Satakunnan ammattikorkeakoulu
Satakunta University of Applied Sciences

HANNU HENTTINEN

From SOLIDWORKS® to ROS simulation

HOW TO SIMULATE YOUR MECHANISM WITH REALISTIC PHYSICS

DEGREE PROGRAMME IN ELECTRICAL AND AUTOMATION ENGINEERING
2022

Author(s)	Type of Publication	Date
Henttinen, Hannu	Bachelor's thesis	09 2022
	Number of pages 34	Language of publication: English

Title of publication
From SOLIDWORKS® to ROS simulation

Title of publication

Abstract

This project aims to provide well-written and easy-to-follow documentation for exporting users' models from SolidWorks to ROS, for continued development and testing, providing feedback for the mechanical engineers, if their design works or if it needs improvements.

This is mainly aimed at mechanical engineering students, but it can be also used in electrical and automation engineering, for students that have prior skills with SolidWorks modelling or are working on a project with mechanical engineers, creating a set of design guidelines to make the project run smoother.

The documentation is shared as a website, which is created using Jekyll static site generator framework and hosted in GitHub Pages, with added automation through CI/CD scripts to automatically generate the webpages, making everything easier to update and modify.

Keywords
ROS, robotics, SolidWorks, programming, mechatronics

CONTENTS

TERMINOLOGY

API	Application Programming Interface
CAD	Computer-aided design
CAE	Computer-aided engineering
CD	Continuous delivery (or continuous deployment)
CI	Continuous integration
ICC	Instantaneous centre of curvature
PDM	Product Data Management
PID	Proportional–Integral–Derivative
ROS	Robot operating system
RPC	Remote procedure call
RViz	ROS Visualization
URDF	Unified robot description format
VM	Virtual machine

1 INTRODUCTION

Robot designing has become easier over time, with the introduction of more affordable pricing range for SolidWorks, which is more accessible for hobbyists and small businesses, to start developing robotic systems or other mechanical systems. Combining SolidWorks modelling capabilities with the open-source meta operating system called ROS, Robot Operating System, you can create physics simulations for the mechanical or robotic systems and evaluate their functionality to improve the designs or material choices.

While working on ROS projects for Satakunta University of Applied Sciences, I had the robots designed in SolidWorks and was required to export them as URDF files. The current state of the documentation that I was able to find, was written from the viewpoint of the developer or someone proficient in the process already and didn't incorporate information on how to start exporting your models and explaining the different options in the SW2URDF plugin.

The main purpose of this documentation is to oversee how I designed my guide to teach other people, mainly mechanical engineers, how to export their designs to ROS for further development through simulations and software design, hopefully bridging the gap between mechanical, electrical, and software development and creating an iterative process for better robot development and better workflow between students.

Documentation is shared through Github pages as a website, that is developed using Jekyll, a tool for creating static sites that are easily updated using Markdown syntaxes for creating formatted pages, that can be uploaded to GitHub using git and are automatically processed through Github actions providing a CI/CD environment for faster updating of the website and making sure, the website is functional and up to date.

2 PROJECTS PURPOSE AND OBJECTIVES

The project started when I was unable to find a decent tutorial, stating how to export complex assemblies from SolidWorks to ROS for simulation. There were simplified instructions that gave the result without explaining, what the different options are, from the standpoint of a mechanical engineer giving the model to a programmer or automation engineer.

Skills required to complete these lessons are a basic understanding of how to use SolidWorks to create the models required, an understanding of Linux since ROS is running on Linux and to be able to troubleshoot, if something is not working, and an understanding of Python, for every script is written in it.

The objectives of the project are as follows:

- Instruct what properties are important when exporting.
- Give examples to follow.
- Create an introductory instruction on how to modify the result, with added functionality.

The goal of the project is to give the knowledge to engineers, on how to create a model of their mechanism in SolidWorks, exporting it to ROS for simulation and software development, streamlining the process of creating robots and similar mechanisms. With future learning and practice, the readers could enter a new field of robotics design or could be even more effective in the field of robotics that uses ROS or similar software.

3 SOFTWARE USED IN THE PROJECT

3.1 SolidWorks by Dassault Systems

SolidWorks was founded in 1993 by Jon Hirschtick and recruited software engineers, to develop more accessible 3D CAD software. They achieved this by developing it for the Windows platform, without the requirement of expensive hardware or software to be operational. The first version of SolidWorks was released in 1995, gaining praise for its ease of use, enabling more engineers to benefit from 3D CAD in product design. In 1997 Dassault Systèmes S.A. acquired SolidWorks and has continued the development of the software, having ease of use as a core value for the new expansions in simulation and PDM. (Dassault Systèmes SOLIDWORKS Corp., 2018.)

SolidWorks is a CAD/CAE software, known for its widespread use in industrial use and hobby projects. The user-friendly interface helps with the design process and gives an easy way to understand, what is happening. With the parametric approach, it is easy to design something and retroactively change aspects of your design, by changing a few key values, these changes drive from part level to full assembly level (Figure 1), if designed properly. To create a 3D object, a designer only needs to use basic geometric shapes in 2D to start their project. (Shakurova, 2019, pp. 7–8.)

SolidWorks is currently the default software, which is taught at Satakunta University of Applied Sciences, for mechanical engineering students. The licence is also open for everyone, who is studying in SAMK, whether they are mechanical engineers or not, through the BYOD service.

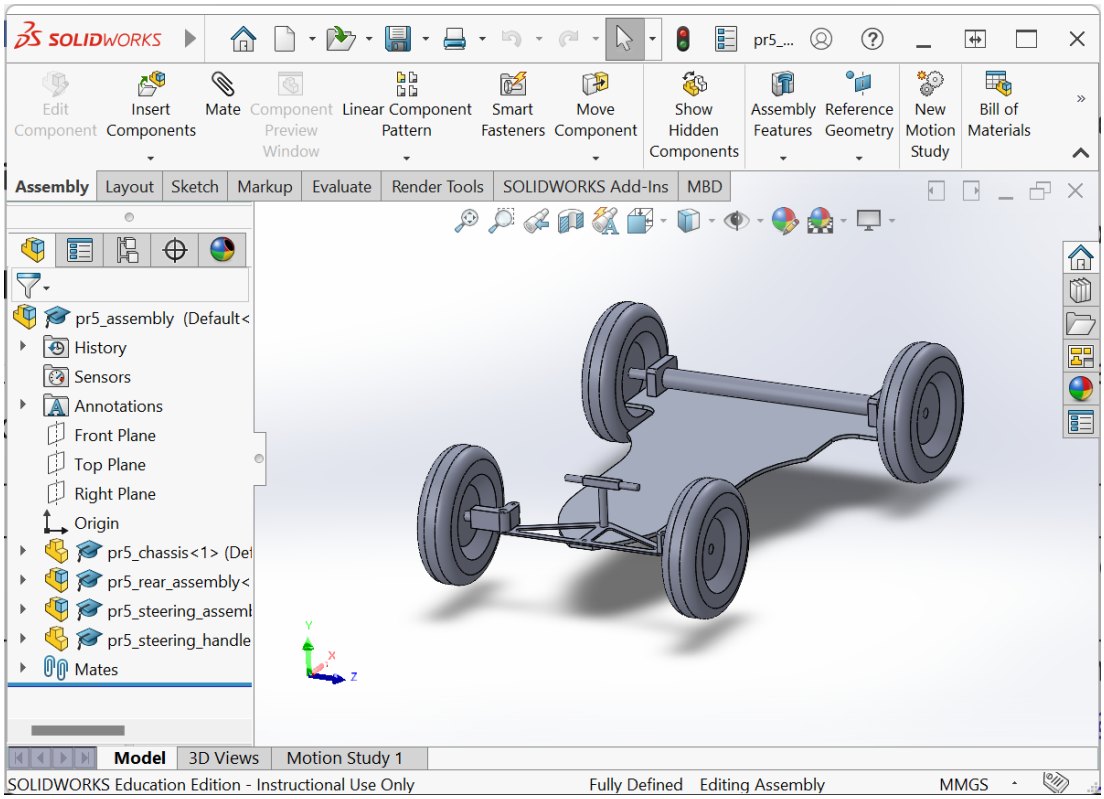


Figure 1. Practice 5 completed assembly in SolidWorks.

3.2 Robot Operating System

ROS is a meta-operating system, which is comprised of multiple open-source software frameworks that are specifically designed for robot software development. It provides services such as hardware abstraction layers (HAL), device controllers, message passing between programs, and package management. ROS runtime method is a peer-to-peer network of processes, which can be divided into other machines, which are loosely coupled with ROS communication node infrastructure. This is achieved by implementing assorted styles of communication, e.g., RPC services, topics that stream data asynchronously, and XML files that are served via a parameter server. (Open Source Robotics Foundation, 2018, sec. What is ROS.)

The goal of ROS is to be a platform, where you can easily create a robot, and reuse old code that you or someone else has created, reducing time spent on re-creating the wheel (see Figure 2). Code that runs with ROS, can be created with different programming languages, but most of the code is done in either Python or C++, as they are fully

implemented in the codebase of ROS. (Open Source Robotics Foundation, 2018, sec. Goals.)

Figure 2. Comic by Jorge Cham creator of "Piled Higher and Deeper" (www.phdcomics.com), illustrating the robotics research re-invention method (Willow Garage & Cham, 2010).

Physical models are generated using an URDF file, which is an XML formatted file, describing various parts of the model. The models are rigid bodies, consisting of links

that are connected to other links with joints. The model is also structured in a tree structure, which rules out parallel robots. URDF models contain the visual representation, kinematic and dynamic information, and collision model of the robot (Open Source Robotics Foundation, 2012).

3.3 Jekyll framework

Jekyll is a framework for creating static websites, using Markdown and Liquid, to create static webpages that can be easily edited (The Jekyll Team, n.d.).The reason I decided to use Jekyll to create the website was, which is natively supported by GitHub Pages, which will host the website and gives the tools for version control (GitHub Inc., n.d.).

For the theme of the site, I am using the TeXt Theme that I found via Google searches, it's design with a sidebar brings usability that other developers have in their documentation, making the browsing much easier. This is illustrated in Figure 3 below.

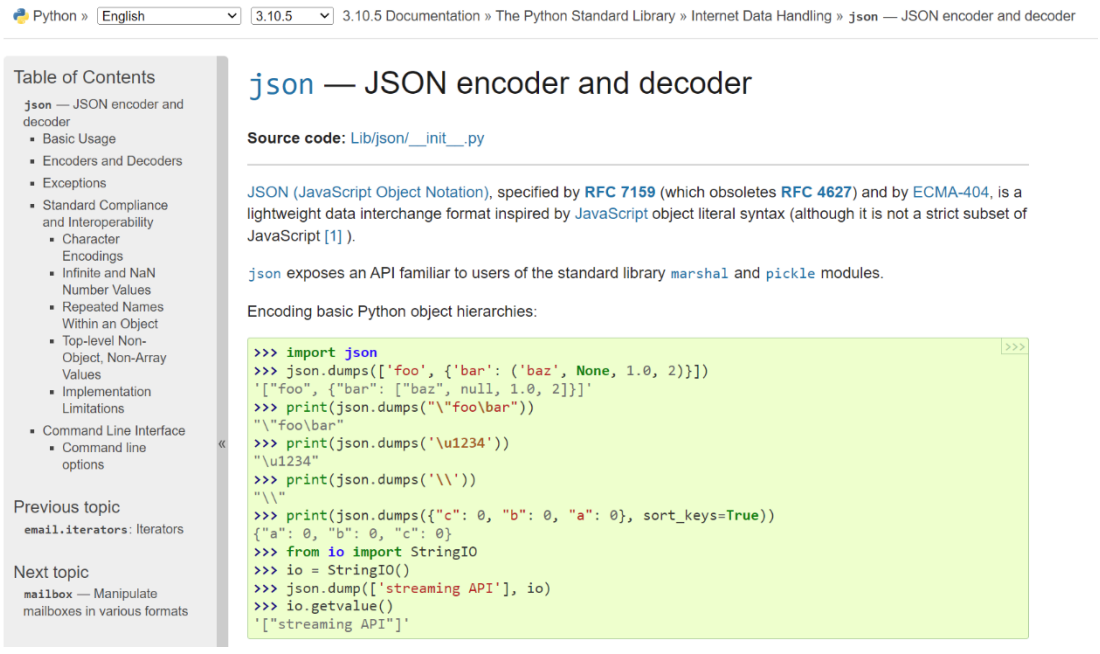


Figure 3. Python 3 documentation page by Python Software Foundation.

4 PLANNING AND IMPLEMENTATION

4.1 Different tasks in the project

All the lessons, that are going to be created in this project consist of creating the model for the practice and evaluating the model for functionality. If the functionality is missing some aspects, the model needs to be adjusted as needed, for example, increasing an angle, and scaling parts. After the model has been made, the next step is to create the drawings of the parts and assemblies, so that students can create the models for their practices.

When introducing a new UI element or option in the documentation, it requires a screenshot, showing the students what it looks like and where it is located. This is also a perfect time to take screenshots of the parts, making sure that all the assemblies will work as intended.

After exporting, it is time to assess the URDF files, to see if all the axes are set properly and everything is working as it should. At this time, the files are also modified a little bit, to add colours to distinct parts, making them pop on the screen and having a clear distinction between parts, and physics parameters are set so that the parts behave as they should and do not break the illusion of a working mechanism.

The final part that is done for the practice, is creating the scripts that the practice requires. These are created, evaluated, and commented, to guide the student to understand what the script is doing, how the object works and what the different settings do.

After everything is working, it is time to write the documentation for the practice, making sure that all the screenshots are easy to read, the code is formatted correctly and, the webpage works.

4.2 Project schedule, timeline, and resources

The documentation concepts were created while working on a workable model for ROS, from SolidWorks models that were created by mechanical engineering students. I gave myself around 6 months as a schedule, to create the documentation.

There is no finite resource on the project, as everything is digital and can be moved to other computers with an external hard drive or version control applications, utilising GitHub's repositories. Using the university's BYOD service, I can use SolidWorks at home and use a virtual machine, created with VMWare Workstation Player, to run Linux with ROS.

Challenges might come from when some pieces of ROS are updated, requiring either changing some parameter names or rewriting the functionality from scratch. There might also be issues with using virtualized Linux, without a dedicated graphics card and sharing the processor load with already limited capacity, grinding everything down to a slow pace. An optimal setup would be to have two separate computers, one running Ubuntu Linux for ROS, another one running Windows for using SolidWorks, never updating the software.

5 PROJECT EXECUTION AND LESSON STRUCTURE

5.1 Core concept of the project

The core concept of teaching is to convey information that the student can use at later date and expand with further studies. The order of the practices is to first introduce the student to exporting a simple model and what parameters can be set in the software. From this, the student moves to more complex models and pieces of code, which challenge them to grasp the information and recall previous information.

The main structures of the lessons are structured so, that first, we introduce the drawings of the parts and assemblies that the student use to create the models. This requires the basic knowledge of using SolidWorks. After this, it is shown how to insert axial and coordinate information, so that when the model is exported, it will contain correct information about where the Gazebo simulation will place the objects and what orientation. After exporting the model, it is tested and extra functionality or controlling scripts are added to it.

5.2 Practice 1 – The simple wheel

The first lesson is a set of wheels, spinning through their centre axis that is perpendicular to the direction of rotation. This is one of the simplest models to make since it only consists of a simple joint that is rotating on an axle.

I wanted to create this, for the sole purpose of giving a basic understanding of how to set a wheel on an axle, give the wheel a rotational speed and see the physics engine working on it. This information is crucial when trying to design a mobile robot platform, which runs on wheels.

The modifications that were made to the model are mostly visual, but we do add a new link and joint so that the model does not move under the forces applied to the wheels. The effect of not locking the base to the world is demonstrated in the lesson and shows, why it is a promising idea to do so. Cosmetic changes are made, to make it easier to visualize the parts moving and have a distinction between them.

In the python script, we create a simple loop that applies a steady force on every wheel, using a ROS service call that was demonstrated earlier through the command-line interface. Figure 4 shows a tool from Gazebo, called Plotting Utility, showing the wheel velocity.

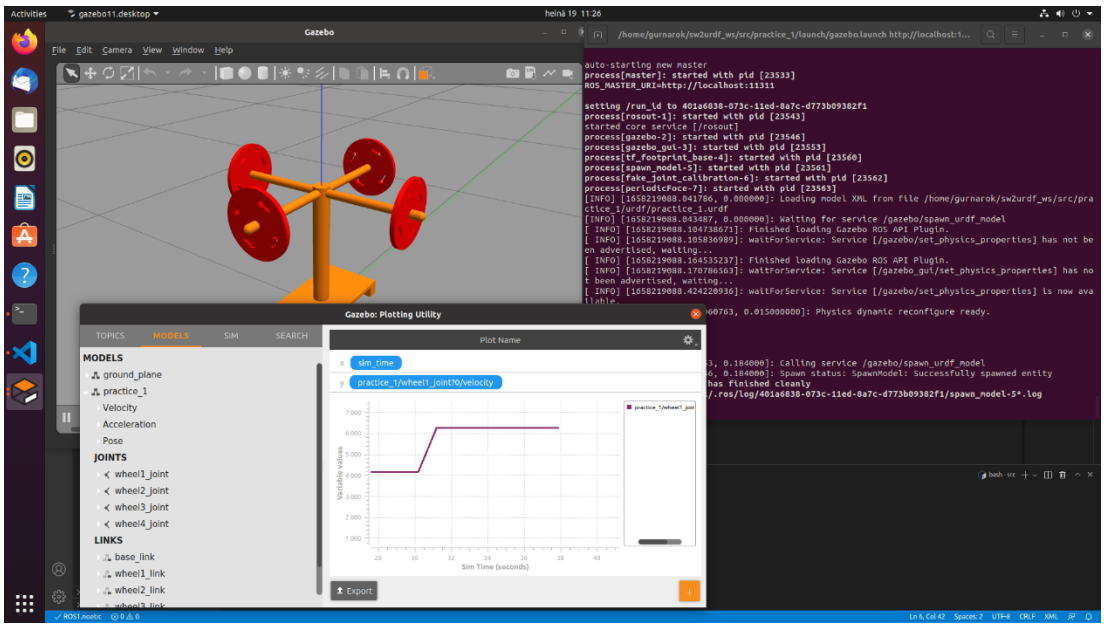


Figure 4. Testing that Practice 1 works.

5.3 Practice 2 The simple wheel, but friction and an angle

The second lesson is a modification of the first lesson, adding an angle to the brackets and demonstrating how easily a simple axial system can start to require more setup before exporting it as a URDF for simulation.

After the exporting, we evaluate the model to see that the export was done properly and that all the axes are aligned correctly and rotating. If the test is successful, the same modifications that lesson 1 had, will be done to the lesson 2 model, but this time, we add dynamics to the model in the form of static friction in newton-meters. The effect that the friction has is demonstrated by a plot line, displaying the wheel velocity of two different wheels, where one has no friction force applied and the second one has a friction force of 0.7 Nm. The lesson also demonstrates that it is possible to reuse code from other packages that have been already made, without the need to waste a lot of time for no reason, which is the reason why ROS was created. Figure 5 shows the effects, of what happens when the base is not locked into the world plane.

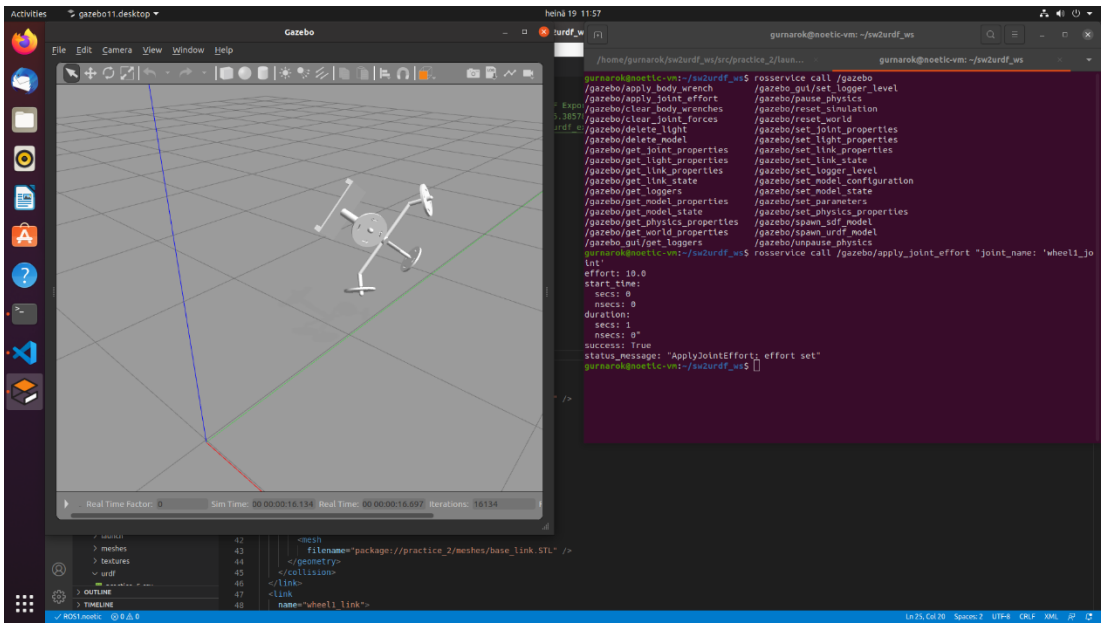


Figure 5. Forgetting to lock the model to the ground plane, before applying force.

5.4 Practice 3 – A 2-axis robot arm

With this lesson, the goal is to demonstrate the hierarchy of links, building a system where one part's movement, affects another part's position and movement range. With the model, we also demonstrate the difference between continuous joint and revolute joint, which are simply put a motor and a servo.

In exporting to URDF, I added a simple JavaScript code, which converts the user's input value from degrees to radians, with the formula that is used to convert it. I also try to show through trigonometry, how to view the final angle of the end effector.

This lesson also introduces limits, transmissions, and controllers. With limits, it can be set what the maximum and minimum angle/position is for a joint, the maximum effort (or force), and the maximum velocity for either movement or rotation. The transmissions and controllers work as an interface between calculating velocity, effort, and position values, needed to achieve a set position, velocity, or effort. With these controllers, there is also a requirement to set PID values.

When testing the model, we come across the inability to load controllers because they are not installed, this emphasizes reading the output of launch logs and thinking are

there missing components that need to be installed. This is also the first time the topics are used to set values and the values are remapped in the launch file. The main difference between topics and services is that services return a value, but topics do not. Figure 6 shows that the errors are displayed in the logs, as warnings and if something does not work, you should read the logs carefully.

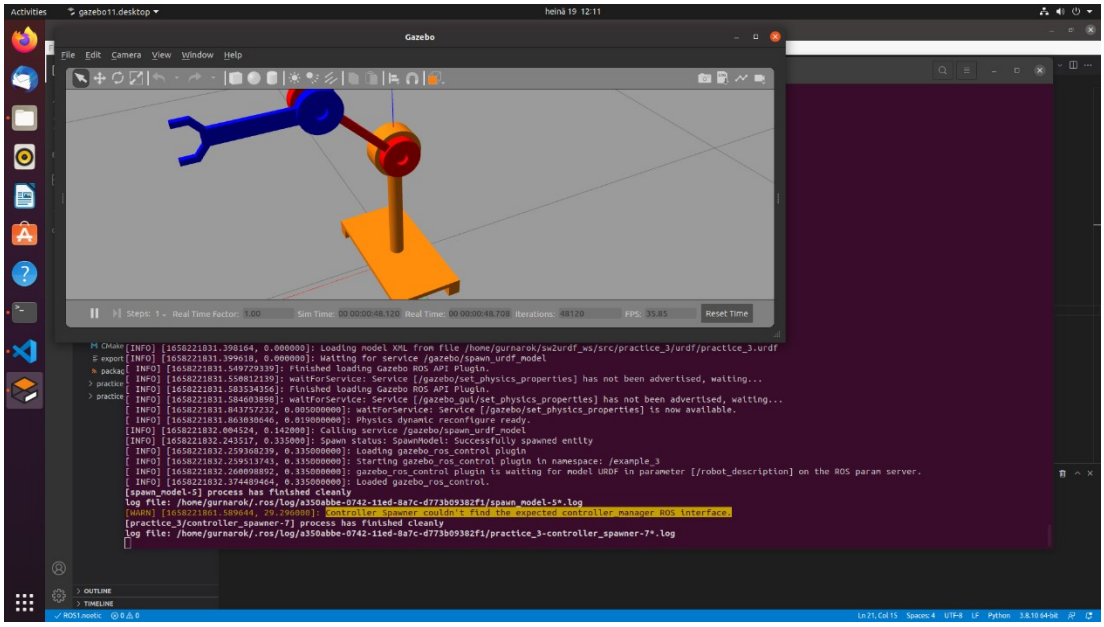


Figure 6. The warnings can sometimes be hard to spot, after a long day of work.

5.5 Practice 4 – Linear movement through a bar

Mechanically, this lesson is not that complex, just a simple bar with an object in it, that can be positioned along the axis of the bar, setting the transmission and controllers to work with positional data, instead of rotational data. Figure 7 shows, that setting the position limit is crucially important, to achieve the correct functionality.

This model can also be used to demonstrate the increased effort required to move an object when friction increases, or the limiting factor when the desired speed is achieved, not utilizing the maximum force a motor can give. The main lesson for this is to show that there are GUI options for sending messages to a topic.

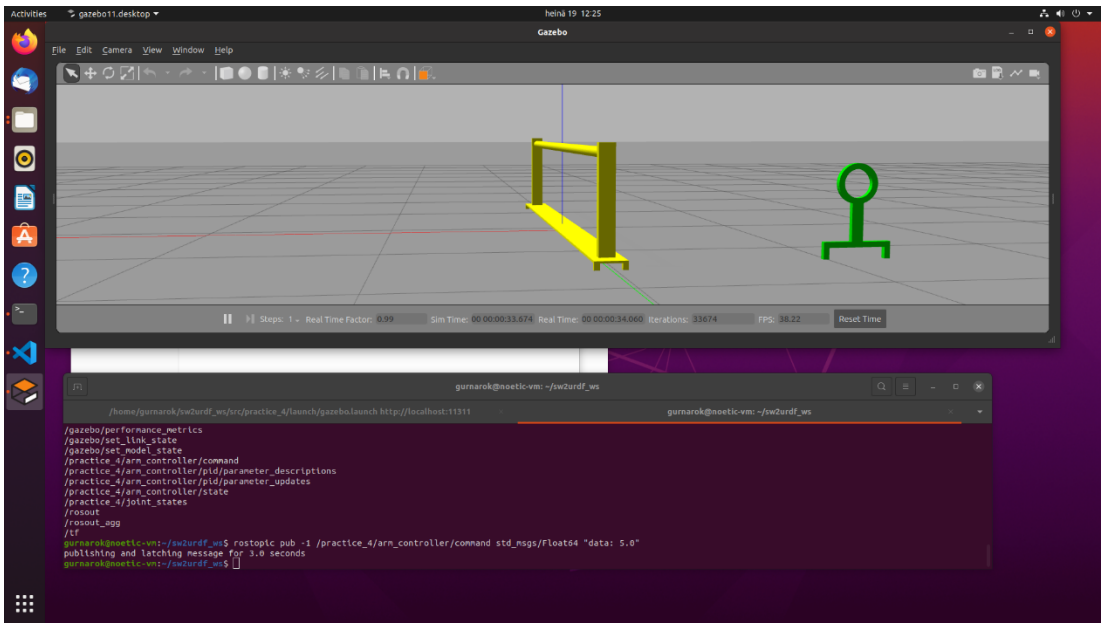


Figure 7. Typos when setting limits can lead to interesting results.

5.6 Practice 5 – A simple vehicle

This is the most complex lesson, where the idea is to create a simple vehicle that has a steerable driving system. The model contains multiple parts and a complicated transmission set for the wheels and steering. This lesson is supposed to be a final test, combining multiple skills learned throughout the previous lessons. Figure 8 shows the visual information added to the model, to make it easier to identify different links and joints.

The control scheme is using Ackermann steering, which is used in car-like vehicles, where the inner wheel is at a sharper angle than the outer wheel. To achieve this, we use the ackermann_steering_controller and create a script, which will orient the wheels as needed.

In the end, the car moves, but not ideally. The documentation for the ackermann_steering_controller is lacking and going through what is given, it is unsure if a key value is missing or one of the values supplied, is too large or too small. It could also be, that the script that was created, is interfering with the controller.

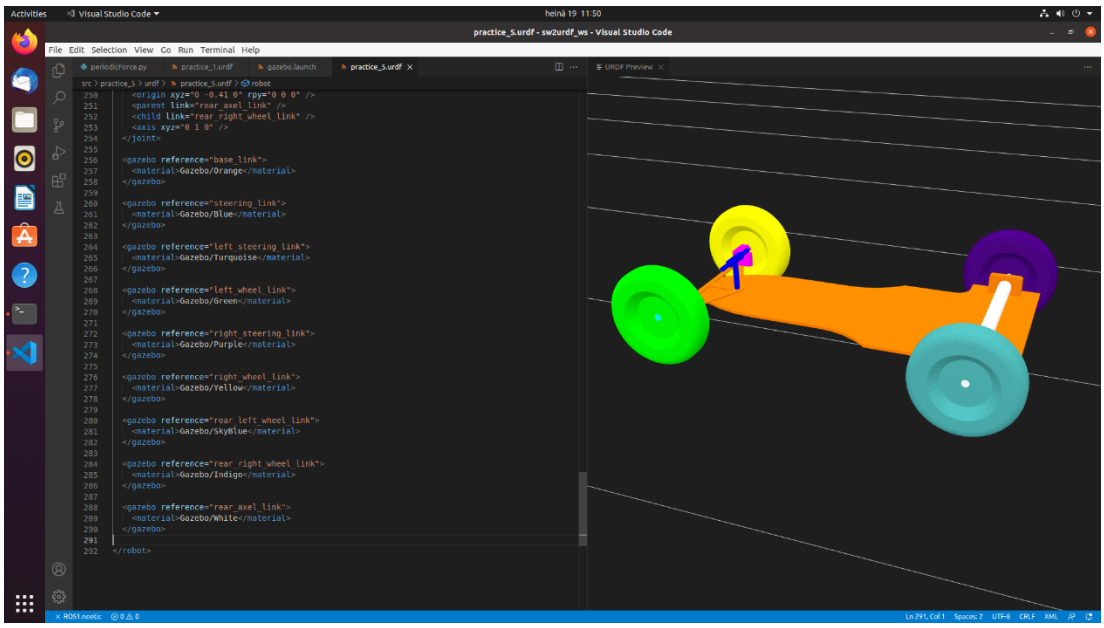
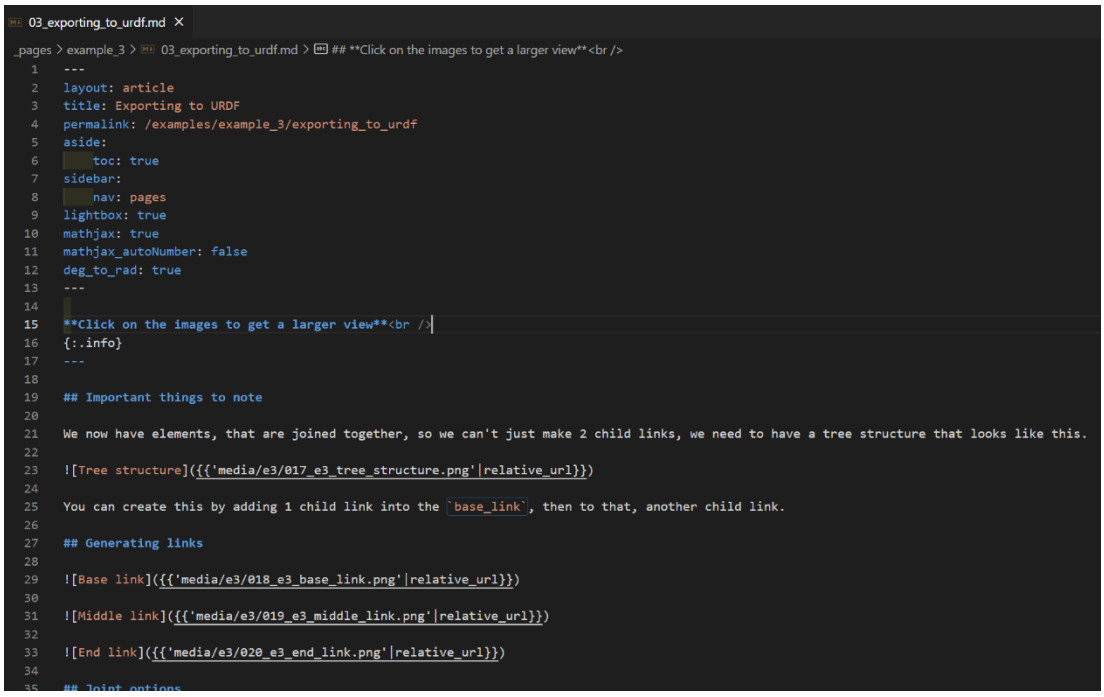


Figure 8. Previewing changes to URDF in Visual Studio Code, with ROS extension installed.

5.7 Writing everything down

Working with Markdown is simple, just write the text and by adding basic keywords or marks, it can create a complex-looking webpage generated by Jekyll, running either on the local machine or on a server. See Figures 9 and 10



```
---
layout: article
title: Exporting to URDF
permalink: /examples/example_3/exporting_to_urdf
aside:
    toc: true
sidebar:
    nav: pages
lightbox: true
mathjax: true
mathjax_autoNumber: false
deg_to_rad: true
---

**Click on the images to get a larger view**<br />
{:.info}
---

## Important things to note

We now have elements, that are joined together, so we can't just make 2 child links, we need to have a tree structure that looks like this.

![Tree structure]({{'media/e3/017_e3_tree_structure.png'|relative_url}})

You can create this by adding 1 child link into the `base_link`, then to that, another child link.

## Generating links

![Base link]({{'media/e3/018_e3_base_link.png'|relative_url}})

![Middle link]({{'media/e3/019_e3_middle_link.png'|relative_url}})

![End link]({{'media/e3/020_e3_end_link.png'|relative_url}})

## Joint options
```

Figure 9. A screenshot of Markdown syntax for Lesson 3 - Exporting to URDF

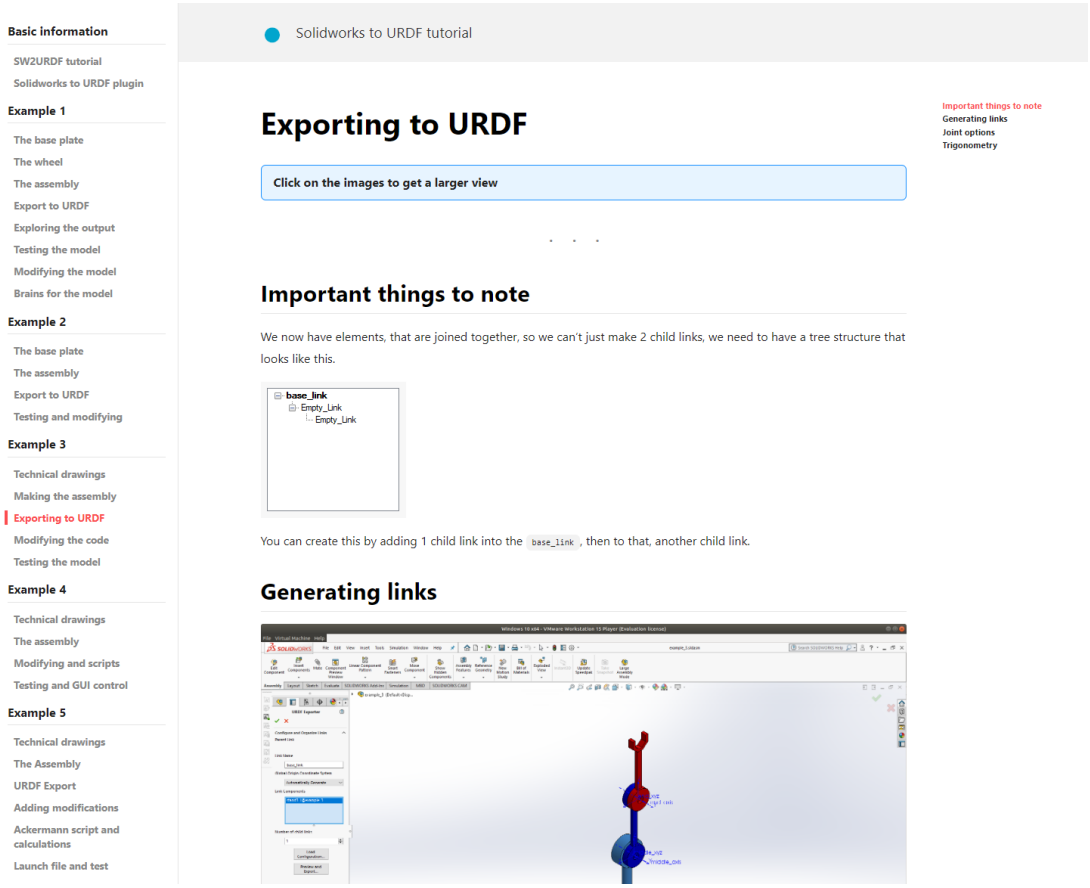


Figure 10. A screenshot of what the Markdown page looks like when rendered.

During the whole writing process, Jekyll is installed in an instance of WSL (Windows Subsystem for Linux), that allows running Linux command-line tools through Windows, without the overhead of a normal virtual machine. With this setup, I can write the documentation and easily check, if the formatting or sample code is working correctly, without needing to upload it to GitHub (see Figure 11), wait for an action to run and convert the pages into a static form and publishing it for everyone to see.

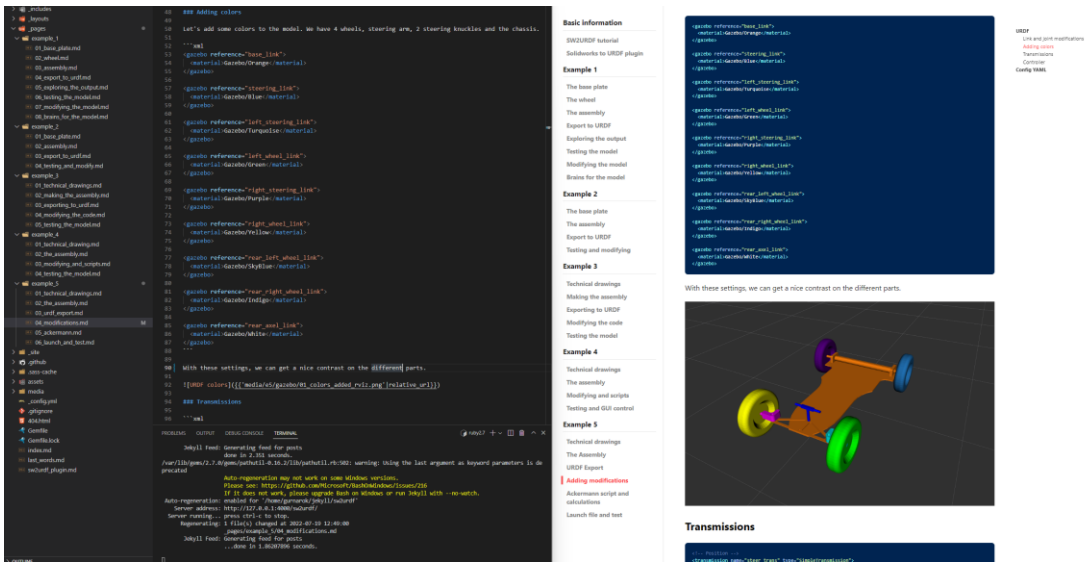


Figure 11. Running a local version of Jekyll makes editing easier.

6 MODELS AND CODE FOR THE PRACTICES

6.1 Structure of the practices and code

This chapter contains an abridged version of the written instructions from the webpage generated. The structure follows the plan that was established in Chapter 5 but is opened to show the equations used and give more information about the individual practices.

6.2 Practice 1

The first practice uses a simple geometry, where everything is lined on a straight axis to each other. The main aspect of this example is to see, how joints are created and how the continuous joint type works.

6.2.1 Modelling

The practice starts by creating the model in SolidWorks, from given engineer drawings that can be found in Appendix 1 and Appendix 2.

When creating the assembly, we give instructions on reference geometry that is required for proper function in ROS, stating the axis of rotation for the two pairs of wheels.

6.2.2 Exporting the model

When the assembly is complete, we go through the export process and state the different options, based on Stephen Brawner's tutorials and personal experience with the software. (Brawner, 2020.)

In the explanations of the different limits for the joint, we must convert values from degrees to radians, which is stated in the ROS wiki for URDF joints (Open Source Robotics Foundation, 2018). We do this with Equation 1.

$$\theta\,^{\circ} = \theta * \frac{\pi}{180^{\circ}} \approx \theta\ rad$$

Equation 1. Converting angle from degrees to radians.

There is also an option, for limiting the velocity of the joint, which must be implemented using radians per second. We have two ways to calculate it, either from RPM or from meters per second. The calculations can be done with Equation 2 and Equation 3.

$$\omega = \frac{2\pi}{60} * x\ [RPM] \approx \omega\ {}^{rad}/_{s}$$

Equation 2. Converting RPM to radians per second

$$\omega = \frac{v\ [{}^{m}/_{s}]}{r\ [m]} = \omega\ {}^{rad}/_{s}$$

Equation 3. Converting meters per second to radians per second.

Moving from joints, we get to set link properties, where through SolidWork's API, the origin point, moment of inertia and mass is filled for every link and can be edited, if needed.

The model can be exported in either a URDF file only or with meshes, which will be in STL format. The mesh detail can be changed from coarse to fine, giving a smoother appearance, if it is required or wanted.

6.2.3 Exploring the output for the first time

After exporting the model, we browse through the output, explaining the different files and the folder structure that the plugin has created. We go over a minority of the commands that are in the CMakeList.txt and package.xml files.

6.2.4 Testing and modifying the model

We instruct the user to build a specific folder structure to compile the examples as individual ROS packages and test that the models are displayed correctly in RViz and Gazebo. If everything works as is supposed to, we modify the URDF to add colour to the model, making parts differ from each other.

6.2.5 Making a custom node

We create a simple Python script, which pauses the Gazebo simulation, applies 1 Nm of force to all wheels for 1 second, and then continues the simulation. The cycle repeats every 10 seconds, having a frequency of 0,1Hz.

After creating the script, we change it to executable and add it as a node to the Gazebo launch file. From this, we move to Practice 2.

6.3 Practice 2

We continue with a similar build as in Practice 1 but add an angle to the wheels, and friction to the joint physics.

6.3.1 Modelling

The engineer drawings can be seen in Appendix 3. We are going to use the same wheels, as in Practice 1, so that we do not need to make them again. Differing from Practice 1, the base has a 45° angle in the arms. This change is to demonstrate the importance of setting the axis for the joints properly, with the corresponding origin.

6.3.2 Exporting the model

The exporting of the model follows the same process as in Practice 1, but here, it is important to notice that the axis of rotation is not in a concentric line with two wheels, so each wheel needs its axis of rotation and a coordinate point for the rotation.

6.3.3 Testing and modifying the model

We make the same tests as with Practice 1, creating the package and adding colours to the Gazebo view, but this time, we add friction to the wheels. For this example, we add a high dynamic friction value of 0.7. When using this, with the periodic force applier that was created in the first practice, we can use Gazebos plotting utility, to visualize the joint velocity and see a clear change between joints that have a friction value and those that do not.
To get a periodic force applied to the wheels, we can use the same script from Practice 1 by adding it to the launch file.

6.4 Practice 3

We will create a simple manipulator with fixed fingers for the third practice. This practice will show us, how child links and joints are controlled.

6.4.1 Modelling

The engineer drawings can be found in Appendix 4, 5 and 6. The assembly drawing is in Appendix 7.

In the documentation, there is a set of images, stating how to assemble the manipulator and how to set the joints to be revolution based.

6.4.2 Exporting the model and trigonometry

We are using set limits for the joints, of 45° angle, which is equal to 0.78540 radians, when using Equation 1. When we set both joints with this limit, we can achieve a maximum angle from the centre pivot to the end of the arm, which is 67.5°, which is illustrated in Figure 12. The angle limit is set as a maximum of 0.7854 rad and the minimum is -0.7854 rad.
We also limit the joint movement speed to 2 rad/s, which is approximately 0.9 m/s at the tip of the tool. The joint effort is limited to 1000 Nm of torque, which is enough for a small model like this.

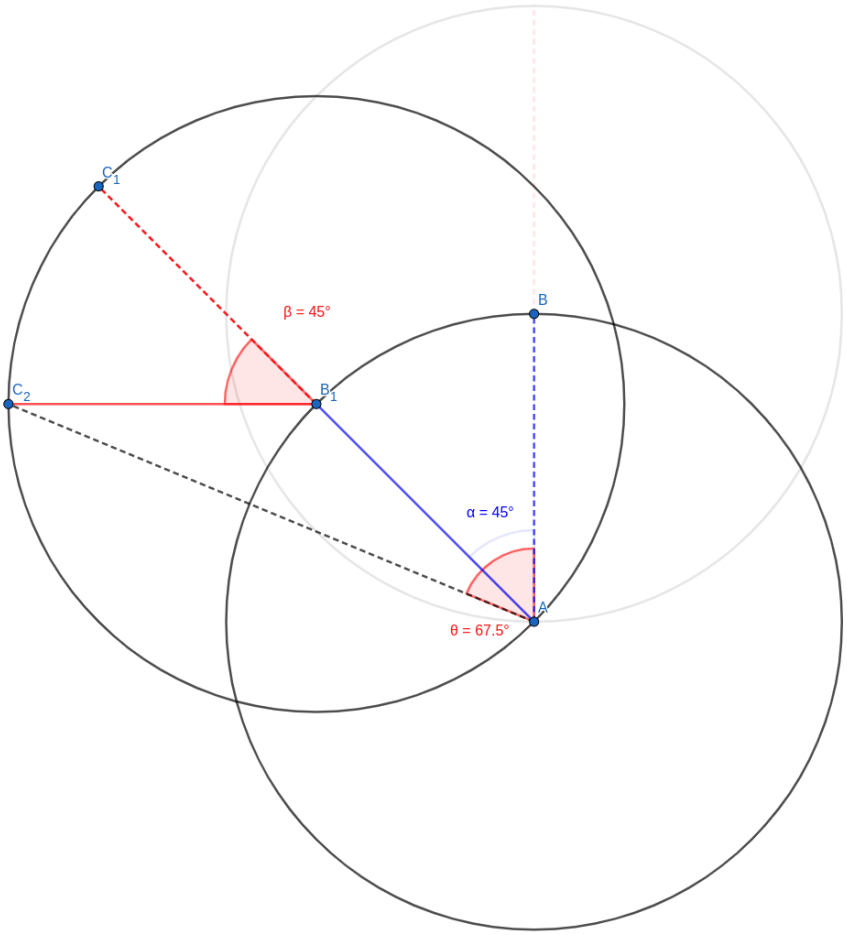


Figure 12. Illustrating manipulator range of movement with unit circles

6.4.3 Modifying URDF

The arm will have set limits and thus, need to be controlled with feedback. To get this working, we will introduce a transmission interface to the model that will provide a simple topic to set the desired position for the hand. We also go through several types of transmission interfaces, which are provided in ROS.

6.4.4 Configuring controller

By modifying joint_names_example3.yaml, we can set parameters for the Gazebo and ROS controllers, setting the namespace, controller name, controller type, controlled joint and PID values.

The two controllers we are using, are,

- Joint state controller, this publishes the joint positions, angles, velocities, and forces that are applied to them. The information is published 20 times per second.
- For the arm and hand, we are using an effort controller, a subset being a joint position controller, which lets us give an angle we want to set the joint and the controller applies torque to the joint, moving it to the desired position. The effort controller also takes in PID values, for a smoother operation and stops the value at the desired position.

6.4.5 Reading topic outputs and analysing it

When we launch the test environment and list topics, we can see two topics, called end_arm_controller/command and middle_arm_controller/command, both take input in std_msgs/Float64 format, stating that the input is a decimal number.

To know where the arm is currently positioned, we echo *_arm_controller/state, which returns us information, but we are only interested in seeing the setpoint, current position and the error value, which is setpoint – current point.

6.4.6 Writing a simple movement script

We write a simple script that gives random positions to both joints in a set interval to control the arm. To achieve this, we publish to end_arm and mid_arm topics a value between -0.7854 and 0.7854 at a 0.2Hz rate, thus changing the value every 5 seconds. We redirect the end_arm and mid_arm topic output into the correct topics, in the launch file.
When the example is launched, it should start moving the arm into random positions every 5 seconds and these can be monitored by echoing joint state topic or by using the rqt_plot application.

6.5 Practice 4

To this point, every practice has used a revolute joint and we wanted to show the prismatic joint, which is a linear motion in a single axis.

6.5.1 Modelling

The engineer drawings for the model can be found in Appendix 8 and Appendix 9. If this was a real device, it would need to be modified to be able to put the moving arm into the axel.

6.5.2 Exporting the model

The assembly goes the same way as with the previous practice, this time when selecting the moving arms joint, we select prismatic instead of revolute. Prismatic joints are linear movements in one plane.

6.5.3 Modifying the URDF

We add colour to the two parts and add joint limits, setting the lower limit to 0 meters and the upper limit to 2.69 meters. The limit is taken from the origin of the part, which is in the midplane of the arm.

6.5.4 Testing and scripting

The scripts and others go in the same way as with previous practises, we configure controllers and set dynamic limits to something that might be a fair value. But instead of using a python script, we are going to use rqt_ez_publisher, which gives a slider that can be used to move the linear arm from one side to the other.

6.6 Practice 5

This practice is the most complex since it contains multiple parts, with coordinate points and multiple axes. This example also contains more math and trigonometry than the rest. For this example, it was decided to use a 4-wheeled robot, which uses Ackermann steering geometry.

6.6.1 Modelling

The engineer drawings to create the parts, are from Appendix 10 to 16 and the assembly drawings are from Appendix 17 to 19. This type of steering was chosen for the use-cases of using an existing steering mechanism from a car or other suitable vehicle and giving large robots steering that would not require multiple motors per wheel, but a single servo to steer.

6.6.2 Ackermann steering geometry

Ackermann steering consists of the concept of rotating the inner wheel more, than the outer wheel, to reduce wheel slippage. In Figure 13, it is shown what the right-angle triangles look and in Equation 4 they are calculated using known lengths from the model.

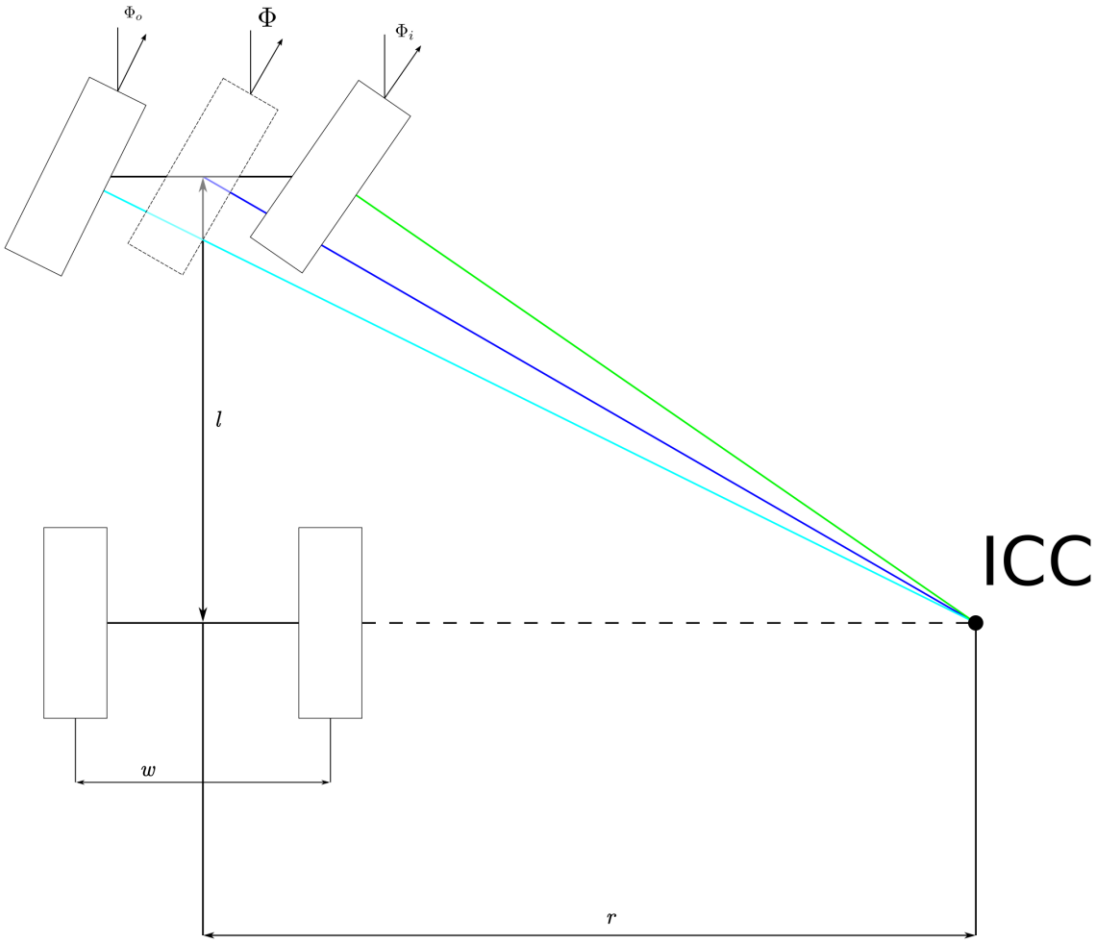


Figure 13. Illustration of the right-angle triangles, required to calculate wheel angles (Eisele, 2020).

$$\Phi_i = \arctan\left(\frac{2l\sin\Phi}{2l\cos\Phi - w\sin\Phi}\right)$$

$$\Phi_o = \arctan\left(\frac{2l\sin\Phi}{2l\cos\Phi + w\sin\Phi}\right)$$

Equation 4. The calculation for the inner and outer wheels angle (Eisele, 2020).

With this equation, the wheel maximum angle can be calculated and set as a limit in the radial joint for the steering knuckle.

6.6.3 Python script

The script created for the wheel angles is simple, reading the angle of the centre steering joint and calculating the correct angles for the wheels using Equation 4.

Since the robot car is a 4-wheel drive, the script also reads the user input for the velocity and sends the data to the wheel controllers. In the launch file, the commands are redirected to the correct topics.

7 CONCLUSION

In the 6-month timeframe, I got the models and code working in a couple of months but writing it open into the Jekyll took more time and debugging the code afterwards when I noticed some of the practices were not working in a clean install of a ROS platform. This required me to go through each practice, writing down what extra packages need to be installed.

The tutorial has not been public access yet, but hopefully, it will be used and given feedback by users. The updating process is simple since I would only need to modify the existing documentation and upload it to GitHub, where there is an automatic action to convert the Jekyll files to a working webpage, which will be hosted on GitHub pages.

The documentation is a living project, where I will update existing practices or add new ones if I see a fit for them. The updates are going to focus on making the documentation more comprehensive and fixing mistakes, I may have missed.

Overall, I am pleased with the outcome of the documentation and the visual style is in my opinion, clean and easy to read, with easily accessible links to share with other people. To make the site more easily shareable and modifiable, the code and models are shared under Gnu General Public License v3.0, which has conditions of sharing the complete source code of licensed work and modifications, which include larger works that use the same license and the licensed work. Table 1 shows the permissions, conditions, and limitations that the license creates.

Table 1. License permissions, conditions, and limitations of GNU GPL v3.0 (Github Inc., n.d.).

Permissions	**Conditions**	**Limitations**
Commercial use	Disclose source	Liability
Distribution	License and copyright notice	Warranty
Modification	Same license	
Patent use	State changes	
Private use		

REFERENCES

Brawner, S. (2020, October 16). sw_urdf_exporter/Tutorials/Export an Assembly - ROS Wiki. Retrieved 14 July 2022, from https://wiki.ros.org/sw_urdf_exporter/Tutorials/Export%20an%20Assembly

Dassault Systèmes SOLIDWORKS Corp. (2018, March 10). Company history. Retrieved 6 June 2022, from https://web.archive.org/web/20180310140957/http://www.solidworks.com:80/sw/656_ENU_HTML.htm

Eisele, R. (2020, September 18). Ackerman Steering • Computer Science and Machine Learning. Retrieved 27 July 2021, from https://www.xarg.org/book/kinematics/ackerman-steering/

GitHub Inc. (9999). About GitHub Pages and Jekyll - GitHub Docs. Retrieved 7 July 2022, from https://docs.github.com/en/pages/setting-up-a-github-pages-site-with-jekyll/about-github-pages-and-jekyll

Github Inc. (9999). GNU General Public License v3.0 | Choose a License. Retrieved 29 August 2022, from https://choosealicense.com/licenses/gpl-3.0/

Open Source Robotics Foundation. (2012, June 19). urdf/XML/model - ROS Wiki. Retrieved 12 June 2022, from http://wiki.ros.org/urdf/XML/model

Open Source Robotics Foundation. (2018, August 8). ROS/Introduction - ROS Wiki. Retrieved 2 June 2022, from http://wiki.ros.org/ROS/Introduction

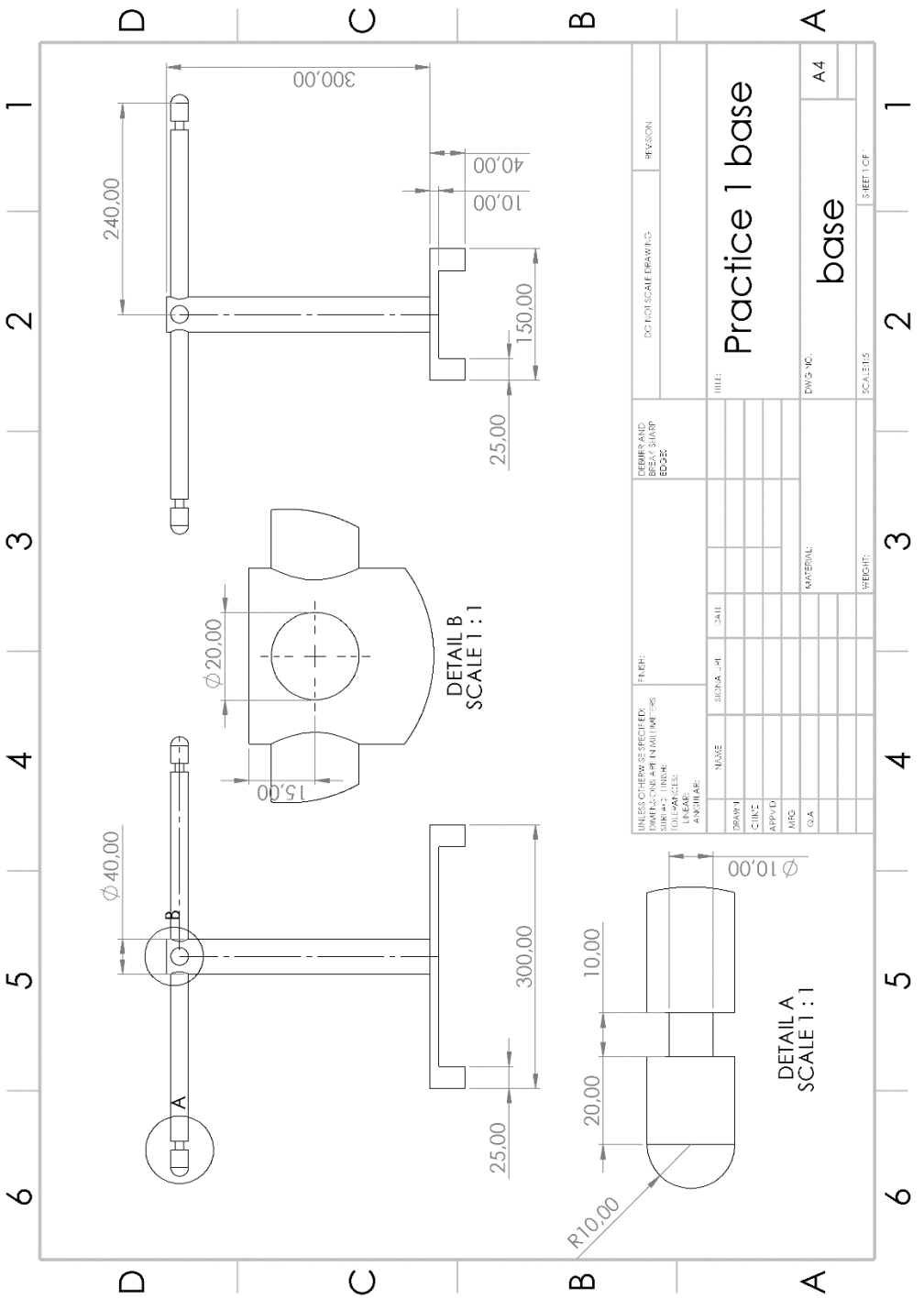
Shakurova, A. (2019). SolidWorks Bottom-Up versus Top-Down Approaches. The Capacity of the Top-Down Modelling. [Bachelor's thesis, Häme University of Applied Sciences]. https://urn.fi/URN:NBN:fi:amk-2019052411791

The Jekyll Team. (9999). Jekyll • Simple, blog-aware, static sites | Transform your plain text into static websites and blogs. Retrieved 7 July 2022, from https://jekyllrb.com/

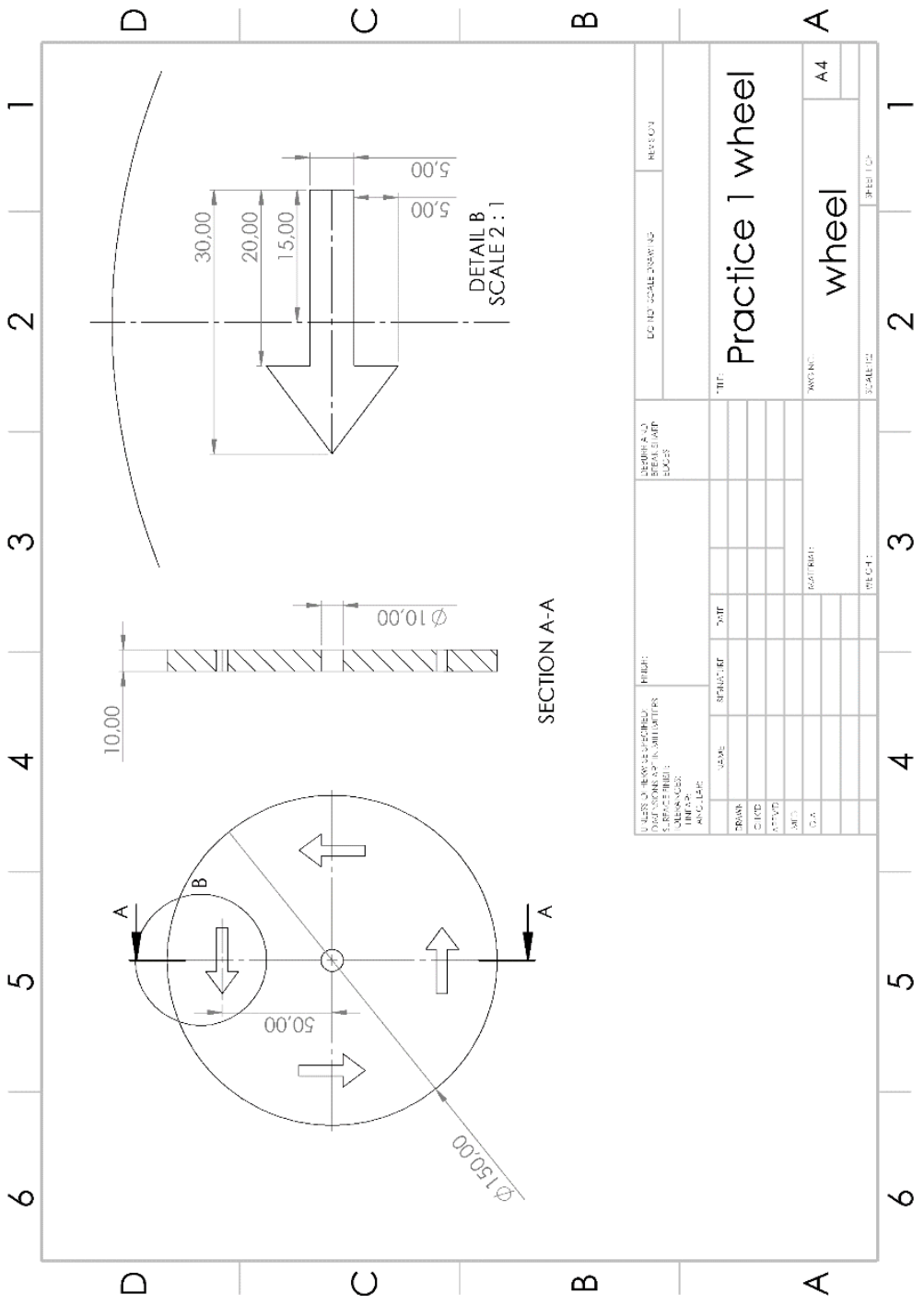
Willow Garage, & Cham, J. (2010, April 27). Comic: Reinventing the wheel. Retrieved 7 July 2022, from

https://web.archive.org/web/20181014170949/http:/www.willowgarage.com:80/blog/2010/04/27/reinventing-wheel

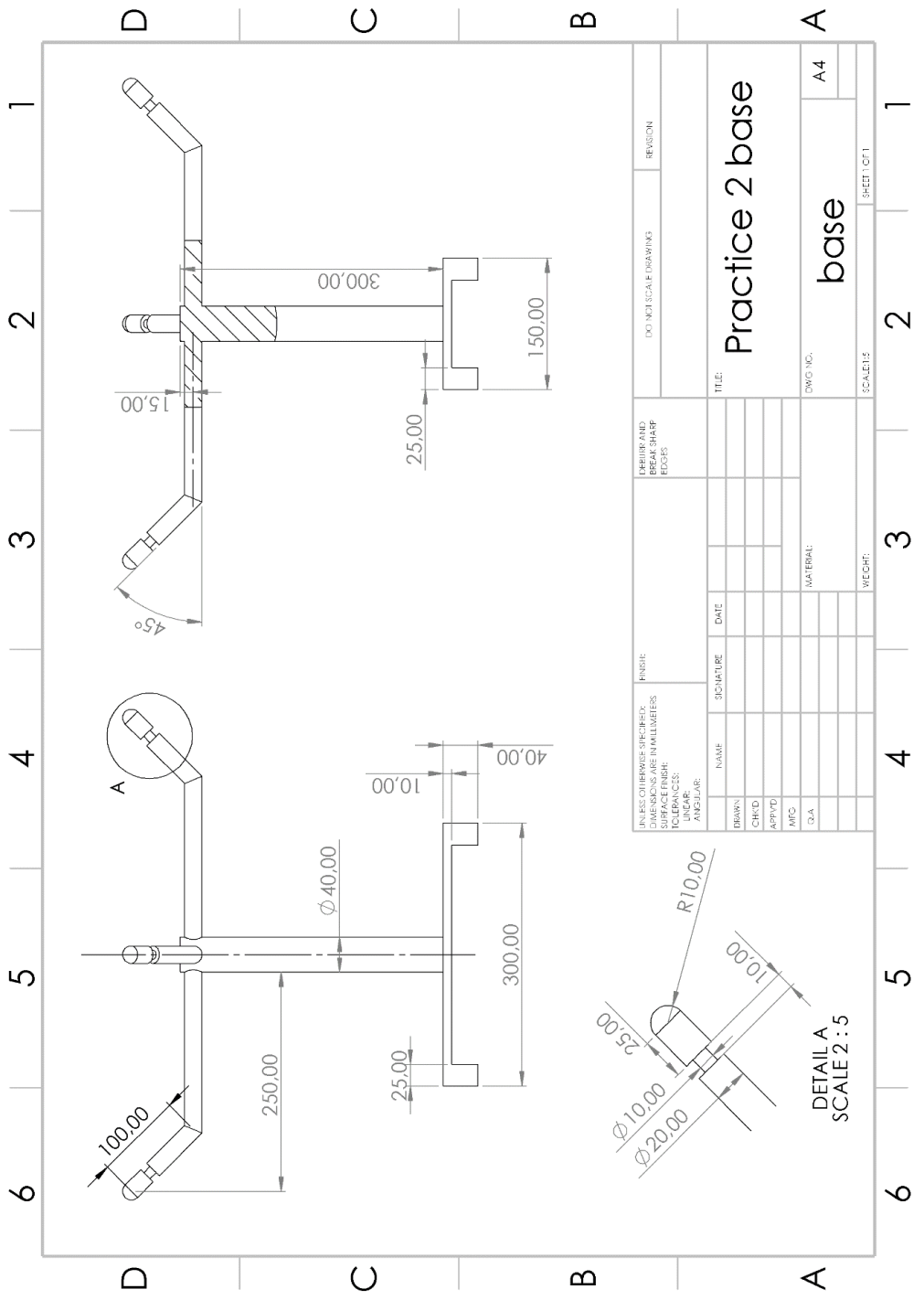
APPENDIX 1


Practice 1 base
base
A4
DO NOT SCALE DRAWING
DEBUR AND BREAK SHARP EDGES
UNLESS OTHERWISE SPECIFIED:
DRAWN
CHK'D
APPV'D
MFG
Q.A
NAME
SIGNATURE
DATE
MATERIAL
WEIGHT:
SCALE:1:5
SHEET 1 OF
DWG NO.
REVISION
DETAIL A
SCALE 1 : 1
DETAIL B
SCALE 1 : 1
300,00
240,00
40,00
10,00
150,00
25,00
Ø20,00
15,00
Ø40,00
300,00
25,00
10,00
20,00
R10,00
Ø10,00

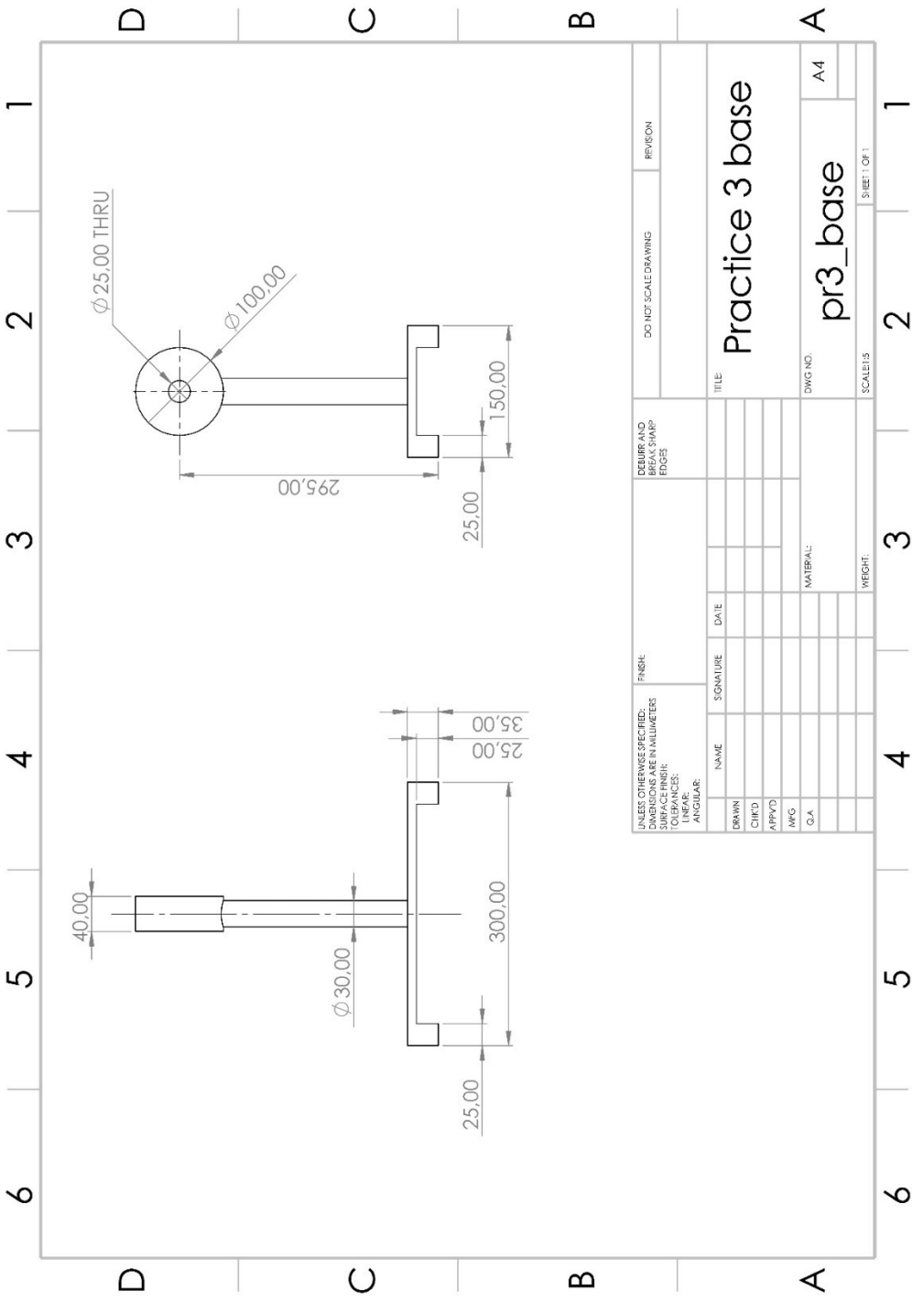
APPENDIX 2


Practice 1 wheel
wheel
A4
DETAIL B
SCALE 2 : 1
30,00
20,00
15,00
5,00
5,00
SECTION A-A
Ø 10,00
10,00
50,00
Ø 150,00
A
B
A

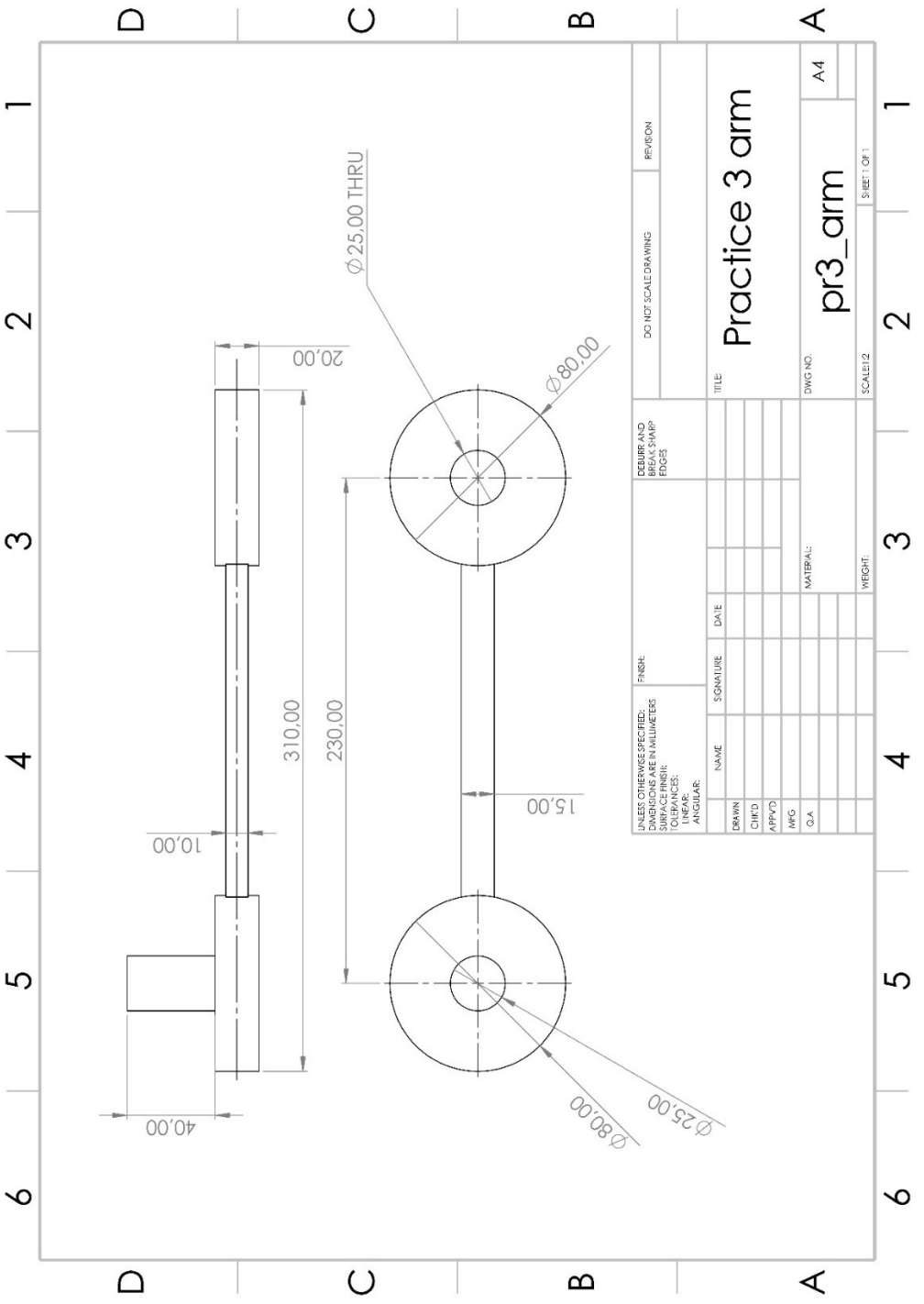
APPENDIX 3


300,00
150,00
15,00
25,00
45°
A
10,00
40,00
Ø40,00
300,00
250,00
25,00
100,00
R10,00
10,00
25,00
Ø10,00
Ø20,00
DETAIL A
SCALE 2 : 5
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
NAME
SIGNATURE
DATE
DRAWN
CHKD
APPVD
MFG
QA
FINISH:
DEBURR AND
BREAK SHARP
EDGES
MATERIAL:
WEIGHT:
DO NOT SCALE DRAWING
REVISION
TITLE:
Practice 2 base
DWG NO.
base
A4
SCALE:1:5
SHEET 1 OF 1

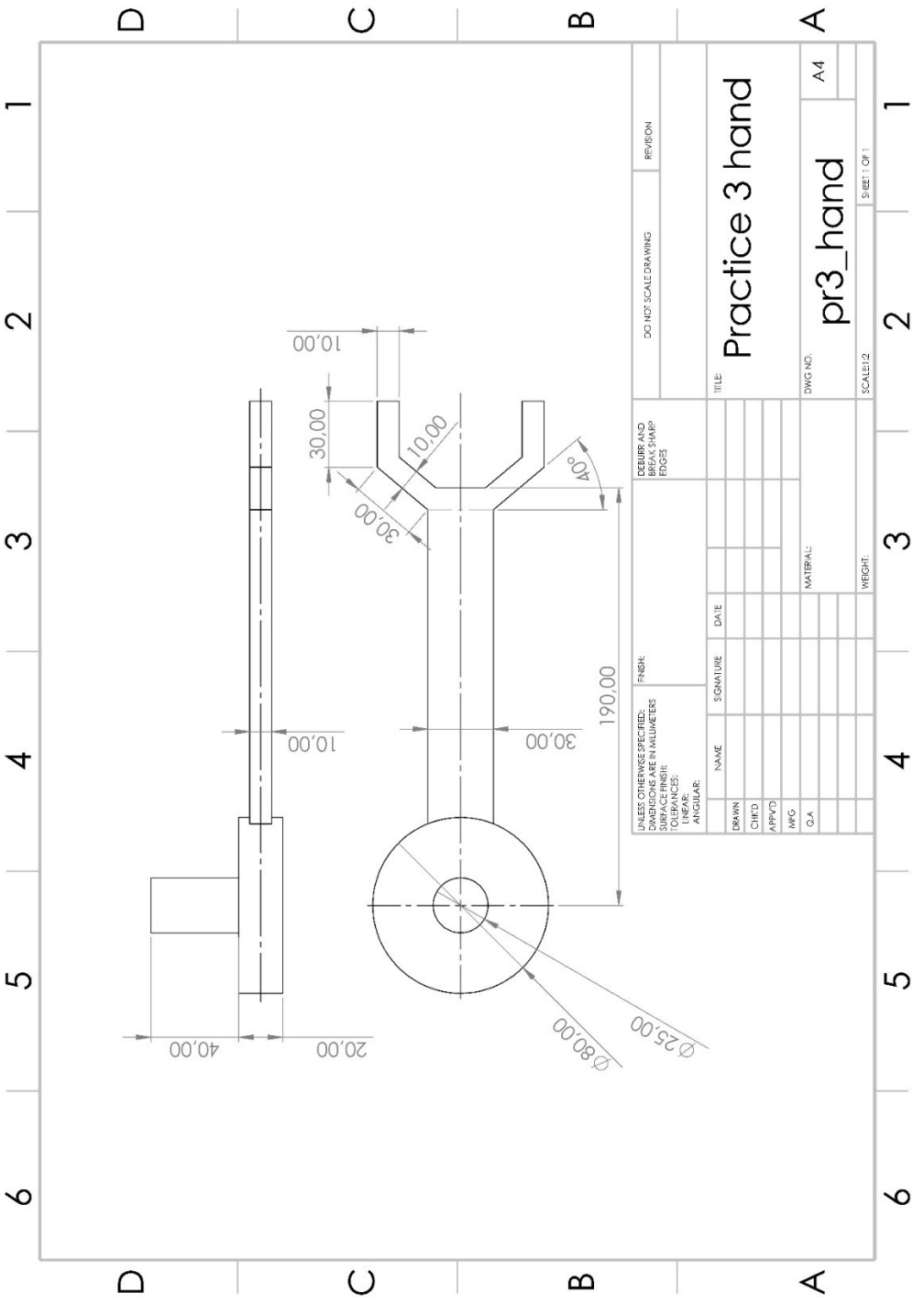
APPENDIX 4

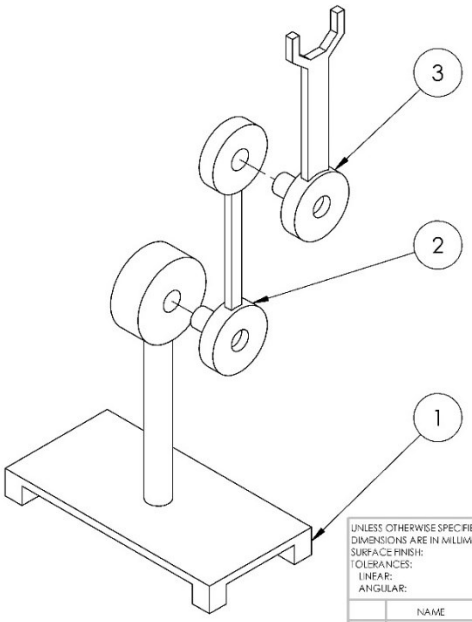

Ø25.00 THRU
Ø100.00
150.00
295.00
25.00
35.00
25.00
40.00
Ø30.00
300.00
25.00
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
FINISH:
DEBURR AND BREAK SHARP EDGES
DO NOT SCALE DRAWING
REVISION
NAME
SIGNATURE
DATE
DRAWN
CHK'D
APPV'D
MFG
Q.A
MATERIAL:
WEIGHT:
TITLE:
Practice 3 base
DWG NO.
pr3_base
A4
SCALE:1:5
SHEET 1 OF 1

APPENDIX 5


Practice 3 arm
pr3_arm
A4
SCALE:1:2
SHEET 1 OF 1
DO NOT SCALE DRAWING
REVISION
DEBURR AND BREAK SHARP EDGES
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
FINISH:
NAME
SIGNATURE
DATE
DRAWN
CHK'D
APPV'D
MFG
Q.A
MATERIAL:
WEIGHT:
DWG NO.
TITLE:
Ø25,00 THRU
Ø80,00
20,00
310,00
230,00
15,00
10,00
40,00
Ø80,00
Ø25,00

APPENDIX 6


Practice 3 hand
pr3_hand
A4
SCALE:1:2
SHEET 1 OF 1
DO NOT SCALE DRAWING
REVISION
DEBURR AND BREAK SHARP EDGES
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
FINISH:
NAME
SIGNATURE
DATE
DRAWN
CHK'D
APPV'D
MFG
Q.A
MATERIAL:
WEIGHT:
DWG NO.
TITLE:
10,00
30,00
10,00
30,00
40°
190,00
30,00
10,00
40,00
20,00
Ø80,00
Ø25,00



ITEM NO.	PART NUMBER	QTY.
1	pr3_frame	1
2	pr3_arm	1
3	pr3_hand	1

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:
Practice 3, manipulator assembly

DWG NO.
pr3_assem

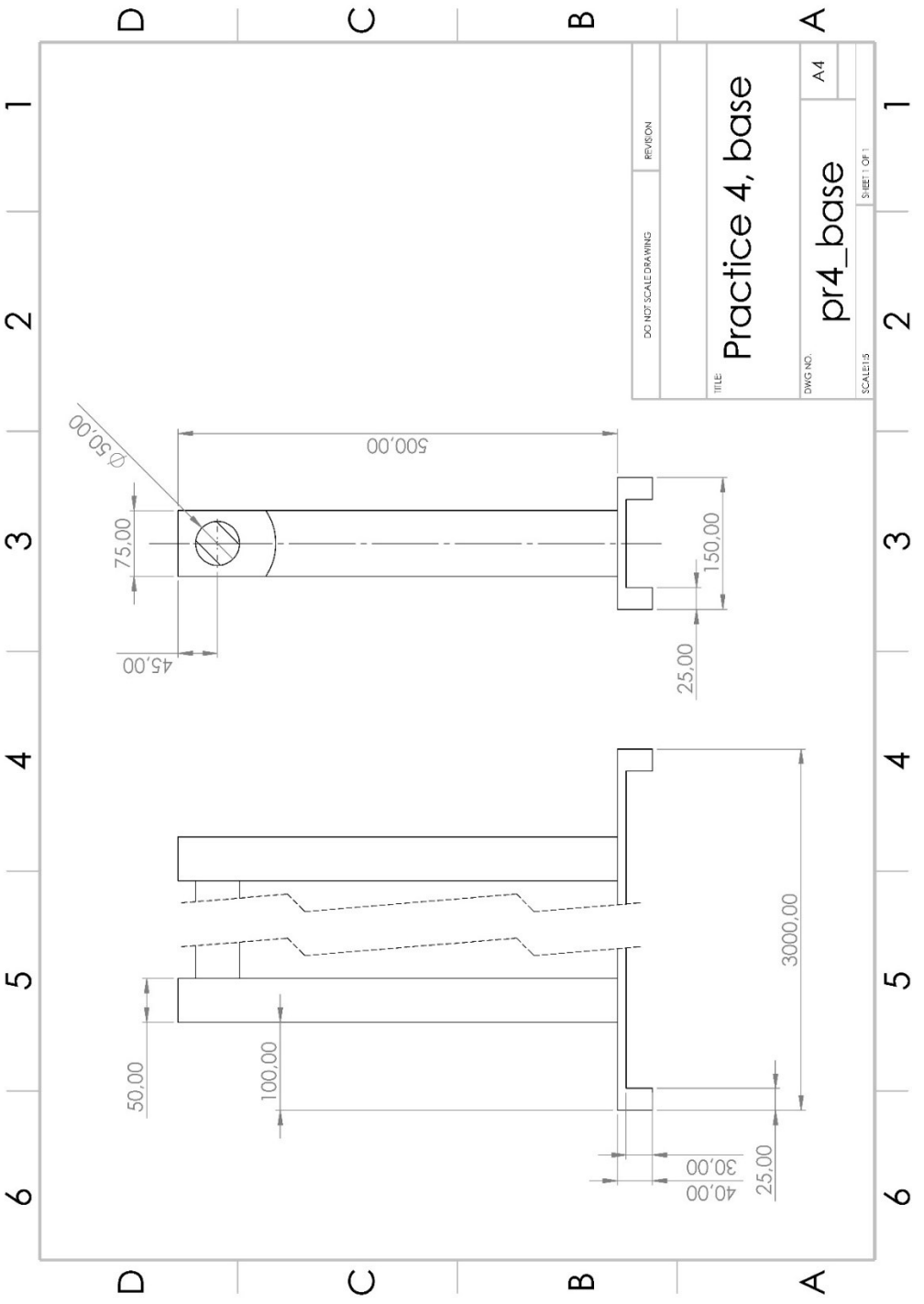
A4

SCALE:1:5

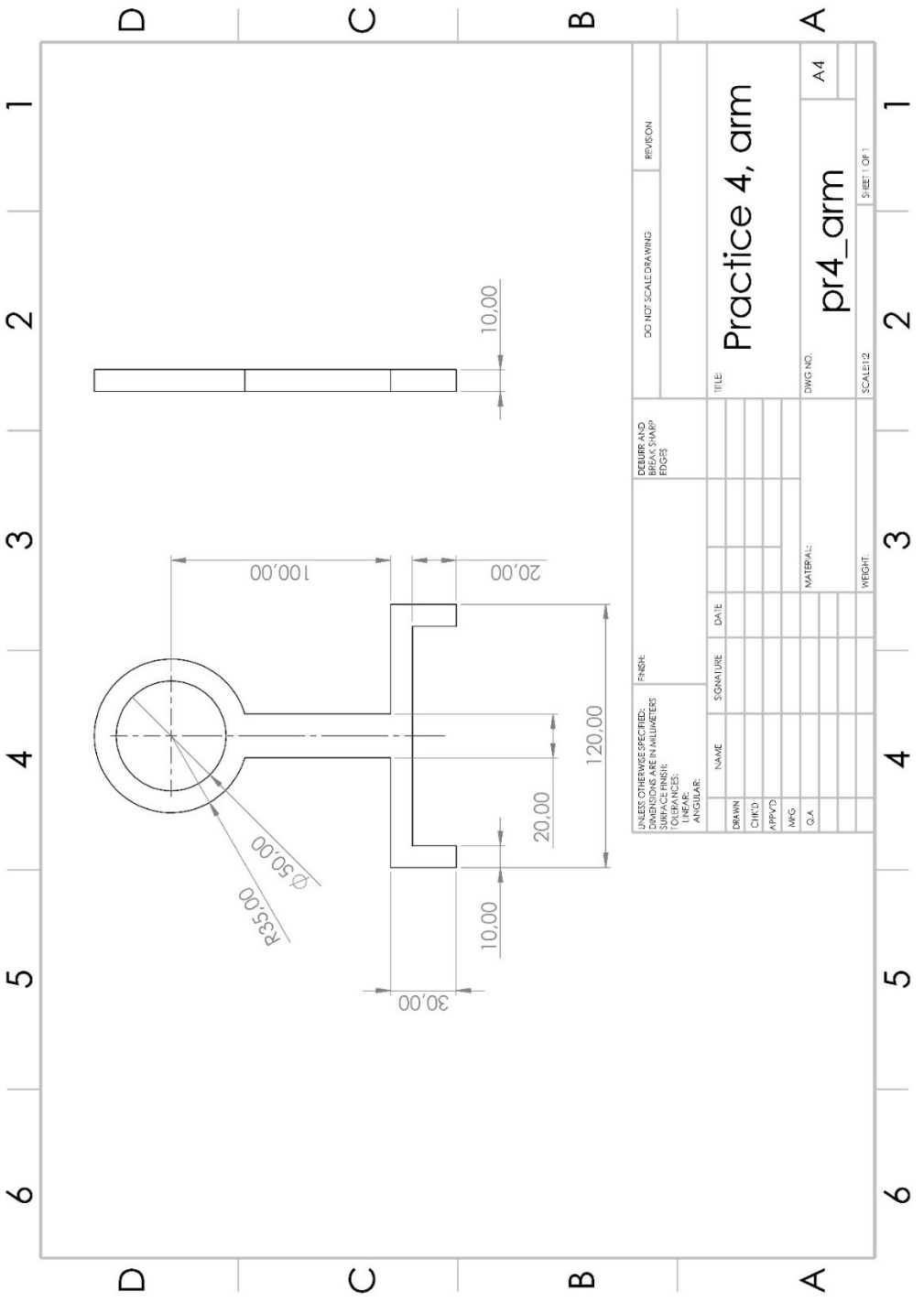
SHEET 1 OF 1

APPENDIX 7

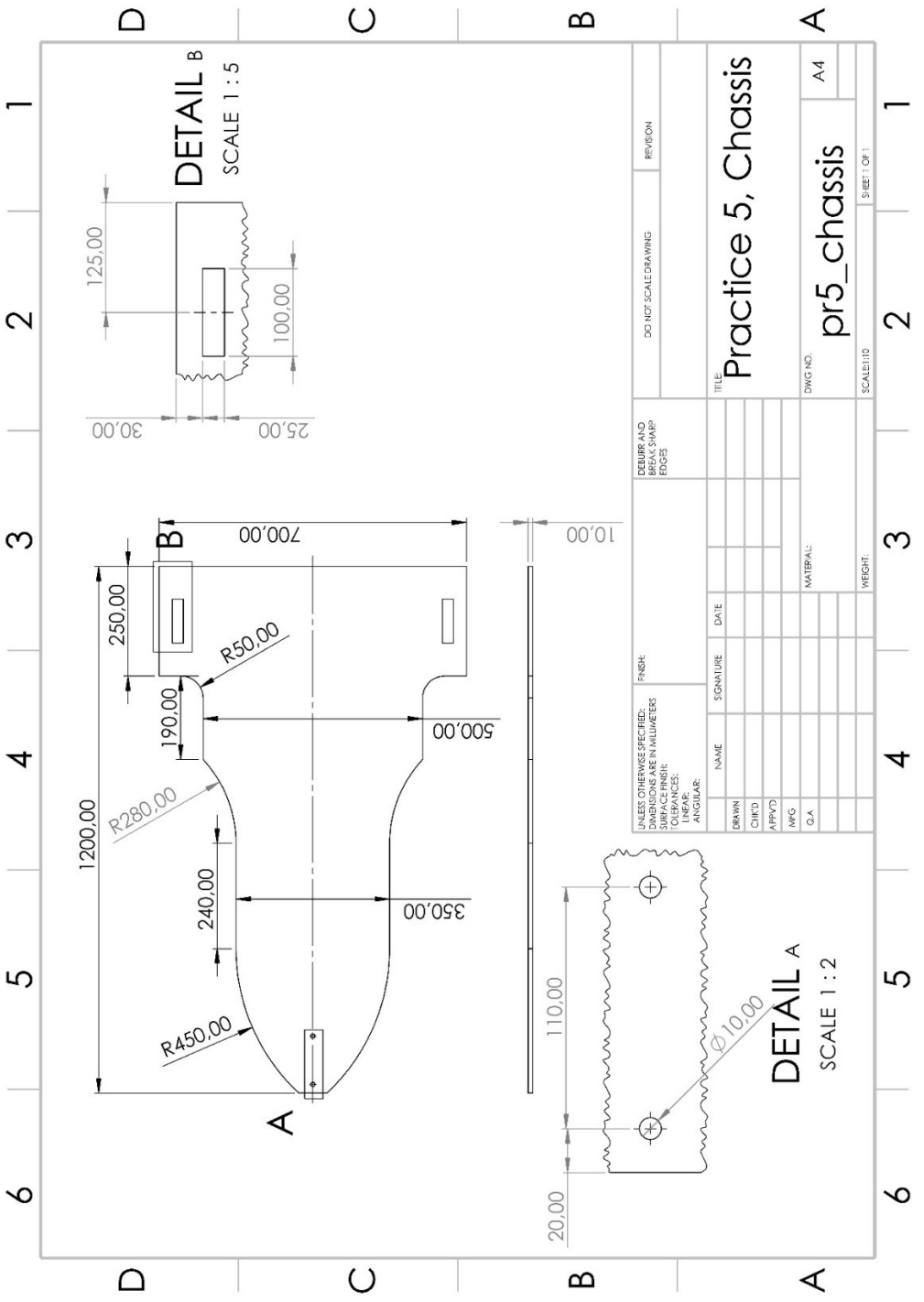
APPENDIX 8


DO NOT SCALE DRAWING
REVISION
TITLE:
Practice 4, base
A4
DWG NO.
pr4_base
SCALE:1:5
SHEET 1 OF 1
Ø50,00
500,00
75,00
45,00
150,00
25,00
3000,00
50,00
100,00
40,00
30,00
25,00

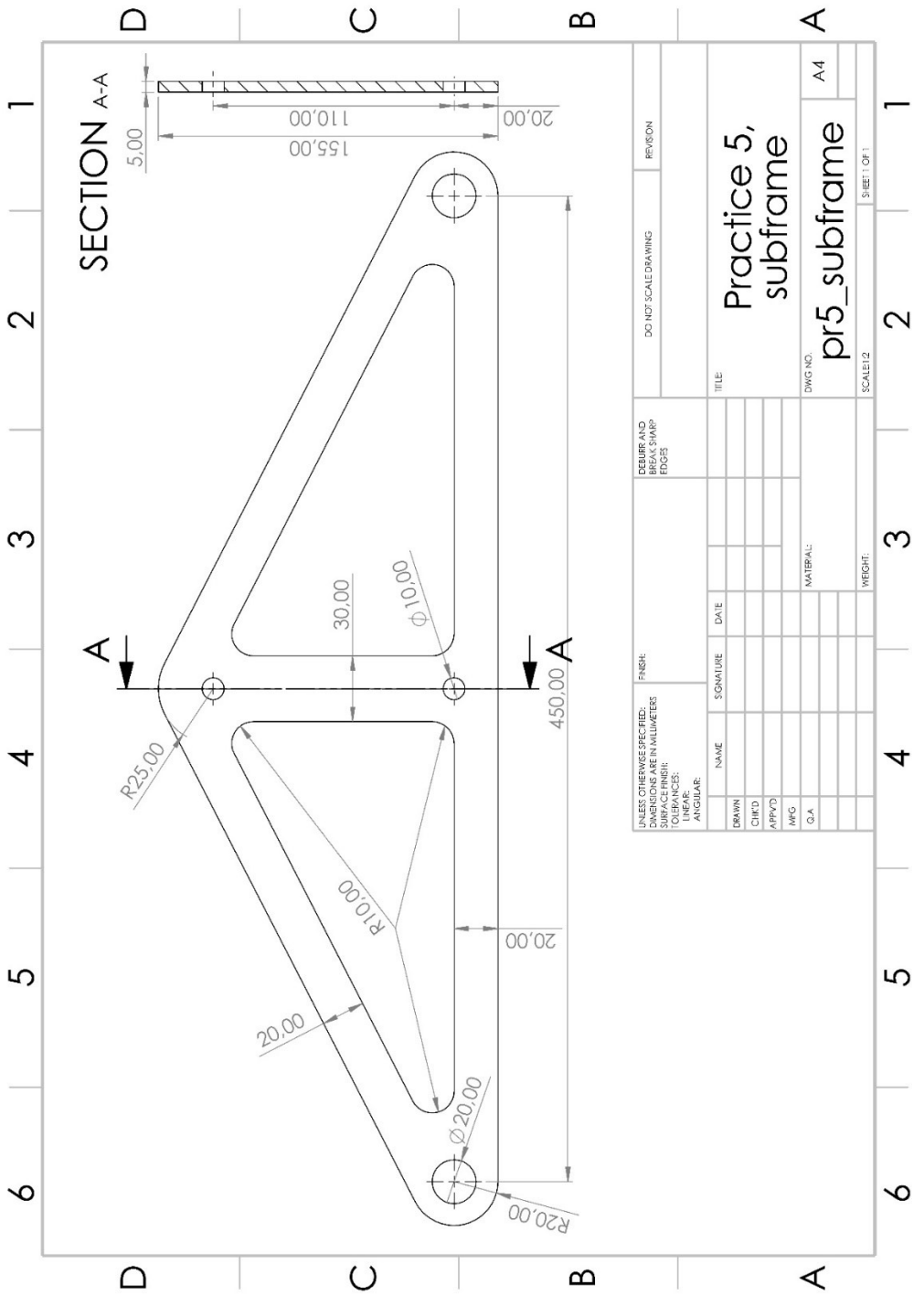
APPENDIX 9


10,00
100,00
20,00
120,00
20,00
10,00
30,00
R35,00
Ø50,00
Practice 4, arm
pr4_arm
A4
SCALE:1:2
SHEET 1 OF 1
DO NOT SCALE DRAWING
REVISION
DEBURR AND BREAK SHARP EDGES
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
FINISH:
NAME
SIGNATURE
DATE
DRAWN
CHK'D
APPV'D
MFG
Q.A
MATERIAL:
WEIGHT:
TITLE:
DWG NO.

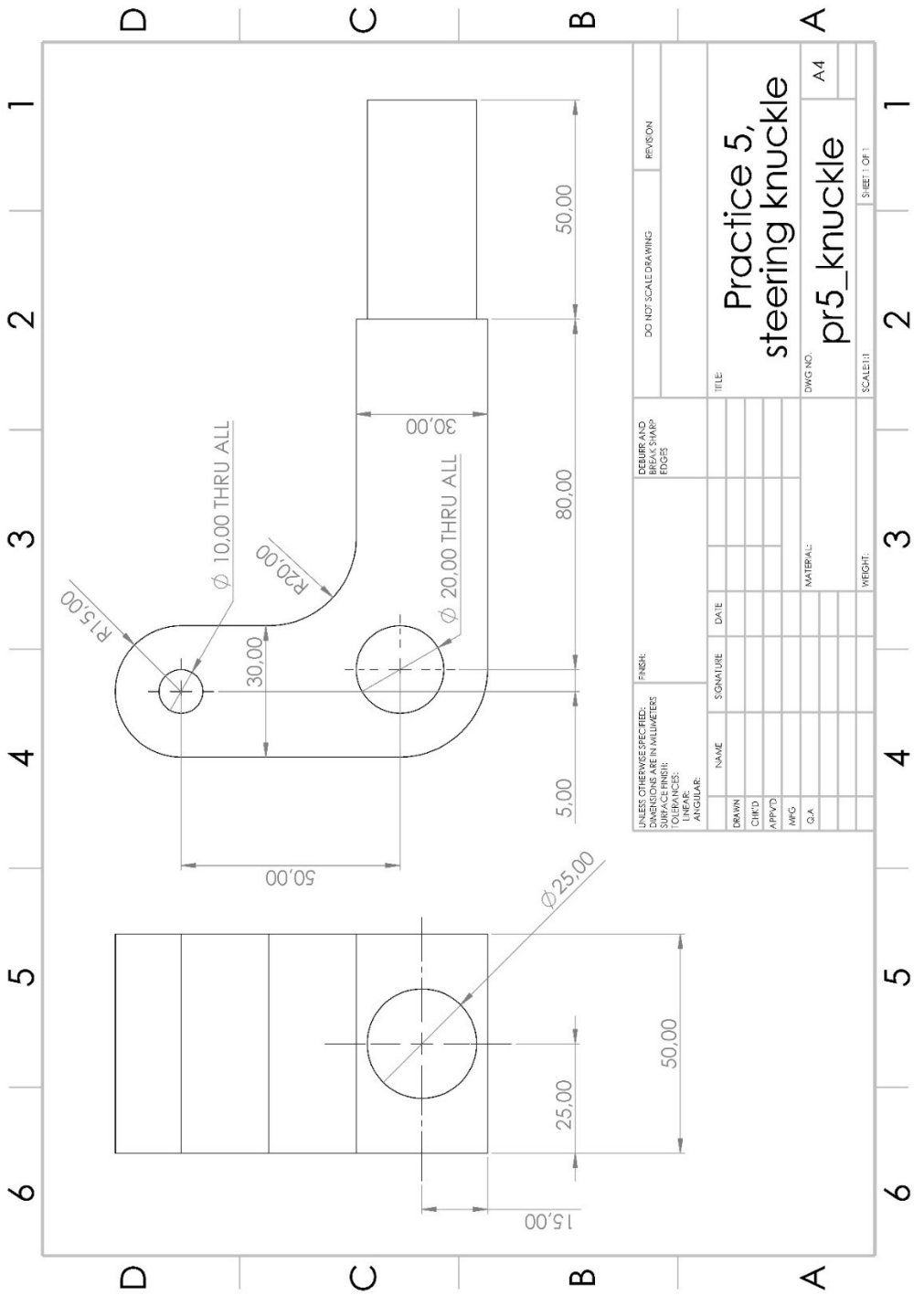
APPENDIX 10


DETAIL B
SCALE 1 : 5
125,00
100,00
30,00
25,00
700,00
250,00
R50,00
190,00
500,00
R280,00
1200,00
240,00
350,00
R450,00
A
B
10,00
110,00
20,00
Ø10,00
DETAIL A
SCALE 1 : 2
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
FINISH:
DEBURR AND BREAK SHARP EDGES
NAME
SIGNATURE
DATE
DRAWN
CHK'D
APPV'D
MFG
Q.A
MATERIAL:
WEIGHT:
DO NOT SCALE DRAWING
REVISION
TITLE:
Practice 5, Chassis
DWG NO.
pr5_chassis
A4
SCALE:1:10
SHEET 1 OF 1

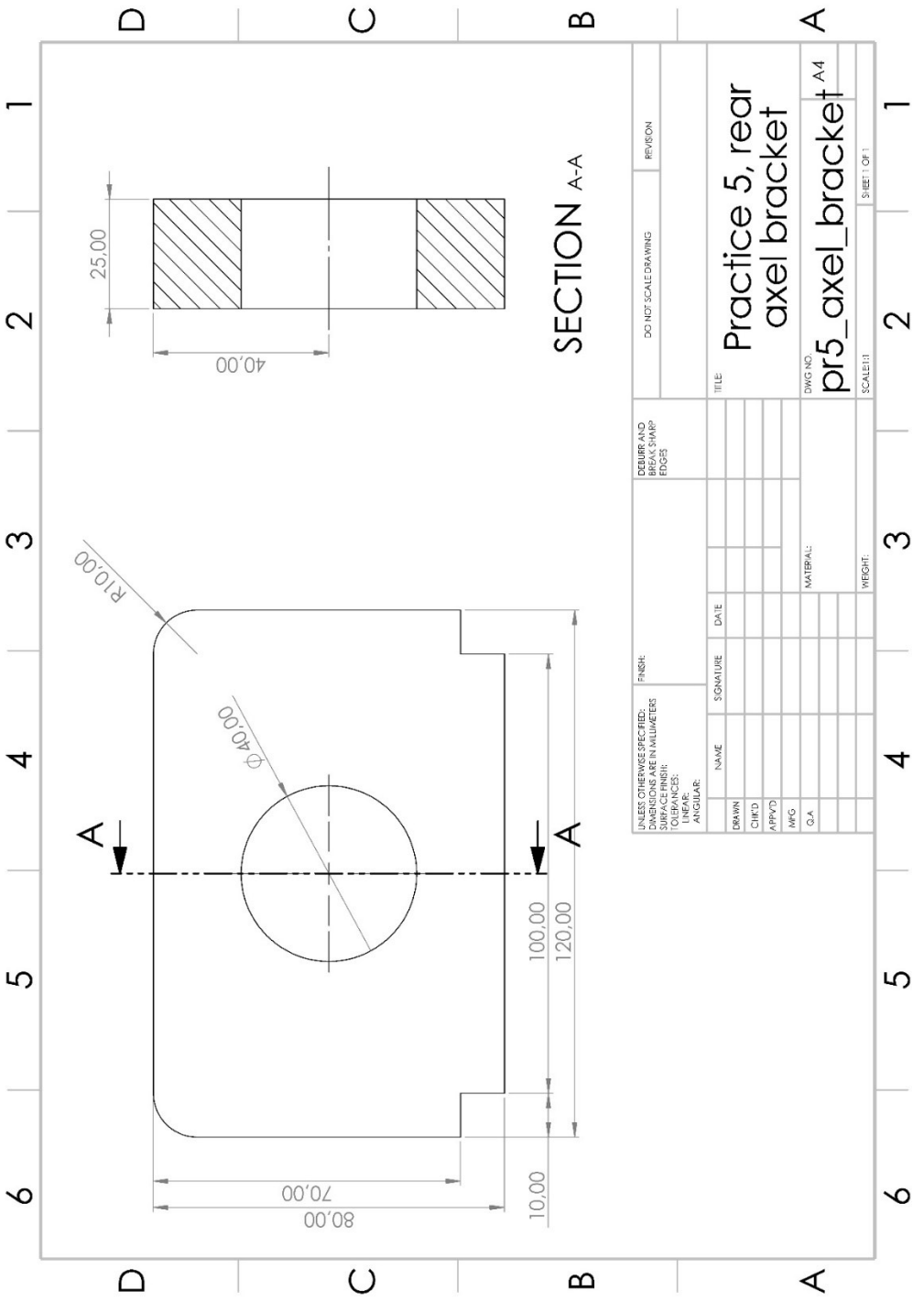
APPENDIX 11


SECTION A-A
5,00
155,00
110,00
20,00
A
A
450,00
30,00
ϕ10,00
R25,00
R10,00
20,00
20,00
Ø20,00
R20,00
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
FINISH:
DEBURR AND BREAK SHARP EDGES
NAME
SIGNATURE
DATE
DRAWN
CHK'D
APPV'D
MFG
Q.A
MATERIAL:
WEIGHT:
DO NOT SCALE DRAWING
REVISION
TITLE:
Practice 5, subframe
DWG NO.
pr5_subframe
A4
SCALE:1:2
SHEET 1 OF 1

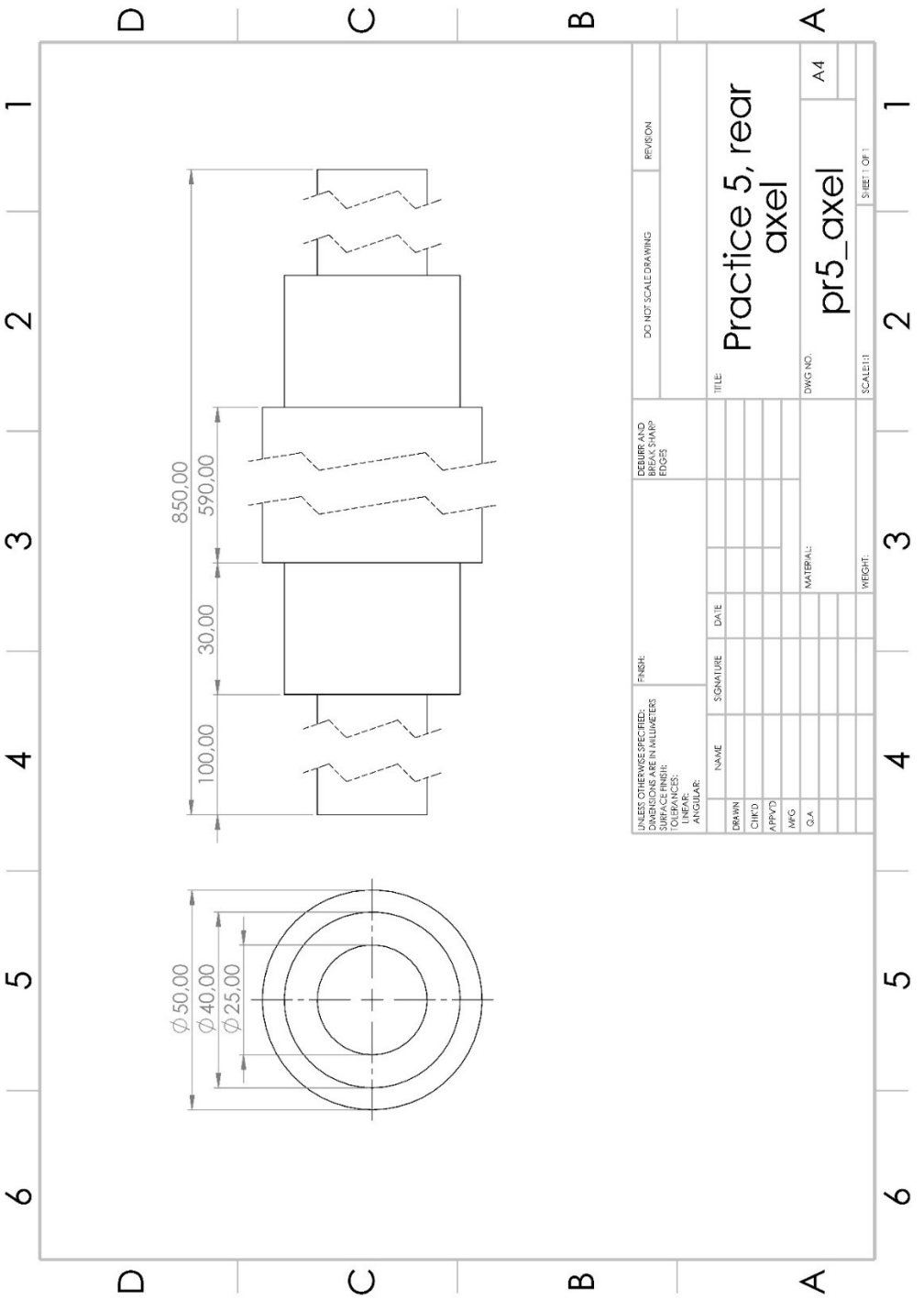
APPENDIX 12


Practice 5,
steering knuckle
pr5_knuckle
A4
DO NOT SCALE DRAWING
REVISION
DEBURR AND
BREAK SHARP
EDGES
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
FINISH:
NAME
SIGNATURE
DATE
DRAWN
CHK'D
APPV'D
MFG
Q.A
MATERIAL:
DWG NO.
WEIGHT:
SCALE:1:1
SHEET 1 OF 1
TITLE:
50,00
80,00
5,00
30,00
50,00
Ø 10,00 THRU ALL
Ø 20,00 THRU ALL
R20,00
R15,00
30,00
50,00
Ø25,00
25,00
50,00
15,00

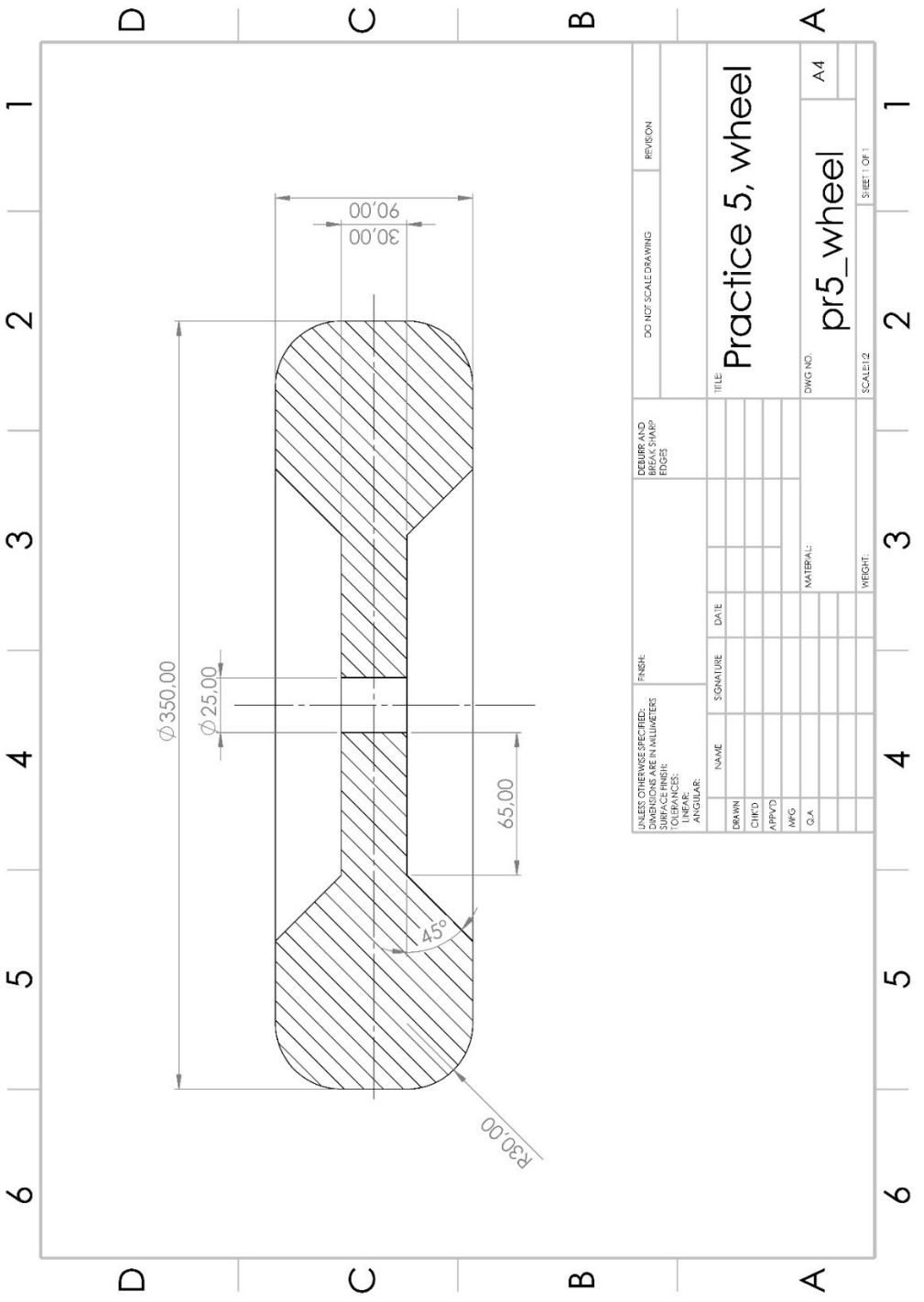
APPENDIX 13


SECTION A-A
25,00
40,00
R10,00
Ø40,00
A
A
100,00
120,00
70,00
80,00
10,00
DO NOT SCALE DRAWING
REVISION
DEBURR AND BREAK SHARP EDGES
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
FINISH:
NAME
SIGNATURE
DATE
DRAWN
CHK'D
APPV'D
MFG
Q.A
MATERIAL:
WEIGHT:
TITLE:
Practice 5, rear axel bracket
DWG NO.
pr5_axel_bracket
A4
SCALE:1:1
SHEET 1 OF 1

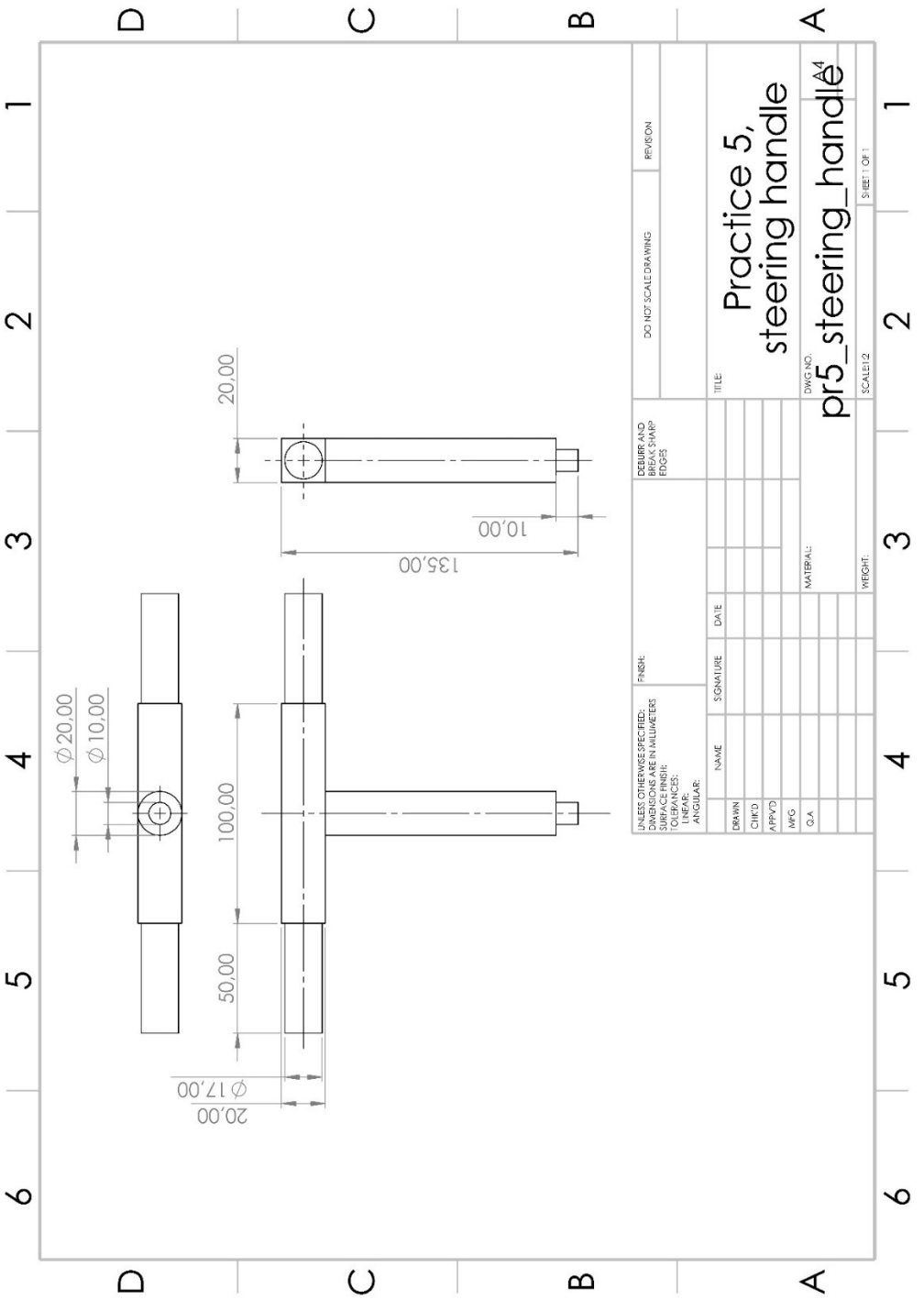
APPENDIX 14

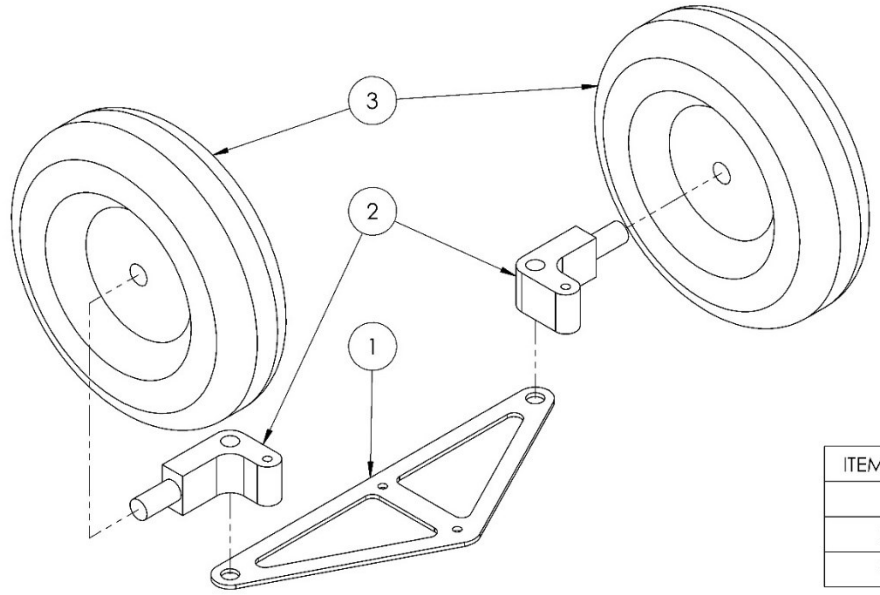

850,00
590,00
30,00
100,00
ϕ50,00
ϕ40,00
ϕ25,00
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
FINISH:
DEBURR AND BREAK SHARP EDGES
NAME
SIGNATURE
DATE
DRAWN
CHK'D
APPV'D
MFG
Q.A
MATERIAL:
WEIGHT:
DO NOT SCALE DRAWING
REVISION
TITLE:
Practice 5, rear axel
DWG NO.
pr5_axel
A4
SCALE:1:1
SHEET 1 OF 1

APPENDIX 15


90,00
30,00
ϕ350,00
ϕ25,00
65,00
45°
R30,00
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
FINISH:
DEBURR AND BREAK SHARP EDGES
NAME
SIGNATURE
DATE
DRAWN
CHK'D
APPV'D
MFG
Q.A
MATERIAL:
WEIGHT:
DO NOT SCALE DRAWING
REVISION
TITLE:
Practice 5, wheel
DWG NO.
pr5_wheel
A4
SCALE:1:2
SHEET 1 OF 1

APPENDIX 16

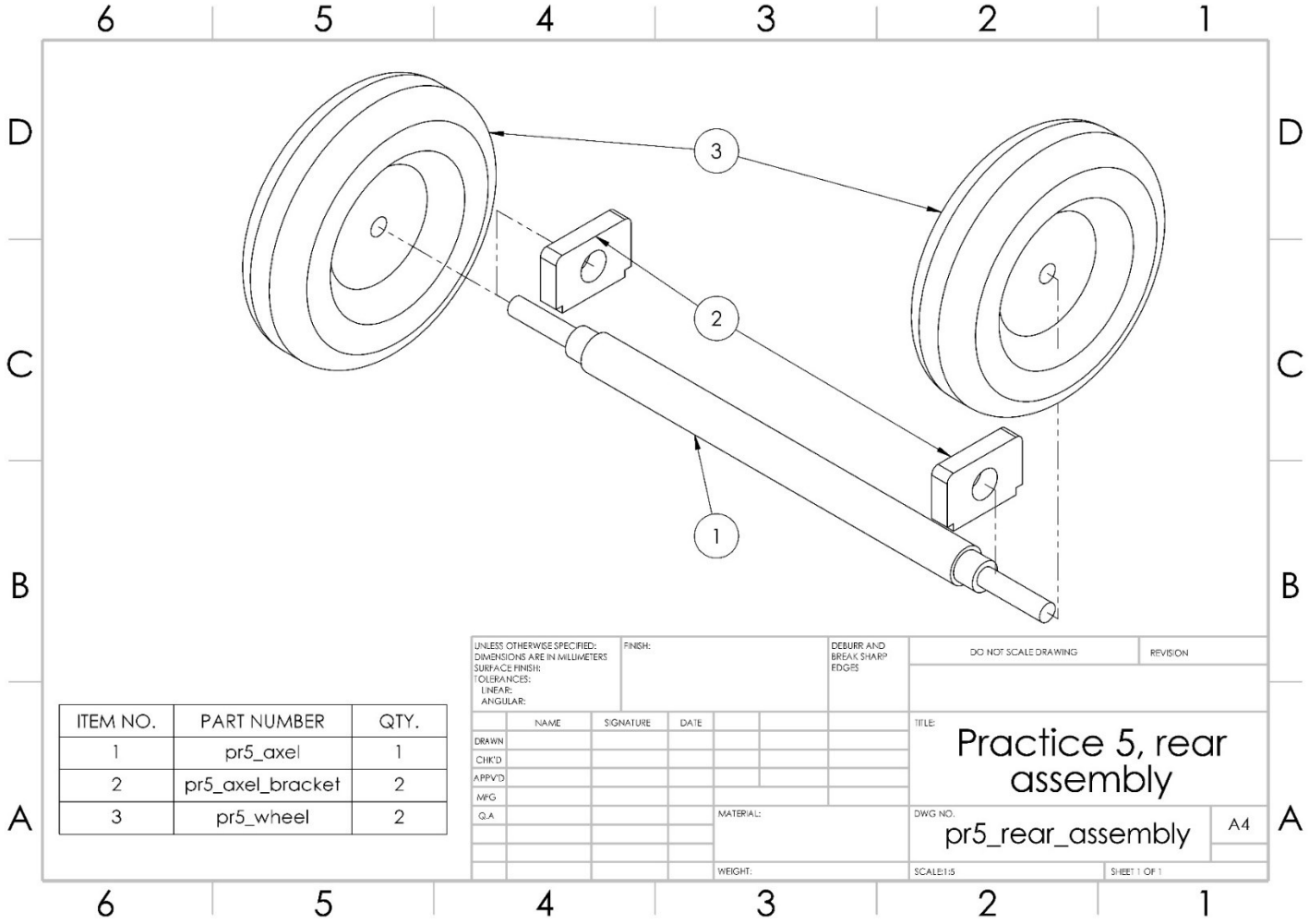

Practice 5,
steering handle
pr5_steering_handle
A4
SCALE:1:2
SHEET 1 OF 1
DO NOT SCALE DRAWING
REVISION
DEBURR AND BREAK SHARP EDGES
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
FINISH:
NAME
SIGNATURE
DATE
DRAWN
CHK'D
APPV'D
MFG
Q.A
MATERIAL:
WEIGHT:
TITLE:
DWG NO.
20,00
10,00
135,00
100,00
50,00
Ø17,00
20,00
Ø20,00
Ø10,00



ITEM NO.	PART NUMBER	QTY.
1	pr5_subframe	1
2	pr5_knuckle	2
3	pr5_wheel	2

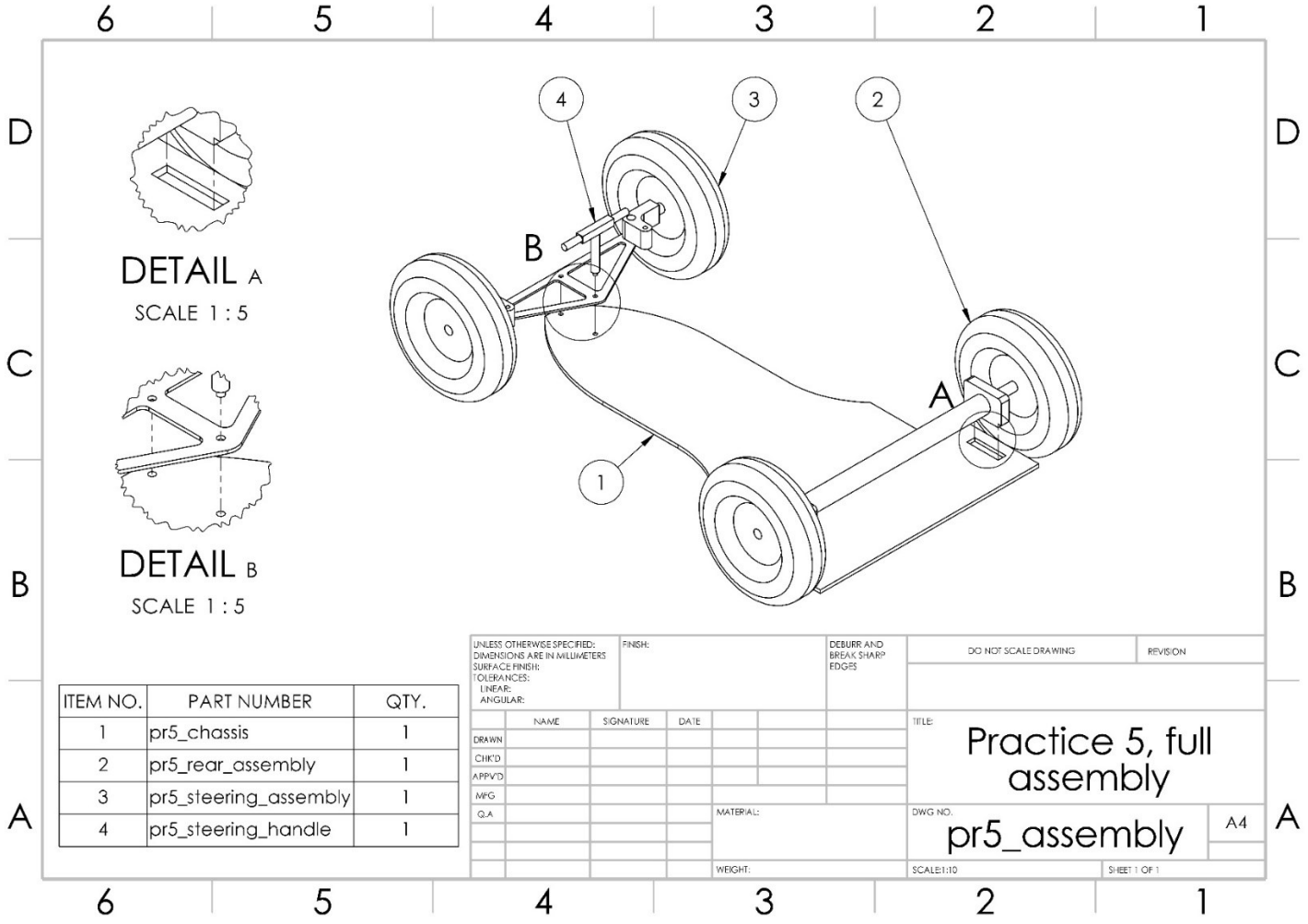
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:	FINISH:				DEBURR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING / REVISION
	NAME	SIGNATURE	DATE			TITLE: Practice 5, steering assembly
DRAWN						
CHK'D						
APPV'D						
MFG						
Q.A				MATERIAL:		DWG NO. pr5_steering_assembly / A4
				WEIGHT:		SCALE:1:5 / SHEET 1 OF 1

APPENDIX 17



ITEM NO.	PART NUMBER	QTY.
1	pr5_axel	1
2	pr5_axel_bracket	2
3	pr5_wheel	2

APPENDIX 18


6
5
4
3
2
1
D
C
B
A
DETAIL A
SCALE 1 : 5
DETAIL B
SCALE 1 : 5
4
3
2
B
A
1
ITEM NO. | PART NUMBER | QTY.
1 | pr5_chassis | 1
2 | pr5_rear_assembly | 1
3 | pr5_steering_assembly | 1
4 | pr5_steering_handle | 1
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:
Practice 5, full assembly
DWG NO.
pr5_assembly
A4
SCALE:1:10
SHEET 1 OF 1

APPENDIX 19