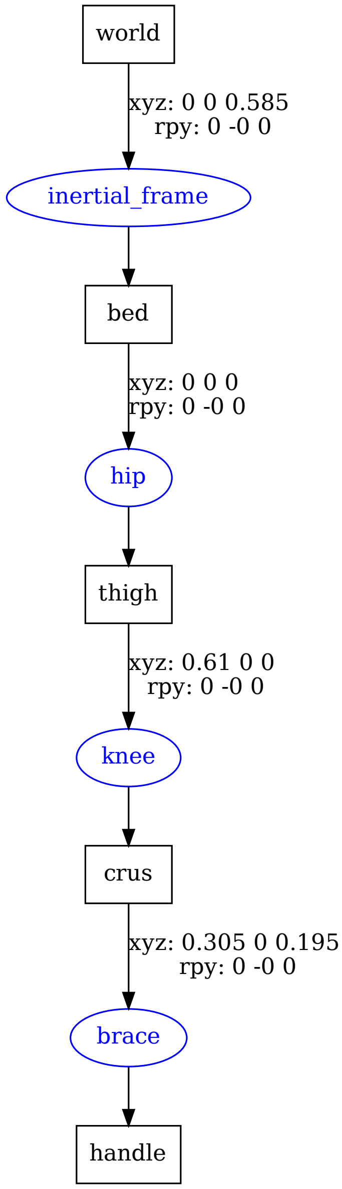


Figure 3.3: The leg model structure visualized by Graphiz

RViz was then used for further inspection and modifications. A world frame was added to the exported code and the CMakeLists.txt and package.xml had to be rewritten to include the required dependencies. A Python launch file was added to start RViz with the model and a view.rviz file was added for RViz configurations. The end effector was positioned at the center of the handle, and the final configuration is displayed in Fig. 3.4. The manipulation of the joints was successful; moving the hip and knee joints, and constraining their movement in the range (0° to 130°) for the hip and (-120° to 0°) for the knee.

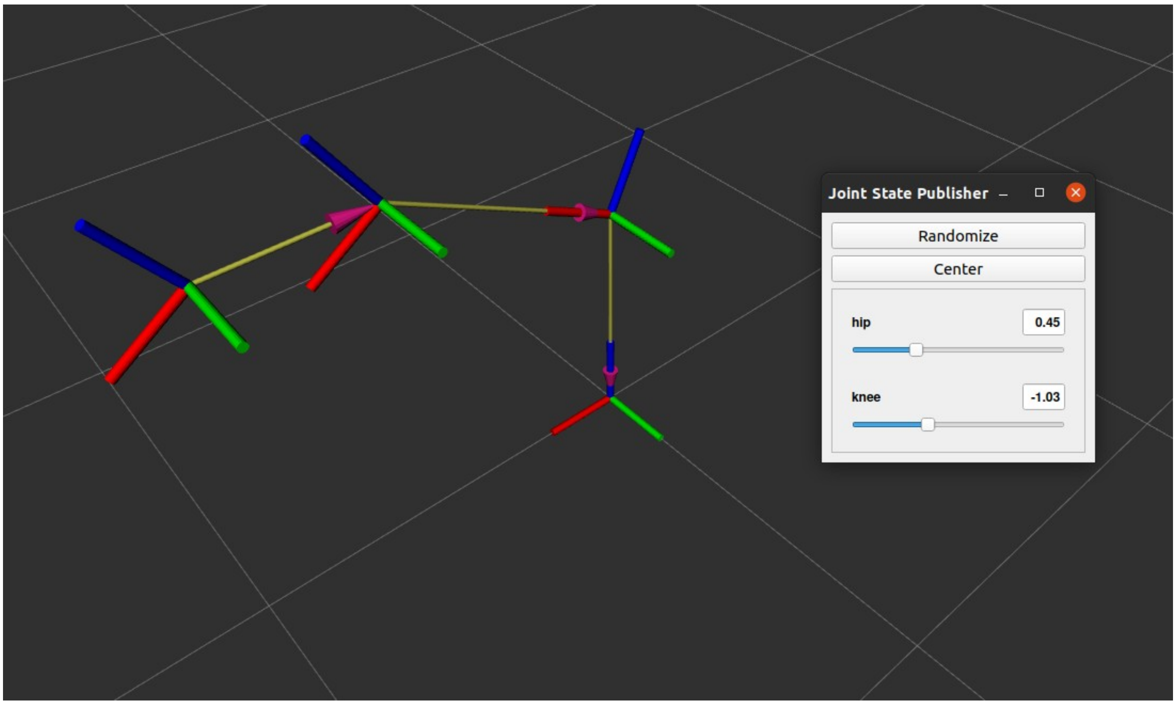


Figure 3.4: RViz representation of the leg model

In order to make the model appear in a Gazebo World, a lot of changes has to be made. It requires a world file with light (sun), ground plane and the model. A short config file must be added to the package along with the STL meshes that was exported from SolidWorks, which can be converted into DAE or OBJ files for better textures. This can be done using an online converter. Also, the CMakeLists.txt and package.xml files must be rewritten to define the required dependencies, and the URDF model file must be modified by adding gazebo tags and their parameters.

3.2 Simulating and Implementing Parts of the Procedure

Halodi Robotics has made messages that was examined in order to find the ones that were required to realize the desired tasks. They have listed all topics with the corresponding message types in the reference [17].

It is not possible to subscribe directly to subordinate message types because it would result in non-atomic measurements, so every desired subordinate message type must be run through the message type that is linked to the topic.

The robot can be controlled with the Trajectory API or the Realtime API, and Halodi generally recommends using the trajectory API, as this does not put realtime constraints on the user application and is easier to use. The HandCommand and the WholeBodyControllerCommand message type are part of the realtime API, while the rest belong to the trajectory API. As stated on the Halodi repository, the trajectory API can interpolate trajectories trough points in task space and joint space. The realtime API allows the fastest control updates for the user, but the user is responsible for updating the setpoints at 250[Hz]. The realtime API is used by the trajectory manager, and therefore cannot be used if the trajectory manager is in use.

3.2.1 Joint and Grasp Manipulation

The WholeBodyTrajectory message type with its subordinate message types are of interest for the joint manipulation, and the HandCommand for closing and opening the hands of EVE. The grasping motion of the EVE robot is not a part of the WholeBodyTrajectory message type, so it must be made separately. It uses the HandCommand message type through the /eve/left_hand_closure and /eve/left_hand_closure topic. Unfortunately, it does not have a status callback message.

The WholeBodyTrajectory message type contains a lot of subordinate task-space objects (e.g.

TaskSpaceCommand and the JointSpaceCommand) that is useful for manipulating the EVE robot through pre-made topics. Each WholeBodyTrajectoryPoint can be composed of desired task space commands and/or desired joint space commands along with a desired time to get there.

Joints were accessed through the WholeBodyTrajectory message type which communicates through the /eve/whole_body_trajectory topic. A callback message for confirmation of the transfer status can be obtained with the GoalStatus message type.

The hip, knee and ankle joints are not possible to manipulate individually due to the balancing of the robot which involves all of these joints simultaneously. Movements that include rotating, delevating or bending the pelvis of EVE has to be done by use of the WholeBodyControllerCommand message type which communicates through the /eve/whole_body_command topic.

The C++ menus were made by using mostly switch statements and do-while loops in order to provide a graphical user interface for testing joint manipulation.

3.2.2 Recording the External Input

The WholeBodyState message type which contains the messages for measurements includes the subordinate JointMeasurement message type which was used to read position and velocity of the joints via the /eve/whole_body_state topic. The stiffness of the joints were relaxed for external input by using the JointSpaceCommand; disabling the use_default_gains and setting the stiffness to zero and damping to 1.0 which is the maximum value. The current version of the digital twin is not optimal in terms of gravity compensation: Gravity is compensated based on a modelled mass. If this mass is slightly off, there will be an overcompensation which gives a floating behaviour, so this must be expected when turning off the use_default_gains option.

The removing of the stiffness was first done by using the JointSpaceCommand: Disabling the use_default_gains, setting the stiffness to 0.0 and damping to 1.0. There was no disturbance in the robot pose when the stiffness run, and the arm was completely vertical and still. Then when the 'use_default_gains' was turned off, the shoulder joint moved about 15 degrees. This behaviour also occurred when the arm was taken high up before taking it down. The unwanted movement disappeared when the arm first was pulled back and then put down. It was consulted with Halodi Robotics, and they said that the motorDampingScale also should be set to 1.0 in order for this not to happen. The damping option is applied at the joint level and quickly becomes unstable, while the motorDamping is applied at the motor level which runs faster than the robot controller. Setting the motorDampingScale to 1.0 diminished the unwanted floating movement in the digital twin, but did not remove the problem. This was then tried out on the real EVEr3 robot. When the stiffness was set to 0.0, damping to 1.0 and the motorDampingScale was set to 1.0, this unwanted movement in the real EVEr3 robot was seemingly low, but the arm was a bit stiffer than without the motorDampingScale involved.

3.2.3 Constraint of the Right Hand of the Digital Twin

It is not currently possible to simulate the hand due to the very complex dynamics and kinematics. Also, some collision tags are missing in the Halodi repository, so they will have to be added to the body of interest if it is not already made, in order for the digital twin to follow along an external movement.

One of the supervisors came up with the idea of replacing the right hand of the digital twin with a hook. There is no way to add friction to a part in SolidWorks, so I made both a circular and a square hook. Both hooks were designed with a width of 50[mm] in order to get a stable connection to the handle of the modelled cuff. The square hook was made to be one of two parts of a puzzle that fit together when the leg is moving. The hooks (Fig. 3.6, 3.6) were made in SolidWorks, exported as STL files which in turn was converted to OBJ files via an online converter, because the OBJ format is what Halodi utilize for their models. Another braced crus was then also made with a square handle that has a diagonal length of 40 [mm].

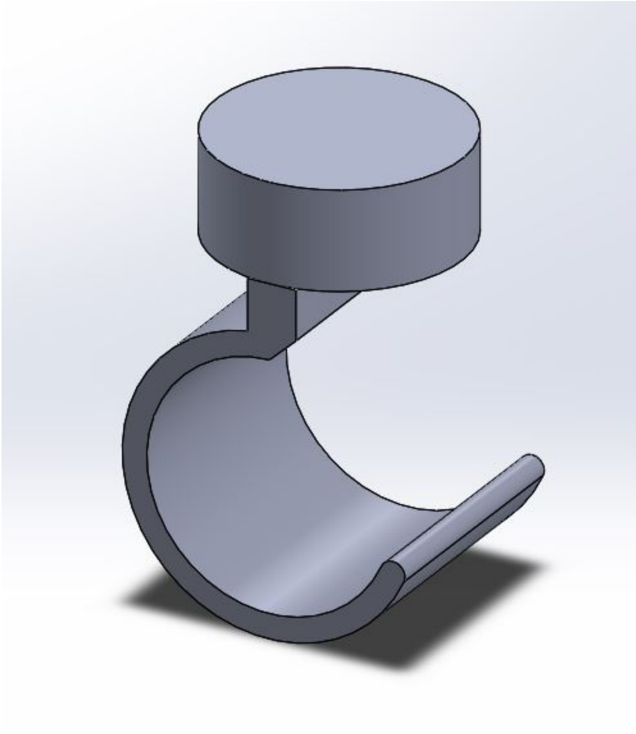


Figure 3.5: Modelled circular hook for simulated human-robot interaction

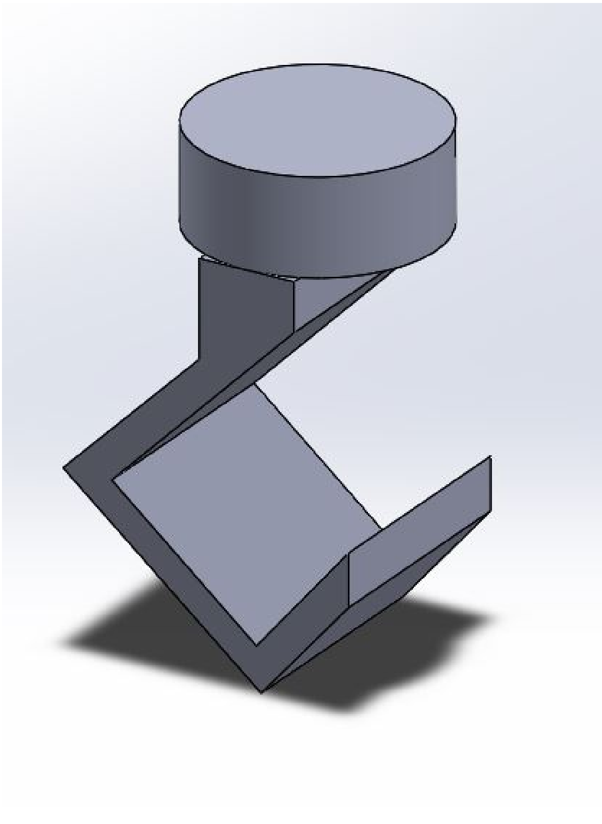


Figure 3.6: Modelled square hook for simulated human-robot interaction

3.3 The Cuff

The design in Fig. 3.7 was first discussed with healthcare professionals via e-mail. They gave me the following general criteria for a cuff that could be used for knee rehabilitation:

- Easily adjustable in circumference
- Easy to take on and off for the patient
- Padded – i.e. no hard parts toward the skin
- A layer toward skin to provide friction (needs to be checked regarding allergens) so that the leg doesn't rotate or slide inside the cuff.

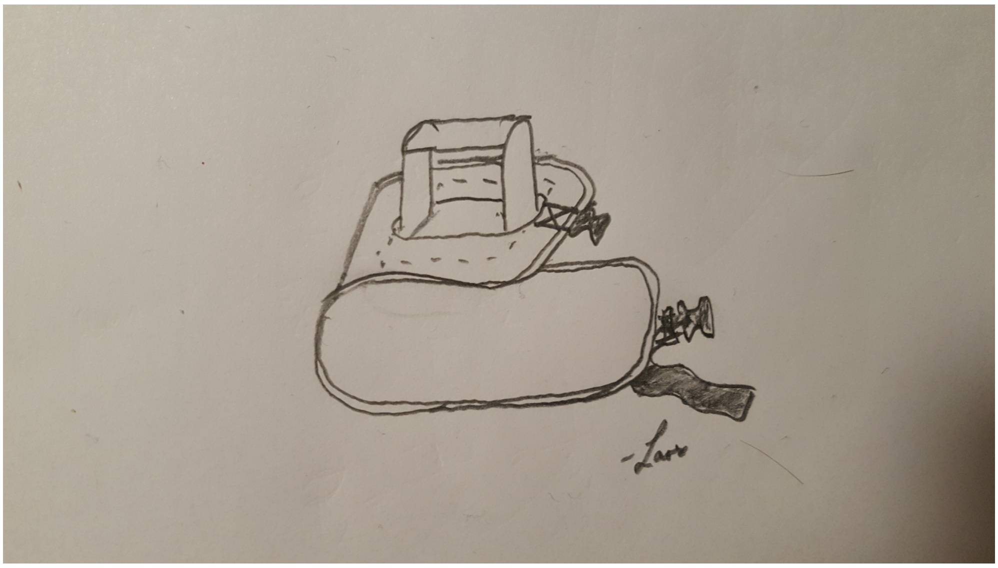


Figure 3.7: First design of the cuff for the human-robot interaction

The cuff was designed as a harness-like contraption made with a 3D-printed handle and base, cotton fabric, foam rubber padding and a lot of cotton stuffed into the sewed cotton fabric. The padded area is thought to be covered with a thin layer of hypoallergenic foam to add friction, and the length can be adjusted by wrapping the excess padded area around the leg and tightening the excess bands with connected locks.

According to a study performed by the Cukurova University School of Medicine, the circumference of the widest part of the calves among students at the time ranged between 280[mm] and 420[mm] which gives the diameter range: (89 - 134)[mm] [23].

The cuff was therefore designed with a minimum diameter of approximately 85[mm] and a maximum diameter of 140[mm]. The width of the cuff was set to 160[mm]. Both the top and base parts of the handle (Fig. 3.8, 3.9) was printed with PLA filaments (white and blue) on a Ultimaker Cura 2+ 3D printer.

The holes for the handle was cut as thin lines which were widened and sewed. All borders were sewed in order to avoid the fringes from separating. The bands were cut to fit the dimensions and sewed on the cotton fabric. The shape of the cushioning part below the base of the handle was cut out of a foam rubber sponge and glued to the base by a cyanoacrylate glue. The top of the base was then glued to the cotton fabric in order to make sure that the base would be fixed to the cuff, and the glue marks were covered with patches. The edges were sewed together but leaving holes for the cotton to be stuffed in for the padding, before sewing it shut. The top of the handle was glued on to the base with a cyanoacrylate glue. The assembled prototype can be seen in Fig. 3.10, 3.11 and 3.12.

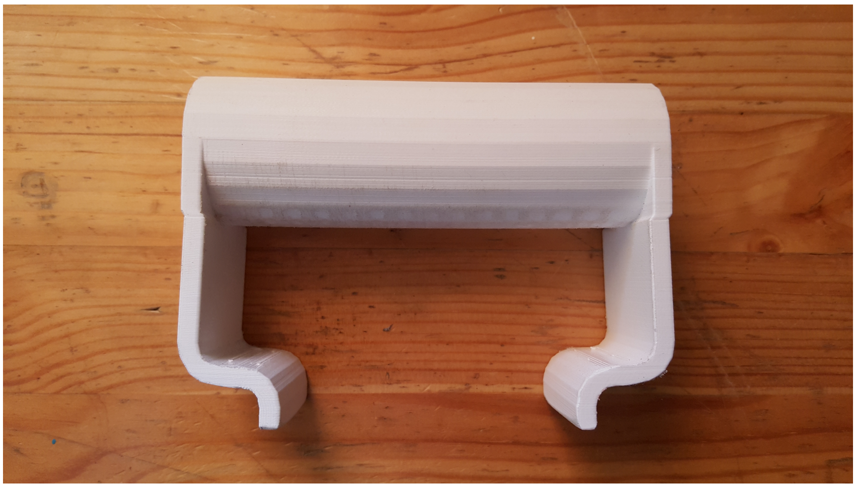


Figure 3.8: 3D printed top of the handle

Figure 3.9: 3D printed base of the handle

Figure 3.10: Assembled cuff with 3D printed handle

Figure 3.11: Assembled cuff with 3D printed handle (top projection)

Figure 3.12: Assembled cuff with 3D printed handle (bottom projection)

3.3.1 An Alternative Cuff

In order to present an alternative to the healthcare professionals, a second cuff had to be designed. It was inspired by a brace that was used for upper limb rehabiliation in a master's thesis by a student named Rodrigo Goncalves Cerejo Antunes. It consists of a hard casing made from a two-part hollow cylinder structure where inside of each part is smooth and there are two carved 25x1[mm] tracks for a band of velcro to be inserted. The velcro is added for surrounding the parts and locking the casing in place. The handle is fixed to the top part and has a diameter of 40[mm] with 5[mm] side supports. The inside diameter is adjusted by inserting more or less foam rubber padding. Including a minimum padding of 10[mm] on each side, the diameter should be 160[mm] in order to make it wide enough to be applicable. The parts were designed in SolidWorks as shown in Fig 3.13 and 3.14.

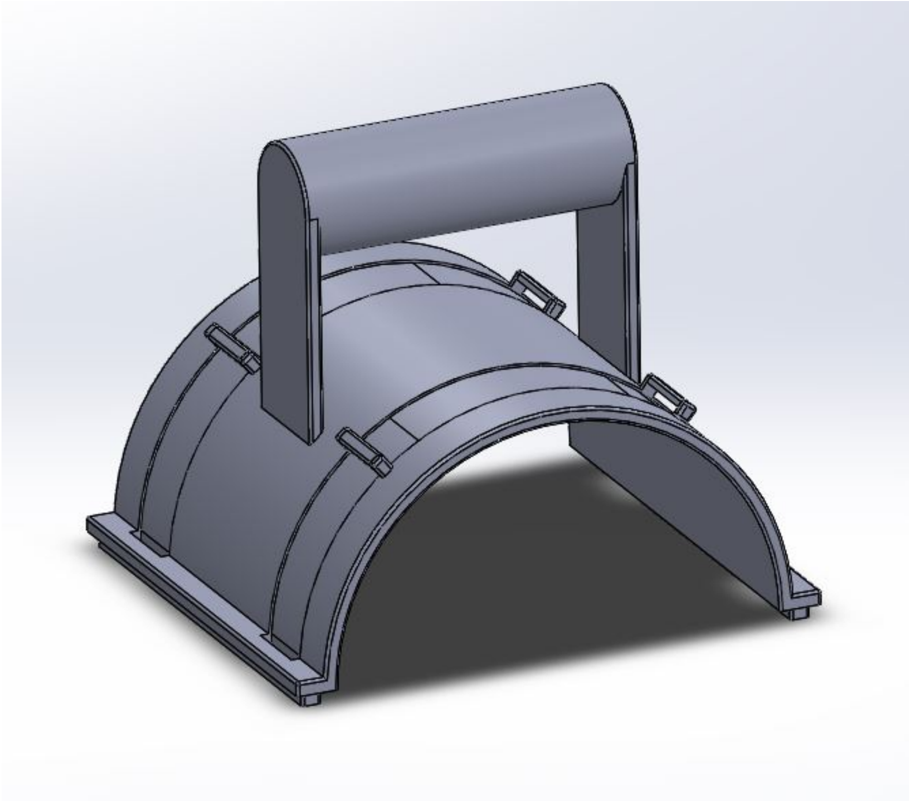


Figure 3.13: The alternative cuff (upper part)

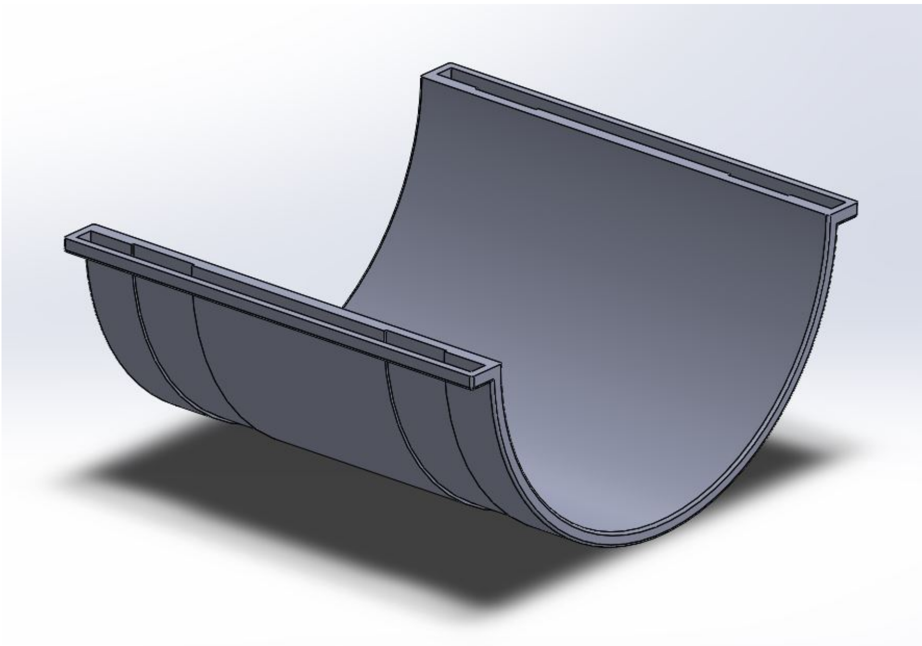


Figure 3.14: The alternative cuff (lower part)

Chapter 4

Results and Discussion

4.1 Objective 1

1) Create a code that can manipulate the joints and hands of EVEr3
   a) Simulate the transfer of data to digital twin
   b) Test on real EVEr3 robot

A menu was succesfully made that could provide an interface for input to the joints of both the EVE Gazebo Model and the real EVE Robot. When the task is completed, the connection shuts down before going back to the menu. It was simulated on the digital twin before using it on the real EVEr3 robot.

Another menu was successfully made to test the grasping and figure out the best closing ratio for a 40[mm] handle. It uses the HandCommand message type which sends the closure rato, velocity and force to the robot through the /eve/left_hand_closure and /eve/right_hand_closure topics for the left and right hand. After a few trials, the best closing ratio for the handle was found to be 0.75-0.8. Also the closure force and velocity can be set, but altering them did not make a significant change in the execution of the grasp. As stated, the Halodi API is unable to simulate a grasping movement with the digital twin, so this was implemented directly on the EVEr3 robot.

4.2 Objective 2

2) Create a leg model for a Gazebo simulation

A leg model was succesfully made as an URDF model and inspected with the RViz tool. It then worked as intended, but the launch file was a remaining problem when trying to insert the model into Gazebo. Even after many weeks with attempts, I could not manage to get the model to appear. It appeared in the menu over models when I inserted a folder containing it into the Gazebo model path, but I could not make it appear in the world through several attempts for the CMakeLists.txt and package.xml files and launch scripts written both in Python and XML. One thing I did not try or think of until the last days of the project, is that I could have tried using a ROS bridge which enables the exchange of messages between ROS and ROS2 and checked if using ROS2 was the problem when using the launch scripts described in Gazebo tutorials.

4.3 Objective 3 and 4

3) Design a cuff for the human-robot interaction
4) Collaborate with healthcare professionals
   a) Input on the created cuff
   b) Input for a rehabilitative movement

The healthcare professionals rated my harness-like design (section 3.3) higher than the alternative cuff with a hard casing that was suggested (subsection 3.3.1) via e-mail, and it was evaluated as a good design. They had two improvements: Due to its loose structure, the cotton stuffing should be replaced or appended with a a more rigid support. The outer cotton fabric is also not hygienic, meaning that it cannot be cleaned using hygiene protocols between patients. Synthetic materials are often used for this purpose. My prototype would then have to be covered with a synthetic cover which should have another kind of hypoallergenic, hygienic fabric glued to the whole inner part in order to increase friction between the patient and the cuff.

A second cuff could also be required for the ankle or foot of the patient. As explained to me, a rehabilitative movement will often include two hands for holding and pushing the leg, where one hand holds the ankle or heel for the pushing movement and the other holds on to the leg to guide it like the example in Fig. 4.1.

The rehabilitative movement was requested at the physical meeting and will be designed for the continuing studies.

Figure 4.1: Hand placement during a rehabilitative movement (©CanStockPhoto)

4.4 Objective 5

5) Create an interaction between the leg model and the digital twin
   a) Constrain one hand of the digital twin onto a limb of the leg model
   b) Move the limb of the human model
   c) Make the digital twin follow along
   d) Detect and store the movement of digital twin into an array
   e) Make the digital twin perform the movement a set number of times

This objective was not finished in time for the deadline of the thesis. I could not figure out how to import and utilize the leg model in Gazebo as explained in section 4.2. I designed two hooks which seems like usable options for attaching the digital twin to a leg model as long as the grasping simulation is not available. The square handle can be used horizontally or vertically. This is not an optimal solution for the system as a vertical positioning could contribute to awkward joint angles if the robot is not placed perfectly. The connection between a normal circular hook and a circular handle could be too unstable in terms of lack of friction, but both alternatives would have to be tested out in Gazebo. It was however difficult to sort out exactly what is required to replace the hand due to the number of folders that is containing the quite complex modelling of the digital twin.

4.5 Objective 6

6) Implement on EVEr3
   a) Perform the procedure
   b) Adjust the control parameters

This objective was not completed. A menu for removing the stiffness of the joints for external input was made. It also has the options to add the stiffness again and to set all joints to zero position. The stiffness of the joints was relaxed for external input by using the JointSpaceCommand and disabling the use_default_gains before setting the stiffness to 0.0 and both motorDampingScale and damping to 1.0. The menus for joint manipulation for the first objective, the grasping and the removing of the stiffness were gathered into one functional main menu (Fig. 4.2).

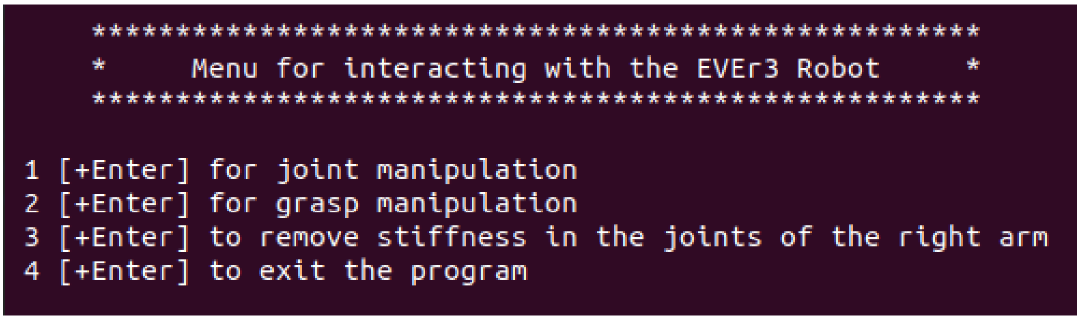


Figure 4.2: The main menu for manipulating the EVEr3 robot

4.6 Remarks and Suggestions for Future Works

It is necessary to get a proper understanding of the whole repository of Halodi Robotics prior to the process of designing a complete rehabilitative system involving the EVEr3 robot. It is also crucial to have an early dialogue with Halodi about all possibilities and limitations regarding what one would like to achieve and not just ask general questions.

There is an error in the current version of the Halodi API which demands several trials if the commands are not executed. This was detected from the GoalStatus callback which sometimes halts at the first status: "Sending trajectory, listening for whole_body_trajectory_status...". Whenever this happens, one has to press [Ctrl]+c and retry until it is successful.

4.6.1 Objective 2

It seems better to create a leg model after a proper rehabilitative exercise has been issued, so that the handles can be placed at the relevant positions. The healthcare professionals pointed out that it could actually be sufficient with one cuff controlled by one arm, but this would have to be tried out and tested to see if the lower extremity can be satisfactorily controlled with only one point of contact in the human-robot interaction.

When trying to include the URDF model in the world of the digital twin, I tried following the package provided by Halodi. In the CMakeLists.txt in the ever3 description package provided by Halodi, they include a URDF to SDF converter. Unfortunately I did not manage to implement this on my model when I eventually discovered this aspect of the model for the digital twin. I recommend to look at the eve_r3 model package files and use these as a side tool in addition to tutorials for creating a functional model.

4.6.2 Objective 3 and 4

The made prototype can be useful for testing a proof of concept on a doll in future works and was left in the cage of the Mechatronics Lab at UiA Grimstad where EVEr3 was stationed. The bands could be cut off and sealed with a lighter flame if they are too long and just are a nuisance.

I was given a general comment from the healthcare professionals regarding the rehabilitative procedure: All rehabilitative exercises involves three steps. First there is a passive movement before an assistive movement which precedes a resistive movement.

4.6.3 Objective 5 and 6

Regarding the digital twin, the mentioned floating behaviour in subsection 3.2.2 could be diminished when the model is constrained, as the arm would be locked to the leg which is in turn joined to the bed. But it could still affect the recorded values, so it will have to be tested if a model is to be used as the external input to the digital twin. If it is impossible to get rid of in terms of jerking behaviour or impacts on the issuing/reading of joint values, then another approach must be taken. In worst case, the procedure cannot be simulated in a satisfactory way and must be developed on the real EVEr3 in combination with the digital twin, as the case is with the grasping for the current version of the digital twin.

The manipulation of the body frame was not made in time for the hand-in of the thesis. For future works, the units for the TaskSpaceCommand are as follows: pose (x,y,z) are issued in meters and pose rotation are given as quaternions (unitless). Angular velocity, linear velocity, angular acceleration and linear acceleration are issued in SI units. It could then be better to use the Realtime API rather than the Trajectory API for the recording and replaying the movement. I was advised that one could record pelvis height and orientation and hand poses, and then use task space control by sending the recorded values to the robot. Also to express the hands in either pelvis or base frame (enable z-up to align the base frame with gravity).

A menu for reading issued values for joint angles and angular velocities for the pitch of the right arm was made and is appended in the folders as the joint_read_menu. It uses the WholeBodyState message type to access the subordinate JointMeasurement message type which contains the messages for measurements. The transferred joint movement is transferred to a public function of the node class and stored in a global vector. The ROS2 QoS was utilized to set the ROS2 reliability setting to best effort in order to receive the messages. The only relevant data is the stop- and turning points, so the vector values are sorted out to not store duplicate values and the value is truncated to three decimals. This was also to avoid a std::bad_alloc error which occures whenever there is an issue with the memory. The values can be stored in an external file which in turn can be loaded into the script whenever the rehabilitative movement is to be replayed, but I could not figure out how to issue them as an array of values in time for the thesis. I think the best way is to issue them as trajectory points, but it was not figured out how to issue them from a vector.

Another problem arose when I was going to implement the recording for an external input when the stiffness is removed. Because the joint_read_menu records when joint values are issued, it turned out that it will not work for recording external inputs. The external movement can however be detected by using the ros2 echo topic /eve/whole_body_state in the terminal. The only way that I have found to record topics is to use ros2 bag, which is a command line tool that can be used for recording values from topics, and store them in a file to be used for later use. Because the /eve/whole_body_state topic contains a lot more information than the joint positions and velocities, it would require a script that can remove the huge amount of excess data. If this cannot be solved by a C++ script, it could be an idea to ask Halodi Robotics to create a specialized topic for the desired values.

4.6.4 Safety Assessment

During the envisioned human-robot interaction, the biggest danger is at the contact point between the patient and the EVEr3 robot. The patient is the most endangered person in interaction with the robot, and should have the wireless emergency button (Fig. 4.3) at his or her disposal in case of errors during the procedure or if the patient wants to end the procedure before it has finished. This was pointed out by one of the healthcare professionals. The worst case scenario that I can think of, is that the control system is temporary malfunctional in addition to the two AA batteries for the emergency button running out. Those two errors must then be avoided. For the stop button, rechargeable batteries instead of regular batteries could be used to make sure that they do not run out at the wrong time. The EVEr3 robot and the batteries could then be recharged simultaneously prior to each procedure.

Figure 4.3: The wireless emergency stop button provided by Halodi Robotics

It should be mentioned that in an article by Senanayake (2009) it is stated that manual switches may not be practical due to the slow motions and reflexes of the physically impaired, and that an automatic shutdown as a result of exceeding pre-defined limits of angles or forces applied on human joints could be a safer approach [40].

4.6.5 Motion Control

A control system should be applied to gently adjust the rehabilitative movement based on measurements of the force or changes in the joint positions due to the motion of the patient. In the current version of EVEr3, the joint torques cannot be read or issued, but the angular position and velocity of the joints are possible to measure, and the angular acceleration, velocity and position can be issued.

Position and torque sensors could be integrated into the cuff instead of just using the joint sensors. An example of this is given in a paper by Bolus et.al. (2008) where they created an adjustable-stiffness knee brace with an embedded magnetic angle sensor [5]. It can however only be used for inspiration, as the angle sensor is only capable of measuring the angular deflection of the device created by bending, leaving a possibility for error if the anatomical angle does not correspond exactly to the angle of the device.

When calculating the kinematics for the EVEr3 robot, the lengths of the links can be found in the provided URDF for the digital twin by reading the <origin> tags in the joint elements, e.g. r_elbow_y for the _upper_arm, j_r_wrist_y for the r_lower_arm, and the <origin> tag for the distance to the center of mass of the hand found in the <inertial> tag for the <link> element of r_palm. A positive angle is defined in the clockwise direction and a negative angle in the anti-clockwise direction when the arm is projected from the right side. The given lengths for the digital twin are identical to the EVEr3 robot.

Chapter 5

Conclusions

A complete rehabilitative system was not realized in time for the hand-in of the thesis. The intended interaction with the leg model in Gazebo was a big obstacle. Many weeks went to going back and fourth on the scripts that are required, and in retrospect I should not have placed so much focus on the simulation of the leg movement and instead made an earlier effort to investigate the possibilities for creating a functional ROS2 system on the EVEr3 robot as a proof of concept.

I have learned about many aspects of creating a rehabilitative system with the use of a robotic infrastructure, notably impedance control, C++ and ROS2 programming, Gazebo, the David-Hartenberg convention, knee anatomy and rehabilitation robots in general.

It seems like the EVEr3 robot could be used as the basis for a robotic infrastructure which can aid physiotherapists by issuing and adjusting assistive or resistive exercises, but it requires more knowledge about the possibilities within ROS2 programming, and this field is being updated all the time as is the case with the EVEr3 software. It also requires more studies on the EVEr3 robot with regards to gentle movement control with a human in the loop and other means for the highest patient safety possible.

Appendix A

Appendices

A.1 MATLAB Scripts

A.1.1 Transformation Matrix

```
%GENERATION OF TRANSFORMATION MATRIX
%The function takes in the link length and joint angle, and produces the
%homogeneous transformation matrix
function [A] = transformation_matrix(a, theta)
A = [cos(theta) -sin(theta) 0 a*cos(theta);...
     sin(theta) cos(theta) 0 a*sin(theta);...
     0 0 1 0;...
     0 0 0 1];
end
```

appendices/MATLAB/transformation_matrix.tex

A.1.2 Matrix Multiplication

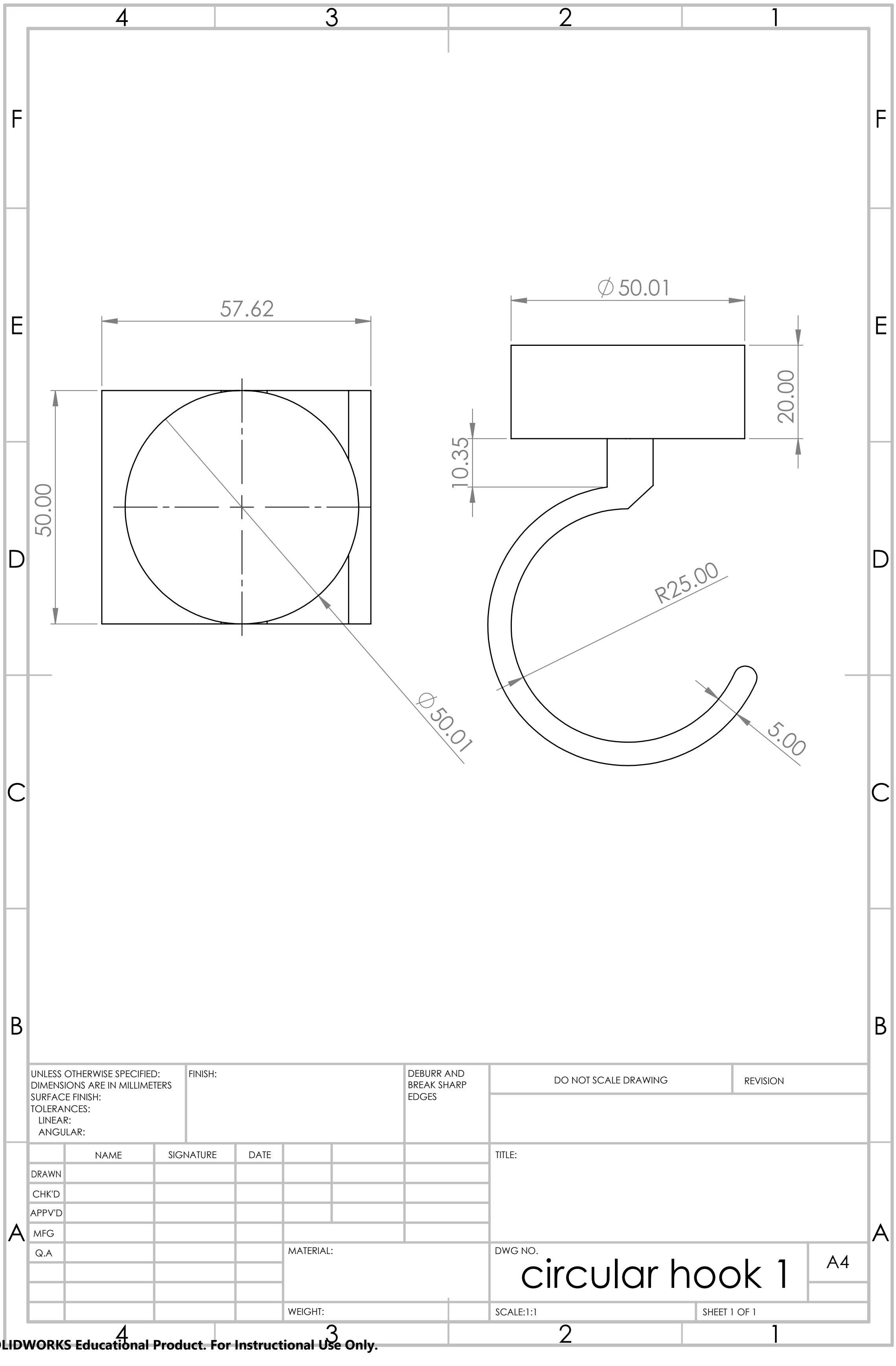
```
%MATRIX MULTIPLICATION FOR TRANSFORMATION MATRICES
close all; %closes all figures whose handles are visible
clear; %removes all variables from the current workspace
clc; %clears all the text from the Command Window
%Declaring symbolic variables for the lengths and angles
syms L1
syms L2
syms theta1
syms theta2
%The matrices are calculated by transformation_matrix(a, theta)
A_1 = transformation_matrix(L1,theta1);
A_2 = transformation_matrix(L2,theta2);
%The total transformation matrix is calculated as T_prelim
T_prelim = A_1*A_2;
%MATLAB simplifies the solution with trigonometric addition
T = simplify(T_prelim)
```

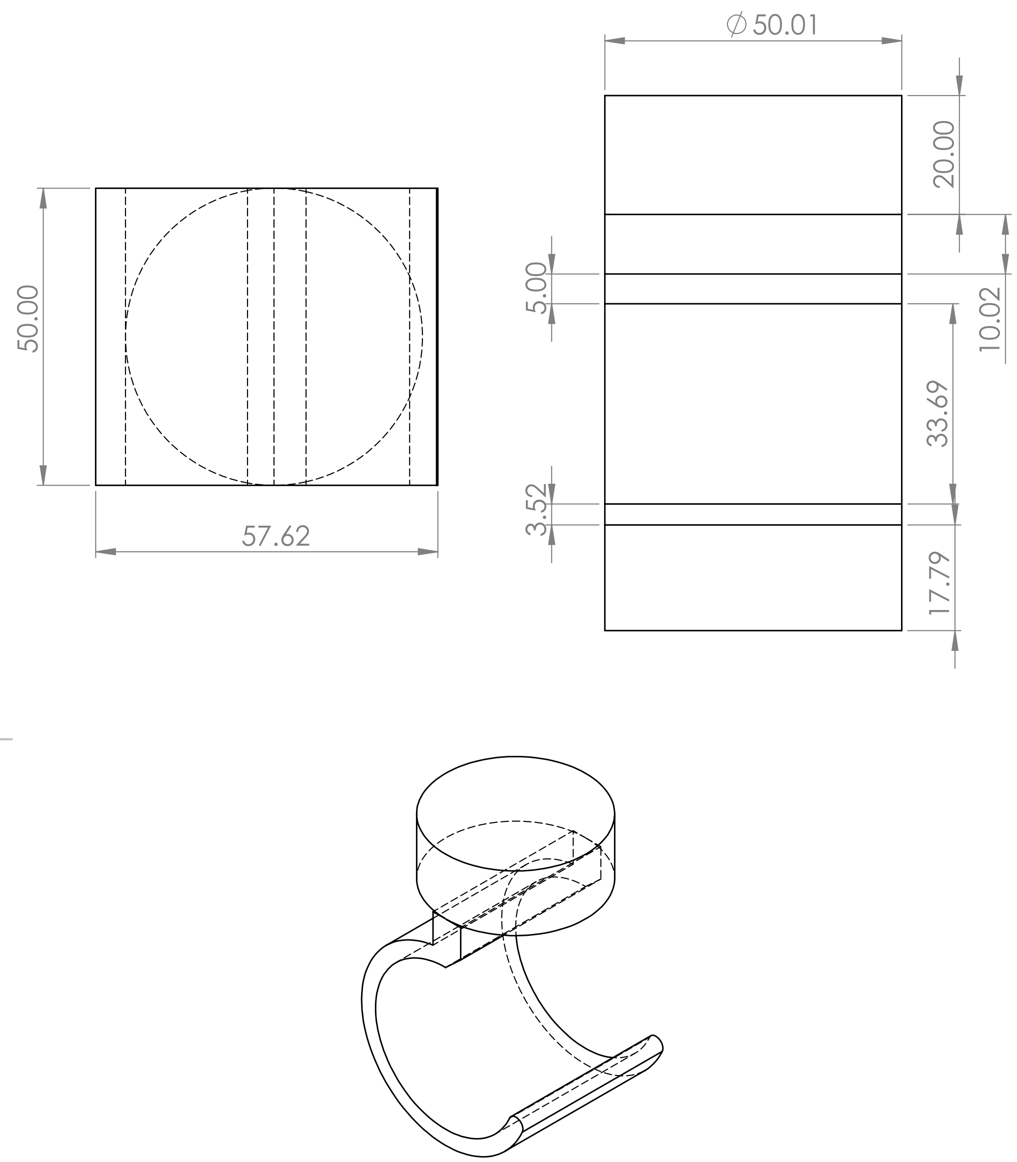
appendices/MATLAB/matrix_multiplication.tex

A.2 SolidWorks Drawings

A.2.1 The Hook Models

Circular Hook Model





UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:

FINISH:

DEBURR AND BREAK SHARP EDGES

DO NOT SCALE DRAWING

REVISION

	NAME	SIGNATURE	DATE
DRAWN			
CHK'D			
APPV'D			
MFG			
Q.A			

TITLE:

MATERIAL:

DWG NO.

circular hook 2

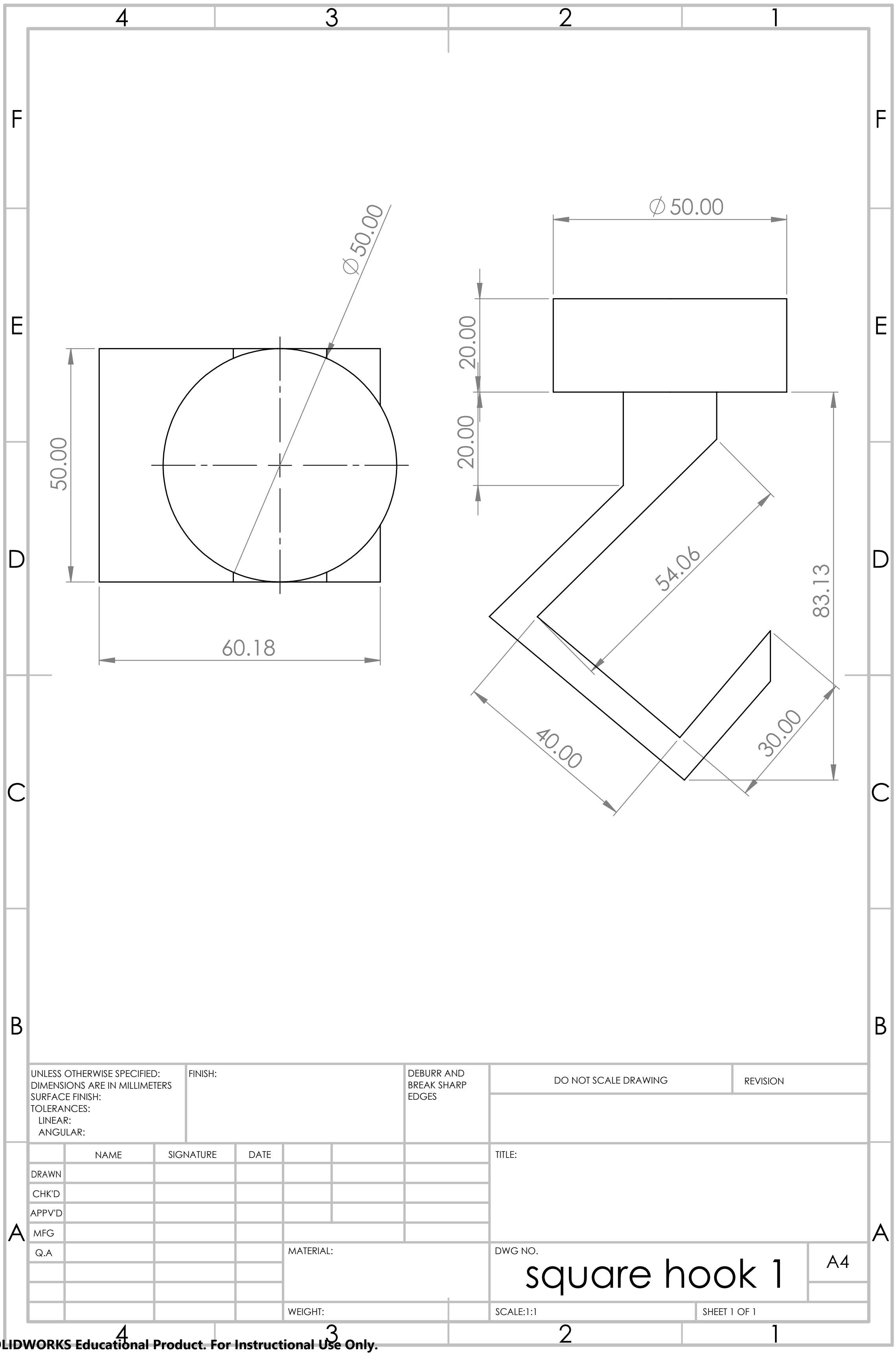
A4

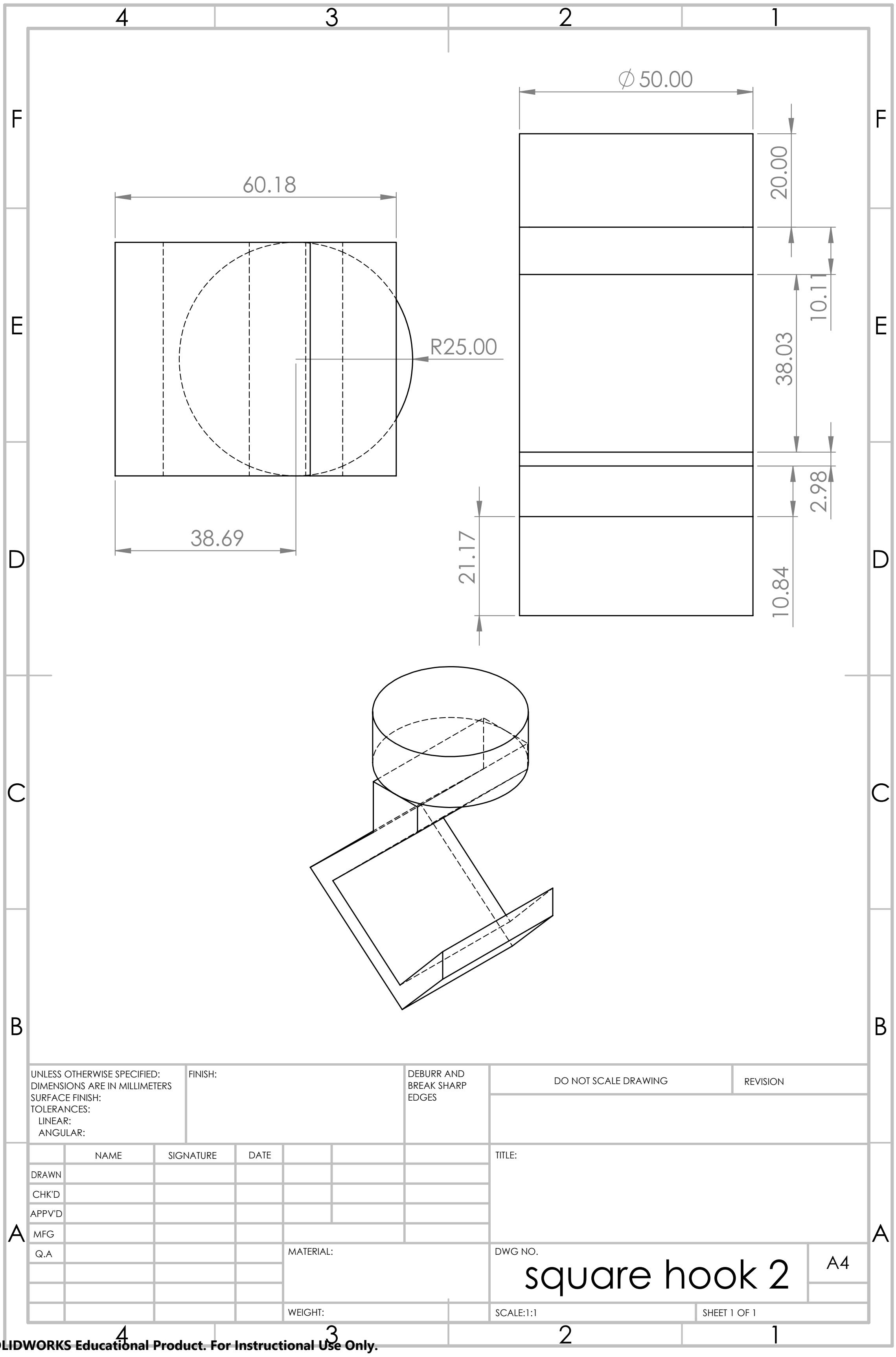
WEIGHT:

SCALE:1:1

SHEET 1 OF 1

Square Hook Model

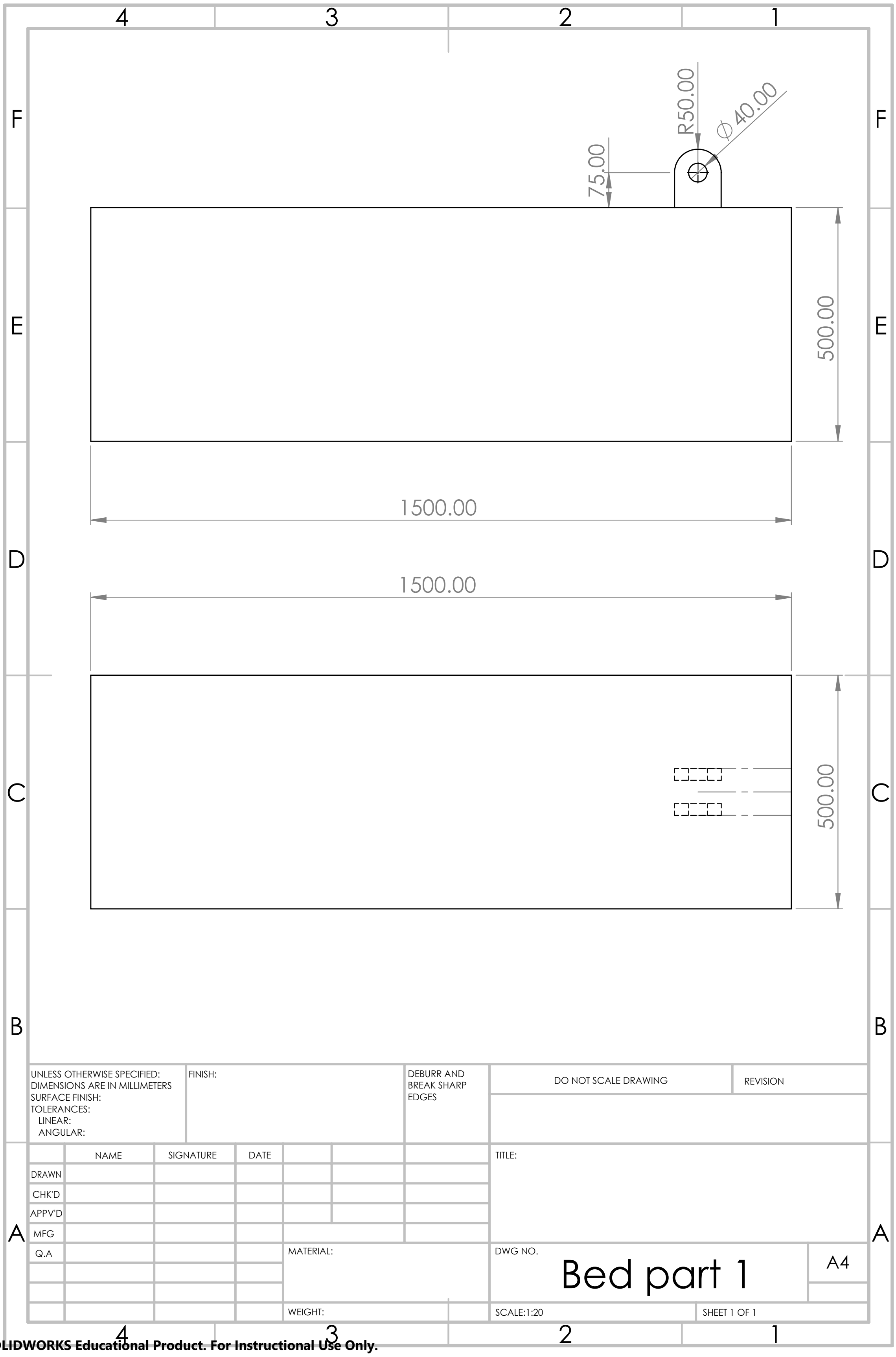




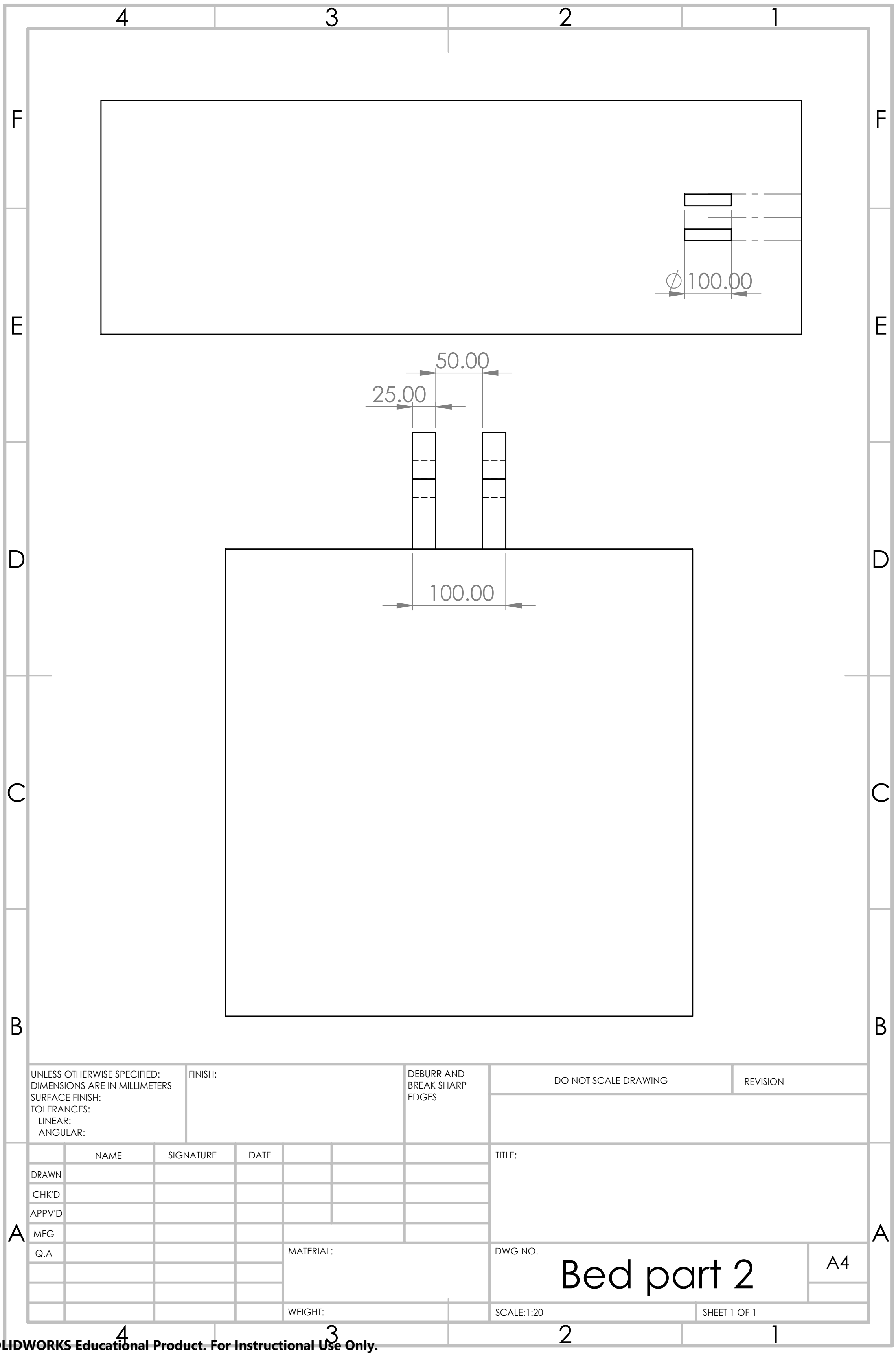
Ø50.00
60.18
20.00
10.11
R25.00
38.03
2.98
38.69
21.17
10.84
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
FINISH:
DEBURR AND BREAK SHARP EDGES
DO NOT SCALE DRAWING
REVISION
NAME
SIGNATURE
DATE
TITLE:
DRAWN
CHK'D
APPV'D
MFG
Q.A
MATERIAL:
DWG NO.
square hook 2
A4
WEIGHT:
SCALE:1:1
SHEET 1 OF 1

A.2.2 The Leg Model

Bed Part 1

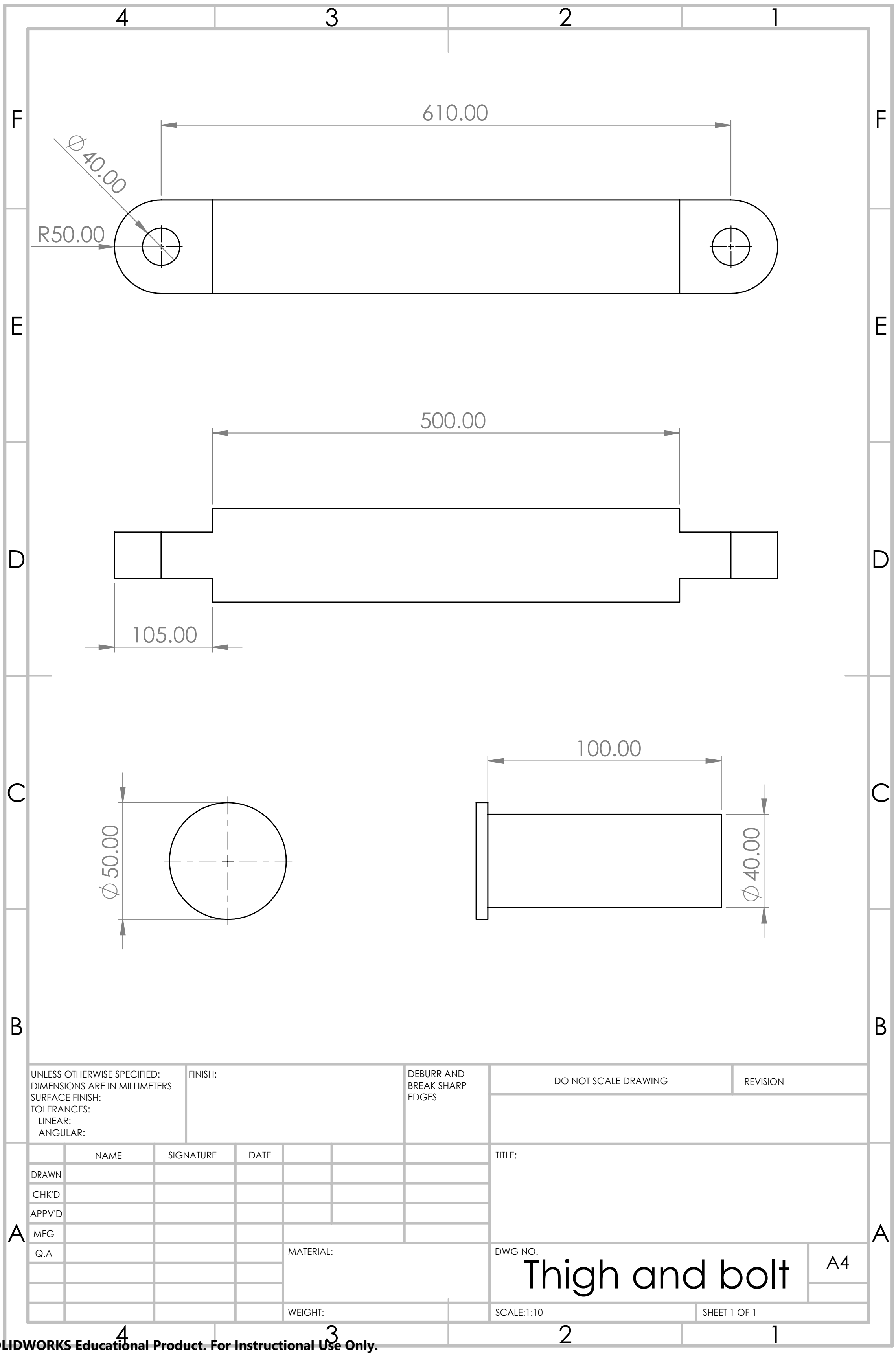


Bed Part 2

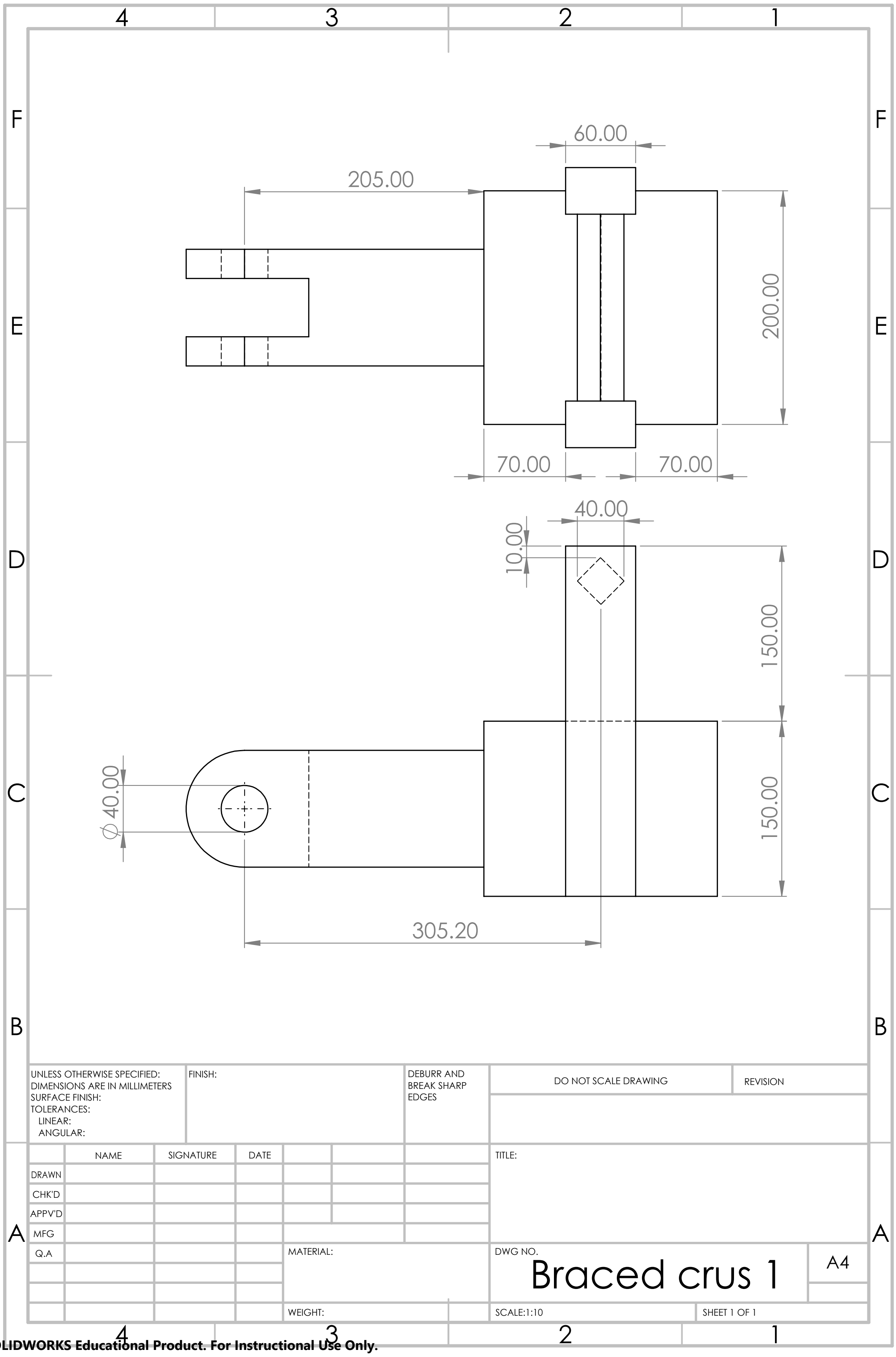


∅100.00
50.00
25.00
100.00
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
FINISH:
DEBURR AND BREAK SHARP EDGES
DO NOT SCALE DRAWING
REVISION
NAME
SIGNATURE
DATE
TITLE:
DRAWN
CHK'D
APPV'D
MFG
Q.A
MATERIAL:
DWG NO.
Bed part 2
A4
WEIGHT:
SCALE:1:20
SHEET 1 OF 1
SOLIDWORKS Educational Product. For Instructional Use Only.

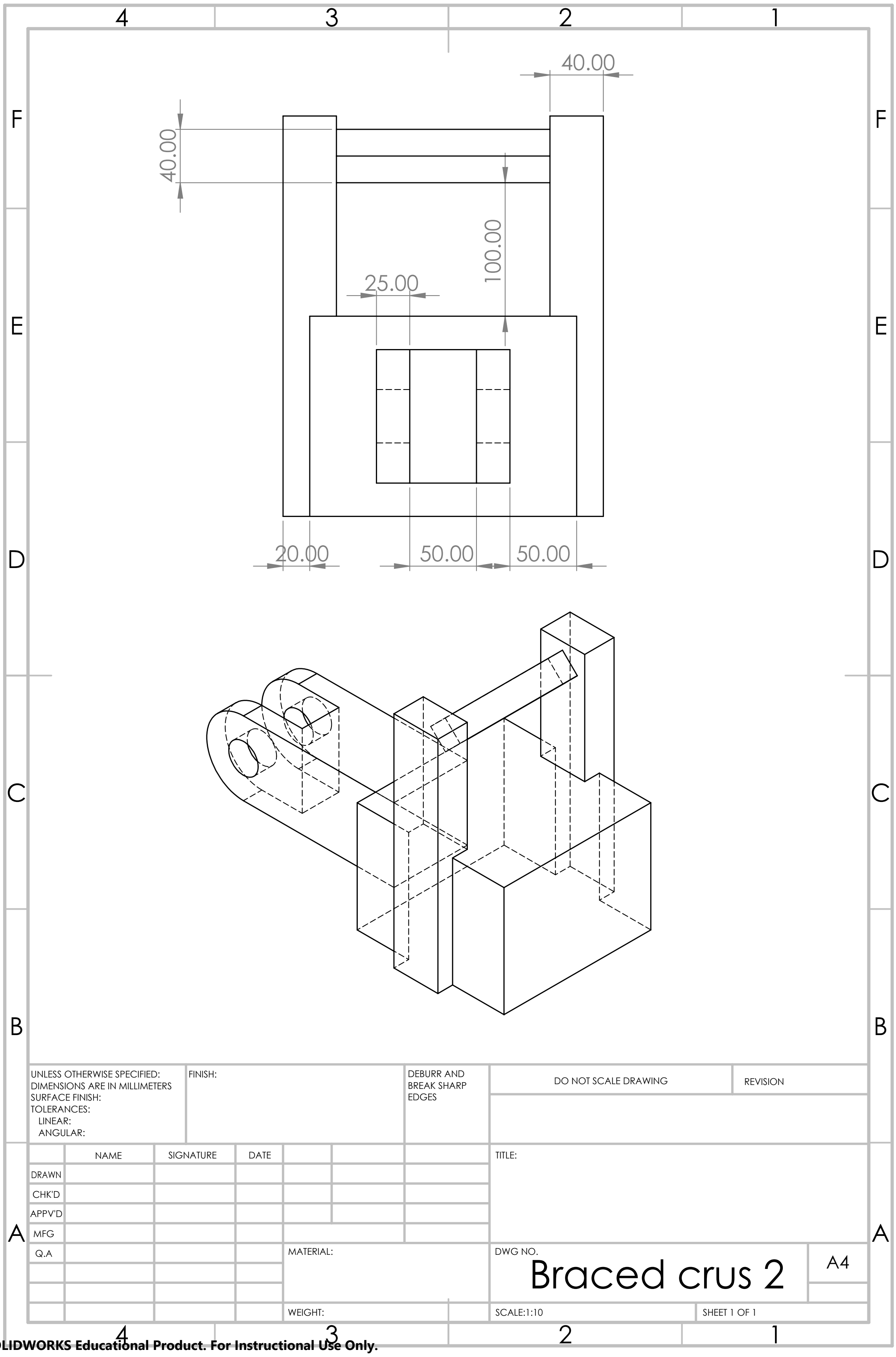
Thigh and Bolt



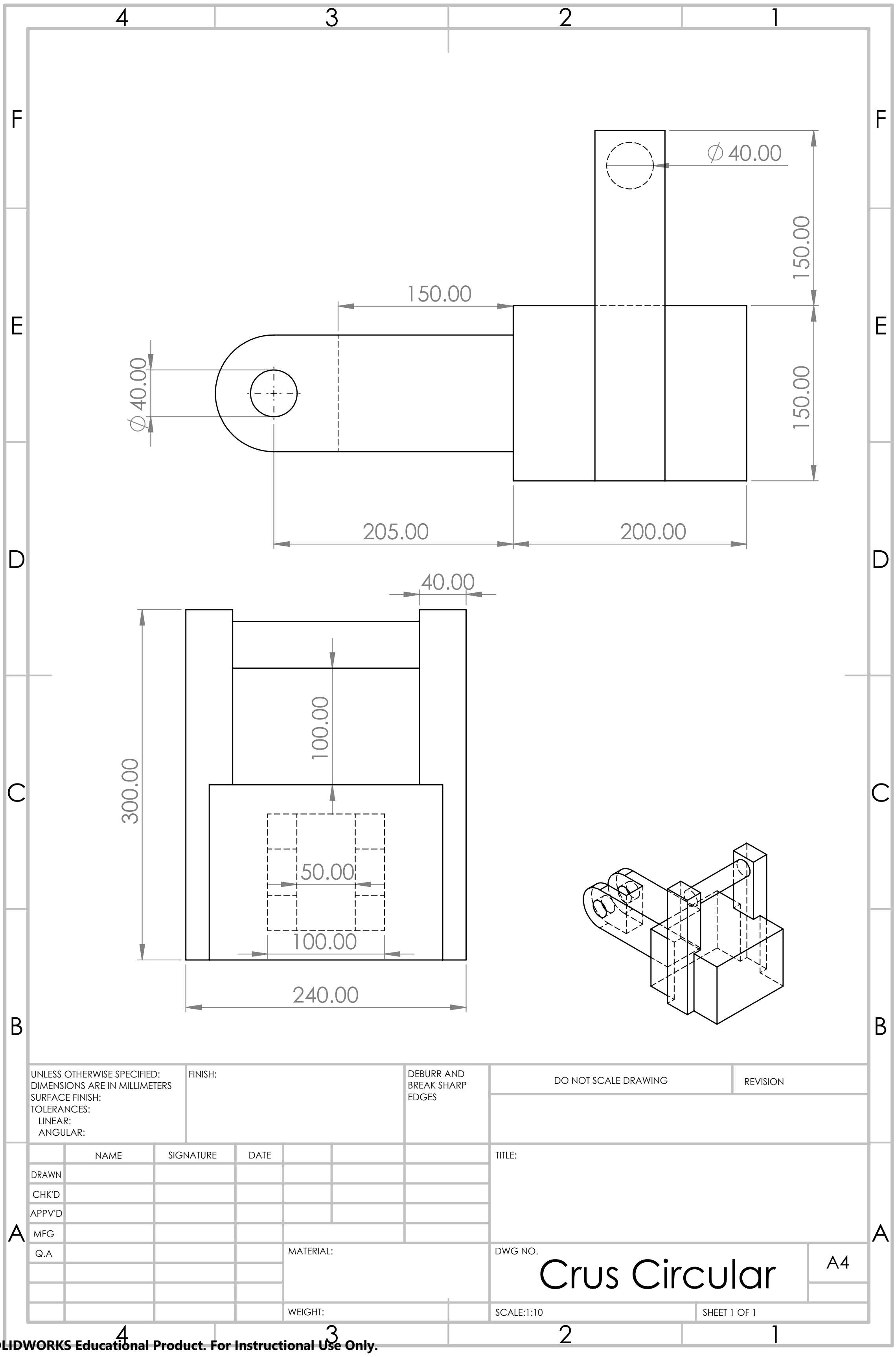
Braced Crus Square Part 1



Braced Crus Square Part 2

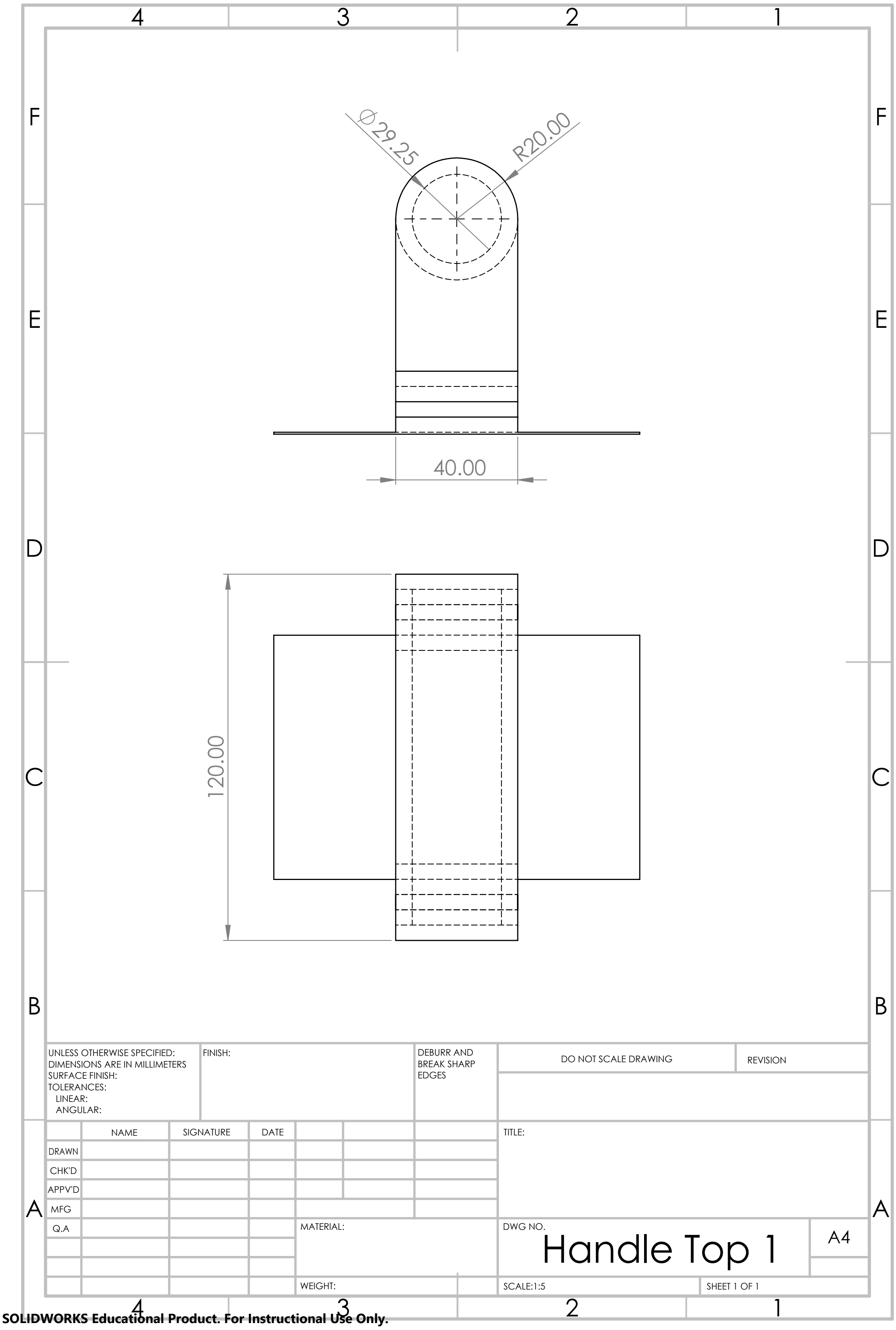


Braced Crus Circular

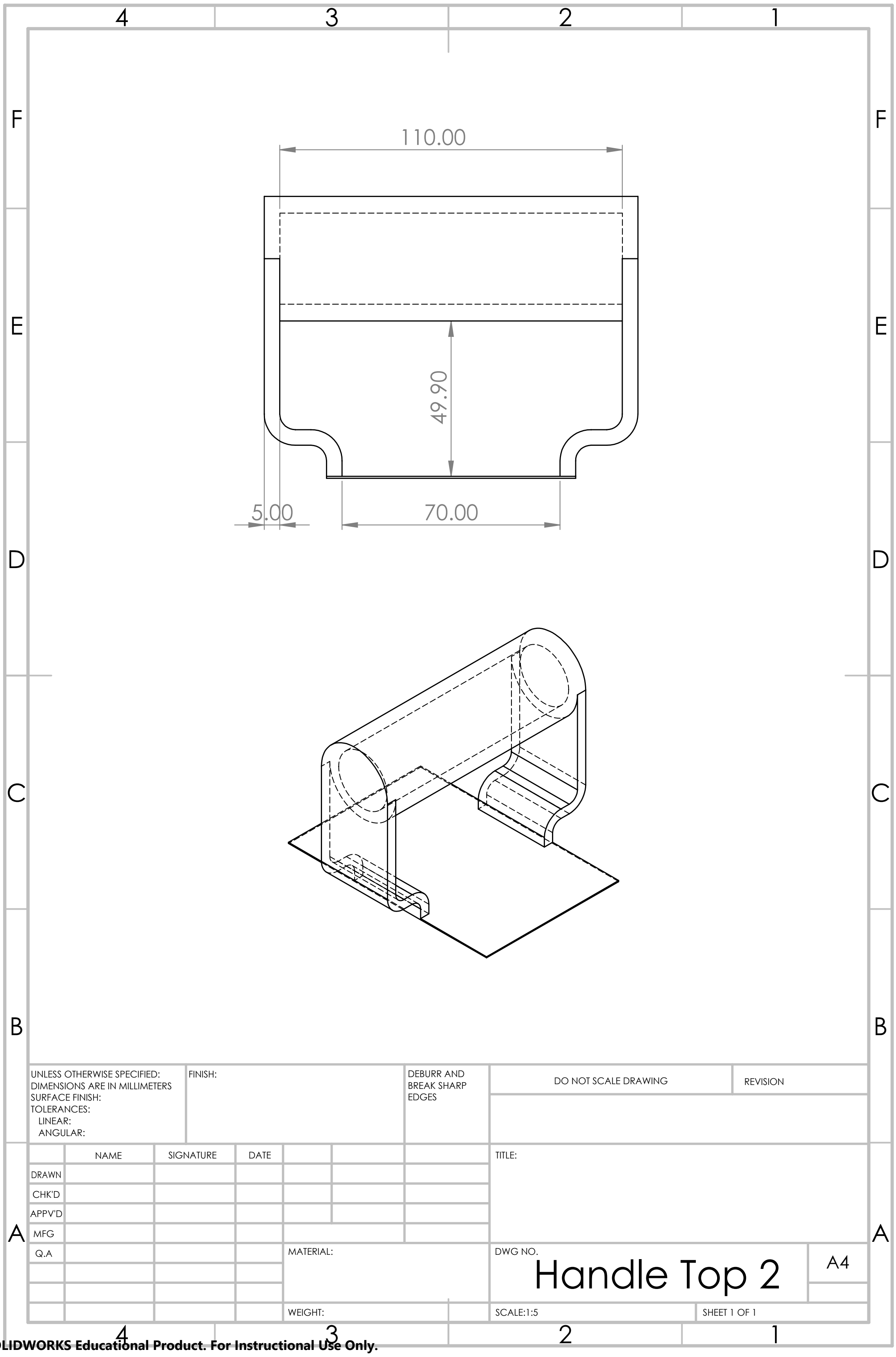


A.2.3 The Cuff

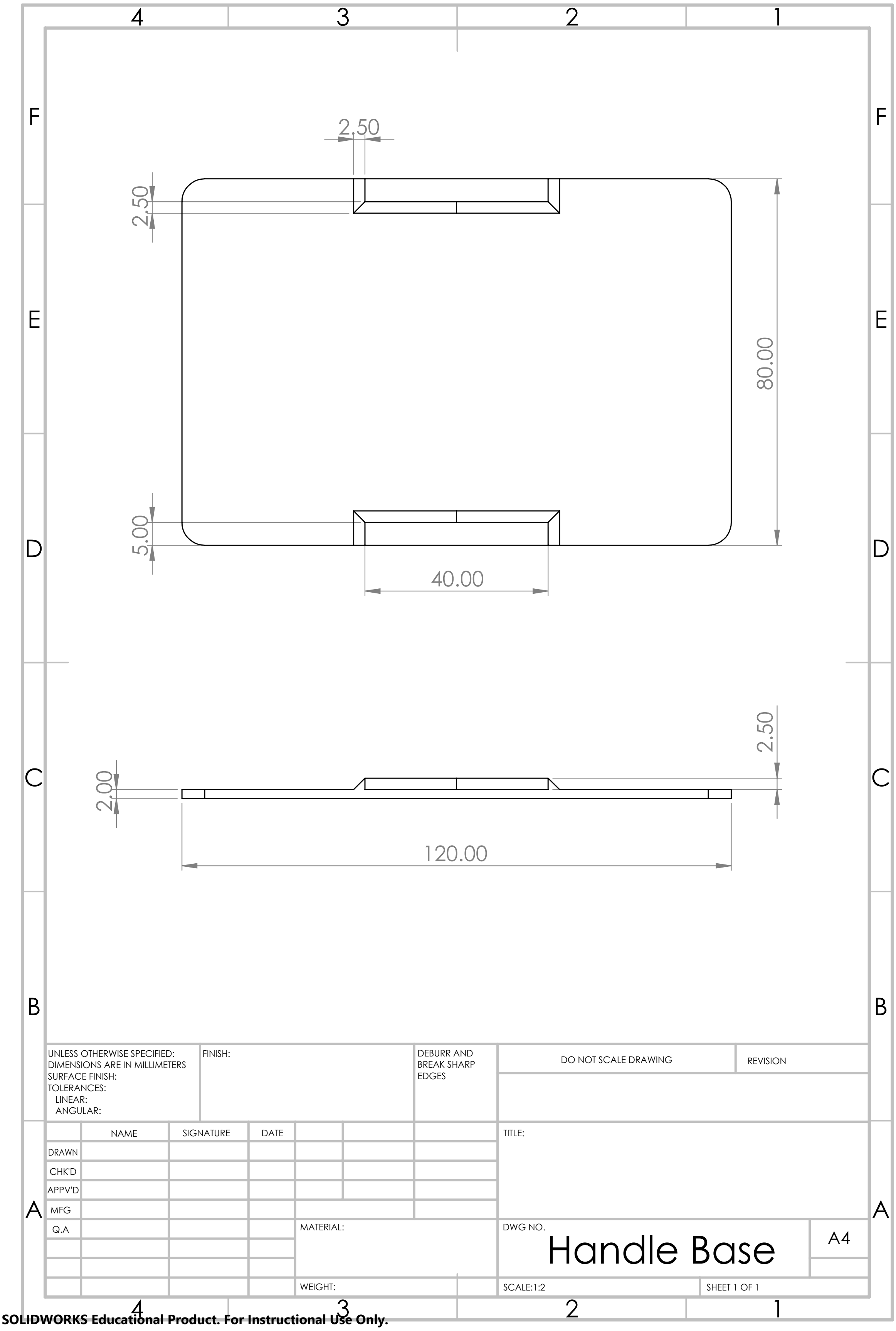
Handle Top Part 1



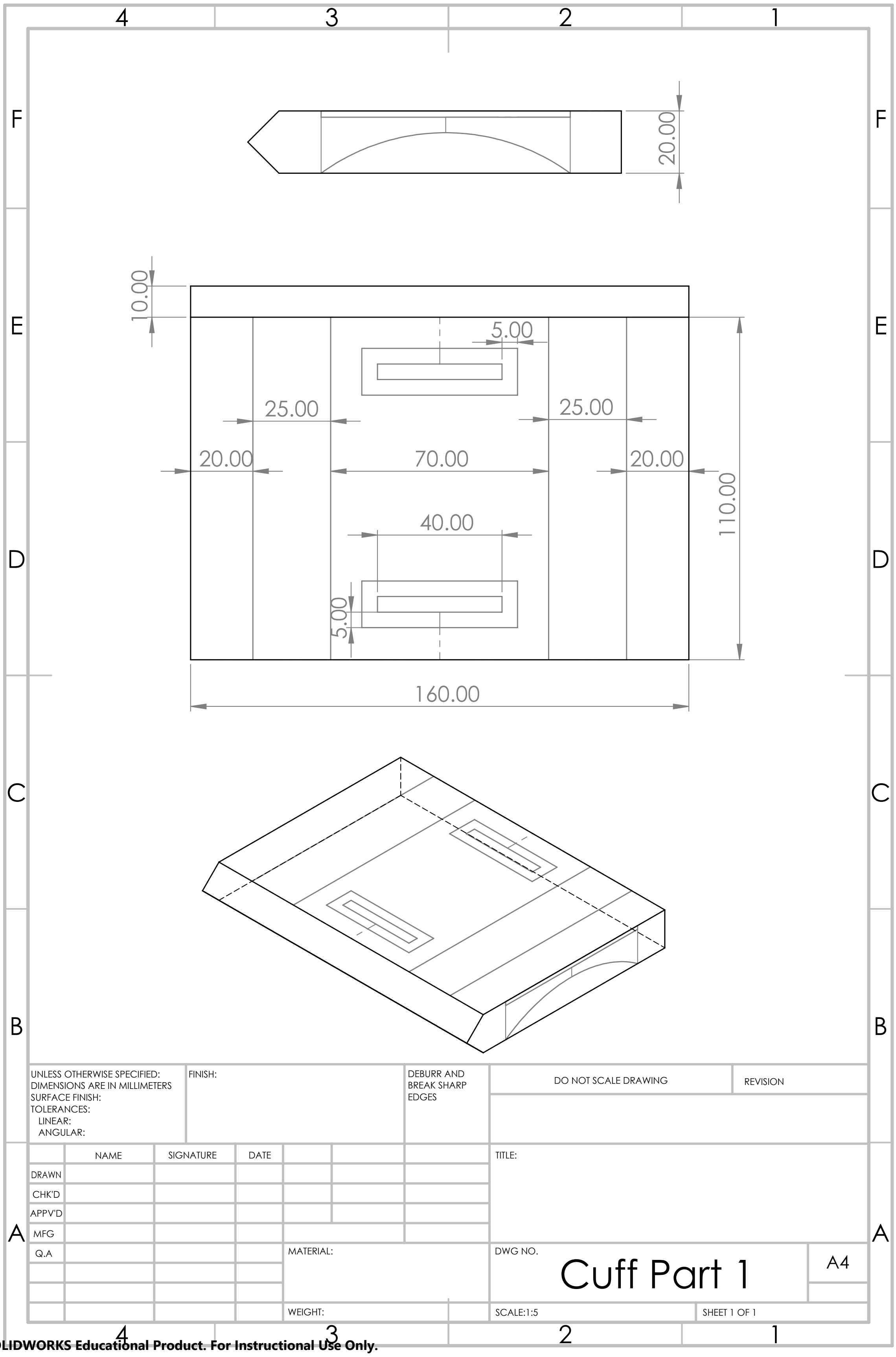
Handle Top Part 2



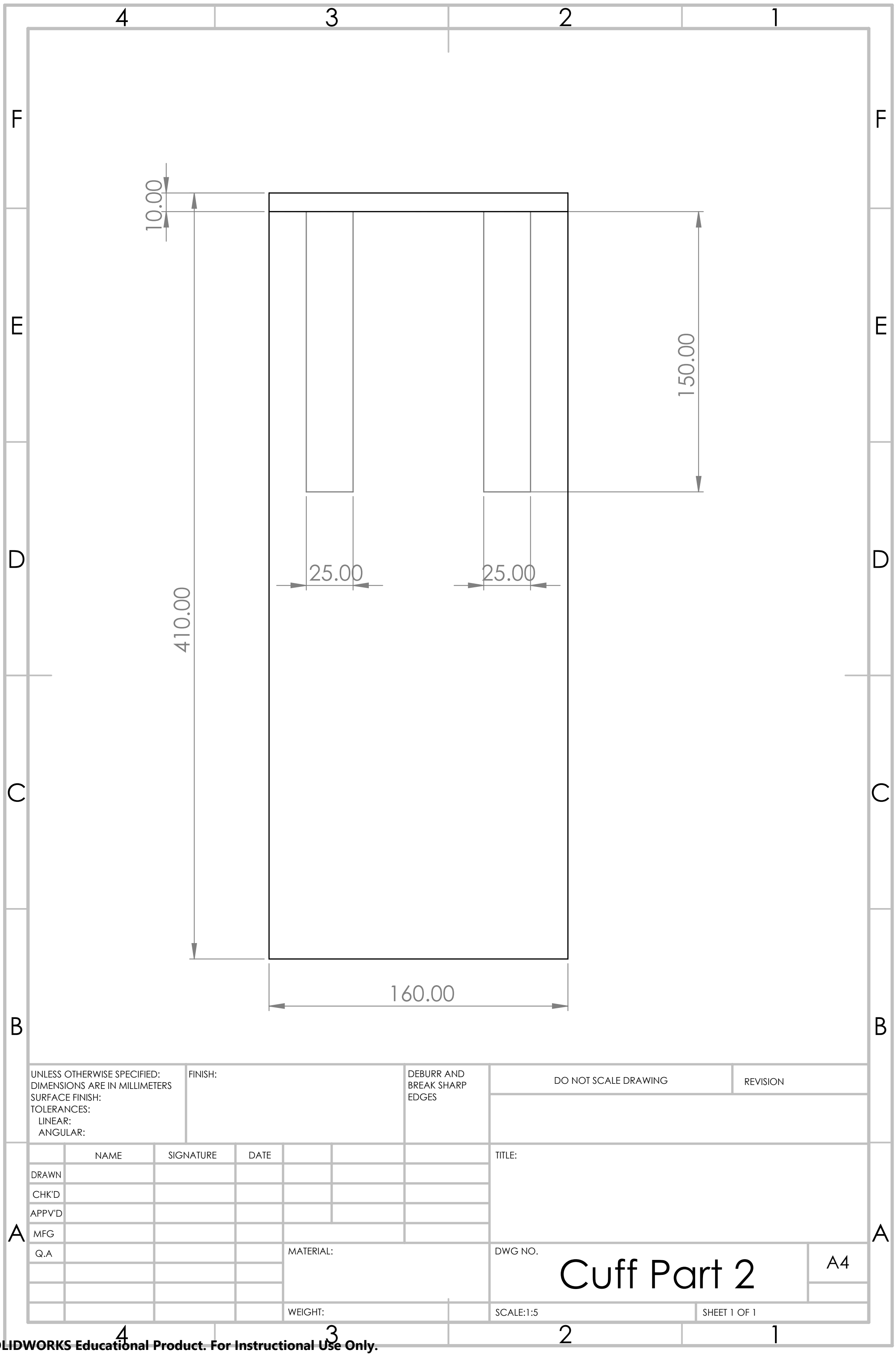
Handle Base


2.50
2.50
80.00
5.00
40.00
2.50
2.00
120.00
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
FINISH:
DEBURR AND BREAK SHARP EDGES
DO NOT SCALE DRAWING
REVISION
NAME
SIGNATURE
DATE
DRAWN
CHK'D
APPV'D
MFG
Q.A
MATERIAL:
WEIGHT:
TITLE:
DWG NO.
Handle Base
A4
SCALE:1:2
SHEET 1 OF 1
SOLIDWORKS Educational Product. For Instructional Use Only.

Cuff Part 1


4
3
2
1
F
E
D
C
B
A
20.00
10.00
5.00
25.00
25.00
20.00
70.00
20.00
110.00
40.00
5.00
160.00
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
FINISH:
DEBURR AND BREAK SHARP EDGES
DO NOT SCALE DRAWING
REVISION
NAME
SIGNATURE
DATE
DRAWN
CHK'D
APPV'D
MFG
Q.A
TITLE:
MATERIAL:
DWG NO.
Cuff Part 1
A4
WEIGHT:
SCALE:1:5
SHEET 1 OF 1
SOLIDWORKS Educational Product. For Instructional Use Only.

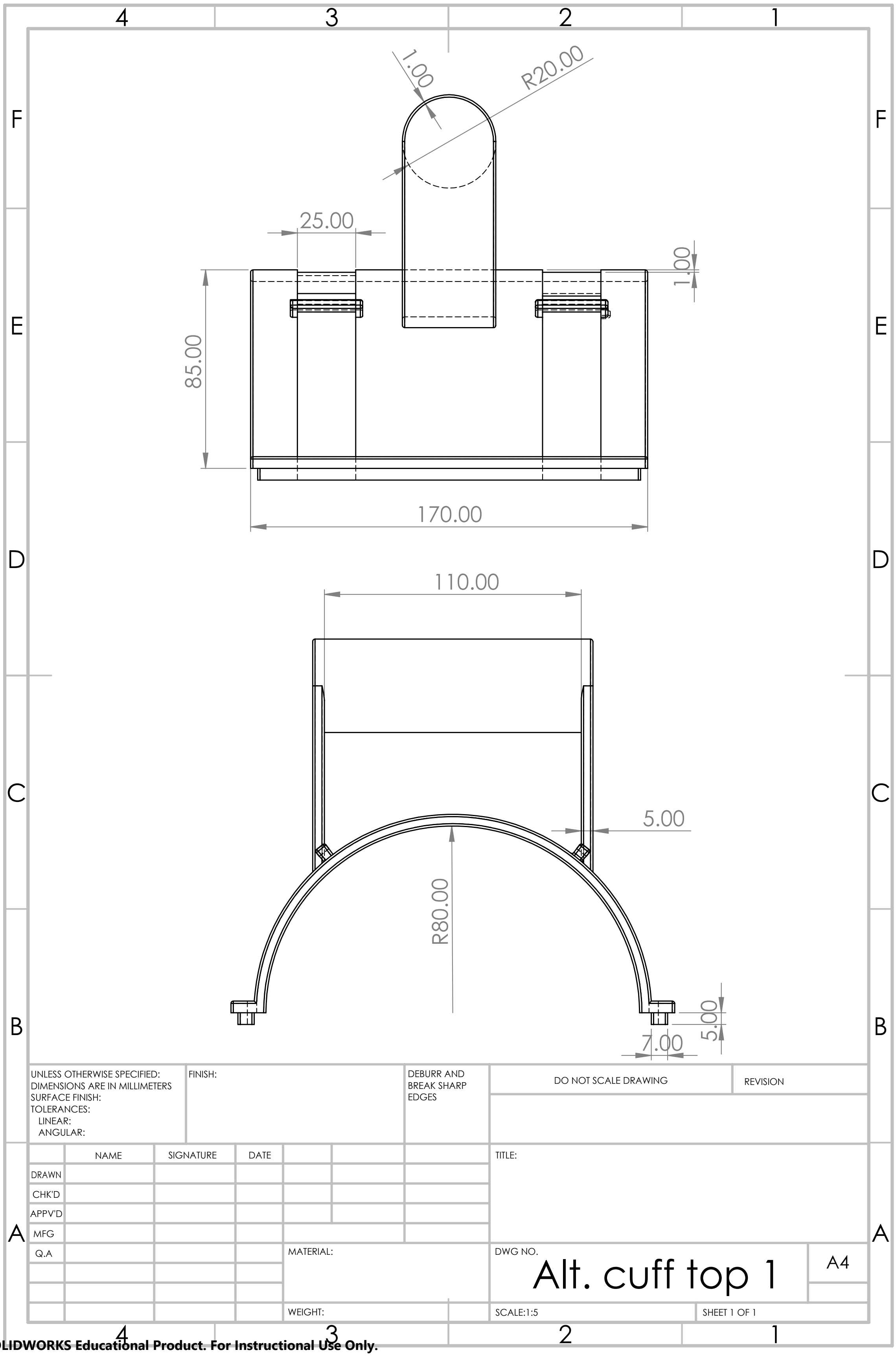
Cuff Part 2



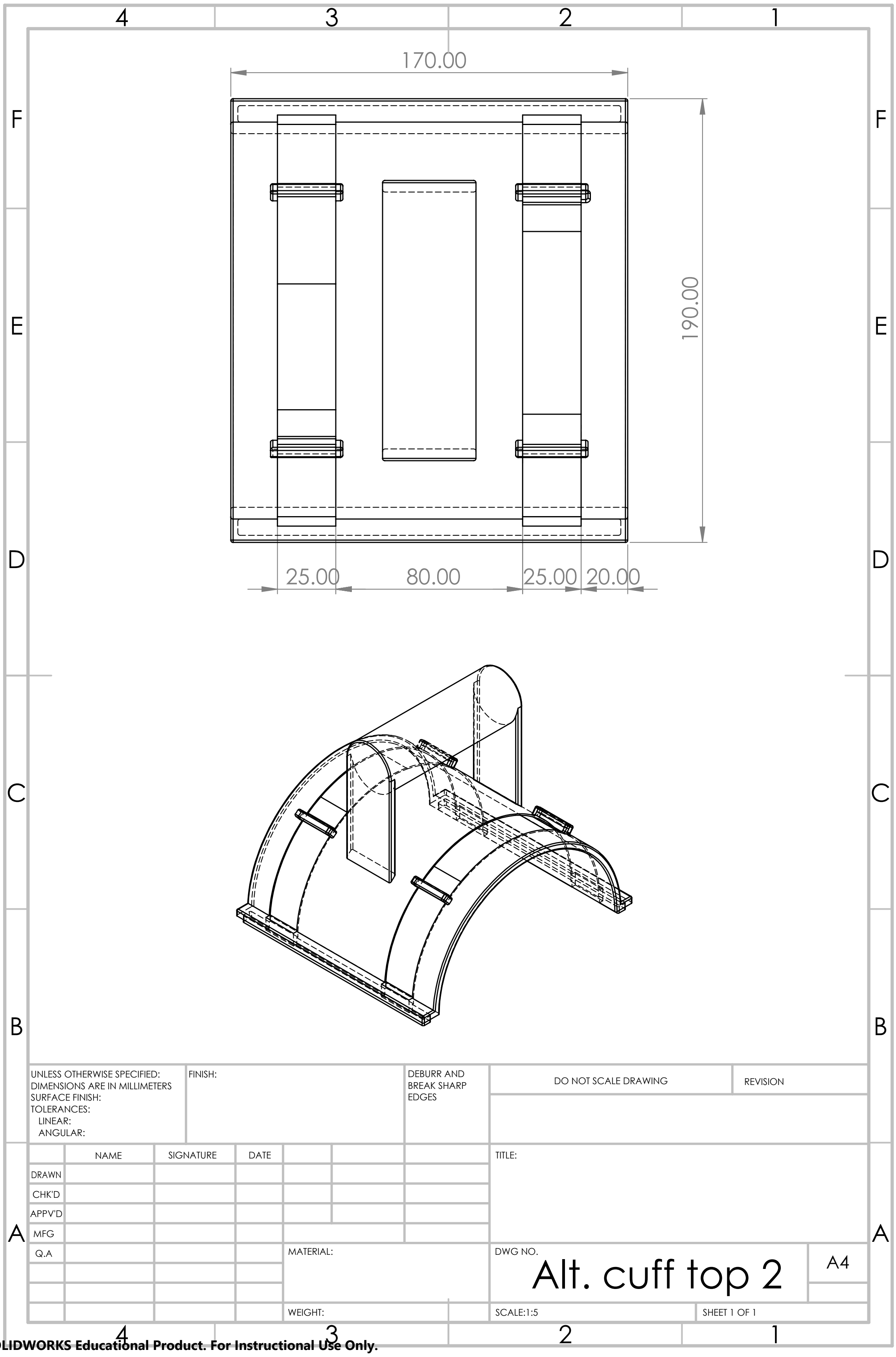
10.00
410.00
150.00
25.00
25.00
160.00
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
FINISH:
DEBURR AND BREAK SHARP EDGES
DO NOT SCALE DRAWING
REVISION
NAME
SIGNATURE
DATE
DRAWN
CHK'D
APPV'D
MFG
Q.A
TITLE:
MATERIAL:
DWG NO.
Cuff Part 2
A4
WEIGHT:
SCALE:1:5
SHEET 1 OF 1
SOLIDWORKS Educational Product. For Instructional Use Only.

A.2.4 The Alternative Cuff

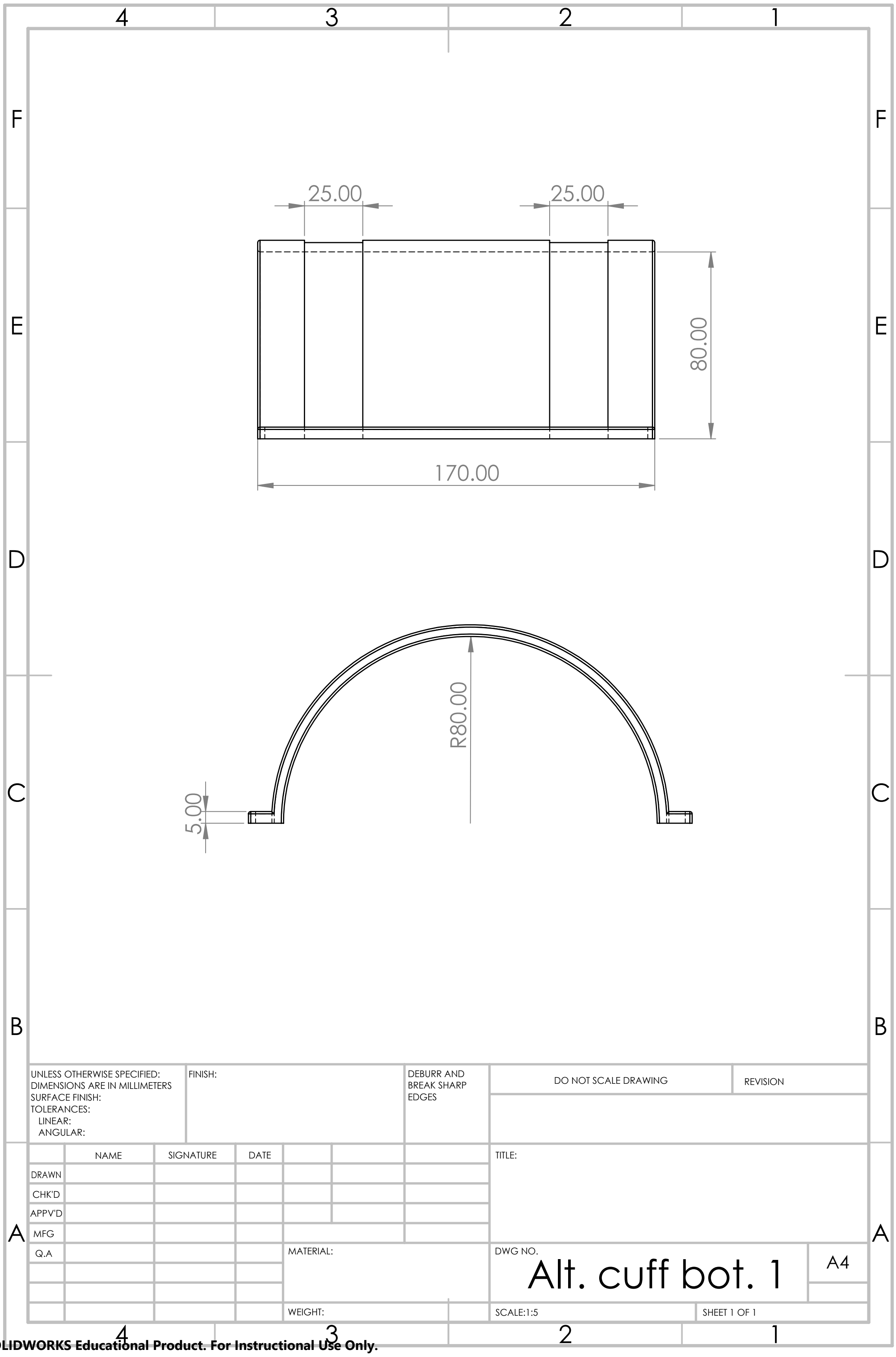
Alternative Cuff Top Part 1



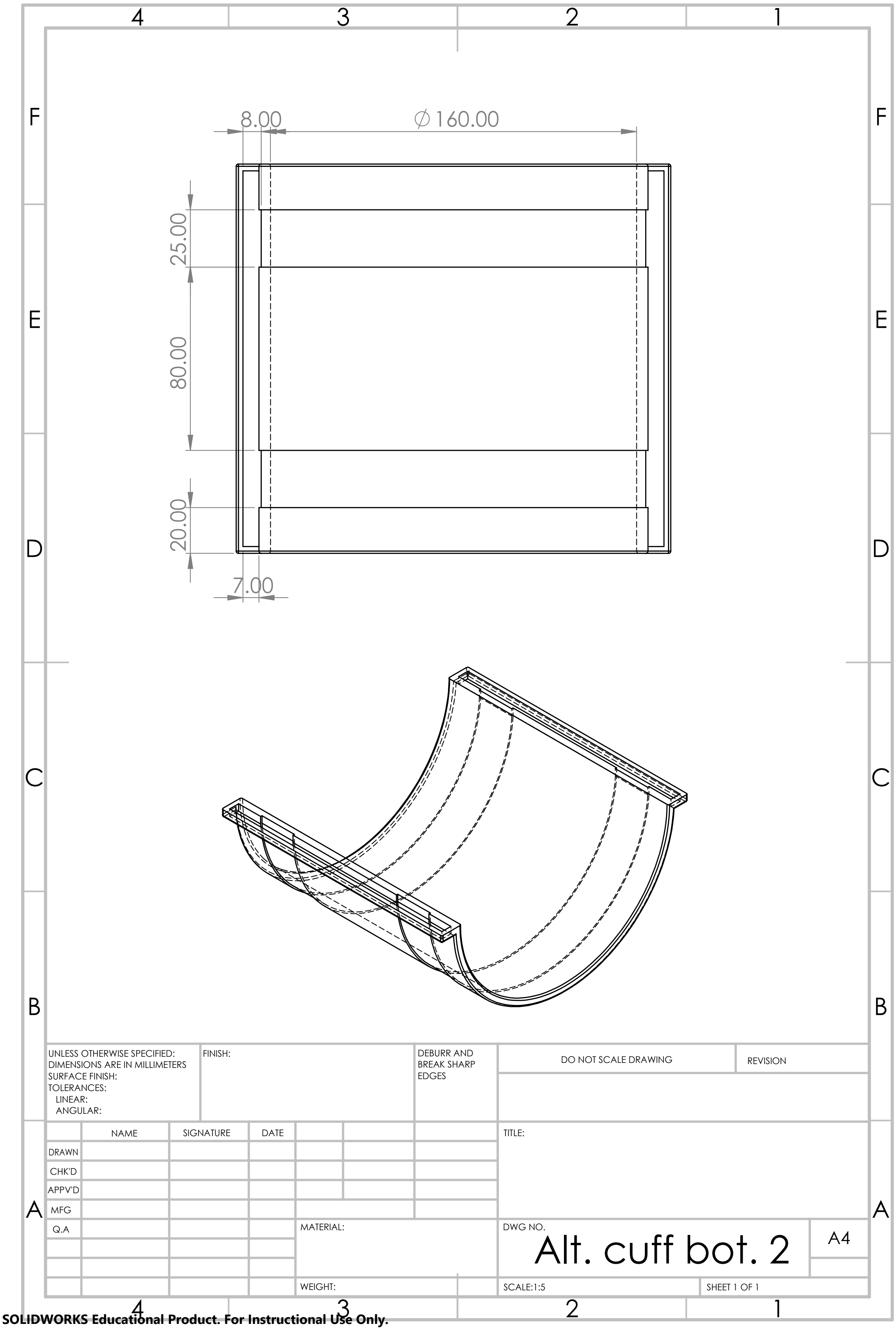
Alternative Cuff Top Part 2



Alternative Cuff Bottom Part 1



Alternative Cuff Bottom Part 2



A.3 Eve Menu Package

A.3.1 README.md

```
##EVE menu for Rehabilitation Project at UiA

* Author: Lars Bleie Andersen <larsa09@uia.no>

## Contents
- Joint Manipulation
- Grasp Manipulation
- Removing or adding stiffness in the right arm

## Instructions

- Copy the eve_menu folder into eve_ws/src
- Navigate back to eve_ws
- Run colcon build --packages-select eve_menu
- Run the program with:
  ros2 run eve_menu eve
```

appendices/eve_menu/README.tex

A.3.2 package.xml

```
<?xml version="1.0"?>
<?xml-model href="http://download.ros.org/schema/package_format3.xsd" schematypens="http://www.... 
    w3.org/2001/XMLSchema"?>
<package format="3">
  <name>eve_menu</name>
  <version>0.0.0</version>
  <description>Menu for Manipulating EVEr3 at UiA Grimstad</description>
  <maintainer email="larsa09@uia.no"> Lars Bleie Andersen </maintainer>
  <license>UiA</license>

  <buildtool_depend>ament_cmake</buildtool_depend>

  <depend>rclcpp</depend>
  <depend>std_msgs</depend>
  <depend>action_msgs</depend>
  <depend>halodi_msgs</depend>

  <test_depend>ament_lint_auto</test_depend>
  <test_depend>ament_lint_common</test_depend>

  <export>
    <build_type>ament_cmake</build_type>
  </export>
</package>
```

appendices/eve_menu/package.tex

A.3.3 CMakeLists.txt

```
cmake_minimum_required(VERSION 3.5)
project(eve_menu)

# Default to C99
if(NOT CMAKE_C_STANDARD)
  set(CMAKE_C_STANDARD 99)
endif()

# Default to C++14
if(NOT CMAKE_CXX_STANDARD)
  set(CMAKE_CXX_STANDARD 14)
```

```
endif()

if(CMAKE_COMPILER_IS_GNUCXX OR CMAKE_CXX_COMPILER_ID MATCHES "Clang")
  add_compile_options(-Wall -Wextra -Wpedantic)
endif()

# find dependencies
find_package(ament_cmake REQUIRED)
find_package(rclcpp REQUIRED)
find_package(std_msgs REQUIRED)
find_package(action_msgs REQUIRED)
find_package(halodi_msgs REQUIRED)
find_package(tf2 REQUIRED)
find_package(tf2_geometry_msgs REQUIRED)

add_executable(eve src/eve.cpp)
target_include_directories(eve PUBLIC
  $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/include>
  $<INSTALL_INTERFACE:include>)
ament_target_dependencies(eve rclcpp halodi_msgs action_msgs)

install(TARGETS eve
  DESTINATION lib/${PROJECT_NAME})



if(BUILD_TESTING)
  find_package(ament_lint_auto REQUIRED)
  ament_lint_auto_find_test_dependencies()
endif()

ament_package()
```

appendices/eve_menu/CMakeLists.tex

A.3.4 eve.cpp

```
#include <iostream>
#include <memory>
#include <boost/uuid/uuid.hpp>
#include <boost/uuid/uuid_generators.hpp>
#include "rclcpp/rclcpp.hpp"
#include "action_msgs/msg/goal_status.hpp"
#include "unique_identifier_msgs/msg/uuid.hpp"
#include "halodi_msgs/msg/whole_body_trajectory.hpp"
#include "halodi_msgs/msg/whole_body_trajectory_point.hpp"
#include "halodi_msgs/msg/joint_space_command.hpp"
#include "halodi_msgs/msg/joint_name.hpp"
#include "halodi_msgs/msg/hand_command.hpp"

using namespace halodi_msgs::msg; //Declaring the 'msg' command from the 'halodi_msgs'
using std::placeholders::_1; //Declaring the placeholder '_1' which directs the position of a ...
    value in a function to the first parameter
double storage[20]; //Declaring the array for 21 joint values, stored as radians
double closure_ratio = 0.75; //Setting the default values. 0.75 was found to be the optimal ...
    ratio for a handle with 40[mm] diameter.
double closure_velocity = 0.5;
double closure_force = 0.5;
bool right_hand = true; //right is ambiguous, but right_hand is ok
bool left_hand = false;
bool both_hands = false;
bool relax = false; //Declaring the boolean statements for selections
bool downward = false;
bool firm = false;

class Grasp_publisher : public rclcpp::Node //Creating the node class 'Grasp_publisher' by ...
    inheriting from 'rclcpp::Node'
```

```
{
public:
  Grasp_publisher()
  : Node("grasp_publisher") //The public constructor names the node 'grasp_publisher'
  {
    publisher_ = this->create_publisher<HandCommand>("/eve/right_hand_closure", 10); //The ...
        first publisher is initialized with the 'HandCommand' message type, the topic name '/...
        eve/right_hand_closure', and the required queue size to limit messages in the event of ...
        a backup.
    publisher2_ = this->create_publisher<HandCommand>("/eve/left_hand_closure", 10);//The ...
        second publisher is initialized with the topic name '/eve/left_hand_closure'
    publish_hand_command(); //Running the 'publish_hand_command' function
  }

private:
  void publish_hand_command() //The function sets the desired parameters and publishes the ...
      message(s). void has not return type.
  {
    HandCommand grasp_msg;
    grasp_msg.closure = closure_ratio;
    grasp_msg.speed = closure_velocity;
    grasp_msg.force = closure_force;
    if(right_hand==true)
    {
      publisher_->publish(grasp_msg); //The message is published to the right hand
    }
    if(left_hand==true)
    {
      publisher2_->publish(grasp_msg); //The message is published to the left hand
    }
    if(both_hands == true)
    {
       publisher_->publish(grasp_msg); //The message is published to both hands
       publisher2_->publish(grasp_msg);
    }
  }

  rclcpp::Publisher<HandCommand>::SharedPtr publisher_; //Declaration of the 'publisher_' ...
      publisher
  rclcpp::Publisher<HandCommand>::SharedPtr publisher2_; //Declaration of the 'publisher2_' ...
      publisher
};


class JointManipulator : public rclcpp::Node //Creating the node class 'JointManipulator' by ...
    inheriting from 'rclcpp::Node'
{
public:
  JointManipulator()
  : Node("joint_manipulator") //The public constructor names the node 'joint_manipulator'
  {
    // set up publisher to trajectory topic
    publisher_ = this->create_publisher<WholeBodyTrajectory>("/eve/whole_body_trajectory", 10);...
         //The publisher is initialized with the 'WholeBodyTrajectory' message type, the topic ...
        name '/eve/whole_body_trajectory', and the required queue size to limit messages in the...
         event of a backup.
    //The subscriber is initialized with the 'action_msgs::msg::GoalStatus' message type. 'std...
        ::bind' is used to register the '&Joint_menu_publisher::status_callback' reference as a...
         callback. It provides feedback of commands send to /eve/whole_body_trajectory
    subscription_ = this->create_subscription<action_msgs::msg::GoalStatus>("/eve/...
        whole_body_trajectory_status", 10, std::bind(&JointManipulator::status_callback, this, ...
        _1));
    uuid_msg_ = create_random_uuid(); //A UUID (universally unique identifier) is created ...
        though the create_random_uuid function
    publish_trajectory(uuid_msg_); //The publish_trajectory function is run with the UUID as ...
        the trajectory_id
  }
```

```
private:

void status_callback(action_msgs::msg::GoalStatus::SharedPtr msg)
  {
    switch(msg->status){ //The arrow operator (->) is used to access the member of the 'status'...
         data structure pointed to by a pointer. A pointer is used because the data cannot be ...
        directly copied and a pointer is what is really transferred. The 'status' data ...
        structure has 7 different states.
      case 0:
        RCLCPP_INFO(this->get_logger(), "GoalStatus: STATUS_UNKNOWN"); //The 'this' pointer is ...
            here used to retrieve the 'get_logger()' information
        break;
      case 1:
        RCLCPP_INFO(this->get_logger(), "GoalStatus: STATUS_ACCEPTED");
        break;
      case 2:
        RCLCPP_INFO(this->get_logger(), "GoalStatus: STATUS_EXECUTING");
        break;
      case 3:
        RCLCPP_INFO(this->get_logger(), "GoalStatus: STATUS_CANCELING");
        break;
      case 4:
        //If the uuid of the received GoalStatus STATUS_SUCCEEDED Msg is the same as the uuid of...
             the command that was sent
        RCLCPP_INFO(this->get_logger(), "GoalStatus: STATUS_SUCCEEDED");
        RCLCPP_INFO(this->get_logger(), "Shutting down...");
        rclcpp::shutdown();
        break;
      case 5:
        RCLCPP_INFO(this->get_logger(), "GoalStatus: STATUS_CANCELED");
        break;
      case 6:
        RCLCPP_INFO(this->get_logger(), "GoalStatus: STATUS_ABORTED");
        break;
      default:
        break;
    }
  }

  unique_identifier_msgs::msg::UUID create_random_uuid() //The function creates a random UUID ...
      to track msg
  {
    boost::uuids::random_generator gen; boost::uuids::uuid u = gen();
    unique_identifier_msgs::msg::UUID uuid_msg;
    std::array<uint8_t, 16> uuid; std::copy(std::begin(u.data), std::end(u.data), uuid.begin())...
        ;
    uuid_msg.uuid = uuid;
    return uuid_msg;
  }

  void publish_trajectory(unique_identifier_msgs::msg::UUID uuid_msg) //The function sets all ...
      required parameters for the WholeBodyTrajectory structure and send them to 'publisher_'
  {
    // begin construction of the publsihed msg
    WholeBodyTrajectory trajectory_msg;
    trajectory_msg.append_trajectory = false; //If set to false, the existing trajectory is ...
        cancelled and this trajectory is immediatly executed.
    //MINIMUM_JERK_CONSTRAINED mode is recommended to constrain joint velocities and ...
        accelerations between each waypoint (make the movement smoother)
    //It specifies how the trajectory is interpolated from the previous objective
    trajectory_msg.interpolation_mode.value = TrajectoryInterpolation::MINIMUM_JERK_CONSTRAINED...
        ;
    trajectory_msg.trajectory_id = uuid_msg;
    // Ading waypoint targets. The desired times (in seconds) are provided in terms of
    // offset from time at which this published message is received
    trajectory_msg.trajectory_points.push_back(targetall_(5));
```

```
  RCLCPP_INFO(this->get_logger(), "Sending trajectory, listening for ...
      whole_body_trajectory_status...");
  publisher_->publish(trajectory_msg); //The trajectory message is published
}

//This generates the individual single joint command. It takes in the required parameters and...
     returns the message used for the taskspace trajectory point in the 'targetall_' function
JointSpaceCommand generate_joint_space_command(int32_t joint_id, double q_des, double qd_des ...
    = 0.0, double qdd_des = 0.0)
{
  JointSpaceCommand ret_msg;
  JointName name;
  name.joint_id = joint_id;
  ret_msg.joint = name;
  ret_msg.q_desired = q_des;
  ret_msg.qd_desired = qd_des;
  ret_msg.qdd_desired = qdd_des;
  ret_msg.use_default_gains = true;
  return ret_msg;
}

//Each target, in the form of a single WholeBodyTrajectoryPoint msg, consists of a ...
    concatenation of desired joint configurations with no more than one desired value per ...
    joint. The desired time at which we want to reach these joint targets is also specified.
//This function then takes in the execution time and assembles the name of the desired joint ...
    along with the desired position, velocity, acceleration etc. through the '...
    generate_joint_space_command' function
WholeBodyTrajectoryPoint targetall_(int32_t t)
{
  WholeBodyTrajectoryPoint ret_msg;

  builtin_interfaces::msg::Duration duration;
  duration.sec = t; //Sets the execution time for the trajectory, relative to the start time.
  ret_msg.time_from_start = duration;

  ret_msg.joint_space_commands.push_back(generate_joint_space_command(JointName::HIP_YAW, ...
      storage[0])); //The 'push_back' pushes elements from the back of the ...
      generate_joint_space_command contents
  ret_msg.joint_space_commands.push_back(generate_joint_space_command(JointName::HIP_ROLL, ...
      storage[1]));
  ret_msg.joint_space_commands.push_back(generate_joint_space_command(JointName::HIP_PITCH, ...
      storage[2]));
  ret_msg.joint_space_commands.push_back(generate_joint_space_command(JointName::KNEE_PITCH, ...
      storage[3]));
  ret_msg.joint_space_commands.push_back(generate_joint_space_command(JointName::ANKLE_ROLL, ...
      storage[4]));
  ret_msg.joint_space_commands.push_back(generate_joint_space_command(JointName::ANKLE_PITCH,...
       storage[5]));
  ret_msg.joint_space_commands.push_back(generate_joint_space_command(JointName::...
      LEFT_SHOULDER_PITCH, storage[6]));
  ret_msg.joint_space_commands.push_back(generate_joint_space_command(JointName::...
      LEFT_SHOULDER_ROLL, storage[7]));
  ret_msg.joint_space_commands.push_back(generate_joint_space_command(JointName::...
      LEFT_SHOULDER_YAW, storage[8]));
  ret_msg.joint_space_commands.push_back(generate_joint_space_command(JointName::...
      LEFT_ELBOW_PITCH, storage[9]));
  ret_msg.joint_space_commands.push_back(generate_joint_space_command(JointName::...
      LEFT_ELBOW_YAW, storage[10]));
  ret_msg.joint_space_commands.push_back(generate_joint_space_command(JointName::...
      LEFT_WRIST_PITCH, storage[11]));
  ret_msg.joint_space_commands.push_back(generate_joint_space_command(JointName::...
      LEFT_WRIST_ROLL, storage[12]));
  ret_msg.joint_space_commands.push_back(generate_joint_space_command(JointName::...
      RIGHT_SHOULDER_PITCH, storage[13]));
  ret_msg.joint_space_commands.push_back(generate_joint_space_command(JointName::...
      RIGHT_SHOULDER_ROLL, storage[14]));
```

```
    ret_msg.joint_space_commands.push_back(generate_joint_space_command(JointName::...
        RIGHT_SHOULDER_YAW, storage[15]));
    ret_msg.joint_space_commands.push_back(generate_joint_space_command(JointName::...
        RIGHT_ELBOW_PITCH, storage[16]));
    ret_msg.joint_space_commands.push_back(generate_joint_space_command(JointName::...
        RIGHT_ELBOW_YAW, storage[17]));
    ret_msg.joint_space_commands.push_back(generate_joint_space_command(JointName::...
        RIGHT_WRIST_PITCH, storage[18]));
    ret_msg.joint_space_commands.push_back(generate_joint_space_command(JointName::...
        RIGHT_WRIST_ROLL, storage[19]));
    ret_msg.joint_space_commands.push_back(generate_joint_space_command(JointName::NECK_PITCH, ...
        storage[20]));

    return ret_msg;
  }

  rclcpp::Publisher<WholeBodyTrajectory>::SharedPtr publisher_; //Declaration of the '...
      publisher_' publisher
  rclcpp::Subscription<action_msgs::msg::GoalStatus>::SharedPtr subscription_; //Declaration of...
       the 'subscription_' subscriber
  unique_identifier_msgs::msg::UUID uuid_msg_; //Declaration of the UUID generator
};

class Relaxing_publisher : public rclcpp::Node //Creating the node class 'Relaxing_publisher' ...
     by inheriting from 'rclcpp::Node'
{
public:
  Relaxing_publisher()
  : Node("relaxing_publisher") //The public constructor names the node 'relaxing_publisher'
  {
    publisher_ = this->create_publisher<WholeBodyTrajectory>("/eve/whole_body_trajectory", 10);...
         //The publisher is initialized with the 'WholeBodyTrajectory' message type, the topic ...
        name '/eve/whole_body_trajectory', and the required queue size to limit messages in the...
         event of a backup.
    subscriber_ = this->create_subscription<action_msgs::msg::GoalStatus>("/eve/...
        whole_body_trajectory_status", 10, std::bind(&Relaxing_publisher::status_callback, this...
        , _1)); //The subscriber is initialized with the 'action_msgs::msg::GoalStatus' message...
         type. 'std::bind' is used to register the '&Joint_menu_publisher::status_callback' ...
        reference as a callback. It provides feedback of commands send to /eve/...
        whole_body_trajectory
    uuid_msg_ = create_random_uuid(); //A UUID (universally unique identifier) is created ...
        though the create_random_uuid function
    publish_message(uuid_msg_); //The publish_trajectory function is run with the UUID as the ...
        trajectory_id
  }

private:

void status_callback(action_msgs::msg::GoalStatus::SharedPtr msg)
  {
    switch(msg->status){ //The arrow operator (->) is used to access the member of the 'status'...
         data structure pointed to by a pointer. A pointer is used because the data cannot be ...
        directly copied and a pointer is what is really transferred. The 'status' data ...
        structure has 7 different states.
      case 0:
        RCLCPP_INFO(this->get_logger(), "GoalStatus: STATUS_UNKNOWN"); //The 'this' pointer is ...
            here used to retrieve the 'get_logger()' information
        break;
      case 1:
        RCLCPP_INFO(this->get_logger(), "GoalStatus: STATUS_ACCEPTED");
        break;
      case 2:
        RCLCPP_INFO(this->get_logger(), "GoalStatus: STATUS_EXECUTING");
        break;
      case 3:
        RCLCPP_INFO(this->get_logger(), "GoalStatus: STATUS_CANCELING");
        break;
```

```
    case 4:
      RCLCPP_INFO(this->get_logger(), "GoalStatus: STATUS_SUCCEEDED");
      RCLCPP_INFO(this->get_logger(), "Shutting down...");
      rclcpp::shutdown();
      break;
    case 5:
      RCLCPP_INFO(this->get_logger(), "GoalStatus: STATUS_CANCELED");
      break;
    case 6:
      RCLCPP_INFO(this->get_logger(), "GoalStatus: STATUS_ABORTED");
      break;
    default:
      break;
  }
}

unique_identifier_msgs::msg::UUID create_random_uuid() //The function creates UUID's
{
  boost::uuids::random_generator gen; boost::uuids::uuid u = gen();
  unique_identifier_msgs::msg::UUID uuid_msg;
  std::array<uint8_t, 16> uuid; std::copy(std::begin(u.data), std::end(u.data), uuid.begin())...
      ;
  uuid_msg.uuid = uuid;
  return uuid_msg;
}

void publish_message(unique_identifier_msgs::msg::UUID uuid_msg ) //The function sets all ...
    required parameters for the WholeBodyTrajectory structure and send them to 'publisher_'
{
  WholeBodyTrajectory trajectory_msg;
  trajectory_msg.append_trajectory = false; //If set to false, the existing trajectory is ...
      cancelled and this trajectory is immediatly executed.
  trajectory_msg.interpolation_mode.value = TrajectoryInterpolation::MINIMUM_JERK_CONSTRAINED...
      ; //Specifies how the trajectory is interpolated from the previous objective
  trajectory_msg.trajectory_id = uuid_msg;

  if (downward==true)
  {
      trajectory_msg.trajectory_points.push_back(targetall_(5));
  }

  if (relax==true)
  {
      trajectory_msg.trajectory_points.push_back(relax_right_arm_(5));
  }

  if (firm==true)
  {
      trajectory_msg.trajectory_points.push_back(stiff_right_arm_(5));
  }

  RCLCPP_INFO(this->get_logger(), "Sending trajectory, listening for ...
      whole_body_trajectory_status...");
  publisher_->publish(trajectory_msg); //The trajectory message is published
}


JointSpaceCommand generate_joint_space_command(int32_t joint_id, double q_des, double qd_des ...
    = 0.0, double qdd_des = 0.0) //The function takes in the required parameters and returns ...
    the message used for the taskspace trajectory point in the 'targetall_' function
{
  JointSpaceCommand ret_msg;
  JointName name;
  name.joint_id = joint_id;
  ret_msg.joint = name;
  ret_msg.q_desired = q_des;
  ret_msg.qd_desired = qd_des;
```

```
  ret_msg.qdd_desired = qdd_des;
  ret_msg.use_default_gains = true;
  return ret_msg;
}

JointSpaceCommand add_stiffness(int32_t joint_id) //The function takes in the name of the ...
    desired joint and sets the use_default_gains to true
{
  JointSpaceCommand ret_msg;
  JointName name;
  name.joint_id = joint_id;
  ret_msg.joint = name;
  ret_msg.use_default_gains = true;
  return ret_msg;
}

JointSpaceCommand remove_stiffness(int32_t joint_id) //The function takes in the name of the ...
    desired joint and sets the use_default_gains to false, enabling the stiffness and damping...
     adjustment
{
  JointSpaceCommand ret_msg;
  JointName name;
  name.joint_id = joint_id;
  ret_msg.joint = name;
  ret_msg.use_default_gains = false; //The default gains must be set to false in order to set...
      the stiffness and damping of the joints
  ret_msg.stiffness = 0.0; //((rad/s^2)/rad) Stiffness = 0 puts the joints in zero gravity ...
      and available to be pushed by external force
  ret_msg.damping = 1.0; // ((rad/s^2)/rad) Desired damping of the PD controller for the ...
      joint
  ret_msg.motorDampingScale = 1.0; //A value between 0.0 and 1.0 which gives the damping to ...
      the motor for the joint.
  return ret_msg;
}

WholeBodyTrajectoryPoint targetall_(int32_t t) //The function takes in the execution time and...
     assembles the name of the desired joint along with the zero position
{
  WholeBodyTrajectoryPoint ret_msg;

  builtin_interfaces::msg::Duration duration;
  duration.sec = t;
  ret_msg.time_from_start = duration;

  ret_msg.joint_space_commands.push_back(generate_joint_space_command(JointName::HIP_YAW, 0))...
      ;
  ret_msg.joint_space_commands.push_back(generate_joint_space_command(JointName::HIP_ROLL, 0)...
      );
  ret_msg.joint_space_commands.push_back(generate_joint_space_command(JointName::HIP_PITCH, ...
      0));
  ret_msg.joint_space_commands.push_back(generate_joint_space_command(JointName::KNEE_PITCH, ...
      0));
  ret_msg.joint_space_commands.push_back(generate_joint_space_command(JointName::ANKLE_ROLL, ...
      0));
  ret_msg.joint_space_commands.push_back(generate_joint_space_command(JointName::ANKLE_PITCH,...
       0));
  ret_msg.joint_space_commands.push_back(generate_joint_space_command(JointName::...
      LEFT_SHOULDER_PITCH, 0));
  ret_msg.joint_space_commands.push_back(generate_joint_space_command(JointName::...
      LEFT_SHOULDER_ROLL, 0));
  ret_msg.joint_space_commands.push_back(generate_joint_space_command(JointName::...
      LEFT_SHOULDER_YAW, 0));
  ret_msg.joint_space_commands.push_back(generate_joint_space_command(JointName::...
      LEFT_ELBOW_PITCH, 0));
  ret_msg.joint_space_commands.push_back(generate_joint_space_command(JointName::...
      LEFT_ELBOW_YAW, 0));
  ret_msg.joint_space_commands.push_back(generate_joint_space_command(JointName::...
```

```
        LEFT_WRIST_PITCH, 0));
    ret_msg.joint_space_commands.push_back(generate_joint_space_command(JointName::...
        LEFT_WRIST_ROLL, 0));
    ret_msg.joint_space_commands.push_back(generate_joint_space_command(JointName::...
        RIGHT_SHOULDER_PITCH, 0));
    ret_msg.joint_space_commands.push_back(generate_joint_space_command(JointName::...
        RIGHT_SHOULDER_ROLL, 0));
    ret_msg.joint_space_commands.push_back(generate_joint_space_command(JointName::...
        RIGHT_SHOULDER_YAW, 0));
    ret_msg.joint_space_commands.push_back(generate_joint_space_command(JointName::...
        RIGHT_ELBOW_PITCH, 0));
    ret_msg.joint_space_commands.push_back(generate_joint_space_command(JointName::...
        RIGHT_ELBOW_YAW, 0));
    ret_msg.joint_space_commands.push_back(generate_joint_space_command(JointName::...
        RIGHT_WRIST_PITCH, 0));
    ret_msg.joint_space_commands.push_back(generate_joint_space_command(JointName::...
        RIGHT_WRIST_ROLL, 0));
    ret_msg.joint_space_commands.push_back(generate_joint_space_command(JointName::NECK_PITCH, ...
        0));

    return ret_msg;
  }

  WholeBodyTrajectoryPoint stiff_right_arm_(int32_t t) //The function takes in the execution ...
      time and sends the name of the desired joint that will have its stiffness and damping ...
      adjusted by means of the 'generate_joint_space_command' function
  {
    WholeBodyTrajectoryPoint ret_msg;

    builtin_interfaces::msg::Duration duration;
    duration.sec = t;
    ret_msg.time_from_start = duration;

    ret_msg.joint_space_commands.push_back(add_stiffness(JointName::RIGHT_SHOULDER_PITCH));
    ret_msg.joint_space_commands.push_back(add_stiffness(JointName::RIGHT_SHOULDER_ROLL));
    ret_msg.joint_space_commands.push_back(add_stiffness(JointName::RIGHT_SHOULDER_YAW));
    ret_msg.joint_space_commands.push_back(add_stiffness(JointName::RIGHT_ELBOW_PITCH));
    ret_msg.joint_space_commands.push_back(add_stiffness(JointName::RIGHT_ELBOW_YAW));
    ret_msg.joint_space_commands.push_back(add_stiffness(JointName::RIGHT_WRIST_PITCH));
    ret_msg.joint_space_commands.push_back(add_stiffness(JointName::RIGHT_WRIST_ROLL));

    return ret_msg;
  }

  WholeBodyTrajectoryPoint relax_right_arm_(int32_t t) //The function takes in the execution ...
      time and sends the name of the desired joint that will be set back to normal ...
      use_default_gains by means of the 'generate_joint_space_command' function
  {
    WholeBodyTrajectoryPoint ret_msg;

    builtin_interfaces::msg::Duration duration;
    duration.sec = t;
    ret_msg.time_from_start = duration;

    ret_msg.joint_space_commands.push_back(remove_stiffness(JointName::RIGHT_SHOULDER_PITCH));
    ret_msg.joint_space_commands.push_back(remove_stiffness(JointName::RIGHT_SHOULDER_ROLL));
    ret_msg.joint_space_commands.push_back(remove_stiffness(JointName::RIGHT_SHOULDER_YAW));
    ret_msg.joint_space_commands.push_back(remove_stiffness(JointName::RIGHT_ELBOW_PITCH));
    ret_msg.joint_space_commands.push_back(remove_stiffness(JointName::RIGHT_ELBOW_YAW));
    ret_msg.joint_space_commands.push_back(remove_stiffness(JointName::RIGHT_WRIST_PITCH));
    ret_msg.joint_space_commands.push_back(remove_stiffness(JointName::RIGHT_WRIST_ROLL));

    return ret_msg;
  }

  rclcpp::Publisher<WholeBodyTrajectory>::SharedPtr publisher_; //Declaration of the '...
      publisher_' publisher
```

```
  rclcpp::Subscription<action_msgs::msg::GoalStatus>::SharedPtr subscriber_; //Declaration of ...
      the 'subscription_' subscriber
  unique_identifier_msgs::msg::UUID uuid_msg_; //Declaration of the UUID generator
};


void wait_for_enter() //Function for pausing the program
{
    std::string wait;
    std::cout << "[Enter] to continue..." << std::endl;
    getline(std::cin, wait);
}

void menu_space() //This is used to make the menu better looking, assuming that 1000 lines will...
     be enough to clear the screen
{
    std::cout << std::string(1000, '\n');
}

void check_single(int number) //Print single values for joint manipulation
{
double checkvalue;

switch(number)
    {
      case 1 :
         checkvalue = storage[0]*(180/3.141592654);
         std::cout << "HIP_YAW is set to be moved " << checkvalue << " degrees"<< std::endl;
         break;
      case 2 :
         checkvalue = storage[1]*(180/3.141592654);
         std::cout << "HIP_ROLL is set to be moved " << checkvalue << " degrees"<< std::endl;
         break;
      case 3 :
         checkvalue = storage[2]*(180/3.141592654);
         std::cout << "HIP_PITCH is set to be moved " << checkvalue << " degrees"<< std::endl;
         break;
      case 4 :
         checkvalue = storage[3]*(180/3.141592654);
         std::cout << "KNEE_PITCH is set to be moved " << checkvalue << " degrees"<< std::endl;
         break;
      case 5 :
         checkvalue = storage[4]*(180/3.141592654);
         std::cout << "ANKLE_ROLL is set to be moved " << checkvalue << " degrees"<< std::endl;
         break;
      case 6 :
         checkvalue = storage[5]*(180/3.141592654);
         std::cout << "ANKLE_PITCH is set to be moved " << checkvalue << " degrees"<< std::endl;
         break;
      case 7 :
         checkvalue = storage[6]*(180/3.141592654);
         std::cout << "LEFT_SHOULDER_PITCH is set to be moved " << checkvalue << " degrees"<< ...
             std::endl;
         break;
      case 8 :
         checkvalue = storage[7]*(180/3.141592654);
         std::cout << "LEFT_SHOULDER_ROLL is set to be moved " << checkvalue << " degrees"<< std...
             ::endl;
         break;
      case 9 :
         checkvalue = storage[8]*(180/3.141592654);
         std::cout << "LEFT_SHOULDER_YAW is set to be moved " << checkvalue << " degrees"<< std...
             ::endl;
         break;
      case 10 :
         checkvalue = storage[9]*(180/3.141592654);
         std::cout << "LEFT_ELBOW_PITCH is set to be moved " << checkvalue << " degrees"<< std::...
```

```
                endl;
            break;
        case 11 :
            checkvalue = storage[10]*(180/3.141592654);
            std::cout << "LEFT_ELBOW_YAW is set to be moved " << checkvalue << " degrees"<< std::...
                endl;
            break;
        case 12 :
            checkvalue = storage[11]*(180/3.141592654);
            std::cout << "LEFT_WRIST_PITCH is set to be moved " << checkvalue << " degrees"<< std::...
                endl;
            break;
        case 13 :
            checkvalue = storage[12]*(180/3.141592654);
            std::cout << "LEFT_WRIST_ROLL is set to be moved " << checkvalue << " degrees"<< std::...
                endl;
            break;
        case 14 :
            checkvalue = storage[13]*(180/3.141592654);
            std::cout << "RIGHT_SHOULDER_PITCH is set to be moved " << checkvalue << " degrees"<< ...
                std::endl;
            break;
        case 15 :
            checkvalue = storage[14]*(180/3.141592654);
            std::cout << "RIGHT_SHOULDER_ROLL is set to be moved " << checkvalue << " degrees"<< ...
                std::endl;
            break;
        case 16 :
            checkvalue = storage[15]*(180/3.141592654);
            std::cout << "RIGHT_SHOULDER_YAW is set to be moved " << checkvalue << " degrees"<< std...
                ::endl;
            break;
        case 17 :
            checkvalue = storage[16]*(180/3.141592654);
            std::cout << "RIGHT_ELBOW_PITCH is set to be moved " << checkvalue << " degrees"<< std...
                ::endl;
            break;
        case 18 :
            checkvalue = storage[17]*(180/3.141592654);
            std::cout << "RIGHT_ELBOW_YAW is set to be moved " << checkvalue << " degrees"<< std::...
                endl;
            break;
        case 19 :
            checkvalue = storage[18]*(180/3.141592654);
            std::cout << "RIGHT_WRIST_PITCH is set to be moved " << checkvalue << " degrees"<< std...
                ::endl;
            break;
        case 20 :
            checkvalue = storage[19]*(180/3.141592654);
            std::cout << "RIGHT_WRIST_ROLL is set to be moved " << checkvalue << " degrees"<< std::...
                endl;
            break;
        case 21 :
            checkvalue = storage[20]*(180/3.141592654);
            std::cout << "NECK_PITCH is set to be moved " << checkvalue << " degrees"<< std::endl;
            break;
        default :
            std::cout << "Invalid input" << std::endl;
    }
}

void check_all() //Print all stored values for joint manipulation
{
double checkvalue;

    checkvalue = storage[0]*(180/3.141592654);
    std::cout << "HIP_YAW is set to be moved " << checkvalue << " degrees"<< std::endl;
```

```
    checkvalue = storage[1]*(180/3.141592654);
    std::cout << "HIP_ROLL is set to be moved " << checkvalue << " degrees"<< std::endl;
    checkvalue = storage[2]*(180/3.141592654);
    std::cout << "HIP_PITCH is set to be moved " << checkvalue << " degrees"<< std::endl;
    checkvalue = storage[3]*(180/3.141592654);
    std::cout << "KNEE_PITCH is set to be moved " << checkvalue << " degrees"<< std::endl;
    checkvalue = storage[4]*(180/3.141592654);
    std::cout << "ANKLE_ROLL is set to be moved " << checkvalue << " degrees"<< std::endl;
    checkvalue = storage[5]*(180/3.141592654);
    std::cout << "ANKLE_PITCH is set to be moved " << checkvalue << " degrees"<< std::endl;
    checkvalue = storage[6]*(180/3.141592654);
    std::cout << "LEFT_SHOULDER_PITCH is set to be moved " << checkvalue << " degrees"<< std::...
        endl;
    checkvalue = storage[7]*(180/3.141592654);
    std::cout << "LEFT_SHOULDER_ROLL is set to be moved " << checkvalue << " degrees"<< std::...
        endl;
    checkvalue = storage[8]*(180/3.141592654);
    std::cout << "LEFT_SHOULDER_YAW is set to be moved " << checkvalue << " degrees"<< std::...
        endl;
    checkvalue = storage[9]*(180/3.141592654);
    std::cout << "LEFT_ELBOW_PITCH is set to be moved " << checkvalue << " degrees"<< std::endl...
        ;
    checkvalue = storage[10]*(180/3.141592654);
    std::cout << "LEFT_ELBOW_YAW is set to be moved " << checkvalue << " degrees"<< std::endl;
    checkvalue = storage[11]*(180/3.141592654);
    std::cout << "LEFT_WRIST_PITCH is set to be moved " << checkvalue << " degrees"<< std::endl...
        ;
    checkvalue = storage[12]*(180/3.141592654);
    std::cout << "LEFT_WRIST_ROLL is set to be moved " << checkvalue << " degrees"<< std::endl;
    checkvalue = storage[13]*(180/3.141592654);
    std::cout << "RIGHT_SHOULDER_PITCH is set to be moved " << checkvalue << " degrees"<< std::...
        endl;
    checkvalue = storage[14]*(180/3.141592654);
    std::cout << "RIGHT_SHOULDER_ROLL is set to be moved " << checkvalue << " degrees"<< std::...
        endl;
    checkvalue = storage[15]*(180/3.141592654);
    std::cout << "RIGHT_SHOULDER_YAWis set to be moved " << checkvalue << " degrees"<< std::...
        endl;
    checkvalue = storage[16]*(180/3.141592654);
    std::cout << "RIGHT_ELBOW_PITCH is set to be moved " << checkvalue << " degrees"<< std::...
        endl;
    checkvalue = storage[17]*(180/3.141592654);
    std::cout << "RIGHT_ELBOW_YAW is set to be moved " << checkvalue << " degrees"<< std::endl;
    checkvalue = storage[18]*(180/3.141592654);
    std::cout << "RIGHT_WRIST_PITCH is set to be moved " << checkvalue << " degrees"<< std::...
        endl;
    checkvalue = storage[19]*(180/3.141592654);
    std::cout << "RIGHT_WRIST_ROLL is set to be moved " << checkvalue << " degrees"<< std::endl...
        ;
    checkvalue = storage[20]*(180/3.141592654);
    std::cout << "NECK_PITCH is set to be moved " << checkvalue << " degrees"<< std::endl;
}

void check(int sel) //This function gathers all printing of joint manipulation values into one ...
    menu
{
int joint_number;
bool done = false;

do {
     switch(sel)
     {
       case 1 :
               menu_space();
               check_all();
               std::cin.get(); //This makes the program wait for input from user
               wait_for_enter();
```

```
                done = true;
            break;
        case 2 :
                menu_space();
                std::cout << "Desired joint number (1-21) [+Enter]: " << std::endl;
                std::cout << "1: HIP_YAW" << std::endl;
                std::cout << "2: HIP_ROLL" << std::endl;
                std::cout << "3: HIP_PITCH" << std::endl;
                std::cout << "4: KNEE_PITCH" << std::endl;
                std::cout << "5: ANKLE_ROLL" << std::endl;
                std::cout << "6: ANKLE_PITCH" << std::endl;
                std::cout << "7: LEFT_SHOULDER_PITCH" << std::endl;
                std::cout << "8: LEFT_SHOULDER_ROLL" << std::endl;
                std::cout << "9: LEFT_SHOULDER_YAW" << std::endl;
                std::cout << "10: LEFT_ELBOW_PITCH" << std::endl;
                std::cout << "11: LEFT_ELBOW_YAW" << std::endl;
                std::cout << "12: LEFT_WRIST_PITCH" << std::endl;
                std::cout << "13: LEFT_WRIST_ROLL" << std::endl;
                std::cout << "14: RIGHT_SHOULDER_PITCH" << std::endl;
                std::cout << "15: RIGHT_SHOULDER_ROLL" << std::endl;
                std::cout << "16: RIGHT_SHOULDER_YAW" << std::endl;
                std::cout << "17: RIGHT_ELBOW_PITCH" << std::endl;
                std::cout << "18: RIGHT_ELBOW_YAW" << std::endl;
                std::cout << "19: RIGHT_WRIST_PITCH" << std::endl;
                std::cout << "20: RIGHT_WRIST_ROLL" << std::endl;
                std::cout << "21: NECK_PITCH" << std::endl;
                std::cin >> joint_number;
                if (joint_number > 0 && joint_number < 22)
                    {check_single(joint_number);
                    std::cin.get(); //This makes the program wait for input from user
                    wait_for_enter();
                    done = true;}
                else
                    {std::cout << "Invalid input" << std::endl;}
            break;
        case 3 :
            done = true;
            break;
        default :
               std::cout << "Invalid input" << std::endl;
      }
} while(!done);
}


void store_single(int number, double angle) //This function stores single inputs for joint manipulation
{
switch(number)
    {
      case 1 :
         std::cout << "Storing value for HIP_YAW: " << angle << " degrees" << std::endl;
         storage[0] = angle*(3.141592654/180);
         break;
      case 2 :
         std::cout << "Storing value for HIP_ROLL: " << angle << " degrees" << std::endl;
         storage[1] = angle*(3.141592654/180);
         break;
      case 3 :
         std::cout << "Storing value for HIP_PITCH: " << angle << " degrees" << std::endl;
         storage[2] = angle*(3.141592654/180);
         break;
      case 4 :
         std::cout << "Storing value for KNEE_PITCH: " << angle << " degrees" << std::endl;
         storage[3] = angle*(3.141592654/180);
         break;
      case 5 :
```

```
            std::cout << "Storing value for ANKLE_ROLL: " << angle << " degrees" << std::endl;
            storage[4] = angle*(3.141592654/180);
            break;
        case 6 :
            std::cout << "Storing value for ANKLE_PITCH: " << angle << " degrees" << std::endl;
            storage[5] = angle*(3.141592654/180);
            break;
        case 7 :
            std::cout << "Storing value for LEFT_SHOULDER_PITCH: " << angle << " degrees" << std::...
                endl;
            storage[6] = angle*(3.141592654/180);
            break;
        case 8 :
            std::cout << "Storing value for LEFT_SHOULDER_ROLL: " << angle << " degrees" << std::...
                endl;
            storage[7] = angle*(3.141592654/180);
            break;
        case 9 :
            std::cout << "Storing value for LEFT_SHOULDER_YAW: " << angle << " degrees" << std::...
                endl;
            storage[8] = angle*(3.141592654/180);
            break;
        case 10 :
            std::cout << "Storing value for LEFT_ELBOW_PITCH: " << angle << " degrees" << std::endl...
                ;
            storage[9] = angle*(3.141592654/180);
            break;
        case 11 :
            std::cout << "Storing value for LEFT_ELBOW_YAW: " << angle << " degrees" << std::endl;
            storage[10] = angle*(3.141592654/180);
            break;
        case 12 :
            std::cout << "Storing value for LEFT_WRIST_PITCH: " << angle << " degrees" << std::endl...
                ;
            storage[11] = angle*(3.141592654/180);
            break;
        case 13 :
            std::cout << "Storing value for LEFT_WRIST_ROLL: " << angle << " degrees" << std::endl;
            storage[12] = angle*(3.141592654/180);
            break;
        case 14 :
            std::cout << "Storing value for RIGHT_SHOULDER_PITCH: " << angle << " degrees" << std::...
                endl;
            storage[13] = angle*(3.141592654/180);
            break;
        case 15 :
            std::cout << "Storing value for RIGHT_SHOULDER_ROLL: " << angle << " degrees" << std::...
                endl;
            storage[14] = angle*(3.141592654/180);
            break;
        case 16 :
            std::cout << "Storing value for RIGHT_SHOULDER_YAW: " << angle << " degrees" << std::...
                endl;
            storage[15] = angle*(3.141592654/180);
            break;
        case 17 :
            std::cout << "Storing value for RIGHT_ELBOW_PITCH: " << angle << " degrees" << std::...
                endl;
            storage[16] = angle*(3.141592654/180);
            break;
        case 18 :
            std::cout << "Storing value for RIGHT_ELBOW_YAW: " << angle << " degrees" << std::endl;
            storage[17] = angle*(3.141592654/180);
            break;
        case 19 :
            std::cout << "Storing value for RIGHT_WRIST_PITCH: " << angle << " degrees" << std::...
                endl;
```

```
        storage[18] = angle*(3.141592654/180);
        break;
      case 20 :
        std::cout << "Storing value for RIGHT_WRIST_ROLL: " << angle << " degrees" << std::endl...
            ;
        storage[19] = angle*(3.141592654/180);
        break;
      case 21 :
        std::cout << "Storing value for NECK_PITCH: " << angle << " degrees" << std::endl;
        storage[20] = angle*(3.141592654/180);
        break;
      default :
        std::cout << "Invalid input" << std::endl;
    }
}

void store_all() //This function stores every joint value intended to be executed
{
double angle;

    std::cout << "Desired angle in degrees for HIP_YAW (-60 to 60) [+Enter]: " << std::endl;
    std::cin >> angle;
    if (angle >= -60 && angle <= 60)
        {std::cout << "Storing value for HIP_YAW: " << angle << " degrees" << std::endl;
        storage[0] = angle*(3.141592654/180);}
    else
        {std::cout << "Invalid input for HIP_YAW" << std::endl;}
    std::cout << "Desired angle in degrees for HIP_ROLL (-30 to 30) [+Enter]: " << std::endl;
    std::cin >> angle;
    if (angle >= -30 && angle <= 30)
        {std::cout << "Storing value for HIP_ROLL: " << angle << " degrees" << std::endl;
        storage[1] = angle*(3.141592654/180);}
    else
        {std::cout << "Invalid input for HIP_ROLL" << std::endl;}
    std::cout << "Desired angle in degrees for HIP_PITCH (-89 to 10) [+Enter]: " << std::endl;
    std::cin >> angle;
    if (angle >= -89 && angle <= 10)
        {std::cout << "Storing value for HIP_PITCH: " << angle << " degrees" << std::endl;
        storage[2] = angle*(3.141592654/180);}
    else
        {std::cout << "Invalid input for HIP_PITCH" << std::endl;}
    std::cout << "Desired angle in degrees for KNEE_PITCH (0 to 147) [+Enter]: " << std::endl;
    std::cin >> angle;
    if (angle >= 0 && angle <= 147)
        {std::cout << "Storing value for KNEE_PITCH: " << angle << " degrees" << std::endl;
        storage[3] = angle*(3.141592654/180);}
    else
        {std::cout << "Invalid input for KNEE_PITCH" << std::endl;}
    std::cout << "Desired angle in degrees for ANKLE_ROLL (-30 to 30) [+Enter]: " << std::endl;
    std::cin >> angle;
    if (angle >= -30 && angle <= 30)
        {std::cout << "Storing value for ANKLE_ROLL " << angle << " degrees" << std::endl;
        storage[4] = angle*(3.141592654/180);}
    else
        {std::cout << "Invalid input for ANKLE_ROLL" << std::endl;}
    std::cout << "Desired angle in degrees for ANKLE_PITCH (-89 to 10) [+Enter]: " << std::endl...
        ;
    std::cin >> angle;
    if (angle >= -89 && angle <= 10)
        {std::cout << "Storing value for ANKLE_PITCH: " << angle << " degrees" << std::endl;
        storage[5] = angle*(3.141592654/180);}
    else
        {std::cout << "Invalid input for ANKLE_PITCH" << std::endl;}
    std::cout << "Desired angle in degrees for LEFT_SHOULDER_PITCH (-138 to 56) [+Enter]: " << ...
        std::endl;
    std::cin >> angle;
    if (angle >= -138 && angle <= 56)
```

```
        {std::cout << "Storing value for LEFT_SHOULDER_PITCH: " << angle << " degrees" << std::...
            endl;
        storage[6] = angle*(3.141592654/180);}
    else
        {std::cout << "Invalid input for LEFT_SHOULDER_PITCH" << std::endl;}
    std::cout << "Desired angle in degrees for LEFT_SHOULDER_ROLL (-5 to 119) [+Enter]: " << ...
        std::endl;
    std::cin >> angle;
    if (angle >= -5 && angle <= 119)
        {std::cout << "Storing value for LEFT_SHOULDER_ROLL: " << angle << " degrees" << std::...
            endl;
        storage[7] = angle*(3.141592654/180);}
    else
        {std::cout << "Invalid input for LEFT_SHOULDER_ROLL" << std::endl;}
    std::cout << "Desired angle in degrees for LEFT_SHOULDER_YAW (-83 to 31) [+Enter]: " << std...
        ::endl;
    std::cin >> angle;
    if (angle >= -83 && angle <= 31)
        {std::cout << "Storing value for LEFT_SHOULDER_YAW: " << angle << " degrees" << std::...
            endl;
        storage[8] = angle*(3.141592654/180);}
    else
        {std::cout << "Invalid input for LEFT_SHOULDER_YAW" << std::endl;}
    std::cout << "Desired angle in degrees for LEFT_ELBOW_PITCH (-124 to 0) [+Enter]: " << std...
        ::endl;
    std::cin >> angle;
    if (angle >= -124 && angle <= 0)
        {std::cout << "Storing value for LEFT_ELBOW_PITCH: " << angle << " degrees" << std::endl...
            ;
        storage[9] = angle*(3.141592654/180);}
    else
        {std::cout << "Invalid input for LEFT_ELBOW_PITCH" << std::endl;}
    std::cout << "Desired angle in degrees for LEFT_ELBOW_YAW (-90 to 45) [+Enter]: " << std::...
        endl;
    std::cin >> angle;
    if (angle >= -90 && angle <= 45)
        {std::cout << "Storing value for LEFT_ELBOW_YAW: " << angle << " degrees" << std::endl;
        storage[10] = angle*(3.141592654/180);}
    else
        {std::cout << "Invalid input for LEFT_ELBOW_YAW" << std::endl;}
    std::cout << "Desired angle in degrees for LEFT_WRIST_PITCH (-44 to 44) [+Enter]: " << std...
        ::endl;
    std::cin >> angle;
    if (angle >= -44 && angle <= 44)
        {std::cout << "Storing value for LEFT_WRIST_PITCH: " << angle << " degrees" << std::endl...
            ;
        storage[11] = angle*(3.141592654/180);}
    else
        {std::cout << "Invalid input for LEFT_WRIST_PITCH" << std::endl;}
    std::cout << "Desired angle in degrees for LEFT_WRIST_ROLL (-90 to 30) [+Enter]: " << std::...
        endl;
    std::cin >> angle;
    if (angle >= -90 && angle <= 30)
        {std::cout << "Storing value for LEFT_WRIST_ROLL: " << angle << " degrees" << std::endl;
        storage[12] = angle*(3.141592654/180);}
    else
        {std::cout << "Invalid input for LEFT_WRIST_ROLL" << std::endl;}
    std::cout << "Desired angle in degrees for RIGHT_SHOULDER_PITCH (-138 to 56) [+Enter]: " <<...
        std::endl;
    std::cin >> angle;
    if (angle >= -138 && angle <= 56)
        {std::cout << "Storing value for RIGHT_SHOULDER_PITCH: " << angle << " degrees" << std::...
            endl;
        storage[13] = angle*(3.141592654/180);}
    else
        {std::cout << "Invalid input for RIGHT_SHOULDER_PITCH" << std::endl;}
    std::cout << "Desired angle in degrees for RIGHT_SHOULDER_ROLL (-119 to 5) [+Enter]: " << ...
```

```
        std::endl;
    std::cin >> angle;
    if (angle >= -119 && angle <= 5)
        {std::cout << "Storing value for RIGHT_SHOULDER_ROLL: " << angle << " degrees" << std::...
            endl;
        storage[14] = angle*(3.141592654/180);}
    else
        {std::cout << "Invalid input for RIGHT_SHOULDER_ROLL" << std::endl;}
    std::cout << "Desired angle in degrees for RIGHT_SHOULDER_YAW (-31 to 83) [+Enter]: " << ...
        std::endl;
    std::cin >> angle;
    if (angle >= -31 && angle <= 83)
        {std::cout << "Storing value for RIGHT_SHOULDER_YAW: " << angle << " degrees" << std::...
            endl;
        storage[15] = angle*(3.141592654/180);}
    else
        {std::cout << "Invalid input for RIGHT_SHOULDER_YAW" << std::endl;}
    std::cout << "Desired angle in degrees for RIGHT_ELBOW_PITCH (-124 to 0) [+Enter]: " << std...
        ::endl;
    std::cin >> angle;
    if (angle >= -124 && angle <= 0)
        {std::cout << "Storing value for RIGHT_ELBOW_PITCH: " << angle << " degrees" << std::...
            endl;
        storage[16] = angle*(3.141592654/180);}
    else
        {std::cout << "Invalid input for RIGHT_ELBOW_PITCH" << std::endl;}
    std::cout << "Desired angle in degrees for RIGHT_ELBOW_YAW (-44 to 90) [+Enter]: " << std::...
        endl;
    std::cin >> angle;
    if (angle >= -44 && angle <= 90)
        {std::cout << "Storing value for RIGHT_ELBOW_YAW: " << angle << " degrees" << std::endl;
        storage[17] = angle*(3.141592654/180);}
    else
        {std::cout << "Invalid input for RIGHT_ELBOW_YAW" << std::endl;}
    std::cout << "Desired angle in degrees for RIGHT_WRIST_PITCH (-44 to 44) [+Enter]: " << std...
        ::endl;
    std::cin >> angle;
    if (angle >= -44 && angle <= 44)
        {std::cout << "Storing value for RIGHT_WRIST_PITCH: " << angle << " degrees" << std::...
            endl;
        storage[18] = angle*(3.141592654/180);}
    else
        {std::cout << "Invalid input for RIGHT_WRIST_PITCH" << std::endl;}
    std::cout << "Desired angle in degrees for RIGHT_WRIST_ROLL (-30 to 90) [+Enter]: " << std...
        ::endl;
    std::cin >> angle;
    if (angle >= -30 && angle <= 90)
        {std::cout << "Storing value for RIGHT_WRIST_ROLL: " << angle << " degrees" << std::endl...
            ;
        storage[19] = angle*(3.141592654/180);}
    else
        {std::cout << "Invalid input for RIGHT_WRIST_ROLL" << std::endl;}
    std::cout << "Desired angle in degrees for NECK_PITCH (-19 to 28) [+Enter]: " << std::endl;
    std::cin >> angle;
    if (angle >= -19 && angle <= 28)
        {std::cout << "Storing value for NECK_PITCH: " << angle << " degrees" << std::endl;
        storage[20] = angle*(3.141592654/180);}
    else
        {std::cout << "Invalid input for NECK_PITCH" << std::endl;}
}

void store(int sel) //This function gathers all storing of joint manipulation values into one ...
    menu
{
double joint_angle;
int joint_number;
bool done = false;
```

```
do {
    switch(sel)
    {
        case 1 :
                menu_space();
                store_all();
                std::cin.get(); //This makes the program wait for input from user
                wait_for_enter();
                done = true;
            break;
        case 2 :
                menu_space();
                std::cout << "Desired joint number (1-21) [+Enter]: " << std::endl;
                std::cout << "1: HIP_YAW (Range: (-60 to 60))" << std::endl;
                std::cout << "2: HIP_ROLL (Range: (-30 to 30))" << std::endl;
                std::cout << "3: HIP_PITCH (Range: (-89 to 10))" << std::endl;
                std::cout << "4: KNEE_PITCH (Range: ( 0 to 147))" << std::endl;
                std::cout << "5: ANKLE_ROLL (Range: (-30 to 30))" << std::endl;
                std::cout << "6: ANKLE_PITCH (Range: (-89 to 10))" << std::endl;
                std::cout << "7: LEFT_SHOULDER_PITCH (Range: (-138 to 56))" << std::endl;
                std::cout << "8: LEFT_SHOULDER_ROLL (Range: (-5 to 119))" << std::endl;
                std::cout << "9: LEFT_SHOULDER_YAW (Range: (-83 to 31))" << std::endl;
                std::cout << "10: LEFT_ELBOW_PITCH (Range: (-124 to 0))" << std::endl;
                std::cout << "11: LEFT_ELBOW_YAW (Range: (-90 to 45))" << std::endl;
                std::cout << "12: LEFT_WRIST_PITCH (Range: (-44 to 44))" << std::endl;
                std::cout << "13: LEFT_WRIST_ROLL (Range: (-90 to 30))" << std::endl;
                std::cout << "14: RIGHT_SHOULDER_PITCH (Range: (-138 to 56))" << std::endl;
                std::cout << "15: RIGHT_SHOULDER_ROLL (Range: (-119 to 5))" << std::endl;
                std::cout << "16: RIGHT_SHOULDER_YAW (Range: (-31 to 83))" << std::endl;
                std::cout << "17: RIGHT_ELBOW_PITCH (Range: (-124 to 0))" << std::endl;
                std::cout << "18: RIGHT_ELBOW_YAW (Range: (-44 to 90))" << std::endl;
                std::cout << "19: RIGHT_WRIST_PITCH (Range: (-44 to 44))" << std::endl;
                std::cout << "20: RIGHT_WRIST_ROLL (Range: (-30 to 90))" << std::endl;
                std::cout << "21: NECK_PITCH (Range: (-19 to 28))" << std::endl;
                std::cin >> joint_number;
                if (joint_number > 0 && joint_number < 22)
                    {std::cout << "Desired angle in degrees [+Enter] : " << std::endl;
                    std::cin >> joint_angle;
                    store_single(joint_number, joint_angle);
                    std::cin.get(); //This makes the program wait for input from user
                    wait_for_enter();
                    done = true;}
                else
                    {std::cout << "Invalid input" << std::endl;}

            break;
        case 3 ://Setting the joint positions to a default value (elbows in, hands out)
            std::cout << "Setting joint angles to default pose" << std::endl;
            storage[0] = 0.0;
            storage[1] = 0.0;
            storage[2] = -0.1;
            storage[3] = 0.15;
            storage[4] = 0.0;
            storage[5] = -0.05;
            storage[6] = 0.3;
            storage[7] = 0.0;
            storage[8] = 0.0;
            storage[9] = -1.3;
            storage[10] = 0.0;
            storage[11] = 0.2;
            storage[12] = 0.0;
            storage[13] = 0.3;
            storage[14] = 0.0;
            storage[15] = 0.0;
            storage[16] = -1.3;
            storage[17] = 0.0;
```

```
            storage[18] = 0.2;
            storage[19] = 0.0;
            storage[20] = 0.0;
            done = true;
            break;
        case 4 : //Setting the arm ready for grasp and everything else to zero
            std::cout << "Setting joint angles to position for grasp" << std::endl;
            storage[0] = 0.0;
            storage[1] = 0.0;
            storage[2] = 0.0;
            storage[3] = 0.0;
            storage[4] = 0.0;
            storage[5] = 0.0;
            storage[6] = 0.0;
            storage[7] = 0.0;
            storage[8] = 0.0;
            storage[9] = 0.0;
            storage[10] = 0.0;
            storage[11] = 0.0;
            storage[12] = 0.0;
            storage[13] = 0.3;
            storage[14] = 0.0;
            storage[15] = 0.0;
            storage[16] = -1.3;
            storage[17] = 1.57;
            storage[18] = 0.0;
            storage[19] = 0.0;
            storage[20] = 0.0;
            break;
        case 5 :
            done = true;
            break;
        default :
            std::cout << "Invalid input" << std::endl;
     }
} while(!done);
}



void store_grasp() //The function receives grasp manipulation input from user and stores them
{
    bool done = false;
    int choice;
    int choice_2;
    do
   {
    menu_space();
    std::cout <<"\n 1 [+Enter] to select hand(s) (default: right)\n 2 [+Enter] Set closure ...
        ratio (default: 0.75)\n 3 [+Enter] to set the closure speed (optional)\n 4 [+Enter] to ...
        set the closure force (optional)\n 5 [+Enter] to go back\n" << std::endl;
    std::cin >> choice;
    switch(choice)
    { case 1:
          std::cout << "1. Right hand\n" << "2. Left hand\n" << "3. Both hands\n" << std::endl;
          std::cin >> choice_2;
          if (choice_2 == 1)
          {
              right_hand = true;
              left_hand = false;
              both_hands = false;
              std::cout << "Right hand was selected" << std::endl;
          }
          if (choice_2 == 2)
          {
              right_hand = false;
              left_hand = true;
```

```
            both_hands = false;
            std::cout << "Left hand was selected" << std::endl;
         }
         if (choice_2 == 3)
         {
            right_hand = false;
            left_hand = false;
            both_hands = true;
            std::cout << "Both hands were selected" << std::endl;
         }
         break;
       case 2:
         std::cout << "Set the closure ratio (Open: 0.0, Fully closed: 1.0) \n" << std::endl;
         std::cin >> closure_ratio;
         break;

       case 3:
         std::cout << "Set the closure velocity (Zero: 0.0, Max: 1.0) \n" << std::endl;
         std::cin >> closure_velocity;
         break;

       case 4:
         std::cout << "Set the closure force (Zero: 0.0, Max: 1.0) \n" << std::endl;
         std::cin >> closure_force;
         break;

       case 5:
         done = true;
         break;

       default:
         break;
    }
    }
    while(!done);
}


void main_menu_header()
{
  std::cout << "****************************************************" << std::endl;
  std::cout << "* Menu for interacting with the EVEr3 Robot *" << std::endl;
  std::cout << "****************************************************" << std::endl;
}


void print_grasp() //The function prints the grasp parameters
{
    double checkvalue;
    menu_space();
    if(right_hand == true)
    {
       std::cout << "The right hand is selected\n" << std::endl;
    }
    if(left_hand == true)
    {
       std::cout << "The left hand is selected\n" << std::endl;
    }
    if(both_hands== true)
    {
       std::cout << "Both hands are selected\n" << std::endl;
    }
    checkvalue = closure_ratio;
    std::cout << "The closure ratio is set to " << checkvalue << "\n" << std::endl;
    checkvalue = closure_velocity;
    std::cout << "The closure velocity is set to " << checkvalue << "\n" << std::endl;
    checkvalue = closure_force;
```

```
    std::cout << "The closure force is set to " << checkvalue << "\n" << std::endl;
    std::cin.get(); //This makes the program wait for input from user
    wait_for_enter();
}


int main (int argc, char * argv[])
{
int sel,sel1,sel2,sel3,sel4; //Declaring all variables used in main()
bool exit = false;
bool done = false;
bool done1 = false;
bool done2 = false;
bool done3 = false;

do {
   menu_space();
   main_menu_header();
   std::cout << "\n 1 [+Enter] for joint manipulation\n 2 [+Enter] for grasp manipulation\n 3 ...
       [+Enter] to remove stiffness in the joints of the right arm\n 4 [+Enter] to exit the ...
       program\n" << std::endl;
   std::cin >> sel;
   switch(sel)
   {
   case 1 :
      do {
      menu_space();
      std::cout << "This is a program for joint manipulation of the EVE Robot\nDo you want to ...
          store, check or execute values?" << std::endl;
      std::cout << "\n 1 [+Enter] to store joint angles\n 2 [+Enter] to check stored joint ...
          angles\n 3 [+Enter] to execute stored joint angles\n 4 [+Enter] to go back to the ...
          main menu\n ";
      std::cin >> sel1;
      switch(sel1)
      {
      case 1 :
         menu_space();
         std::cout << "Do you want to store values for all 21 joints or select one in ...
             specific?" << std::endl;
         std::cout << "\n 1 [+Enter] to store values for all joints\n 2 [+Enter] to store a ...
             value for one joint\n 3 [+Enter] to set all joint angles to the default pose\n 4...
              [+Enter] to position the arm for grasp k\n 5 [+Enter] to go back\n ";
         std::cin >> sel1;
         store(sel1);
         break;
      case 2 :
         menu_space();
         std::cout << "Do you want to print stored values for all 21 joints or select one in ...
             specific?" << std::endl;
         std::cout << "\n 1 [+Enter] to print values for all joints\n 2 [+Enter] to print the...
              value for one joint\n 3 [+Enter] to go back\n ";
         std::cin >> sel1;
         check(sel1);
         break;
      case 3 :
         menu_space();
         check_all();
         std::cout << "\n 1 [+Enter] to execute these values\n 2 [+Enter] to go back\n";
         std::cin >> sel2;
         do {
            switch(sel2)
               {
               case 1 :
               //The 'Joint_menu_publisher' node is executed in these steps. 'rclcpp::init' ...
                   initializes ROS 2, and 'rclcpp::spin' starts processing data from the ...
                   node
                  rclcpp::init(argc, argv);
```

```
                    rclcpp::spin(std::make_shared<JointManipulator>());
                    rclcpp::shutdown();
                    std::cin.get(); //This makes the program wait for input from user
                    wait_for_enter();
                    done1 = true;
                break;
                case 2 :
                    done1 = true;
                break;
                default :
                    std::cout << "Invalid input" << std::endl;
                }
            } while(!done1);
        break;
    case 4 :
        done = true;
    break;
    default :
        std::cout << "Invalid input" << std::endl;
    }

    } while(!done);
    break;
case 2:
 do
    {
        menu_space();
        std::cout << "This is a menu for grasp manipulation of the EVE Robot\nDo you want to...
             store, check or execute values?" << std::endl;
        std::cout << "\n 1 [+Enter] to execute values\n 2 [+Enter] to print stored values\n ...
            3 [+Enter] store values\n 4 [+Enter] to go back to the main menu\n ";
        std::cin >> sel3;

        switch (sel3)
        {
        case 1:
        rclcpp::init(argc, argv); //The 'Grasp_publisher' node is executed in these steps. '...
            rclcpp::init' initializes ROS 2, and 'rclcpp::spin' starts processing data from ...
            the node
        rclcpp::spin(std::make_shared<Grasp_publisher>());
        rclcpp::shutdown();
        break;

        case 2:
        print_grasp();
        break;

        case 3:
        store_grasp();
        break;

        case 4:
        done2 = true;
        break;

        default:
        std::cout << "Invalid input" << std::endl;

        }
    }while(!done2);
    break;

    case 3:
    do
    {
        menu_space();
        std::cout << "This is a menu for altering the stiffness in the right arm pitch of ...
```

```
                EVE\nDo you want to remove or add stiffness, or set the arm downward?" << std::...
                endl;
            std::cout << "\n 1 [+Enter] to remove the stiffness on the pitch of right arm\n 2 [+...
                Enter] to add stiffness on the pitch of right arm\n 3 [+Enter] to execute zero ...
                angle on all joints to set the arm downward\n 4 [+Enter] to go back to the main ...
                menu\n ";
            std::cin >> sel4;

            switch (sel4)
            {
            case 1:
            relax = true;
            firm = false;
            downward = false;
            rclcpp::init(argc, argv); //'rclcpp::init' initializes ROS 2, and 'rclcpp::spin' ...
                starts processing data from the Relaxing_publisher node
            rclcpp::spin(std::make_shared<Relaxing_publisher>());
            rclcpp::shutdown();
            break;

            case 2:
            relax = false;
            firm = true;
            downward = false;
            rclcpp::init(argc, argv);
            rclcpp::spin(std::make_shared<Relaxing_publisher>());
            rclcpp::shutdown();
            break;

            case 3:
            relax = false;
            firm = false;
            downward = true;
            rclcpp::init(argc, argv);
            rclcpp::spin(std::make_shared<Relaxing_publisher>());
            rclcpp::shutdown();
            break;

            case 4:
            done3 = true;
            break;

            default:
            std::cout << "Invalid input" << std::endl;

            }
        }while(!done3);
    break;

    case 4 :
    exit = true;
    break;

    default :
    std::cout << "Invalid input" << std::endl;
    }

} while(!exit);
return 0;
}
```

appendices/eve_menu/eve.tex

A.4 Joint Read Menu Package

A.4.1 README.md

```
##Test menu for storing joint values for Rehabilitation Project at UiA

* Author: Lars Bleie Andersen <larsa09@uia.no>

## Instructions
- Open the joint_read.cpp file and edit the file paths to where the external storage file ...
    should be created

- Copy the joint_read_menu folder into eve_ws/src
- Navigate back to eve_ws
- Run colcon build --packages-select joint_read_menu
- Run the program with:
  ros2 run joint_read_menu joint_read
```

appendices/joint_read_menu/README.tex

A.4.2 package.xml

```xml
<?xml version="1.0"?>
<?xml-model href="http://download.ros.org/schema/package_format3.xsd" schematypens="http://www....
    w3.org/2001/XMLSchema"?>
<package format="3">
  <name>joint_read_menu</name>
  <version>0.0.0</version>
  <description>Test menu for reading joint values of EVE</description>
  <maintainer email="larsa09@uia.no"> Lars Bleie Andersen </maintainer>
  <license>UiA</license>

  <buildtool_depend>ament_cmake</buildtool_depend>

  <depend>rclcpp</depend>
  <depend>std_msgs</depend>
  <depend>action_msgs</depend>
  <depend>halodi_msgs</depend>

  <test_depend>ament_lint_auto</test_depend>
  <test_depend>ament_lint_common</test_depend>

  <export>
    <build_type>ament_cmake</build_type>
  </export>
</package>
```

appendices/joint_read_menu/package.tex

A.4.3 CMakeLists.txt

```cmake
cmake_minimum_required(VERSION 3.5)
project(joint_read_menu)

# Default to C99
if(NOT CMAKE_C_STANDARD)
  set(CMAKE_C_STANDARD 99)
endif()

# Default to C++14
if(NOT CMAKE_CXX_STANDARD)
  set(CMAKE_CXX_STANDARD 14)
endif()

if(CMAKE_COMPILER_IS_GNUCXX OR CMAKE_CXX_COMPILER_ID MATCHES "Clang")
```

```
  add_compile_options(-Wall -Wextra -Wpedantic)
endif()

# find dependencies
find_package(ament_cmake REQUIRED)
find_package(rclcpp REQUIRED)
find_package(std_msgs REQUIRED)
find_package(action_msgs REQUIRED)
find_package(halodi_msgs REQUIRED)
find_package(tf2 REQUIRED)
find_package(tf2_geometry_msgs REQUIRED)

#add executable command
add_executable(joint_read src/joint_read.cpp)
target_include_directories(joint_read PUBLIC
  $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/include>
  $<INSTALL_INTERFACE:include>)
ament_target_dependencies(joint_read rclcpp halodi_msgs action_msgs)

install(TARGETS joint_read
  DESTINATION lib/${PROJECT_NAME})



if(BUILD_TESTING)
  find_package(ament_lint_auto REQUIRED)
  ament_lint_auto_find_test_dependencies()
endif()

ament_package()
```

appendices/joint_read_menu/CMakeLists.tex

A.4.4 joint_read.cpp

```
#include <iostream>
#include <memory>
#include <vector>
#include <fstream>
#include <iterator>
#include <string>
#include <math.h>
#include <algorithm>
#include <boost/uuid/uuid.hpp>
#include <boost/uuid/uuid_generators.hpp>
#include "rclcpp/rclcpp.hpp"
#include "rclcpp/qos.hpp"
#include "action_msgs/msg/goal_status.hpp"
#include "unique_identifier_msgs/msg/uuid.hpp"
#include "halodi_msgs/msg/whole_body_trajectory.hpp"
#include "halodi_msgs/msg/whole_body_trajectory_point.hpp"
#include "halodi_msgs/msg/joint_space_command.hpp"
#include "halodi_msgs/msg/joint_name.hpp"
#include "halodi_msgs/msg/hand_command.hpp"
#include "halodi_msgs/msg/whole_body_state.hpp"

using namespace halodi_msgs::msg; //Declaring the 'msg' command from the 'halodi_msgs'
using std::placeholders::_1; //Declaring the placeholder '_1' which directs the position of a ...
    value in a function to the first parameter

double storage[20]; //Declaring the array for 21 joint values, stored as radians

std::vector<double> Joint13_pos; //declaring vectors for position and velocity data
std::vector<double> Joint13_vel;
std::vector<double> Joint16_pos;
std::vector<double> Joint16_vel;
std::vector<double> Joint18_pos;
```

```
std::vector<double> Joint18_vel;

/**EDIT THESE PATHS TO STORE FILES**/
const char *path13p = "/home/lba/eve_ws/src/joint_read_menu/storage/...
    RIGHT_SHOULDER_PITCH_position.txt"; //Declaring file paths to store position and velocity ...
    data
const char *path13v = "/home/lba/eve_ws/src/joint_read_menu/storage/...
    RIGHT_SHOULDER_PITCH_velocity.txt";
const char *path16p = "/home/lba/eve_ws/src/joint_read_menu/storage/RIGHT_ELBOW_PITCH_position....
    txt";
const char *path16v = "/home/lba/eve_ws/src/joint_read_menu/storage/RIGHT_ELBOW_PITCH_velocity....
    txt";
const char *path18p = "/home/lba/eve_ws/src/joint_read_menu/storage/RIGHT_WRIST_PITCH_position....
    txt";
const char *path18v = "/home/lba/eve_ws/src/joint_read_menu/storage/RIGHT_WRIST_PITCH_velocity....
    txt";

void remove(std::vector<double> &v)//Function to remove duplicate numbers
{
    auto end = v.end();
    for (auto it = v.begin(); it != end; ++it)
    {
        end = std::remove(it + 1, end, *it);
    }
    v.erase(end, v.end());
}


class JointSubscriber : public rclcpp::Node //Creating the node class 'JointSubscriber' by ...
    inheriting from 'rclcpp::Node'
{
public:

  JointSubscriber()
  : Node("joint_subscriber") //The public constructor names the node 'joint_subscriber'
  {
    rclcpp::QoS qos(1);
    qos.best_effort();//Setting the quality of service to best effort in order to receive the ...
        WholeBodyState messages (see Halodi API in the repository for QoS settings for the ...
        topics)

    publisher_ = this->create_publisher<WholeBodyTrajectory>("/eve/whole_body_trajectory", 10);...
         //The publisher is initialized with the 'WholeBodyTrajectory' message type, the topic ...
        name '/eve/whole_body_trajectory', and the required queue size to limit messages in the...
         event of a backup.
    //The subscriber is initialized with the 'action_msgs::msg::GoalStatus' message type. 'std...
        ::bind' is used to register the '&Joint_menu_publisher::status_callback' reference as a...
         callback. It provides feedback of commands send to /eve/whole_body_trajectory
    subscription_ = this->create_subscription<action_msgs::msg::GoalStatus>("/eve/...
        whole_body_trajectory_status", 10, std::bind(&JointSubscriber::status_callback, this, ...
        _1));
    subscription2_ = this->create_subscription<halodi_msgs::msg::WholeBodyState>("/eve/...
        whole_body_state", qos, std::bind(&JointSubscriber::joint_callback, this, _1));
    uuid_msg_ = create_random_uuid(); //A UUID (universally unique identifier) is created ...
        though the create_random_uuid function
    publish_trajectory(uuid_msg_); //The publish_trajectory function is run with the UUID as ...
        the trajectory_id
  }

   void joint_callback(halodi_msgs::msg::WholeBodyState::SharedPtr msg) //The function that ...
       stores the joint values via callback
  {
    Joint13_pos.shrink_to_fit();//Remove possible excess numbers in vector
    Joint13_vel.shrink_to_fit();
    Joint16_pos.shrink_to_fit();
    Joint16_vel.shrink_to_fit();
    Joint18_pos.shrink_to_fit();
```

```
    Joint18_vel.shrink_to_fit();
    //The arrow operator (->) is used to access the member of the data structure pointed to by ...
        a pointer.
    double J13p = roundf(msg->joint_states[13].position * 1000) / 1000; //round to 3 decimals
    double J13v = roundf(msg->joint_states[13].velocity * 1000) / 1000; //round to 3 decimals
    double J16p = roundf(msg->joint_states[16].position * 1000) / 1000; //round to 3 decimals
    double J16v = roundf(msg->joint_states[16].velocity * 1000) / 1000; //round to 3 decimals
    double J18p = roundf(msg->joint_states[18].position * 1000) / 1000; //round to 3 decimals
    double J18v = roundf(msg->joint_states[18].velocity * 1000) / 1000; //round to 3 decimals
    //Storing the values into vectors. push_back means that it adds new elements at the end of ...
        the vector
    Joint13_pos.push_back(J13p);
    Joint13_vel.push_back(J13v);
    Joint16_pos.push_back(J16p);
    Joint16_vel.push_back(J16v);
    Joint18_pos.push_back(J18p);
    Joint18_vel.push_back(J18v);
    //Removing duplicate values
    remove(Joint13_pos);
    remove(Joint13_vel);
    remove(Joint16_pos);
    remove(Joint16_vel);
    remove(Joint18_pos);
    remove(Joint18_vel);

  }

void status_callback(action_msgs::msg::GoalStatus::SharedPtr msg)
  {
    switch(msg->status){ //The arrow operator (->) is used to access the member of the 'status'...
         data structure pointed to by a pointer
      case 0:
        RCLCPP_INFO(this->get_logger(), "GoalStatus: STATUS_UNKNOWN"); //The 'this' pointer is ...
            here used to retrieve the 'get_logger()' information
        break;
      case 1:
        RCLCPP_INFO(this->get_logger(), "GoalStatus: STATUS_ACCEPTED");
        break;
      case 2:
        RCLCPP_INFO(this->get_logger(), "GoalStatus: STATUS_EXECUTING");
        break;
      case 3:
        RCLCPP_INFO(this->get_logger(), "GoalStatus: STATUS_CANCELING");
        break;
      case 4:
        //If the uuid of the received GoalStatus STATUS_SUCCEEDED Msg is the same as the uuid of...
             the command that was sent
        RCLCPP_INFO(this->get_logger(), "GoalStatus: STATUS_SUCCEEDED");
        RCLCPP_INFO(this->get_logger(), "Shutting down...");
        rclcpp::shutdown();
        break;
      case 5:
        RCLCPP_INFO(this->get_logger(), "GoalStatus: STATUS_CANCELED");
        break;
      case 6:
        RCLCPP_INFO(this->get_logger(), "GoalStatus: STATUS_ABORTED");
        break;
      default:
        break;}
  }
private:
  unique_identifier_msgs::msg::UUID create_random_uuid() //The function creates a random UUID ...
      to track msg
  {
    boost::uuids::random_generator gen; boost::uuids::uuid u = gen();
    unique_identifier_msgs::msg::UUID uuid_msg;
    std::array<uint8_t, 16> uuid; std::copy(std::begin(u.data), std::end(u.data), uuid.begin())...
```

```
      ;
  uuid_msg.uuid = uuid;
  return uuid_msg;
}

void publish_trajectory(unique_identifier_msgs::msg::UUID uuid_msg) //The function sets all ...
    required parameters for the WholeBodyTrajectory structure and send them to 'publisher_'
{
  // begin construction of the publsihed msg
  WholeBodyTrajectory trajectory_msg;
  trajectory_msg.append_trajectory = false; //If set to false, the existing trajectory is ...
      cancelled and this trajectory is immediatly executed.
  //MINIMUM_JERK_CONSTRAINED mode is recommended to constrain joint velocities and ...
      accelerations between each waypoint (make the movement smoother)
  //It specifies how the trajectory is interpolated from the previous objective
  trajectory_msg.interpolation_mode.value = TrajectoryInterpolation::MINIMUM_JERK_CONSTRAINED...
      ;
  trajectory_msg.trajectory_id = uuid_msg;
  // Ading waypoint targets. The desired times (in seconds) are provided in terms of
  // offset from time at which this published message is received
  trajectory_msg.trajectory_points.push_back(targetall_(5));

  RCLCPP_INFO(this->get_logger(), "Sending trajectory, listening for ...
      whole_body_trajectory_status...");
  publisher_->publish(trajectory_msg); //The trajectory message is published
}

//This generates the individual single joint command. It takes in the required parameters and...
     returns the message used for the taskspace trajectory point in the 'targetall_' function
JointSpaceCommand generate_joint_space_command(int32_t joint_id, double q_des, double qd_des ...
    = 0.0, double qdd_des = 0.0)
{
  JointSpaceCommand ret_msg;
  JointName name;
  name.joint_id = joint_id;
  ret_msg.joint = name;
  ret_msg.q_desired = q_des;
  ret_msg.qd_desired = qd_des;
  ret_msg.qdd_desired = qdd_des;
  ret_msg.use_default_gains = true;
  return ret_msg;
}

//Each target, in the form of a single WholeBodyTrajectoryPoint msg, consists of a ...
    concatenation of desired joint configurations with no more than one desired value per ...
    joint. The desired time at which we want to reach these joint targets is also specified.
//This function then takes in the execution time and assembles the name of the desired joint ...
    along with the desired position, velocity, acceleration etc. through the '...
    generate_joint_space_command' function
WholeBodyTrajectoryPoint targetall_(int32_t t)
{
  WholeBodyTrajectoryPoint ret_msg;

  builtin_interfaces::msg::Duration duration;
  duration.sec = t; //Sets the execution time for the trajectory, relative to the start time.
  ret_msg.time_from_start = duration;

  //The 'push_back' pushes elements from the back of the generate_joint_space_command ...
      contents
  ret_msg.joint_space_commands.push_back(generate_joint_space_command(JointName::...
      RIGHT_SHOULDER_PITCH, storage[13]));
  ret_msg.joint_space_commands.push_back(generate_joint_space_command(JointName::...
      RIGHT_ELBOW_PITCH, storage[16]));
  ret_msg.joint_space_commands.push_back(generate_joint_space_command(JointName::...
      RIGHT_WRIST_PITCH, storage[18]));

  return ret_msg;
```

```cpp
  }

  rclcpp::Publisher<WholeBodyTrajectory>::SharedPtr publisher_; //Declaration of the 'publisher_' publisher
  rclcpp::Subscription<action_msgs::msg::GoalStatus>::SharedPtr subscription_; //Declaration of the 'subscription_' subscriber
  rclcpp::Subscription<halodi_msgs::msg::WholeBodyState>::SharedPtr subscription2_; //Declaration of the 'subscription_' subscriber
  unique_identifier_msgs::msg::UUID uuid_msg_; //Declaration of the UUID generator
};

class JointIssuer : public rclcpp::Node //Creating the node class 'JointSubscriber' by inheriting from 'rclcpp::Node'
{
public:

  JointIssuer()
  : Node("joint_issuer") //The public constructor names the node 'joint_subscriber'
  {
    publisher_ = this->create_publisher<WholeBodyTrajectory>("/eve/whole_body_trajectory", 10); //The publisher is initialized with the 'WholeBodyTrajectory' message type, the topic name '/eve/whole_body_trajectory', and the required queue size to limit messages in the event of a backup.
    //The subscriber is initialized with the 'action_msgs::msg::GoalStatus' message type. 'std::bind' is used to register the '&Joint_menu_publisher::status_callback' reference as a callback. It provides feedback of commands send to /eve/whole_body_trajectory
    subscription_ = this->create_subscription<action_msgs::msg::GoalStatus>("/eve/whole_body_trajectory_status", 10, std::bind(&JointIssuer::status_callback, this, _1));
    uuid_msg_ = create_random_uuid(); //A UUID (universally unique identifier) is created though the create_random_uuid function
    publish_trajectory(uuid_msg_); //The publish_trajectory function is run with the UUID as the trajectory_id
  }

private:
  void status_callback(action_msgs::msg::GoalStatus::SharedPtr msg)
  {
    switch(msg->status){ //The arrow operator (->) is used to access the member of the 'status' data structure pointed to by a pointer
      case 0:
        RCLCPP_INFO(this->get_logger(), "GoalStatus: STATUS_UNKNOWN"); //The 'this' pointer is here used to retrieve the 'get_logger()' information
        break;
      case 1:
        RCLCPP_INFO(this->get_logger(), "GoalStatus: STATUS_ACCEPTED");
        break;
      case 2:
        RCLCPP_INFO(this->get_logger(), "GoalStatus: STATUS_EXECUTING");
        break;
      case 3:
        RCLCPP_INFO(this->get_logger(), "GoalStatus: STATUS_CANCELING");
        break;
      case 4:
        //If the uuid of the received GoalStatus STATUS_SUCCEEDED Msg is the same as the uuid of the command that was sent
        RCLCPP_INFO(this->get_logger(), "GoalStatus: STATUS_SUCCEEDED");
        RCLCPP_INFO(this->get_logger(), "Shutting down...");
        rclcpp::shutdown();
        break;
      case 5:
        RCLCPP_INFO(this->get_logger(), "GoalStatus: STATUS_CANCELED");
        break;
      case 6:
        RCLCPP_INFO(this->get_logger(), "GoalStatus: STATUS_ABORTED");
        break;
      default:
        break;}
```

```
}

unique_identifier_msgs::msg::UUID create_random_uuid() //The function creates a random UUID ...
    to track msg
{
  boost::uuids::random_generator gen; boost::uuids::uuid u = gen();
  unique_identifier_msgs::msg::UUID uuid_msg;
  std::array<uint8_t, 16> uuid; std::copy(std::begin(u.data), std::end(u.data), uuid.begin())...
    ;
  uuid_msg.uuid = uuid;
  return uuid_msg;
}

void publish_trajectory(unique_identifier_msgs::msg::UUID uuid_msg) //The function sets all ...
    required parameters for the WholeBodyTrajectory structure and send them to 'publisher_'
{
  // begin construction of the publsihed msg
  WholeBodyTrajectory trajectory_msg;
  trajectory_msg.append_trajectory = false; //If set to false, the existing trajectory is ...
    cancelled and this trajectory is immediatly executed.
  //MINIMUM_JERK_CONSTRAINED mode is recommended to constrain joint velocities and ...
    accelerations between each waypoint (make the movement smoother)
  //It specifies how the trajectory is interpolated from the previous objective
  trajectory_msg.interpolation_mode.value = TrajectoryInterpolation::MINIMUM_JERK_CONSTRAINED...
    ;
  trajectory_msg.trajectory_id = uuid_msg;
  // Ading waypoint targets. The desired times (in seconds) are provided in terms of
  // offset from time at which this published message is received
  trajectory_msg.trajectory_points.push_back(targetall_(5));

  RCLCPP_INFO(this->get_logger(), "Sending trajectory, listening for ...
    whole_body_trajectory_status...");
  publisher_->publish(trajectory_msg); //The trajectory message is published
}

//This generates the individual single joint command. It takes in the required parameters and...
     returns the message used for the taskspace trajectory point in the 'targetall_' function
JointSpaceCommand generate_joint_space_command(int32_t joint_id, double q_des, double qd_des ...
    = 0.0, double qdd_des = 0.0)
{
  JointSpaceCommand ret_msg;
  JointName name;
  name.joint_id = joint_id;
  ret_msg.joint = name;
  ret_msg.q_desired = q_des;
  ret_msg.qd_desired = qd_des;
  ret_msg.qdd_desired = qdd_des;
  ret_msg.use_default_gains = true;
  return ret_msg;
}

//Each target, in the form of a single WholeBodyTrajectoryPoint msg, consists of a ...
    concatenation of desired joint configurations with no more than one desired value per ...
    joint. The desired time at which we want to reach these joint targets is also specified.
//This function then takes in the execution time and assembles the name of the desired joint ...
    along with the desired position, velocity, acceleration etc. through the '...
    generate_joint_space_command' function
WholeBodyTrajectoryPoint targetall_(int32_t t)
{
  WholeBodyTrajectoryPoint ret_msg;

  builtin_interfaces::msg::Duration duration;
  duration.sec = t; //Sets the execution time for the trajectory, relative to the start time.
  ret_msg.time_from_start = duration;

  /** insert code for issuing values**/

```

```
    ret_msg.joint_space_commands.push_back(generate_joint_space_command(JointName::...
        RIGHT_SHOULDER_PITCH, Joint13_pos[0]));
    ret_msg.joint_space_commands.push_back(generate_joint_space_command(JointName::...
        RIGHT_ELBOW_PITCH, Joint16_pos[0]));
    ret_msg.joint_space_commands.push_back(generate_joint_space_command(JointName::...
        RIGHT_WRIST_PITCH, Joint18_pos[0]));

    return ret_msg;
  }

  rclcpp::Publisher<WholeBodyTrajectory>::SharedPtr publisher_; //Declaration of the '...
      publisher_' publisher
  rclcpp::Subscription<action_msgs::msg::GoalStatus>::SharedPtr subscription_; //Declaration of...
       the 'subscription_' subscriber
  unique_identifier_msgs::msg::UUID uuid_msg_; //Declaration of the UUID generator
};


void store()
{ std::remove(path13p); // delete file
    std::ofstream storeFile13pos(path13p); // Create an output filestream object
    // Check if file is open
    if (storeFile13pos.is_open())
    { Joint13_pos.erase(Joint13_pos.begin()); //delete unwanted value from RCLCPP_INFO transfer
        // Send data to the stream
        for(std::size_t i = 0, max = Joint13_pos.size(); i != max; ++i)
        {
            storeFile13pos << Joint13_pos[i] << "\n";
        }
        // Close the file
        storeFile13pos.close();
     }
    else
    {
        std::cout << "Error! Unable to create file for RIGHT_SHOULDER_PITCH position";
    }

    std::remove(path13v); // delete file
    std::ofstream storeFile13vel(path13v); // Create an output filestream object
    // Check if file is open
    if (storeFile13vel.is_open())
    { Joint13_vel.erase(Joint13_vel.begin()); //delete unwanted value from RCLCPP_INFO transfer
        // Send data to the stream
        for(std::size_t i = 0, max = Joint13_vel.size(); i != max; ++i)
        {
            storeFile13vel << Joint13_vel[i] << "\n";
        }
        // Close the file
        storeFile13vel.close();
     }
    else
    {
        std::cout << "Error! Unable to create file for RIGHT_SHOULDER_PITCH velocity";
    }

    std::remove(path16p); // delete file
    std::ofstream storeFile16pos(path16p); // Create an output filestream object
    // Check if file is open
    if (storeFile16pos.is_open())
    { Joint16_pos.erase(Joint16_pos.begin()); //delete unwanted value from RCLCPP_INFO transfer
        // Send data to the stream
        for(std::size_t i = 0, max = Joint16_pos.size(); i != max; ++i)
        {
            storeFile16pos << Joint16_pos[i] << "\n";
        }
        // Close the file
        storeFile16pos.close();
```

```
    }
    else
    {
        std::cout << "Error! Unable to create file for RIGHT_ELBOW_PITCH position";
    }

    std::remove(path16v); // delete file
    std::ofstream storeFile16vel(path16v); // Create an output filestream object
    // Check if file is open
    if (storeFile16vel.is_open())
    { Joint16_vel.erase(Joint16_vel.begin()); //delete unwanted value from RCLCPP_INFO transfer
        // Send data to the stream
        for(std::size_t i = 0, max = Joint16_vel.size(); i != max; ++i)
        {
            storeFile16vel << Joint16_vel[i] << "\n";
        }
        // Close the file
        storeFile16vel.close();
     }
    else
    {
        std::cout << "Error! Unable to create file for RIGHT_ELBOW_PITCH velocity";
    }

    std::remove(path18p); // delete file
    std::ofstream storeFile18pos(path18p); // Create an output filestream object
    // Check if file is open
    if (storeFile18pos.is_open())
    { Joint18_pos.erase(Joint18_pos.begin()); //delete unwanted value from RCLCPP_INFO transfer
        // Send data to the stream
        for(std::size_t i = 0, max = Joint18_pos.size(); i != max; ++i)
        {
            storeFile18pos << Joint18_pos[i] << "\n";
        }
        // Close the file
        storeFile18pos.close();
     }
    else
    {
        std::cout << "Error! Unable to create file for RIGHT_WRIST_PITCH position";
    }

    std::remove(path18v); // delete file
    std::ofstream storeFile18vel(path18v); // Create an output filestream object
    // Check if file is open
    if (storeFile18vel.is_open())
    { Joint18_vel.erase(Joint18_vel.begin()); //delete unwanted value from RCLCPP_INFO transfer
        // Send data to the stream
        for(std::size_t i = 0, max = Joint18_vel.size(); i != max; ++i)
        {
            storeFile18vel << Joint18_vel[i] << "\n";
        }
        // Close the file
        storeFile18vel.close();
     }
    else
    {
        std::cout << "Error! Unable to create file for RIGHT_WRIST_PITCH velocity";
    }

}

void retrieve()
{
    std::ifstream inputFile13pos(path13p); // Create an input file stream object
    Joint13_pos.clear(); //Clear all values from vector
    // Check if exists and then open the file.
```

```
    if (inputFile13pos.good())
    {
        // Push items into the vector
        double current_number = 0;
        while (inputFile13pos >> current_number)
        {
            Joint13_pos.push_back(current_number);
        }
        // Close the file.
        inputFile13pos.close();
    }
    else
    {
        std::cout << "Error! Unable to open file for RIGHT_SHOULDER_PITCH position";
    }


    std::ifstream inputFile13vel(path13v); // Create an input file stream object
    Joint13_vel.clear(); //Clear all values from vector
    // Check if exists and then open the file.
    if (inputFile13vel.good())
    {
        // Push items into the vector
        double current_number = 0;
        while (inputFile13vel >> current_number)
        {
            Joint13_vel.push_back(current_number);
        }
        // Close the file.
        inputFile13vel.close();
    }
    else
    {
        std::cout << "Error! Unable to open file for RIGHT_SHOULDER_PITCH velocity";
    }


    std::ifstream inputFile16pos(path16p); // Create an input file stream object
    Joint16_pos.clear(); //Clear all values from vector
    // Check if exists and then open the file.
    if (inputFile16pos.good())
    {
        // Push items into the vector
        double current_number = 0;
        while (inputFile16pos >> current_number)
        {
            Joint16_pos.push_back(current_number);
        }
        // Close the file.
        inputFile16pos.close();
    }
    else
    {
        std::cout << "Error! Unable to open file for RIGHT_ELBOW_PITCH position";
    }


    std::ifstream inputFile16vel(path16v); // Create an input file stream object
    Joint16_vel.clear(); //Clear all values from vector
    // Check if exists and then open the file.
    if (inputFile16vel.good())
    {
        // Push items into the vector
        double current_number = 0;
        while (inputFile16vel >> current_number)
        {
            Joint16_vel.push_back(current_number);
```

```
        }
        // Close the file.
        inputFile16vel.close();
    }
    else
    {
        std::cout << "Error! Unable to open file for RIGHT_ELBOW_PITCH velocity";
    }


    std::ifstream inputFile18pos(path18p); // Create an input file stream object
    Joint18_pos.clear(); //Clear all values from vector
    // Check if exists and then open the file.
    if (inputFile18pos.good())
    {
        // Push items into the vector
        double current_number = 0;
        while (inputFile18pos >> current_number)
        {
            Joint18_pos.push_back(current_number);
        }
        // Close the file.
        inputFile18pos.close();
    }
    else
    {
        std::cout << "Error! Unable to open file for RIGHT_WRIST_PITCH position";
    }


    std::ifstream inputFile18vel(path18v); // Create an input file stream object
    Joint18_vel.clear(); //Clear all values from vector
    // Check if exists and then open the file.
    if (inputFile18vel.good())
    {
        // Push items into the vector
        double current_number = 0;
        while (inputFile18vel >> current_number)
        {
            Joint18_vel.push_back(current_number);
        }
        // Close the file.
        inputFile18vel.close();
    }
    else
    {
        std::cout << "Error! Unable to open file for RIGHT_WRIST_PITCH velocity";
    }

    for (std::size_t i = 0, max = Joint13_pos.size(); i != max; ++i)
    {
        std::cout << "Angular position " << i << " for RIGHT_SHOULDER_PITCH is: " << Joint13_pos...
            [i] << std::endl;
    }
    for (std::size_t i = 0, max = Joint13_vel.size(); i != max; ++i)
    {
        std::cout << "Angular velocity " << i << " for RIGHT_SHOULDER_PITCH is: " << Joint13_vel...
            [i] << std::endl;
    }
    for (std::size_t i = 0, max = Joint16_pos.size(); i != max; ++i)
    {
        std::cout << "Angular position " << i << " for RIGHT_ELBOW_PITCH is: " << Joint13_pos[i]...
             << std::endl;
    }
    for (std::size_t i = 0, max = Joint16_vel.size(); i != max; ++i)
    {
        std::cout << "Angular velocity " << i << " for RIGHT_ELBOW_PITCH is: " << Joint13_vel[i]...
```

```
            << std::endl;
    }
    for (std::size_t i = 0, max = Joint18_pos.size(); i != max; ++i)
    {
        std::cout << "Angular position " << i << " for RIGHT_WRIST_PITCH is: " << Joint13_pos[i]...
            << std::endl;
    }
    for (std::size_t i = 0, max = Joint18_vel.size(); i != max; ++i)
    {
        std::cout << "Angular velocity " << i << " for RIGHT_WRIST_PITCH is: " << Joint13_vel[i]...
            << std::endl;
    }
}


void intro() //Intro message
{
    std::cout << "*******************************************************************" << std::...
        endl;
    std::cout << "* Menu for testing the recording of joint values of EVEr3 *" << std::endl;
    std::cout << "*******************************************************************" << std::...
        endl;
}

void menu_space() //This is used to make the menu better looking, assuming that 1000 lines will...
    be enough to clear the screen
{
    std::cout << std::string(1000, '\n');
}

void wait_for_enter()
{

    std::string wait;
    std::cout << "[Enter] to continue..." << std::endl;
    getline(std::cin, wait);
}

void print() //Print transferred values to vector
{
    for (std::size_t i = 0, max = Joint13_pos.size(); i != max; ++i)
    {
        std::cout << "Angular position " << i << " for RIGHT_SHOULDER_PITCH is: " << Joint13_pos...
            [i] << std::endl;
    }
    for (std::size_t i = 0, max = Joint13_vel.size(); i != max; ++i)
    {
        std::cout << "Angular velocity " << i << " for RIGHT_SHOULDER_PITCH is: " << Joint13_vel...
            [i] << std::endl;
    }
    for (std::size_t i = 0, max = Joint16_pos.size(); i != max; ++i)
    {
        std::cout << "Angular position " << i << " for RIGHT_ELBOW_PITCH is: " << Joint13_pos[i]...
            << std::endl;
    }
    for (std::size_t i = 0, max = Joint16_vel.size(); i != max; ++i)
    {
        std::cout << "Angular velocity " << i << " for RIGHT_ELBOW_PITCH is: " << Joint13_vel[i]...
            << std::endl;
    }
    for (std::size_t i = 0, max = Joint18_pos.size(); i != max; ++i)
    {
        std::cout << "Angular position " << i << " for RIGHT_WRIST_PITCH is: " << Joint13_pos[i]...
            << std::endl;
    }
    for (std::size_t i = 0, max = Joint18_vel.size(); i != max; ++i)
    {
```

```
      std::cout << "Angular velocity " << i << " for RIGHT_WRIST_PITCH is: " << Joint13_vel[i] << std::endl;
  }
}

void raise_right_arm() //Raising the arm to check the transfer of values
{
     storage[13] = 0.3;
     storage[16] = -1.3;
     storage[18] = 0.2;
}

void set_to_zero() //Setting all joint values to zero to check the transfer of values
{
     storage[13] = 0.0;
     storage[16] = 0.0;
     storage[18] = 0.0;
}


int main(int argc, char * argv[])
{
 int sel5;
 bool done4 = false;

 do
 {
   menu_space();
   intro();
   std::cout <<" 1[+Enter] to lower right arm and execute" << std::endl;
   std::cout <<" 2[+Enter] to raise right arm and execute" << std::endl;
   std::cout <<" 3[+Enter] to print recorded values" << std::endl;
   std::cout <<" 4[+Enter] to store recorded values" << std::endl;
   std::cout <<" 5[+Enter] to retrieve previously recorded values and print them" << std::endl;
   std::cout <<" 6[+Enter] to issue retrieved values" << std::endl;
   std::cout <<" 7[+Enter] to exit the program" << std::endl;
   std::cin >> sel5;

   switch (sel5)
   {
   case 1:
     set_to_zero();
     rclcpp::init(argc, argv); //'rclcpp::init' initializes ROS 2, and 'rclcpp::spin' starts processing data from the JointSubscriber node
     rclcpp::spin(std::make_shared<JointSubscriber>());
     rclcpp::shutdown();
     break;

   case 2:
     raise_right_arm();
     rclcpp::init(argc, argv); //'rclcpp::init' initializes ROS 2, and 'rclcpp::spin' starts processing data from the JointSubscriber node
     rclcpp::spin(std::make_shared<JointSubscriber>());
     rclcpp::shutdown();
     break;


     case 3:
     menu_space();
     print();
     std::cin.get(); //This makes the program wait for input from user
     wait_for_enter();
     break;

   case 4:
     store();
```

```
        std::cin.get();
        wait_for_enter();
        break;

      case 5:
        menu_space();
        retrieve();
        std::cin.get();
        wait_for_enter();
        break;

      case 6:
        rclcpp::init(argc, argv); //'rclcpp::init' initializes ROS 2, and 'rclcpp::spin' starts processing data from the JointIssuer node
        rclcpp::spin(std::make_shared<JointIssuer>());
        rclcpp::shutdown();
        break;

      case 7:
        done4 = true;
        break;

      default:
         std::cout << "Invalid input" << std::endl;

      }
    }while(!done4);

    return 0;
}
```

appendices/joint_read_menu/joint_read.tex

A.5 The Leg Model

A.5.1 README.md

```
##Leg model for Rehabilitation Project at UiA

* Author: Lars Bleie Andersen <larsa09@uia.no>

## Resources

[URDF EXAMPLE with RVIZ LAUNCH FILE (ROS2)](https://github.com/olmerg/lesson_urdf)
[URDF to GAZEBO TUTORIAL (ROS1)](http://gazebosim.org/tutorials/?tut=ros_urdf)
[ROS CONTROL](http://wiki.ros.org/ros_control)
[ROS2 CONTROL DEMOS](https://github.com/ros-simulation/gazebo_ros2_control/tree/master/...
    gazebo_ros2_control_demos)
[ROS INTERFACES](https://wiki.ros.org/ros_control#Hardware_Interfaces)
[SDF CONVERSION](gz sdf -p /my_urdf.urdf > /my_sdf.sdf)

## Launch commands

Leg Model in Rviz:
  ros2 launch leg_model_description rviz_launch.py

##If the model does not show in Rviz, try to type in the shell terminal: export LC_NUMERIC="...
    en_US.UTF-8"
```

appendices/README.tex

A.6 Leg Model Description Package

A.6.1 package.xml

```
<?xml version="1.0"?>
<?xml-model href="http://download.ros.org/schema/package_format3.xsd" schematypens="http://www....
    w3.org/2001/XMLSchema"?>
<package format="3">
  <name>leg_model_description</name>
  <version>0.0.5</version>
  <description>Description package for leg_model</description>
  <maintainer email="larsa09@uia.no"> Lars Bleie Andersen </maintainer>
  <license>UiA</license>

  <buildtool_depend>ament_cmake</buildtool_depend>
  <depend>urdf</depend>
  <build_depend>launch_ros</build_depend>
  <exec_depend>launch_ros</exec_depend>
  <exec_depend>robot_state_publisher</exec_depend>
  <exec_depend>joint_state_publisher_gui</exec_depend>
  <export>
    <build_type>ament_cmake</build_type>
  </export>
</package>
```

appendices/leg_model_description/package.tex

A.6.2 CMakeLists.txt

```
cmake_minimum_required(VERSION 3.5)
project(leg_model_description)

# Default to C99
if(NOT CMAKE_C_STANDARD)
  set(CMAKE_C_STANDARD 99)
endif()

```

```
# Default to C++14
if(NOT CMAKE_CXX_STANDARD)
  set(CMAKE_CXX_STANDARD 14)
endif()

if(CMAKE_COMPILER_IS_GNUCXX OR CMAKE_CXX_COMPILER_ID MATCHES "Clang")
  add_compile_options(-Wall -Wextra -Wpedantic)
endif()


# Find ament packages and libraries for ament and urdf
find_package(ament_cmake REQUIRED)
find_package(urdf REQUIRED)


# Path to directories
install(DIRECTORY
  urdf
  rviz
  launch
  meshes
  DESTINATION share/${PROJECT_NAME})


if(BUILD_TESTING)
  find_package(ament_lint_auto REQUIRED)
  ament_lint_auto_find_test_dependencies()
endif()
ament_package()
```

appendices/leg_model_description/CMakeLists.tex

A.6.3 rviz_launch.py

```python
import os

from ament_index_python.packages import get_package_share_directory

from launch import LaunchDescription
from launch.actions import DeclareLaunchArgument, ExecuteProcess, IncludeLaunchDescription
from launch.conditions import IfCondition
from launch.launch_description_sources import PythonLaunchDescriptionSource
from launch.substitutions import LaunchConfiguration, PythonExpression
from launch_ros.actions import Node


def generate_launch_description():
    # Defining the launch directory
    bringup_dir = get_package_share_directory('leg_model_description')
    launch_dir = os.path.join(bringup_dir, 'launch')

    # Declaring launch configuration variables specific to simulation
    rviz_config_file = LaunchConfiguration('rviz_config_file')
    use_robot_state_pub = LaunchConfiguration('use_robot_state_pub')
    use_joint_state_pub = LaunchConfiguration('use_joint_state_pub')
    use_rviz = LaunchConfiguration('use_rviz')
    urdf_file= LaunchConfiguration('urdf_file')

    declare_rviz_config_file_cmd = DeclareLaunchArgument(
        'rviz_config_file',
        default_value=os.path.join(bringup_dir, 'rviz', 'view.rviz'),
        description='Full path to the RVIZ config file to use')
    declare_use_robot_state_pub_cmd = DeclareLaunchArgument(
        'use_robot_state_pub',
        default_value='True',
        description='Whether to start the robot state publisher')
    declare_use_joint_state_pub_cmd = DeclareLaunchArgument(
```

```
        'use_joint_state_pub',
        default_value='True',
        description='Whether to start the joint state publisher')
    declare_use_rviz_cmd = DeclareLaunchArgument(
        'use_rviz',
        default_value='True',
        description='Whether to start RVIZ')

    declare_urdf_cmd = DeclareLaunchArgument(
        'urdf_file',
        default_value=os.path.join(bringup_dir, 'urdf', 'leg_model.xacro'),
        description='Whether to start RVIZ')


    start_robot_state_publisher_cmd = Node(
        condition=IfCondition(use_robot_state_pub),
        package='robot_state_publisher',
        executable='robot_state_publisher',
        name='robot_state_publisher',
        output='screen',
        #parameters=[{'use_sim_time': use_sim_time}],
        arguments=[urdf_file])

    start_joint_state_publisher_cmd = Node(
        condition=IfCondition(use_joint_state_pub),
        package='joint_state_publisher_gui',
        executable='joint_state_publisher_gui',
        name='joint_state_publisher_gui',
        output='screen',
        arguments=[urdf_file])

    rviz_cmd = Node(
        condition=IfCondition(use_rviz),
        package='rviz2',
        executable='rviz2',
        name='rviz2',
        arguments=['-d', rviz_config_file],
        output='screen')


    # Creating the launch description
    ld = LaunchDescription()

    # Declaring the launch options
    ld.add_action(declare_rviz_config_file_cmd)
    ld.add_action(declare_urdf_cmd)
    ld.add_action(declare_use_robot_state_pub_cmd)
    ld.add_action(declare_use_joint_state_pub_cmd)
    ld.add_action(declare_use_rviz_cmd)


    # Adding conditioned actions
    ld.add_action(start_joint_state_publisher_cmd)
    ld.add_action(start_robot_state_publisher_cmd)
    ld.add_action(rviz_cmd)

    return ld
```

appendices/leg_model_description/rviz_launch_py.tex

A.6.4 leg_model.xacro

```xml
<?xml version="1.0" encoding="utf-8"?>
<!-- This URDF was automatically created by SolidWorks to URDF Exporter! Originally created by ...
     Stephen Brawner (brawner@gmail.com)
     Commit Version: 1.6.0-1-g15f4949 Build Version: 1.6.7594.29634
     For more information, please see http://wiki.ros.org/sw_urdf_exporter -->
<robot name="leg_model" xmlns:xacro="http://ros.org/wiki/xacro">

  <!-- World (the child element Bed is rigidly attached to the world/base_link)-->
  <link name="world"/>

  <!-- Bed -->
  <link
    name="bed">
    <inertial>
      <origin
        xyz="0.49576 0.76172 1.1371"
        rpy="0 0 0" />
      <mass
        value="100" />
      <inertia
        ixx="0"
        ixy="0"
        ixz="0"
        iyy="34.8958"
        iyz="0"
        izz="0" />
    </inertial>
    <visual>
      <origin
        xyz="0 0 0"
        rpy="0 0 0" />
      <geometry>
        <mesh
          filename="package://leg_model_description/meshes/bed.STL" scale= "0.001 0.001 0.001"/>
      </geometry>
    </visual>
    <collision>
      <origin
        xyz="0 0 0"
        rpy="0 0 0" />
      <geometry>
        <mesh
          filename="package://leg_model_description/meshes/bed.STL" scale= "0.001 0.001 0.001"/>
      </geometry>
    </collision>
  </link>

  <!-- Inertial frame connection -->
  <joint
    name="inertial_frame"
    type="fixed">
    <origin
      xyz="0 0 0.585"
      rpy="0 0 0" />
    <parent
      link="world" />
    <child
      link="bed" />
    <axis
      xyz="0 0 0" />
  </joint>

  <!-- Hip joint -->
  <joint
```

```
  name="hip"
  type="revolute">
  <origin
    xyz="0 0 0"
    rpy="0 0 0" />
  <parent
    link="bed" />
  <child
    link="thigh" />
  <axis
    xyz="0 -1 0" />
 <limit lower="0" upper="2.2689" effort="0" velocity="0.5" />
  <dynamics damping="0.1" friction="0.0"/>
</joint>

<!-- Thigh -->
<link
  name="thigh">
  <inertial>
    <origin
      xyz="0.305 -0.01 2.7456E-13"
      rpy="0 0 0" />
    <mass
      value="10" />
    <inertia
      ixx="0"
      ixy="0"
      ixz="0"
      iyy="0.8396"
      iyz="0"
      izz="0" />
  </inertial>
  <visual>
    <origin
      xyz="0 0 0"
      rpy="0 0 0" />
    <geometry>
      <mesh
        filename="package://leg_model_description/meshes/thigh.STL" scale= "0.001 0.001 0.001"...
             />
    </geometry>
  </visual>
  <collision>
    <origin
      xyz="0 0 0"
      rpy="0 0 0" />
    <geometry>
      <mesh
        filename="package://leg_model_description/meshes/thigh.STL" scale= "0.001 0.001 0.001"...
             />
    </geometry>
  </collision>
</link>

<!-- Knee joint -->
<joint
  name="knee"
  type="revolute">
  <origin
    xyz="0.61 0 0"
    rpy="0 0 0" />
  <parent
    link="thigh" />
  <child
    link="braced_crus_w_handle" />
  <axis
    xyz="0 -1 0" />
```

```
 <limit lower="-2.0944" upper="0" effort="0" velocity="0.5" />
 <dynamics damping="0.1" friction="0.0"/>
</joint>

<!-- Braced Crus w/Handle -->
<link
  name="braced_crus_w_handle">
  <inertial>
    <origin
      xyz="0.26325 0.016019 2.2204E-16"
      rpy="0 0 0" />
    <mass
      value="3.5" />
    <inertia
      ixx="0"
      ixy="0"
      ixz="0"
      iyy="0.1411"
      iyz="0"
      izz="0" />
  </inertial>
  <visual>
    <origin
      xyz="0 0 0"
      rpy="0 0 0" />
    <geometry>
      <mesh
        filename="package://leg_model_description/meshes/braced_crus.STL" scale= "0.001 0.001 ...
            0.001"/>
    </geometry>
  </visual>
  <collision>
    <origin
      xyz="0 0 0"
      rpy="0 0 0" />
    <geometry>
      <mesh
        filename="package://leg_model_description/meshes/braced_crus.STL" scale= "0.001 0.001 ...
            0.001"/>
    </geometry>
  </collision>
</link>

 <!-- End effector connection -->
<joint
  name="handle_end"
  type="fixed">
  <origin
    xyz="0.305 0 0.195"
    rpy="0 0 0" />
  <parent
    link="braced_crus_w_handle" />
  <child
    link="end_effector" />
  <axis
    xyz="0 0 0" />
</joint>

<!-- End effector -->
<link name="end_effector">
  <inertial>
    <origin
      xyz="0 0 0"
      rpy="0 0 0" />
    <mass
      value="0" />
    <inertia
```

```
        ixx="0"
        ixy="0"
        ixz="0"
        iyy="0"
        iyz="0"
        izz="0" />
    </inertial>
    <visual>
      <origin
        xyz="0 0 0"
        rpy="0 0 0" />
    </visual>
    <collision>
      <origin
        xyz="0 0 0"
        rpy="0 0 0" />
    </collision>
  </link>

  <!-- Gazebo Plugin -->
  <gazebo>
    <plugin filename="libgazebo_ros_control.so" name="ros_control">
     </plugin>
  </gazebo>

  <!-- Gazebo Transmissions -->
  <transmission name="hip_trans">
    <type>transmission_interface/SimpleTransmission</type>
    <joint name="hip">
      <hardwareInterface>EffortJointInterface</hardwareInterface>
    </joint>
    <actuator name="hip_motor">
      <mechanicalReduction>1</mechanicalReduction>
    </actuator>
  </transmission>
  <transmission name="knee_trans">
    <type>transmission_interface/SimpleTransmission</type>
    <joint name="knee">
      <hardwareInterface>EffortJointInterface</hardwareInterface>
    </joint>
    <actuator name="knee_motor">
      <mechanicalReduction>1</mechanicalReduction>
    </actuator>
  </transmission>

  <!--Gazebo Reference for every link-->
  <gazebo reference="bed">
    <material>Gazebo/White</material>
    <selfCollide>true</selfCollide>
  </gazebo>
  <gazebo reference="thigh">
    <material>Gazebo/Wood</material>
    <selfCollide>true</selfCollide>
  </gazebo>
  <gazebo reference="braced_crus_w_handle">
    <material>Gazebo/Wood</material>
    <selfCollide>true</selfCollide>
    <mu>0.2</mu>
  </gazebo>
  <gazebo reference="end_effector">
    <material>Gazebo/Wood</material>
    <selfCollide>true</selfCollide>
  </gazebo>
</robot>
```

appendices/leg_model_description/leg_model.tex

Bibliography

[1] Jawad F. Abulhasan and Michael J. Grey. “Anatomy and Physiology of Knee Stability.” en. In: *Journal of Functional Morphology and Kinesiology* 2.4 (Dec. 2017). Number: 4 Publisher: Multidisciplinary Digital Publishing Institute, p. 34. DOI: 10.3390/jfmk2040034.

[2] Erhan Akdoğan, Ertuğrul Taçgın, and M. Arif Adli. “Knee rehabilitation using an intelligent robotic system.” en. In: *Journal of Intelligent Manufacturing* 20.2 (Jan. 2009), p. 195. ISSN: 1572-8145. DOI: 10.1007/s10845-008-0225-y.

[3] P. Beyl et al. “Safe and Compliant Guidance by a Powered Knee Exoskeleton for Robot-Assisted Rehabilitation of Gait.” In: *Advanced Robotics* 25.5 (Jan. 2011), pp. 513–535. ISSN: 0169-1864. DOI: 10.1163/016918611X558225.

[4] Turner A Blackburn and Emily Craig. “Knee Anatomy: A Brief Review.” In: *Physical Therapy* 60.12 (Dec. 1980), pp. 1556–1560. ISSN: 0031-9023. DOI: 10.1093/ptj/60.12.1556.

[5] Nicholas Bolus, Venu Ganti, and Omer Inan. “A 3D-Printed, Adjustable-Stiffness Knee Brace with Embedded Magnetic Angle Sensor.” In: vol. 2018. July 2018, pp. 1624–1627. DOI: 10.1109/EMBC.2018.8512600.

[6] Scott R. Brannan and David A. Jerrard. “Synovial fluid analysis.” en. In: *The Journal of Emergency Medicine* 30.3 (Apr. 2006), pp. 331–339. ISSN: 0736-4679. DOI: 10.1016/j.jemermed.2005.05.029.

[7] Harold Brem and Marjana Tomic-Canic. “Cellular and molecular basis of wound healing in diabetes.” en. In: *The Journal of Clinical Investigation* 117.5 (May 2007). Publisher: American Society for Clinical Investigation, pp. 1219–1222. ISSN: 0021-9738. DOI: 10.1172/JCI32169.

[8] Gong Chen et al. “Mechanical design and evaluation of a compact portable knee–ankle–foot robot for gait rehabilitation.” en. In: *Mechanism and Machine Theory* 103 (Sept. 2016), pp. 51–64. ISSN: 0094-114X. DOI: 10.1016/j.mechmachtheory.2016.04.012.

[9] Kim Seng Chia. “Ziegler-Nichols Based Proportional-Integral-Derivative Controller for a Line Tracking Robot.” In: *Indonesian Journal of Electrical Engineering and Computer Science* 9 (Jan. 2018), pp. 221–226. DOI: 10.11591/ijeecs.v9.i1.pp221-226.

[10] John Clancy, Andrew Mcvicar, and Janice Mooney. “Homeostasis 6: Nurses as external control agents in rheumatoid arthritis.” In: *British journal of nursing (Mark Allen Publishing)* 20 (Apr. 2011), pp. 497–8, 500. DOI: 10.12968/bjon.2011.20.8.497.

[11] Iñaki Díaz, Jorge Juan Gil, and Emilio Sánchez. *Lower-Limb Robotic Rehabilitation: Literature Review and Challenges*. en. Review Article. ISSN: 1687-9600 Pages: e759764 Publisher: Hindawi Volume: 2011. Nov. 2011. DOI: https://doi.org/10.1155/2011/759764.

[12] Christophe Duret, Anne-Gaëlle Grosmaire, and Hermano Igo Krebs. “Robot-Assisted Therapy in Upper Extremity Hemiparesis: Overview of an Evidence-Based Approach.” eng. In: *Frontiers in Neurology* 10 (2019), p. 412. ISSN: 1664-2295. DOI: 10.3389/fneur.2019.00412.

[13] *EVEr3 Humanoid Robot*. https://www.halodi.com/ever3.

[14] Lisa A. Garner. “Contact dermatitis to metals.” eng. In: *Dermatologic Therapy* 17.4 (2004), pp. 321–327. ISSN: 1396-0296. DOI: 10.1111/j.1396-0296.2004.04034.x.

[15] Roger Gassert and Volker Dietz. “Rehabilitation robots for the treatment of sensorimotor deficits: a neurophysiological perspective.” In: *Journal of NeuroEngineering and Rehabilitation* 15.1 (June 2018), p. 46. ISSN: 1743-0003. DOI: 10.1186/s12984-018-0383-x.

[16] A. Goldenberg, B. Benhabib, and R. Fenton. "A complete generalized solution to the inverse kinematics of robots." In: *IEEE Journal on Robotics and Automation* 1.1 (Mar. 1985). Conference Name: IEEE Journal on Robotics and Automation, pp. 14–20. ISSN: 2374-8710. DOI: 10.1109/JRA.1985.1086995.

[17] Halodi Robotics AS. *API*. https://github.com/Halodi/halodi-messages/blob/main/API.md.

[18] N. Hogan. "Impedance Control: An Approach to Manipulation." In: *1984 American Control Conference*. June 1984, pp. 304–313. DOI: 10.23919/ACC.1984.4788393.

[19] N. Hogan et al. "MIT-MANUS: a workstation for manual therapy and training. I." In: *[1992] Proceedings IEEE International Workshop on Robot and Human Communication*. 1992, pp. 161–165. DOI: 10.1109/ROMAN.1992.253895.

[20] Neville Hogan and Hermano I. Krebs. "Interactive robots for neuro-rehabilitation." en. In: *Restorative Neurology and Neuroscience* 22.3-5 (Jan. 2004). Publisher: IOS Press, pp. 349–358. ISSN: 0922-6028.

[21] Prathap Kumar J., Arun Kumar M., and Venkatesh D. "Healthy Gait: Review of Anatomy and Physiology of Knee Joint." In: *International Journal of Current Research and Review* 12.06 (2020), pp. 01–08. DOI: 10.31782/ijcrr.2020.12061.

[22] Leonard E. Kahn et al. "Robot-assisted movement training for the stroke-impaired arm: Does it matter what the robot does?" en. In: *Journal of rehabilitation research and development* 43.5 (Nov. 2014), pp. 619–630. ISSN: 1938-1352.

[23] Pınar Karakaş and M. Gülhal Bozkir. "Determination of Normal Calf and Ankle Values Among Medical Students." en. In: *Aesthetic Plastic Surgery* 31.2 (Apr. 2007), pp. 179–182. ISSN: 1432-5241. DOI: 10.1007/s00266-006-0132-6.

[24] Ian Kimber et al. "Allergic contact dermatitis." en. In: *International Immunopharmacology*. Occupational Immunology 2.2 (Feb. 2002), pp. 201–211. ISSN: 1567-5769. DOI: 10.1016/S1567-5769(01)00173-4.

[25] A. Koller-Hodac et al. "Knee orthopaedic device how robotic technology can improve outcome in knee rehabilitation." In: *2011 IEEE International Conference on Rehabilitation Robotics*. ISSN: 1945-7901. June 2011, pp. 1–6. DOI: 10.1109/ICORR.2011.5975347.

[26] Serdar Kucuk and Zafer Bingul. *Robot Kinematics: Forward and Inverse Kinematics, Industrial Robotics: Theory, Modelling and Control*. Sam Cubero (Ed.), 2006. ISBN: 3-86611-285-8.

[27] Ioan D. Landau and Gianluca Zito. *Digital Control Systems*. Springer-Verlag London Limited, 2006.

[28] Seth Hutchinson Mark W. Spong and M. Vidyasagar. *Robot Modeling and Control*. JOHN WILEY & SONS, INC., 1989.

[29] Yuya Maruyama, Shinpei Kato, and Takuya Azumi. "Exploring the performance of ROS2." In: *Proceedings of the 13th International Conference on Embedded Software*. EMSOFT '16. New York, NY, USA: Association for Computing Machinery, Oct. 2016, pp. 1–10. ISBN: 978-1-4503-4485-2. DOI: 10.1145/2968478.2968502.

[30] "Medical gallery of Blausen Medical 2014." In: (Aug. 2014). ISSN: 2002-4436. DOI: 10.15347/WJM/2014.010.

[31] Abolfazl Mohebbi. "Human-Robot Interaction in Rehabilitation and Assistance: a Review." en. In: *Current Robotics Reports* 1.3 (Sept. 2020), pp. 131–144. ISSN: 2662-4087. DOI: 10.1007/s43154-020-00015-4. URL: https://doi.org/10.1007/s43154-020-00015-4 (visited on 05/27/2021).

[32] Tobias Nef, Matjaz Mihelj, and Robert Riener. "ARMin: a robot for patient-cooperative arm therapy." en. In: *Medical & Biological Engineering & Computing* 45.9 (Sept. 2007), pp. 887–900. ISSN: 0140-0118, 1741-0444. DOI: 10.1007/s11517-007-0226-6. URL: http://link.springer.com/10.1007/s11517-007-0226-6 (visited on 05/27/2021).

[33] Olesya Ogorodnikova. "Methodology of safety for a human robot interaction designing stage." In: *2008 Conference on Human System Interactions*. ISSN: 2158-2254. May 2008, pp. 452–457. DOI: 10.1109/HSI.2008.4581481.

[34] Open Robotics. *About Quality of Service settings*. https://docs.ros.org/en/foxy/Concepts/About-Quality-of-Service-Settings.html.

[35] Lizheng Pan et al. "Patient-Centered Robot-Aided Passive Neurorehabilitation Exercise Based on Safety-Motion Decision-Making Mechanism." en. In: *BioMed Research International* 2017 (Jan. 2017). Publisher: Hindawi, e4185939. ISSN: 2314-6133. DOI: 10.1155/2017/4185939. URL: https://www.hindawi.com/journals/bmri/2017/4185939/ (visited on 05/27/2021).

[36] Stanley Plagenhoef, F. Gaynor Evans, and Thomas Abdelnour. "Anatomical Data for Analyzing Human Motion." In: *Research Quarterly for Exercise and Sport* 54.2 (June 1983). Publisher: Routledge, pp. 169–178. ISSN: 0270-1367. DOI: 10.1080/02701367.1983.10605290.

[37] Prof. Wyatt Newman, Ph.D., P.E. *Robotics competitions showcase the collaborative power of ROS—the Robot Operating System*. https://insights.roboglobal.com/robotics-competitions-showcase-power-of-ros.

[38] Kathryn Roach and Toni Miles. "Normal hip and knee active range of motion: The relationship to age." In: *Physical therapy* 71 (Oct. 1991), pp. 656–65. DOI: 10.1093/ptj/71.9.656.

[39] Kengo Sato et al. "Anatomical study of the proximal origin of hamstring muscles." In: *Journal of orthopaedic science : official journal of the Japanese Orthopaedic Association* 17 (June 2012), pp. 614–8. DOI: 10.1007/s00776-012-0243-7.

[40] Chathuri Senanayake and S. M. Namal Senanayake. "Emerging robotics devices for therapeutic rehabilitation of the lower extremity." In: Aug. 2009, pp. 1142–1147. DOI: 10.1109/AIM.2009.5229740.

[41] Vinay Setty and Vishal Kumar. "Study of Range of Motion of Knee Joint in South Indian Male Subjects." In: *Journal of Karnataka Chapter of Anatomists* 6 (Jan. 2012), pp. 81–85.

[42] Hayder F. N. Al-Shuka et al. "Active Impedance Control of Bioinspired Motion Robotic Manipulators: An Overview." en. In: *Applied Bionics and Biomechanics* 2018 (Oct. 2018). Publisher: Hindawi, e8203054. ISSN: 1176-2322. DOI: 10.1155/2018/8203054. URL: https://www.hindawi.com/journals/abb/2018/8203054/ (visited on 04/14/2021).

[43] Joel Stein et al. "Comparison of Two Techniques of Robot-Aided Upper Limb Exercise Training After Stroke." en-US. In: *American Journal of Physical Medicine & Rehabilitation* 83.9 (Sept. 2004), pp. 720–728. ISSN: 0894-9115. DOI: 10.1097/01.PHM.0000137313.14480.CE.

[44] Wen Tan et al. "Comparison of some well-known PID tuning formulas." en. In: *Computers & Chemical Engineering* 30.9 (July 2006), pp. 1416–1423. ISSN: 0098-1354. DOI: 10.1016/j.compchemeng.2006.04.001.

[45] GM Van der Zalm. "Tuning of PID-type controllers: Literature overview." In: *Eindhoven* (2004).

[46] Milos Vasic and Aude Billard. "Safety issues in human-robot interactions." In: *2013 IEEE International Conference on Robotics and Automation*. ISSN: 1050-4729. May 2013, pp. 197–204. DOI: 10.1109/ICRA.2013.6630576.

[47] B. Weinberg et al. "Design, Control and Human Testing of an Active Knee Rehabilitation Orthotic Device." In: *Proceedings 2007 IEEE International Conference on Robotics and Automation*. ISSN: 1050-4729. Apr. 2007, pp. 4126–4133. DOI: 10.1109/ROBOT.2007.364113.