# Development of a Semi-autonomous Holonomic Load Carrier with Multi-camera Perception

**Sondre Johnsen**
**Didrik Efjestad Fjereide**
**Mikal Sørensen**

**Supervisor**
Morten H. Rudolfsen, UiA

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

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

# Abstract

This thesis documents the development of a load carrier capable of carrying ten advanced personal robots. The robots of concern are Segway Robotics' Loomo, which are used for education purposes at the University of Agder. They are used at multiple locations on campus, and it is desired to develop a system that can effectively transport them around. The report covers the concept generation, mechanical design, electrical design and development of a navigation system.

Figure 1: Loomo rig

A simple and compact design was developed and built. To achieve holonomic drive, the rig was equipped with four mecanum wheels. A mechanical design process was performed to come up with a solution for mounting the wheels to the rig. This included design of an axle and bearing calculations. Additionally, the stability of the rig had to be verified. Further on, to drive the mecanum wheels, four brushless dc motors were utilized. The system consist of multiple hardware components, which needed to communicate with each other. The solution for this is documented in this report.

The rig features three different operating modes: manual control, assistive actuation and autonomous navigation. Manual control is based on controlling the rig using a hand held controller. Assistive actuation utilized a motion tracking camera to interpret hand gestures. These hand gestures can be used to control the rig, without physical contact. The autonomous mode lets the operator set a goal position in a provided map, and the robot will plan and execute a path accordingly.

Utilization of the Robot Operating System (ROS) frame work in conjunction with the simulation program GAZEBO and visualization tool RVIZ is elaborated. Semi-autonomous navigation was achieved in simulation, by utilizing the navigation stack available in ROS. The robot is considered as semi-autonomous because it relays on instructions from the operator, such as an initial position at start-up and the desired goal position in the map.

A video of the robot is available at: https://www.youtube.com/watch?v=LGs4QlntGkw

The source code is available at: https://github.com/didrif/megatronds

# Acknowledgements

We appreciate the support and guidance provided by our supervisor Morten Hallquist Rudolfsen at the University of Agder, who has given us valuable input and providing resources throughout the thesis.

We also want to thank the engineers at the mechatronics lab at University of Agder; Carl Thomas Duus, Roy Werner Folgerø, Jan Christian Strandene and Harald Sauvik, for the assistance with machining of essential mechanical parts, in addition to their assistance with training to use necessary equipment, as well as answers to design questions.

# Table of Contents

# List of Figures

# List of Tables

# 1. Introduction

In the early stages of automation, mobile robots could not perceive their surroundings due to the lack of sensors. They had no information about where humans were at any given time, and because of this, they were typically enclosed inside an area where humans did not stray. Humans crossed the path of a robot at their own risk. Due to the huge advancement in sensor technology, robots can now be taken out of their fences and integrated on the production floor. As a result, human-robot-interaction becomes more critical. Today, the tables have turned, and it has now become the responsibility of the robot not to collide with a human. For cooperative work between robots and humans, interaction systems through the use of advanced sensors and data processing have become vital.

## 1.1 Background

The University of Agder possesses ten units of Segway Robotics' Loomo that are used for educational purposes. A Loomo is an advanced personal robot that is programmable by its user. Users can create a range of features and practical solutions through access to advanced sensors and features such as robust motion control, intelligent planning, and obstacle avoidance as well as visual perception [19]. At the university, it is used to spark interest and emphasize the power of programming in the introductory course for engineers. The devices are used at several different locations on campus, which requires them to be transported effectively. A simple transportation rig for the Loomos is available and in frequent use at the university today. However, the construction is subjected to heavy loads, which makes it hard to maneuver manually and generally requires two people to operate. Because of this, the university wishes to develop a new rig with assistive actuation and steering. State of the art embedded system-on-module and different modern camera sensors are readily available at the university, and there is a desire to look at the possibility of using these to automate the rig further.

## 1.2 Objective

The main objective is to develop a load carrying vehicle, which can efficiently transport the ten Loomos used for educational purposes. The vehicle must be equipped with assistive actuation and should have a low level steering mechanism, such as a hand-held controller, in order to provide easy manual control. In addition, the rig should be able to follow a designated person, object or Loomo. An evaluation of a solution for autonomous guidance should be performed. For instance a feature which lets the user order, or request the rig to a specific location at the university.

The rig must be able to enter a designated storage room. The room is located in a quite narrow hallway, which set the following size requirements for the design:

- Width: 850 mm
- Length: 1750 mm
- Heigth: 2000 mm

An additional feature is desired for the rig. It should be possible to lock the Loomos to the rig. Preferably the locking system should be universal, so that all Loomos are locked by one lock.

## 1.3 Report Structure

The complete system documented in this report includes a wide range of subjects and to give a complete overview of the thesis, the report structure and a short summary of each chapter is presented in this section.

**Chapter 2: Product Development**

The product development chapter covers the technique used to generate a feasible idea for the design of the rig. This includes detailed descriptions of the evaluated concepts and the additional extension features included in the final design. Additionally, a brief overview of the project planning and execution is presented.

**Chapter 3: Mechanical Design**

The third chapter focus on the design of necessary mechanical components to realise the chosen concept. Evaluation of material selection, shaft design, bearing lifetime calculations, and weld durability are covered. Additionally, an introduction to mecanum wheels as well as a detailed force analysis for the wheels is documented in this chapter.

**Chapter 4: Electrical Design**

The electrical design chapter concentrate on the required components to motorize the rig. The selection and functionality of speed controllers and brushless DC motors are covered together with the drive train sizing. Verification of the power source and overall power consumption of the electronic hardware component is performed.

**Chapter 5: Modeling**

In the modeling chapter, the kinematic constraints of a mecanum wheel is derived. Further on, the process of creating a simulation model of the plant is documented. Also, the software used to develop the simulation model and the framework for the overall robot system is introduced in this chapter.

**Chapter 6: System Architecture**

The communication interface created between the hardware components are covered. This involves communication between the processing unit and the base controller, as well as the communication from the base controller to the motors. The techniques used are UART communication and CAN-bus communication, and both are described. Further, the methods concerning the perception of the environment are presented. This deals with the selection and placement of the perception sensors, as well as the processing of the perceived data and the space transforms in order to relate it correctly.

**Chapter 7: Localization and Navigation**

The localization and navigation chapter aims to cover the features used to achieve the overall functionality of the navigation system. Two navigation modes are presented. ArUco

tracking and following, as well as a semi-autonomous navigation. For the latter mode, map generation, solving the localization problem and path planning is documented. Lastly, a method of assistive actuation by the use of hand gestures is presented.

### Chapter 8: Results

The results chapter present the results obtained throughout the project.

### Chapter 9: Discussion

The discussion chapter contemplate the achieved results. It also elaborates factors which could have been done better and recommendations for future work.

### Chapter 10: Conclusion

This chapter draws a conclusion from the achieved results and discussion.

## 1.4 System Overview

This section aim to give a brief overview of the system as a whole. The system consist of multiple advanced sensors and software modules. To get an understanding of their purpose in this project, a block diagram is presented in Figure (1.1).

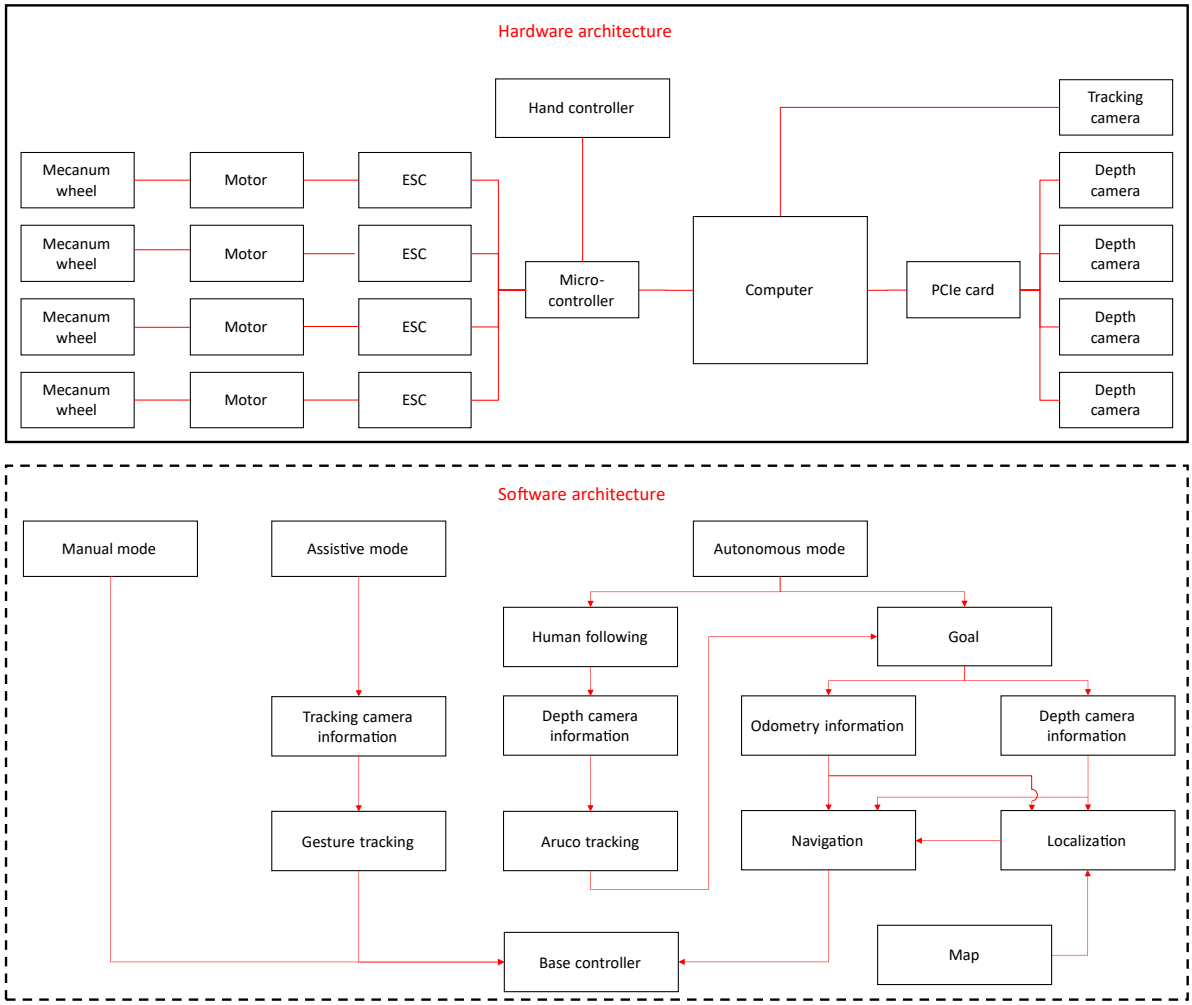



Figure 1.1: System architecture

The body of the robot consist of a steel frame, which is equipped with four mecanum wheels. Due to the geometry of mecanum wheels, the locomotion mechanism enables the vehicle to achieve omni-directional movement. A processing unit serves as the brain of the system and will process, send, and receive information from sensors and to actuators. Connected to the computer is a microcontroller, which serves as the control unit. The control unit is connected to four electronic speed controllers, which control the four brushless dc motors that provide motion for the rig.

The robot consist of three modes: manual, assistive and autonomous mode. The manual mode is a low level controller which bypass the navigation system to provide simple control of the robot. The assistive actuation rely on a tracking camera to interpret gestures based on hand pose, which are further processed and forwarded to the base controller. The base controller handles the kinematics of the robot and converts linear velocity commands into rotational velocity for the wheels. The base controller is implemented on the micro-controller. The autonomous mode lets the operator set a goal position in a map, and the robot will attempt to reach it. The mode has an additional feature, which is to follow an aruco marker. Four depth cameras are used to perceive the surroundings to avoid collision. The perception data is also used to localize the robot in the known map.

# 2. Product Development

The parking rig should be able to enter an allocated storage room which requires the frame to be small. For that reason, some requirements regarding size had to be defined when developing concepts for the framework. This section will present five different design concepts for a parking rig module. These concepts will further be evaluated, and result in a concept for development.

## 2.1 Project Management

At the start of the project, the overall task was defined and delineated. It was then decided that the project should be divided into three phases. The first phase covers the mechanical design and development of the Loomo rig. This includes mechanical calculations to verify the feasibility of the chosen design, but also the process of building the physical rig. In the second phase, the electrical design, selection and installing of electrical components, programming and rig control is going to be added to the mechanical construction. The third, and last phase, multi-camera perception, localization and navigation for the physical model is going to be realised.

In the initial stage of the thesis, a Gantt chart to distribute the resources was created. Figure (2.1) shows the draft of the first Gantt chart of phase one. The main chart highlights the entire project period with the main tasks. At the start of each of the three phases a sub chart was created to highlight the main focus for resources and tasks at hand. By using the three sub charts, key tasks were broken down to ensure each part was consider. By utilizing sub charts internal deadlines are made to keep progress. A finalized Gantt chart of each individual phase are found in their respective section in Appendix (E). In addition, a Gantt chart giving an overview of the whole project plan is also present.

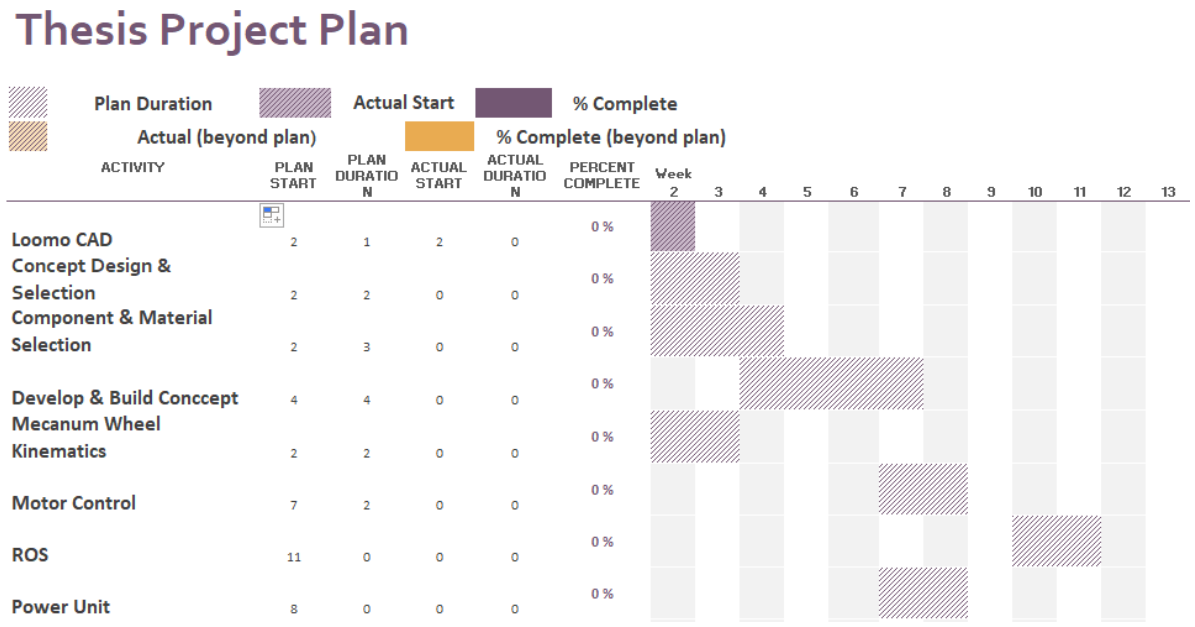



Figure 2.1: Draft of first Gantt chart

## 2.2 Current Rack

A simple storage unit for the Loomos already exists; however, there are some disadvantages to the current design. The size of the rig is almost at it's most permissible, and this makes it hard to maneuver the vehicle into the designated product development room. When developing the new design, the size should be optimized with respect to length and width. Due to the narrowness of the hallway, the length of the storage unit is the most crucial factor. With that said, a reduction in width would make it easier to steer the vehicle into it's allocated space. Further on, the Looms are currently suspended from a hook and have no additional support, which causes the devices to wobble during transportation. For this reason, a suitable support mechanism should be developed for the improved rig. Due to the significant weight of the construction, the current rig is hard to maneuver and requires two persons to operate. To facilitate the operator of the rig, there is a desire to replace the current wheels with a motorized drive system to lay the groundwork for assisted handling and autonomous operations. The current rig design can be viewed in Figure (2.2).

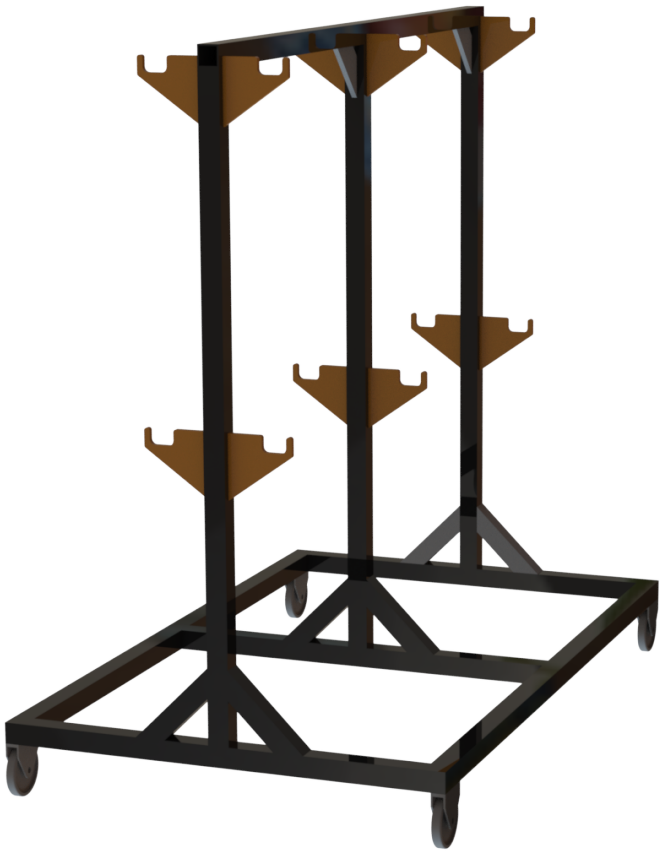


Figure 2.2: Illustration of current rig design

The current dimensions of the rig are: $2000 \times 850 \times 1350\,\text{mm}$. The new design should take into account that the rig will be modified at a later stage with the addition of a charging system for the Loomos. It should also be arranged for a battery system.

## 2.3 Concepts

This section will present the concepts individually, exploiting their features and how they solve the issues regrading the current rack.

**Concept 1**

This concept focuses on minimizing the width of the frame by stacking the Loomo devices in a row configuration. The width of this design is 650 mm while the rig's length is reduced to 1800 mm. Configuring the devices in this manner ensures that the mass is centered on the construction, which means there is a low risk of sideways toppling if the rig is loaded unevenly. To get an impression of how much material is used, and thereof also the price, the weight of the construction is evaluated. The weight of this design is approximately 90 kg. Concept 1 is illustrated in Figure (2.3).

Figure 2.3: Illustration of concept 1

The concept features a mechanism that lets the user slide the lower section to the right and the upper section to the left. This facilitates easy loading/unloading of the devices and does not restrain the mounting sequence. With this design, the devices are easy to mount regardless of which section is loaded first/last. The sliding feature allows for a low height of the frame because the vertical direction can be quite compact when traveling and slides to the side when mounting. A mechanical lock serves to immobilize the sliding during traveling. The sliding mechanism is illustrated in Figure (2.4) and Figure (2.5). To secure the devices to the vehicle, customized stands are used, which are made such that the Loomos can be placed on top of them. In addition, the lifting handle of the Loomo is attached to a bracket to keep it in place. The design in this concept has medium complexity because of the sliding ability and introduce some challenges with regards to electrical wiring and central locking of the Loomos. When locking the devices, it is favorable to use as few locks as possible. Because the Loomos are aligned, a central locking system would be simple to implement, however, the sideways freedom would require at least two locks, one for each section.

Figure 2.4: Traveling configuration

Figure 2.5: Loading configuration

### Concept 2

This concept also minimizes the width of the frame by stacking the devices in a row configuration. The width of the frame is 650 mm whereas the length is 1900 mm. The weight of the construction itself is approximately 132 kg. Configuring the devices in this manner ensures that the mass is centered on the construction, which means there is a low risk of toppling sideways during traveling. The concept described in this section can be seen in Figure (2.6).

Figure 2.6: Illustration of concept 2

This design features two configurations, a loading/unloading configuration and a traveling mode. The two modes can be seen in Figure (2.7) and Figure (2.8) respectively. To change between the modes, a mechanism for simultaneous lowering and rotation is proposed. A screw jack provides vertical movement, whereas a rack and pinion system produces a rotational motion. When the rotational arm is rotating, gravity keeps each row of Looms parallel to ground at all times. The purpose of the design is to grant easy access when loading/unloading the Loomos from the storage unit. In the lowered position, the lifting height required for placing a Loomo onto the storage unit is minimized. This provides an ergonomic position for the operator while loading and unloading the storage unit. Loomos are stabilized on a mount together with a bracket holding the lifting handle in place to secure the Loomos in a vertical position. This design is classified as a high complexity concept. Due to the vertical and rotational movement, electrical wiring and central locking may introduce challenges. Moving between the two modes will also require an additional motor, which further increases the complexity.

Figure 2.7: Travel configuration

Figure 2.8: Loading configuration

### Concept 3

To minimize length and width of the storage unit, the Loomos are placed in a configuration that combines longitudinal and traverses mounting, as shown in Figure (2.9). The length of the frame is 1550 mm, and the width is 750 mm, which makes this the smallest concept. The weight of the construction is approximately 75 kg.

Figure 2.9: Illustration of concept 3

The concept utilizes the lifting handle attached to each Loomo to keep them in place on the storage unit. With the storage unit simplicity, it is manually loaded/unload by lifting each Loomo carefully onto the holding bracket. By their self-weight, the Looms stay attached to the rig. With the handle being curved, the loomo may move sideways during any transportation because of their placement on the vertical beams. In order to keep the Loomos in a stationary position during traveling, a supporting beam which is in contact with the wheels is added. The design has a low complexity, which is favorable considering the manufacturing process. Because the Loomos are stacked quite compact, a specific loading sequence may be required when loading the rig. Considering that the Loomos are mounted statically to the frame, electrical wiring is easy to establish. Contrarily, due to the combination of longitudinal and transverse mounting, central locking may become challenging.

Figure 2.10: Stacking configuration

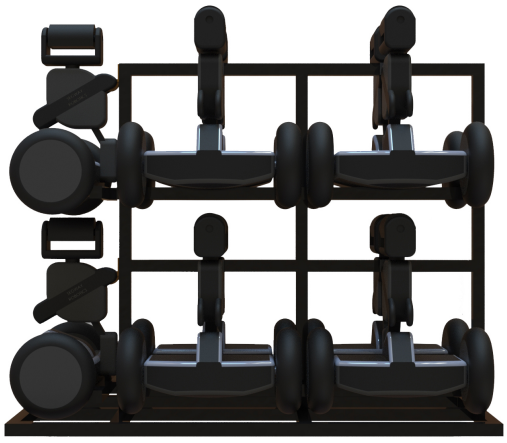


Figure 2.11: Stacking configuration

### Concept 4

This concept minimizes the length and width of the storage unit and features a simple mounting mechanism. The Loomos are stored in two heights where each row consists of five looms stacked in a row arrangement. A length of 1750 mm and a width of 750 mm is achieved by stacking the devices in this manner. Lastly, the construction for this concept weight roughly 70 kg. The design can be viewed in Figure (2.12).

Figure 2.12: Illustration of concept 4

The frame is equipped with a vertical frame containing three load-carrying beams. Each vertical beam is equipped with four holding brackets apart from one, where only two Loomos are attached. To increase the Looms stability during moving, a support beam for each device is mounted strategically to the vertical beams. The support is in contact with the wheels and prevents the unit from wobbling. The design has low complexity and provides the desired improvements concerning size and mounting stability. Traverse mounting of the devices requires some arm extension to attach the Loomos on the storage unit and may result in unfavorable lifting positions for the operator. Further on, the electric wiring is easily achieved because the concept features static mounting of the Loomos. Additionally, the devices are aligned in a row, which provides for easy implementation of a locking system.

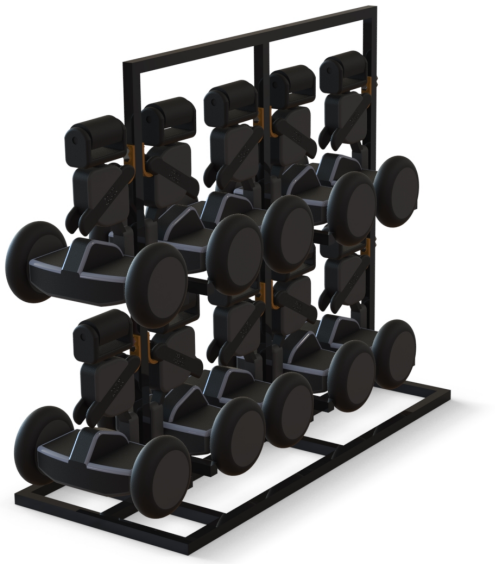


Figure 2.13: Concept 4 in the stacking configuration

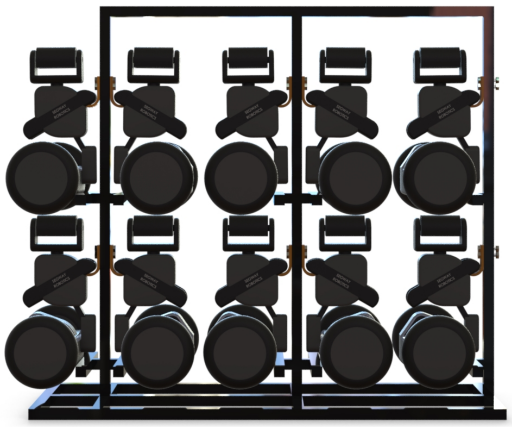


Figure 2.14: Concept 4 in the stacking configuration

### Concept 5

This concept is designed to reduce the length as well as the width of the parking rig. The volume is effectively utilized by stacking five Loomos on each side, where four are placed in a square configuration, and the fifth centralized between them. The result of this configuration is a length of 1615 mm and a width of 750 mm. The construction weighs approximately 80 kg.

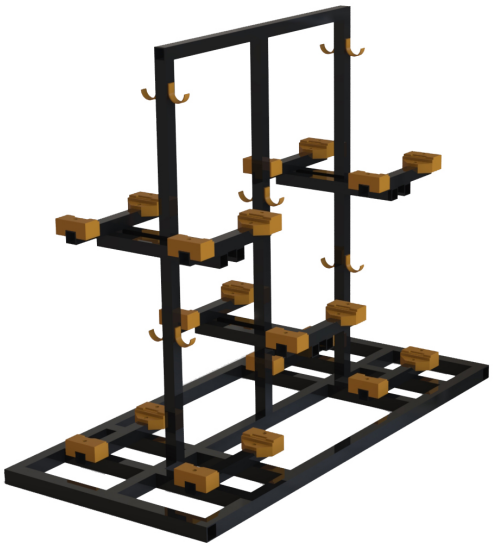


Figure 2.15: Illustration of Concept 5

To ensure that the Loomos are stationary during transportation, each Loomo has a docking block to rest upon. In addition, a bracket is mounted to secure the Loomo on the rig by its lifting handle. The Loomos are stacked relatively compact, which means that some slots may be hard to access if the surrounding slots are occupied. However, the compact and symmetrical configuration facilitates easy electrical wiring and central locking.

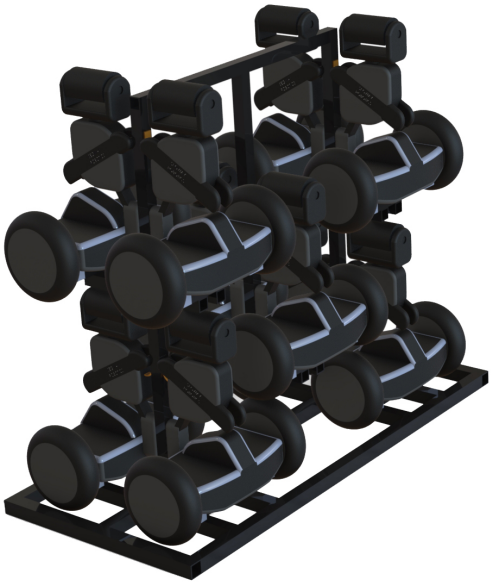


Figure 2.16: Concept 5 in a stacking configuration

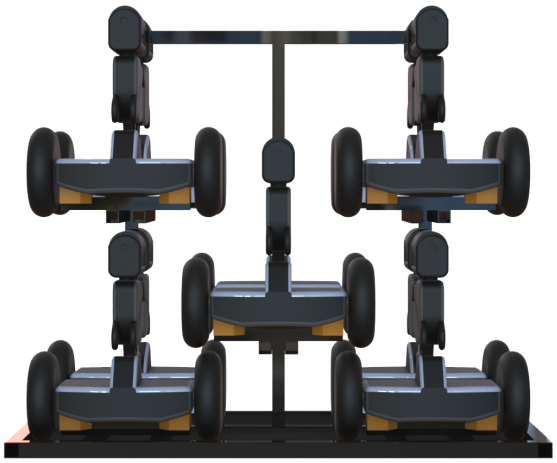


Figure 2.17: Concept 5 in the loading/unloading configuration

## 2.4 Concept Evaluation

A scoring technique was used to choose between the five generated designs. Concept scoring is an excellent method for choosing between multiple concepts and bases the result on an analysis in which the factors are weighted and given ranks. The concepts are judged based on eight evaluation criteria. The most weighted criterion is the size, which considers the length and width of the frame. The next criterion is complexity and concerns how challenging the rig is to construct, for instance, how many movable parts and the geometry of the parts. Another essential factor is to consider how stable the units are mounted to the rig, and this is evaluated with the unit support criterion. How much the rig weights is also important because this has a direct link to the power required to run the platform. To facilitate easy mounting of the Loomos, the concepts are also rated based on how easy the Loomos are to attach/detach and how compact the devices are stacked. If the Loomos are stacked very compact, this may require a specific loading/unloading sequence, and this is evaluated with the stacking sequence criterion. Next, the concepts are evaluated based on how well they relief the operator from unfavorable lifting positions, which is taken into account with the ergonomic criterion. Lastly, the concepts are ranked based on how easy it is to achieve a locking system and electrical wiring. When performing the scoring process, all the evaluation criteria were worked across, i.e all the concepts were ranked by size, then complexity and so on. The scoring results can be seen in Table (2.1).

| | | Concept 1 | | Concept 2 | | Concept 3 | | Concept 4 | | Concept 5 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Evaluationcriteria | Weight | Rank | Weighted score | Rank | Weighted score | Rank | Weighted Score | Rank | Weighted score | Rank | Weighted score |
| Size | 30 | 2 | 60 | 1 | 30 | 5 | 150 | 3 | 90 | 4 | 120 |
| Complexity | 20 | 2 | 40 | 1 | 20 | 4 | 80 | 5 | 100 | 3 | 60 |
| Unit support | 15 | 4 | 60 | 3 | 45 | 2 | 30 | 1 | 15 | 5 | 75 |
| Weight | 10 | 2 | 20 | 1 | 10 | 5 | 50 | 4 | 40 | 3 | 30 |
| Weight distribution | 10 | 4 | 40 | 5 | 50 | 1 | 10 | 3 | 30 | 2 | 20 |
| Stacking sequence | 5 | 4 | 20 | 5 | 25 | 2 | 10 | 1 | 5 | 3 | 15 |
| Ergonomity | 5 | 4 | 20 | 5 | 25 | 2 | 10 | 1 | 5 | 3 | 15 |
| Wiring & locking | 5 | 2 | 10 | 1 | 5 | 3 | 15 | 4 | 20 | 5 | 25 |
| Total score | | 270 | | 210 | | 355 | | 305 | | 360 | |
| Proceed? | | No | | No | | No | | No | | Yes | |

Table 2.1: Concept scoring

The most suitable concept came to be concept 5 because it was concluded to be short in length, simple and symmetrical.

## 2.5 Concept Details

Once a concept was chosen, the details concerning mounting and locking were further developed.

### Loomo stand

The Loomos are designed with the ability to be stored vertically. A support block is designed for the Loomo unit to stand on. The design utilizes the Loomos flatspots on the underside, which provide good stability. Two support blocks is mounted one each side of a metal transverse bar, as shown in Figure (2.18).

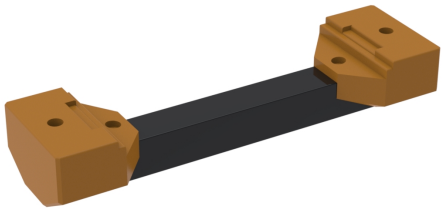


Figure 2.18: Mounting block

### Mounting bracket

In addition to the support blocks which the Loomo stands on, an attaching device was mounted, as seen in Figure 2.19. The mechanism consists of the main bracket (purple), a guide block (green), which guides the lock ring (red) around the lifting handle of the Loomo.

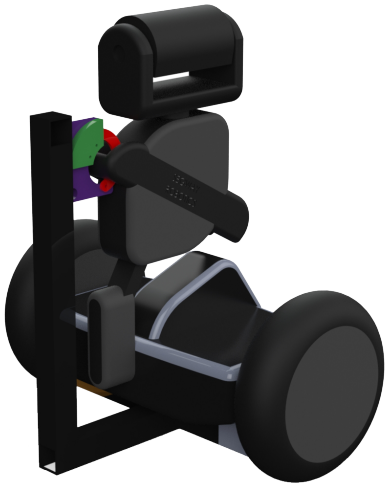


Figure 2.19: Mounting bracket

### Locking mechanism

To prevent unauthorized personnel from accessing the devices, a locking system was designed. Preferably, the mechanism should be able to lock all devices simultaneously and require as few locks as possible. The developed mechanism consist of six rotatable shafts, two for each row and connecting links between them. When the upper shaft is rotated, the links will apply the same rotation to the other rows and enables synchronous rotation. A hook is mounted to the rotating shafts, and will connect to a knob on the lock ring, which will prevent the ring from opening.

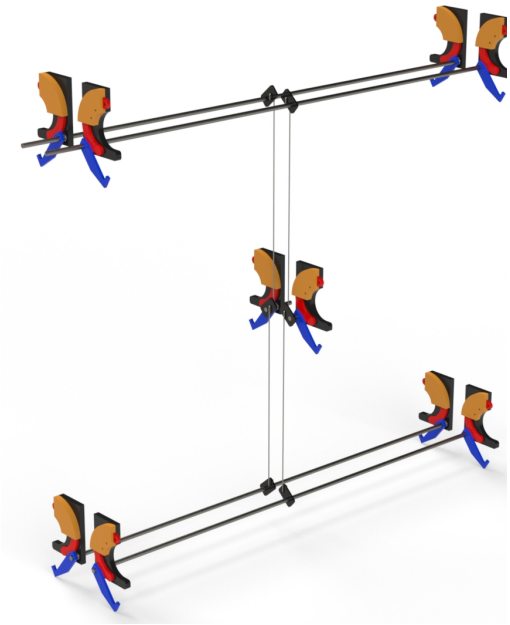


Figure 2.20: Locking mechanism

# 3. Mechanical Design

The mechanical design chapter focuses on the design of necessary mechanical components to realize the selected concept. Evaluation of material selection, shaft design, bearing lifetime calculations,and weld durability are covered. Additionally, an introduction to mecanum wheels as well as a detailed force analysis for the wheels is documented in this chapter.

## 3.1 Material Selection

The material for the frame was selected based on the available material at the university. The material is of type structural steel with a square hollow section of $40 \times 40$ mm and a thickness of 3 mm. The minimum yield strength of the material is 355 MPa. To verify that the chosen material was suitable for the application, a finite element method analysis was performed using ABAQUS. The final concept was created as a shell structure in SOLIDWORKS and imported to ABAQUS as a single part.

Two load cases were analysed, one scenario when the rig is fully loaded and one scenario where the rig is only loaded with Loomos on a single side. The load from each Loomo was applied to a multi-point constraint (MPC) with beam type. The MPC constrains the slave nodes of a region to a single point. Further on, the load was applied to this point. When loading the rig fully, the maximum stresses in the construction are located at the contacting corner of the supporting bracket. The value of the stress is approximately 37 MPa, which is far from the minimum yield strength. The results can be seen in Figure (3.1).

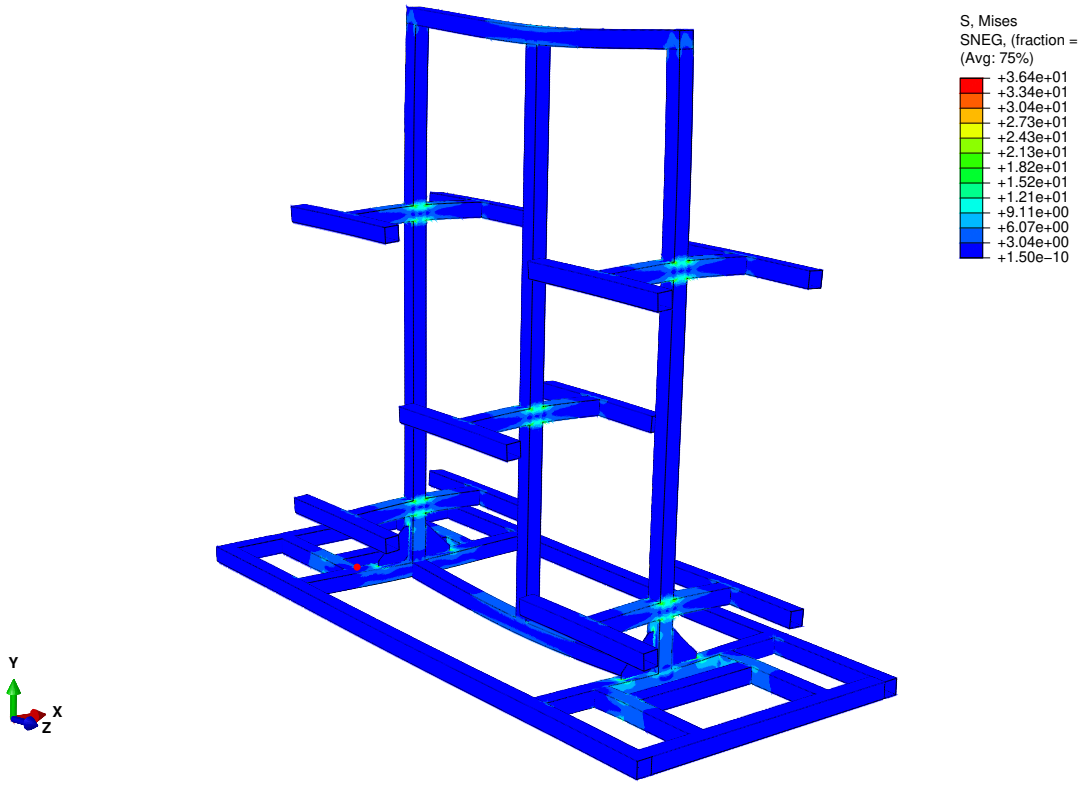



Figure 3.1: Fully loaded Rig, Deformation scale factor: 200

When partially loading the rig, the maximum stresses in the construction are also located in the support bracket. However, the stress is slightly higher and has a value of approximately 63 MPa. Still, the results are far below the maximum limit. The results can be seen in Figure (3.2).

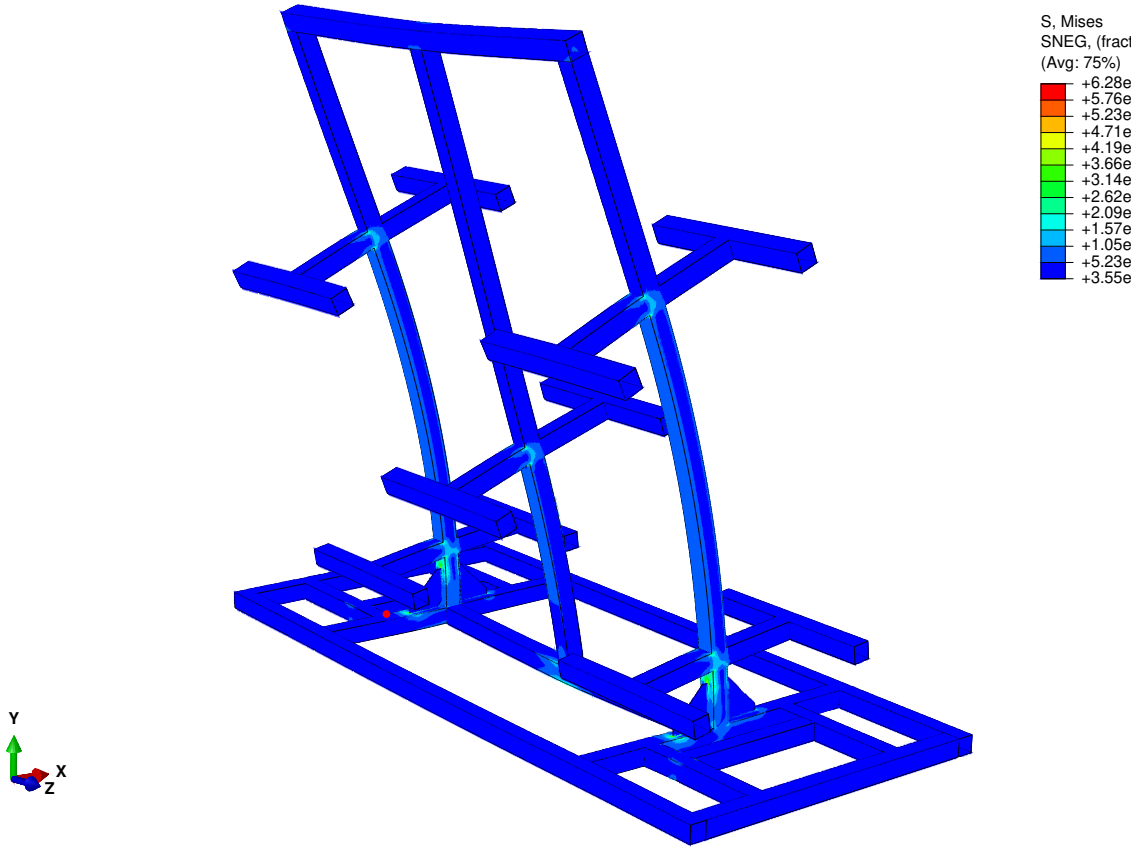



Figure 3.2: Partially loaded rig, Deformation scale factor: 200

## 3.2 Mecanum Wheels

The vehicle in this project is equipped with four mecanum wheels to maximize the rig's mobility. A mecanum wheel is a locomotion mechanism with three degrees of freedom (DOF); rotation around the wheel axle, around the rollers, and the contact point. The mechanism functions as a conventional wheel, but it also has low resistance in the lateral direction. The effect of utilizing mecanum wheels is that the vehicle can move in all directions and rotate independently, also referred to as omnidirectional movement. An image of a mecanum wheel can be seen in Figure (3.3).

Figure 3.3: Mecanum wheel

The omnidirectional behavior is possible because the wheel's external circumference is equipped with multiple rollers that are inclined relative to the axis of rotation. Each roller can rotate freely, and depending on the rotational direction of the wheel, the rollers will apply an axial force to the contact surface and push the wheel to either side. By mounting the four wheels in a specific configuration, holonomic drive can be achieved. To utilize the holonomic ability of mecanum wheels, the system requires one motor for each wheel. All wheels are driven individually, and by controlling the rotating direction in a specific manner, omnidirectional movement is obtained. The main disadvantages of adding the extra degrees of freedom are; an accumulation of slippage, tendencies for reduced dead-reckoning accuracy, and increased design complexity. [20]

The mecanum wheel selected for this project is an 8" wheel from AndyMark. This wheel has a maximum load capacity of approximately 225 kg, per wheel. On the circumference of the wheel, 12 rollers are mounted at an 45° angle. The rollers are molded with a nylon core surrounded by polyurethane which ensures high durability and traction. Further on, the roller rotates on a brass axle tube contained by a screw passing through it. The core of the wheel consist of poly-carbonate and features steel plates riveted on the side. An exploded view of the mecanum wheel assembly is shown in Figure (3.4). [21]

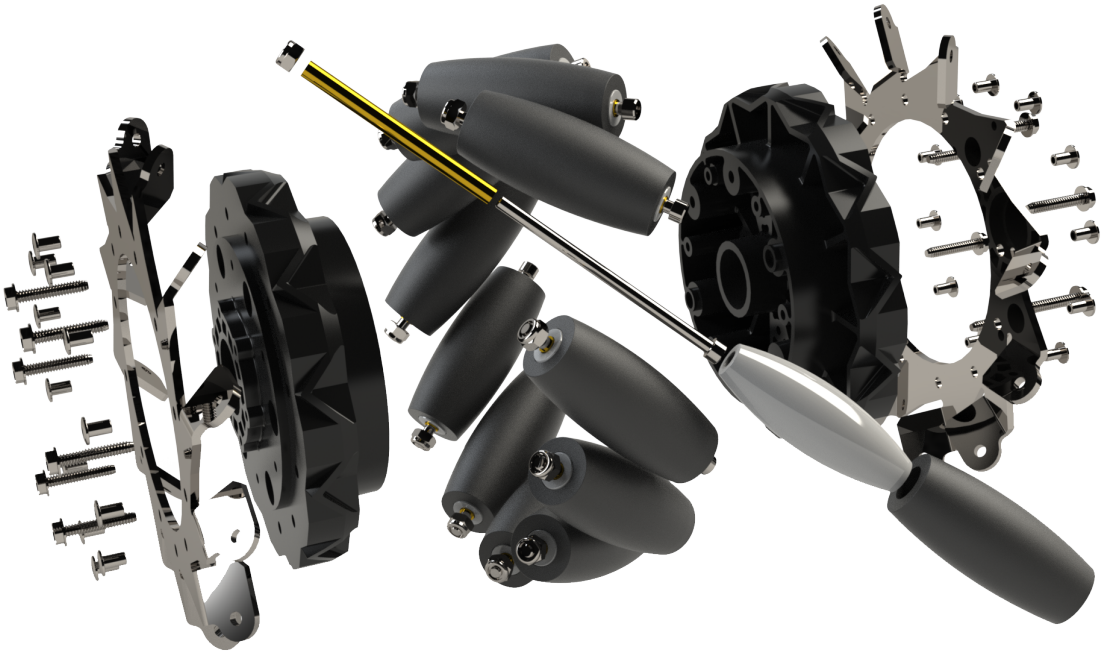


Figure 3.4: Andymark 8" Mecanum wheel exploded view

### 3.2.1 Force Analysis

The dynamics of a mecanum wheel allows it to create a driving force in both the x- and y-direction when torque is applied to the axle. It is due to the orientation of the rollers that the resulting force on the wheel's shaft is acting at an angle [22]. This force is shown in Figure (3.5) as $F_2$, where $F_1$ and $F_3$ are the decomposed force components. By having an angle of $\alpha = 45°$ the force components will always be an equal pair. However, their directions may vary. Hence, $|F_1| = |F_3|$ becomes true for this instance.

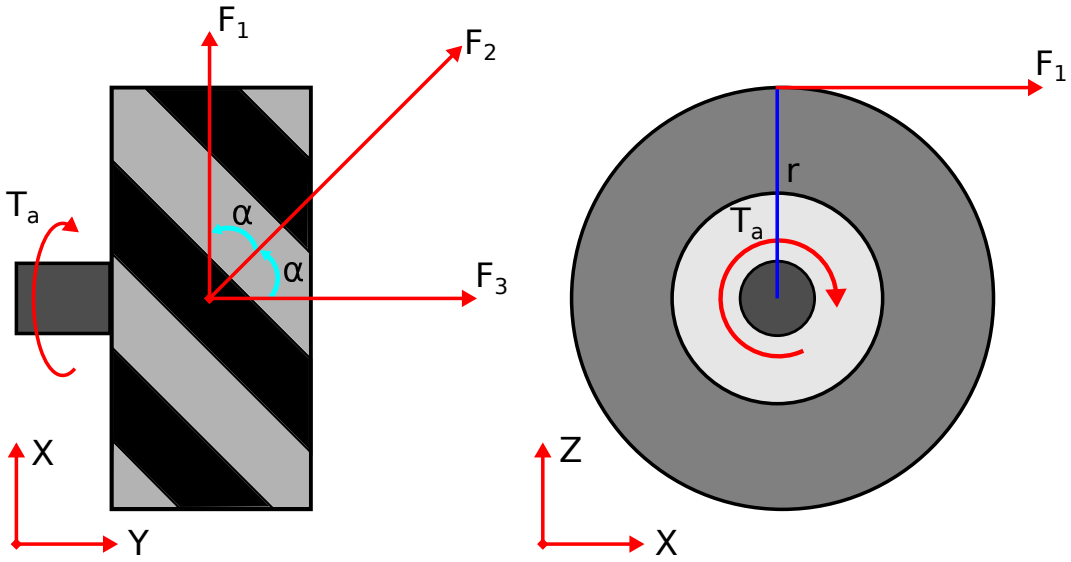



Figure 3.5: Shaft torque to resulting force

The relation between $F_1$ and $F_2$ is used in conjunction with Figure (3.5) in order to derive the relation between the applied wheel shaft torque and the resulting roller forces. The relations are shown in Equation (3.1) - (3.3).

$$F_1 = \frac{T_a}{r} \tag{3.1}$$

$$F_3 = F_1 = \frac{T_a}{r} \tag{3.2}$$

$$F_2 = \frac{\sqrt{2}T_a}{r} = \frac{F_1}{\cos(\alpha)} = \frac{F_3}{\cos(\alpha)} \tag{3.3}$$

where:

| Symbol: | | Description: | Value: | Unit: |
|---|---|---|---|---|
| $F_1$ | - | Tangential force component | $\sim$ | $N$ |
| $F_2$ | - | Total force from applied torque | $\sim$ | $N$ |
| $F_3$ | - | Axial force component | $\sim$ | $N$ |
| $T_a$ | - | Torque applied to wheel shaft | $\sim$ | $Nm$ |
| $r$ | - | Wheel radius with roller | $\sim$ | $m$ |

Before evaluating the forces acting on each wheel, the mounting configuration has to be determined. There are multiple possible configurations which yields a omnidirectional drive. However, their omnidirectional motion capacity differs. There are two configurations which are the most common for use in scientific research and industrial applications [23]. These are going to be evaluated, and are going to be denoted as the square and cross configuration. When discussing the different wheel configurations, they are described from a worm-perspective. More specifically, from the ground up. This view is selected since their differences are better emphasised from this perspective, as these configurations flip when viewed from the top down. It should be noted that this is the only time that the worm-perspective is going to be applied in this chapter. In Figure (3.6) the two configurations are shown. The square configuration creates a square with the lines created by following the roller's angle to where they intersect. Whilst the cross configuration creates a cross by following where the adjacent wheel pair intersect. Hence, the reason for their names. Both configurations achieve omnidirectional ability. However, the square configuration can cause loss of omnidirectional ability. This occurs if the rollers intersects into a single common center. I.e, when the wheels are mounted into a square with equal width and height. This occurs due to the kinematic system matrix transforming into a matrix of not full column rank [23]. Thus, all terms in the column affecting the change in angle becomes zero.

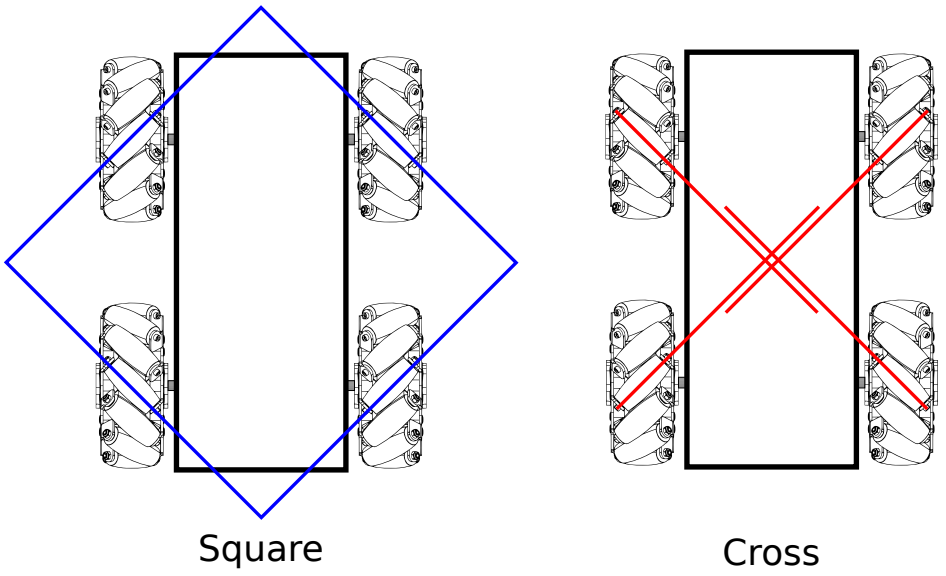



Figure 3.6: Square and cross configuration (Bottom view)

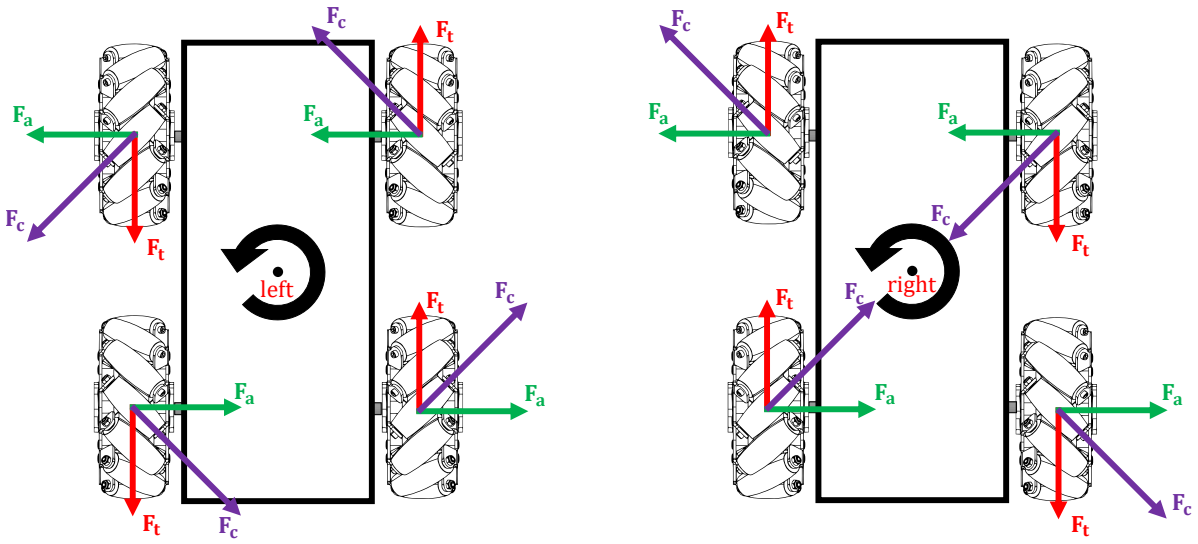



Figure 3.7: Square and cross configuration forces while rotating in place (Bottom view)

The dynamic principles of the two wheel configurations presented are almost identical. They only differ when it comes to rotation around its own axis. In order to evaluate the difference between them, the forces required to rotate in counter clockwise are added as shown in Figure (3.7). Then the differences are evaluated by applying the figure and calculating the sum of moments about the center of rotation. The square and cross configuration moments are calculated in Equation (3.4) and Equation (3.5) respectively, where $h > w$.

$$\sum \overset{+}{\circlearrowleft} T_{left} = wF_t + wF_t + wF_t + wF_t + hF_a + hF_a + hF_a + hF_a \tag{3.4}$$

$$\Downarrow$$

$$\sum \overset{+}{\circlearrowleft} T_{left} = 4wF_t + 4hF_a$$

$$\sum \overset{+}{\circlearrowleft} T_{right} = -wF_t - wF_t - wF_t - wF_t + hF_a + hF_a + hF_a + hF_a \tag{3.5}$$

$$\Downarrow$$

$$\sum \overset{+}{\circlearrowleft} T_{right} = -4wF_t + 4hF_a$$

where:

| Symbol: | | Description: | Value: | Unit: |
|---|---|---|---|---|
| $\sum \overset{+}{\circlearrowleft} T_{left}$ | - | Sum of moments about center (square) | $\sim$ | $Nm$ |
| $\sum \overset{+}{\circlearrowleft} T_{right}$ | - | Sum of moments about center (cross) | $\sim$ | $Nm$ |
| $F_t$ | - | Tangential force | $\sim$ | $N$ |
| $F_a$ | - | Axial force | $\sim$ | $N$ |
| $w$ | - | Width moment arm from center | $\sim$ | $m$ |
| $h$ | - | Height moment arm from center | $\sim$ | $m$ |

As emphasized in Equation (3.4), the cross configuration produces moments working against the wanted rotation, while from Equation (3.5) the square configuration does not. Thus, the latter configuration is selected, since it is more effective. In addition, as previously discussed, the loss of omnidirectional ability can occur for the cross configuration. It can be seen that if; $w = h$, the sum of moments will always become zero. Hence, eliminating the possibility of rotation about center, since $\alpha = 45°$, $F_t = F_a$.

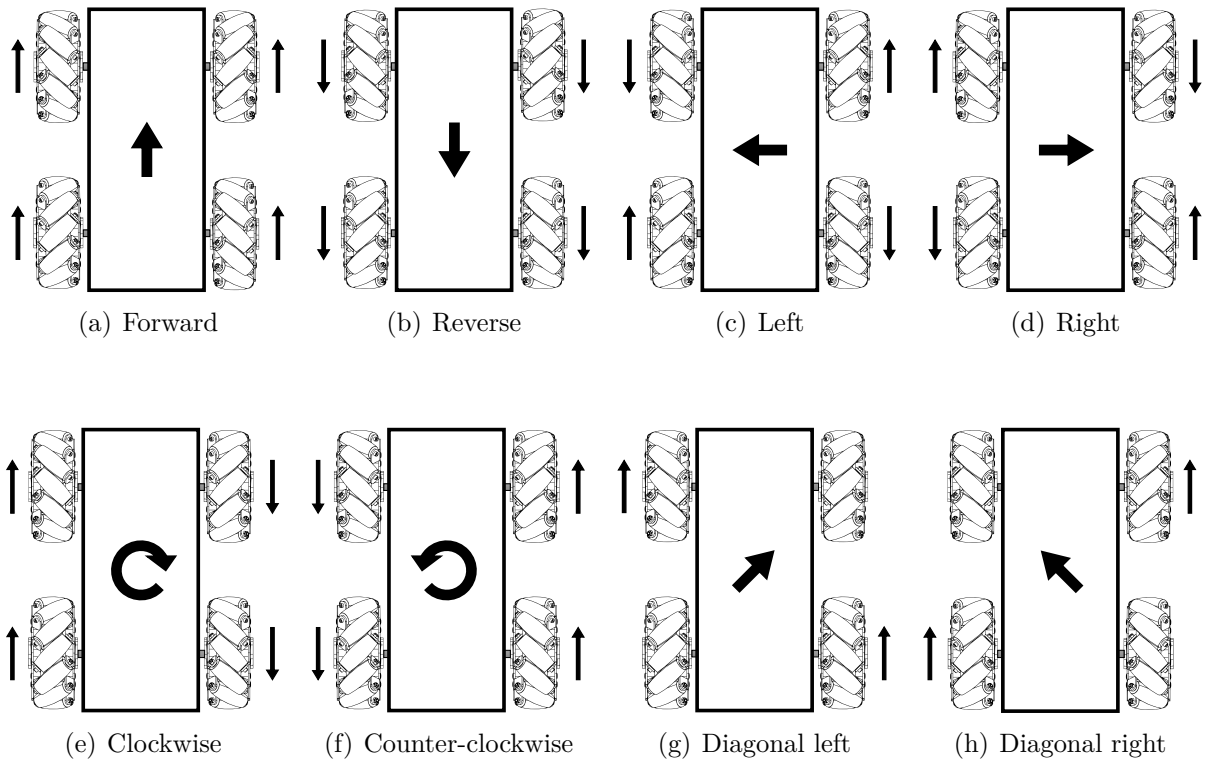
Lastly, the measured length of the Loomo rig is used in order to check the torque difference between the two configurations and how the selection affects the Loomo rig. This is going to be done by calculating the ratio applying the measured width and height values in Equation (3.6). The calculation shows that by selecting the square configuration, the Loomo rig will rotate with a torque sum which is about 2.6 times stronger.

$$k = \frac{\sum \overset{+}{\circlearrowleft} T_{left}}{\sum \overset{+}{\circlearrowleft} T_{right}} = \frac{h + w}{h - w} \tag{3.6}$$

where:

| Symbol: | | Description: | Value: | Unit: |
|---|---|---|---|---|
| $k$ | - | Moment strength difference | 2.63 | $-$ |
| $w$ | - | Width moment arm from center | 0.285 | $m$ |
| $h$ | - | Height moment arm from center | 0.635 | $m$ |

Figure (3.8) illustrates how each individual wheel has to be controlled according to the chosen configuration, in order to obtain specific motion. These are shown in bird-perspective. Specifically, from the top, viewing down towards the ground.



(a) Forward (b) Reverse (c) Left (d) Right

(e) Clockwise (f) Counter-clockwise (g) Diagonal left (h) Diagonal right

Figure 3.8: Mecanum wheel configuration (Top view)

When designing a drive train for a mobile system, it is important to evaluate the torque required to move the platform. If the motor is not able to provide enough torque to the system, the motor may stall or stop. The total weight of the construction is 320 kg, however, the weight is assumed to be 400 kg to ensure that the calculations are conservative. The assumption made implies a safety factor of 1.2. It is also necessary to specify the operating environment of which the product is designed for. More specifically, it is crucial to consider whether the platform is to be used on flat ground or in slopes, as the latter requires more torque. For this project, the vehicle is limited to planar movement and is not designed to operate in slopes.

Assuming that the total weight of the vehicle is equally distributed between the four mecanum wheels, the torque acting on each wheel can be examined separately. Generally, static friction has to be overcome when an object is transitioning between the state of rest to motion. As soon as the object is in motion, the kinetic friction comes into play. For a wheel, the static friction prevents the wheel from sliding along the surface, resulting in the wheel rolling. When examining rolling elements, traction is also introduced. This is described as the amount of force a wheel can apply to a given surface before it slips. The following analysis does not take into account slippage between the rollers and the contact surface, which means that all applied toque is converted into traction force for the wheel. Further on, the kinematic friction for a wheel is known as rolling friction. This type of friction is defined as the restive force that slows down the motion of a rolling ball or wheel. [24]

Although mecanum wheels provide impressive maneuverability and mobility of a vehicle, the locomotion mechanism is prone to shifting of the mass center. If the mass center is substantially shifted, it will influence the force distribution on the wheels and may lead to inadequate control of the motion.

A set of equations are derived to evaluate the force requirements for allowing omnidirectional movement. These were evaluated as the sum of forces in their respective plane of action. In Figure (3.9), the forces acting on a mecanum wheel moving diagonally is shown. Further on, it has been given a load vertically on the driving axle, and the axle itself is able to produce a driving torque. In the calculation of the rolling resistance, denoted $F_{rr}$ in Figure (3.9), the coefficient of polyurethane in contact with steel flooring was found in a table to be 0.03 [25]. However, this value is the greater than the applied coefficient for a car tire in contact with gravel [26]. This lead to the decision of selecting the rolling coefficient of car tire in contact with concrete in the calculations. This coefficient is 0.01. The friction force from the mecanum wheels roller's rolling is displayed. As there is no information given about the rolling friction from the producer, in addition to being typically ignored in dynamic analysis, the friction force from the rollers is going to be neglected [27][28]. It should also be noted that $F_t$ and $F_a$ are the force components of $F_c$ in X- and Y-direction

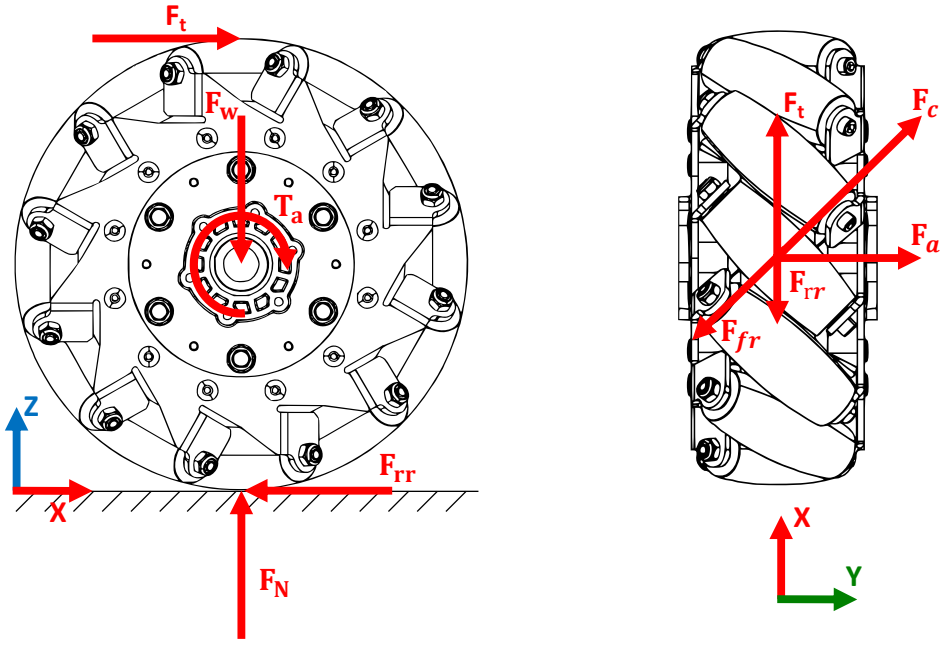



Figure 3.9: Forces on a mecanum wheel

$$F_t = \frac{T_a}{r} \tag{3.7}$$

$$|F_a| = |F_t| \tag{3.8}$$

$$F_c = \sqrt{2}F_t \tag{3.9}$$

$$m = \frac{m_{tot}}{4} \tag{3.10}$$

$$F_{rr} = \mu_r F_n \tag{3.11}$$

$$F_w = F_n = mg \tag{3.12}$$

$$F_{fr} = 0 \quad \text{Since it is neglected} \tag{3.13}$$

In order to move the vehicle as a whole in a straight motion, all wheels have to be driven at the same velocity. The forces acting on the rollers during this scenario is illustrated in Figure (3.10). The figure is shown in bird-perspective and explains how the axial forces on the left and right wheels are opposing each other. This results in the axial forces being canceled and the remaining tangential forces produce a forward motion of the vehicle.

The minimum force required in order to initialize movement of the rig is then calculated with Equation (3.14). In the equation it is assumed that each wheel contributes with equal force. As can be seen from the Figure (3.10), the forces in Y-direction cancel each other out.

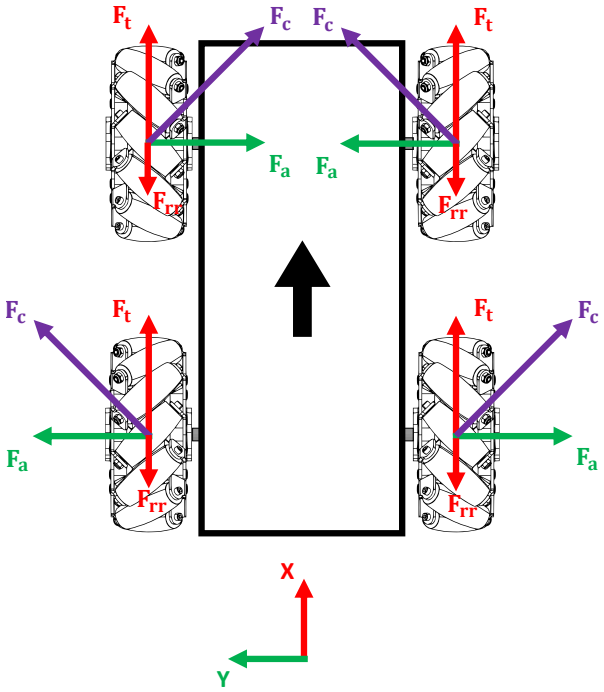


Figure 3.10: Forces acting on roller from contact surface - Forward (Top view)

$$\sum F_x = 0 = 4F_t - 4F_{rr} \Rightarrow F_t = \mu_r mg \tag{3.14}$$

$$\sum F_y = 0 = 2F_a - 2F_a \tag{3.15}$$

$$\sum \overset{+}{\circlearrowleft} T_O = 0 = 2hF_a - 2hF_a + 2wF_t - 2wF_t + 2wF_{rr} - 2wF_{rr} \tag{3.16}$$

The calculations indicates that the total output force produced from all wheels has to be 39.2 N. Then, the sum of the torque each drive shaft has to produce is calculated to be 4.0 Nm with Equation (3.17).

$$T_a = 4F_t r \tag{3.17}$$

where:

| Symbol: | | Description: | Value: | Unit: |
|---|---|---|---|---|
| $F_t$ | - | Tangential force | 9.8 | $N$ |
| $F_n$ | - | Normal force | 3924 | $N$ |
| $T_a$ | - | Torque acting on axle | 1.0 | $Nm$ |
| $m_{tot}$ | - | Loomo rig total mass | 400 | $kg$ |
| $r$ | - | Radius of the wheel | 0.1015 | $m$ |
| $g$ | - | Gravity | 9.81 | $m/s^2$ |
| $\mu_r$ | - | Rolling friction coefficient | 0.01 | $-$ |

In Figure (3.11), the forces which has to be produced from the drive shaft in order to move laterally is shown. When moving laterally, the mecanum wheels act as a wheel rolling along the lateral axis, thus rolling resistance is thus added to work against the rolling direction. The minimum force required in order to initialize movement of the Loomo rig in lateral direction in calculated with Equation (3.19). As can be seen from Figure (3.11), the forces acting in X-direction cancel each other out.

$$\sum F_x = 0 = 2F_t - 2F_t \tag{3.18}$$

$$\begin{aligned} \sum F_y = 0 &= 4F_a - 4F_{rr} \\ \Rightarrow F_a &= \mu_r mg \end{aligned} \tag{3.19}$$

$$\begin{aligned} \sum \overset{+}{\circlearrowleft} T_O = 0 &= 2hF_a - 2hF_a + 2wF_t \\ &- 2wF_t + 2wF_{rr} - 2wF_{rr} \end{aligned} \tag{3.20}$$

The calculations indicates that the total output torque produced from all wheels has to be 39.2 N. Then, the sum of the torque each drive shaft has to produce is calculated to be 4.0 Nm with Equation (3.21). Exactly the same as for moving forwards.

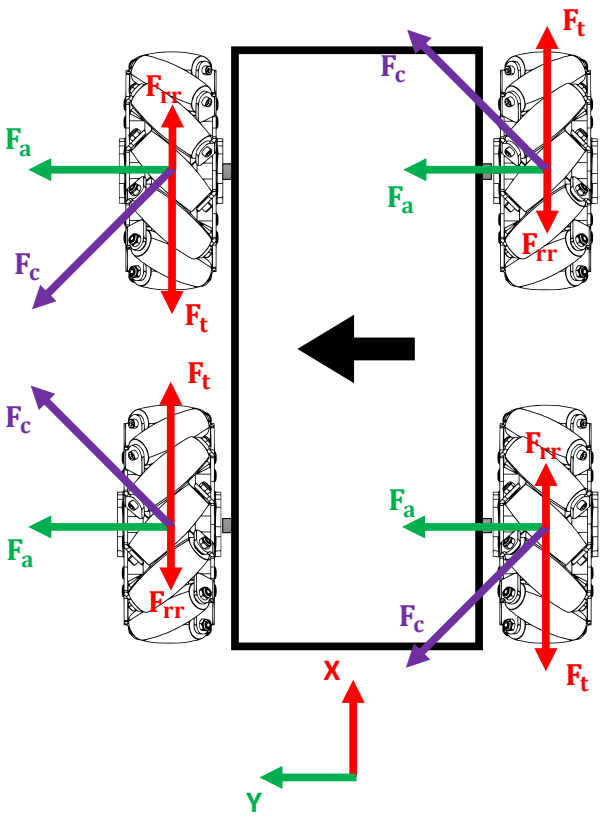



Figure 3.11: Forces acting on roller from contact surface - Sideways (Top view)

$$T_a = \mu_r mgr \tag{3.21}$$

where:

| Symbol: | | Description: | Value: | Unit: |
|---|---|---|---|---|
| $F_a$ | - | Axial force | 9.81 | $N$ |
| $F_n$ | - | Normal force | 3924 | $N$ |
| $T_a$ | - | Torque acting on axle | 1.0 | $Nm$ |
| $m_{tot}$ | - | Loomo rig total mass | 400 | $kg$ |
| $r$ | - | Radius of the wheel | 0.1015 | $m$ |
| $g$ | - | Gravity | 9.81 | $^m/_{s^2}$ |
| $\mu_r$ | - | Rolling friction coefficient | 0.01 | $-$ |

In order to rotate counter clockwise, the forces produced from the driving shafts have to be in accordance with Figure (3.12). The minimum moment required to initialize rotation is calculated with Equation (3.24). The point of rotation in the equation is defined as the center of the Loomo rig. As can be seen from Figure (3.12), all of the forces cancel each other out from moving in the X- and Y-direction.

$$\sum F_x = 0 = 2F_t - 2F_t + 2F_{rr} - 2F_{rr} \quad (3.22)$$

$$\sum F_y = 0 = 2F_a - 2F_a \quad (3.23)$$

$$\sum \overset{+}{\circlearrowleft} T_O = 0 = 4wF_t + 4hF_a - 4wF_{rr} \quad (3.24)$$

Since the magnitude of $F_t$ and $F_a$ are equal, the latter is substituted into the equation for the minimum force each wheel has to produce in order to initialize rotation. It is calculated with in Equation (3.25) to be about 3.0 N .

$$F_t = \frac{wF_{rr}}{(w+h)} \quad (3.25)$$

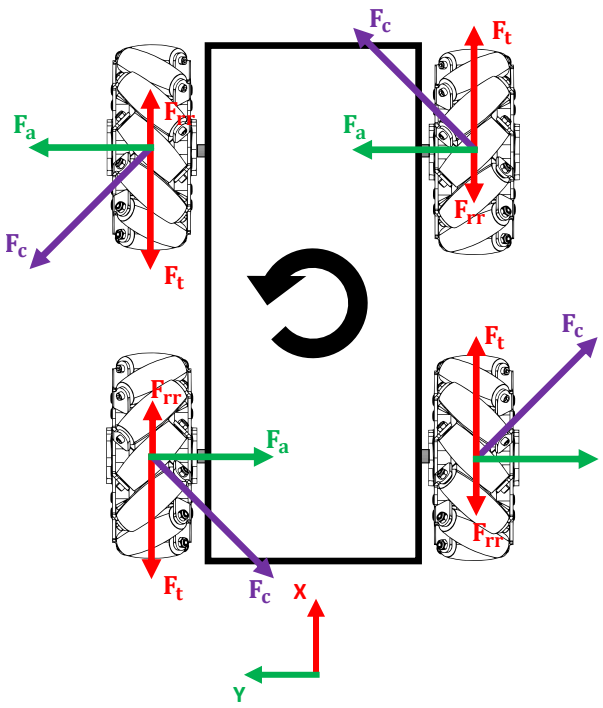



Figure 3.12: Forces acting on roller from contact surface - Rotational (Top view)

Then, the individual torque the drive shafts has to produce is calculated with Equation (3.26) to be 0.3 Nm.

$$T_a = \frac{wF_{rr}}{(w+h)} r \quad (3.26)$$

where:

| Symbol: | | Description: | Value: | Unit: |
|---|---|---|---|---|
| $F_t$ | - | Tangential force | 6.1 | $N$ |
| $F_n$ | - | Normal force | 3924 | $N$ |
| $T_a$ | - | Torque acting on axle | 0.3 | $Nm$ |
| $m_{tot}$ | - | Loomo rig total mass | 400 | $kg$ |
| $r$ | - | Radius of the wheel | 0.1015 | $m$ |
| $g$ | - | Gravity | 9.81 | $m/s^2$ |
| $\mu_r$ | - | Rolling friction coefficient | 0.01 | $-$ |

It is also of interest to evaluate the scenario which requires the most torque to accelerate. Looking at Figure (3.8), scenario (3.8(g)) and (3.8(h)) are driven by only two motors and will require them to provide a larger amount of torque to accelerate. The wheels which are not controlled will only be rolling due to the transverse motion. The force distribution for scenario (3.8(g)) is illustrated in Figure (3.13). Equation (3.27)-(3.29) evaluates the required torque to initialize movment of the vehicle in the diagonal direction.

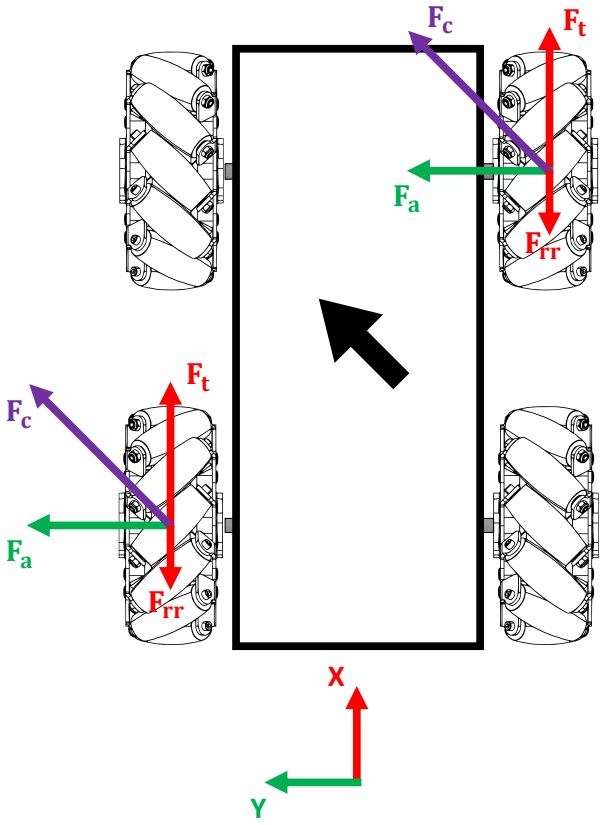



Figure 3.13: Forces acting on roller from contact surface - Diagonally (Top view)

$$\sum F_x = 0 = 2F_t - 2F_{rr} \tag{3.27}$$

$$\sum F_y = 0 = 2F_a \tag{3.28}$$

$$\sum \overset{+}{\circlearrowleft} T_O = 0 = hF_a - hF_a + wF_t - wF_t + wF_{rr} - wF_{rr} \tag{3.29}$$

where:

| Symbol: | | Description: | Value: | Unit: |
|---|---|---|---|---|
| $F_t$ | - | Tangential force | 9.8 | $N$ |
| $F_n$ | - | Normal force | 3924 | $N$ |
| $T_a$ | - | Torque acting on axle | 1.0 | $Nm$ |
| $m_{tot}$ | - | Loomo rig total mass | 400 | $kg$ |
| $r$ | - | Radius of the wheel | 0.1015 | $m$ |
| $g$ | - | Gravity | 9.81 | $^m/_{s^2}$ |
| $\mu_r$ | - | Rolling friction coefficient | 0.01 | $-$ |

The analysis show that the axle has to be subjected to a torque of 1.0 Nm as minimum torque to initialize movement. However, only two wheels are used to accelerate the vehicle in diagonal direction only.

## 3.3 Shaft

To convert the rotational energy from the mecanum wheel into linear motion for the base frame, a shaft that can tolerate the loads acting on the rig was designed with S355 steel. An illustration of a shaft design is presented in Figure (3.14). By analysing the support of the axle and the applied load, diagrams which represent the bending moment, shear force and axial force were derived.

Figure 3.14: Shaft design

An illustration of a wheel suspension design is shown in Figure (3.15). The shaft will be mounted to the frame by two pillow block bearing units, represent in purple. To fix the mecanum wheel to the drive shaft, a set of hubs were designed. These are illustrated in silver at both sides of the mecanum wheel. A flat surface has been machined onto the shaft surface in order to give set screws better contact surface, as shown in Figure (3.14).

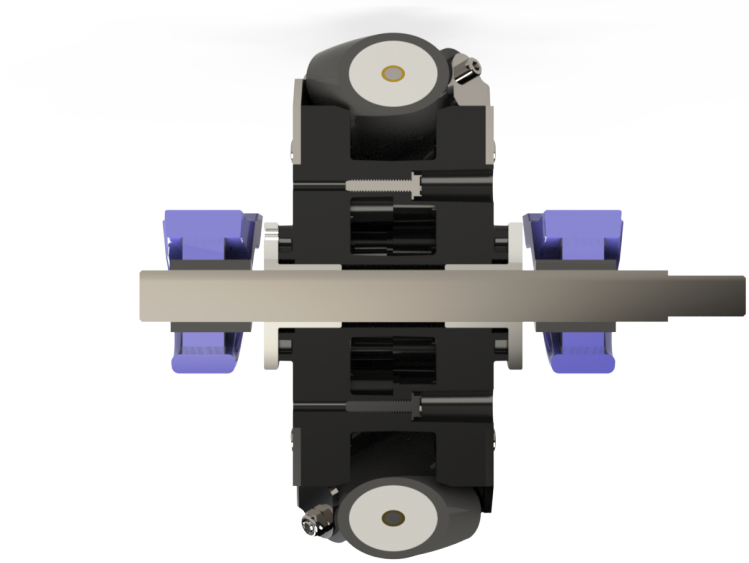


Figure 3.15: Wheel suspension

### Force Analysis

An evaluation of the force and moments produced due to applied load in motion is performed. The evaluation is made for both XY and ZY plane, presented with Figure (3.17) and Figure (3.19) respectively. The drive shaft is supported at both ends and the forces is represent at both sides of a the centered wheel. The results of the analysis in XY are presented with Figure (3.17) and results for ZY in Figure (3.19).

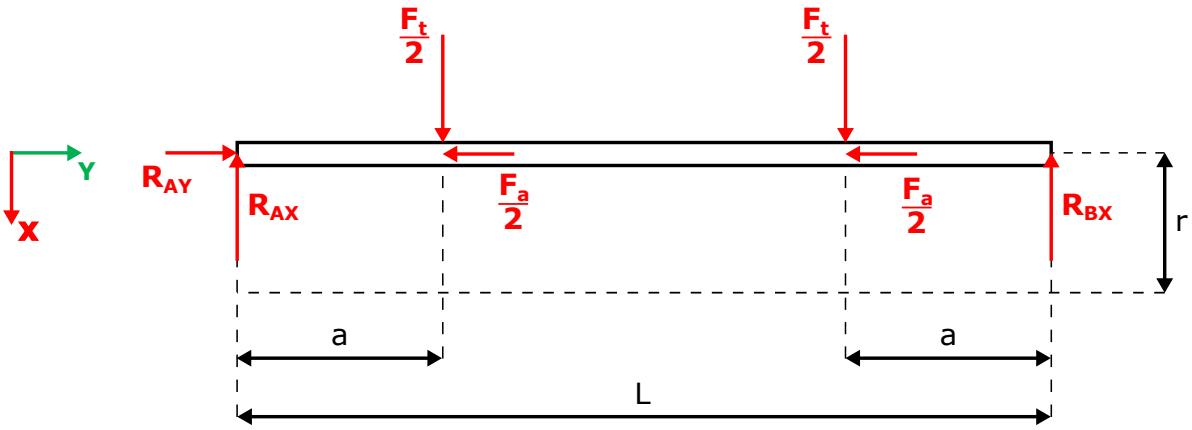



Figure 3.16: Forces acting on the shaft in XY-plane

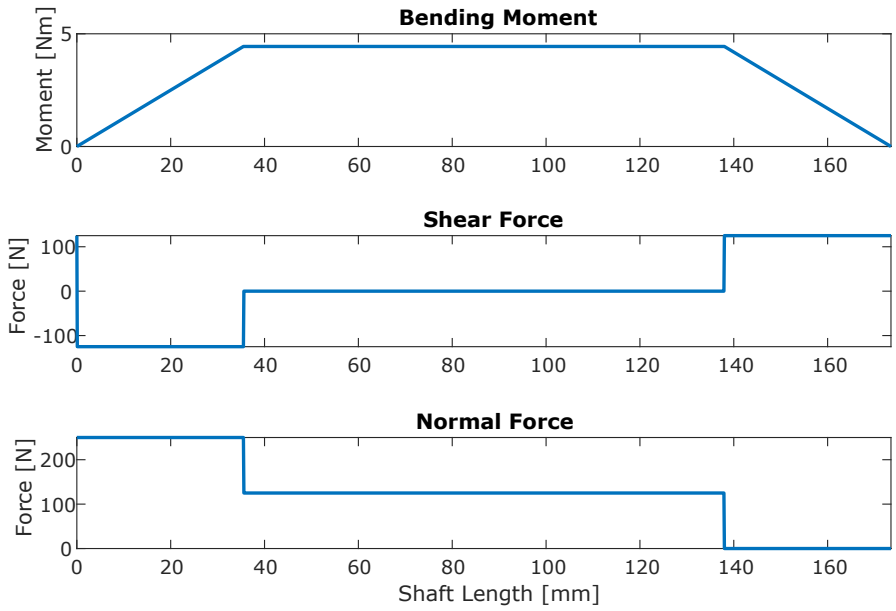



Figure 3.17: Bending, shear and normal diagrams

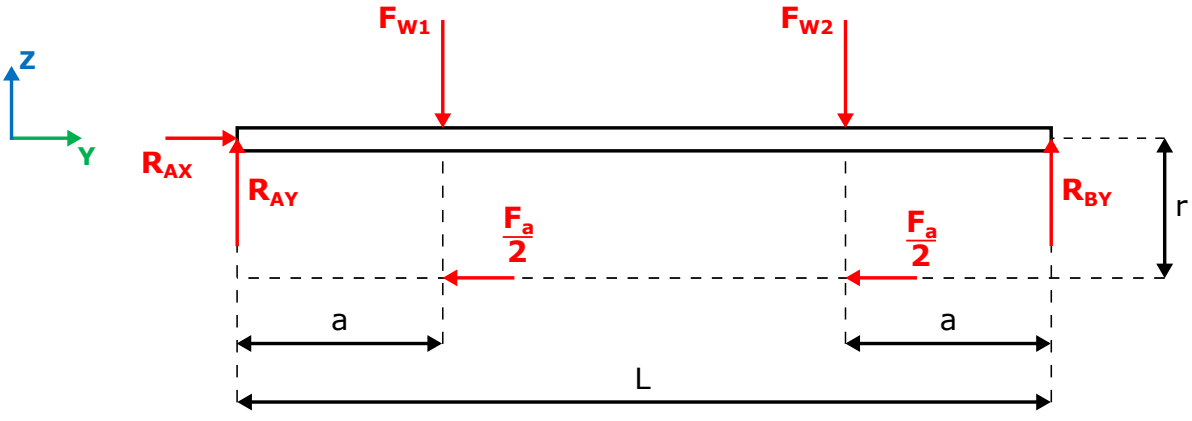



Figure 3.18: Forces acting on the shaft in ZY-plane

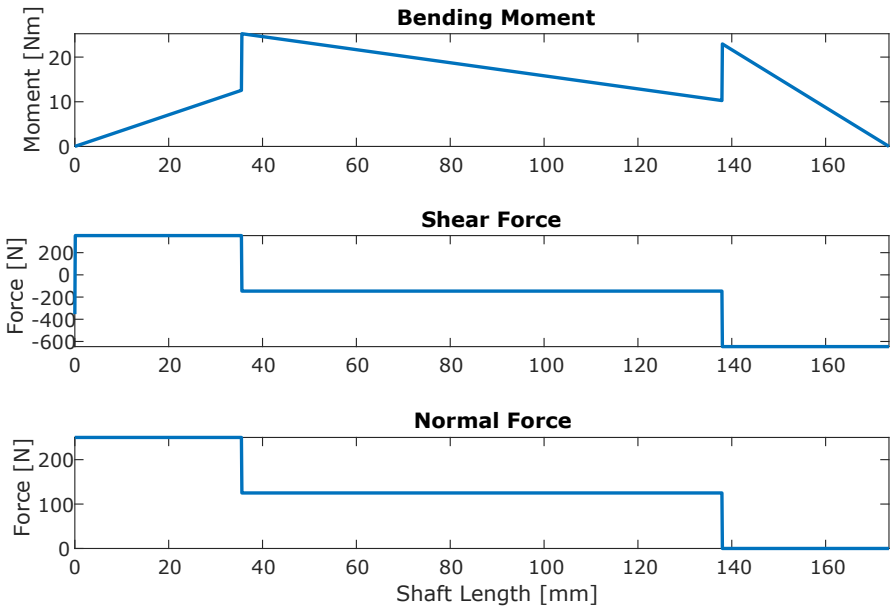



Figure 3.19: Bending, shear and normal diagrams

The resulting maximum bending moment for the shaft is calculated by Equation (3.30).

$$M_{bres} = \sqrt{{M_{bxy}}^2 + {M_{bzy}}^2} \tag{3.30}$$

$$M_{bres} = 25.6\ Nm \tag{3.31}$$

### Diameter

A calculation of the minimum shaft diameter can be determined from previous calculations of maximum bending moment and maximum torsion. The calculations also depend on the torque acting on the axle. The value used for the torque requirement is gathered from a detailed evaluation in Section (4.1). Equation (3.32) serves to find the minimum shaft diameter to ensure unlimited lifetime [29]. In the equation, dynamic load, alternating bending and torsion is assumed.

$$d = \sqrt[3]{32 \cdot \frac{\sqrt{M_b^2 + 0.75 \cdot (\alpha_0 \cdot T)^2}}{\pi \cdot \sigma_{lim}}} \tag{3.32}$$

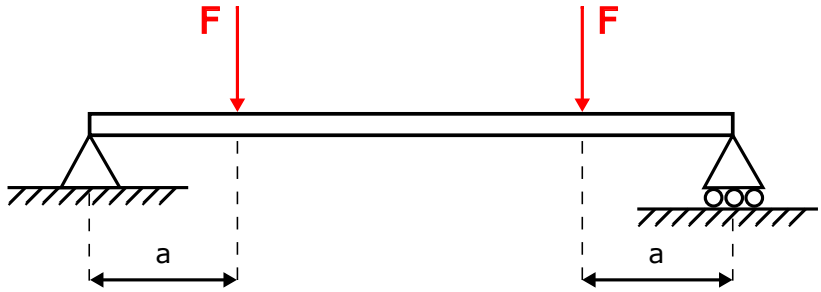
where:

| Symbol: | | Description: | Value: | Unit: |
|---|---|---|---|---|
| $d$ | - | Shaft diameter | 17.76 | $mm$ |
| $M_b$ | - | Maximum bending moment | 25618 | $Nmm$ |
| $\alpha_0$ | - | Fatigue factor related to shaft load | 1.0 | – |
| | - | Alternating bending and static torsion | 0.6 | – |
| | - | Alternating bending and pulsating torsion | 0.75 | – |
| | - | Alternating bending and torsion | 1.0 | – |
| $T$ | - | Maximum torque on shaft | 34000 | $Nmm$ |
| $\sigma_y$ | - | Yield strength material | 355 | $N/mm^2$ |
| $\sigma_{lim}$ | - | Allowable stresses in shaft | 71 | $N/mm^2$ |
| | - | $\sigma_y/4$ for static load | 88.75 | $N/mm^2$ |
| | - | $\sigma_y/5$ for dynamic load | 71 | $N/mm^2$ |

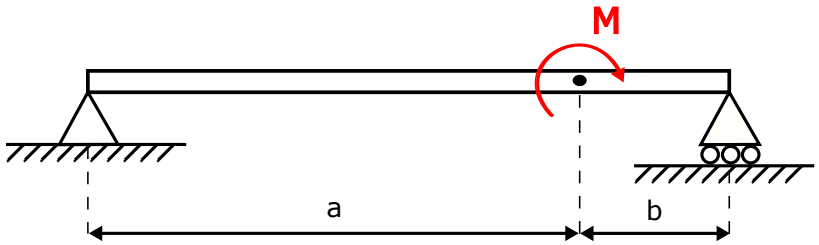
The results imply that a minimum diameter of 17.76 mm is required to withstand the bending and torsion. The closest available shaft diameter was 20 mm, which was chosen.

### Deflection

The principle of super-positioning was used to calculate the deflection of the shaft. The principle is valid for materials where Hooke's law applies and when the deflections and rotations are small. Curves which describes the deflection for any point on the shafts were created in Matlab with respect to Figure (3.20) and Equation (3.33) - (3.37). The script is available in Appendix (G.2). [30]




(a) Force




(b) Moment

Figure 3.20: Deflection figures

$$v(x) = -\frac{F \cdot x}{6 \cdot E \cdot I}\left(3 \cdot a \cdot L - 3 \cdot a^2 - x^2\right) \qquad 0 \leq x \leq a \qquad (3.33)$$

$$v(x) = -\frac{F \cdot a}{6 \cdot E \cdot I}\left(3 \cdot L \cdot x - 3 \cdot x^2 - a^2\right) \qquad a \leq x \leq L - a \qquad (3.34)$$

$$v(x) = \frac{M \cdot x}{6 \cdot L \cdot E \cdot I}\left(3 \cdot L \cdot a - 3 \cdot a^2 - 2 \cdot L^2 - x^2\right) \qquad 0 \leq x \leq a \qquad (3.35)$$

$$v(x) = -\frac{M \cdot (L - x)}{6 \cdot L \cdot E \cdot I}\left(3 \cdot L \cdot b - 3 \cdot b^2 - 2 \cdot L^2 - (L - x)^2\right) \qquad a \leq x \leq L \qquad (3.36)$$

$$I = \frac{\pi}{64} d^4 \qquad (3.37)$$

where:

| Symbol: | | Description: | Value: | Unit: |
|---|---|---|---|---|
| $v$ | - | Deflection curve | $\sim$ | $mm$ |
| $x$ | - | Any point along the length | $\sim$ | $mm$ |
| $F$ | - | Force | $\sim$ | $N$ |
| $L$ | - | Length of the shaft | 173.5 | $mm$ |
| $a$ | - | Distance from the left to the force | 35.5 | $mm$ |
| $b$ | - | Distance from the right to the force | 35.5 | $mm$ |
| $E$ | - | Modulus of elasticity | 210000 | $^N/_{mm^2}$ |
| $I$ | | Inertia of the shaft | $\sim$ | $mm^3$ |
| $d$ | - | Diameter of the shaft | 20 | $mm$ |

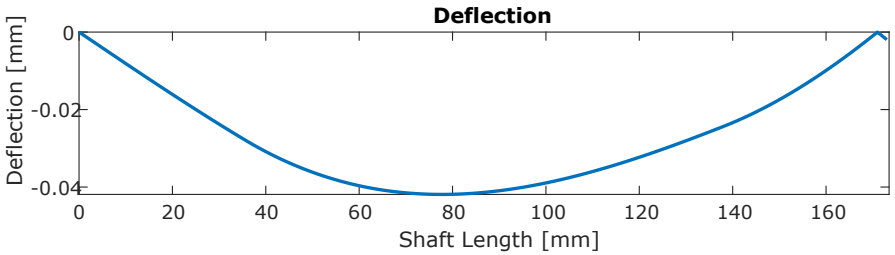



Figure 3.21: Deflection of shaft

The deflection curve is presented in Figure (3.21). The numerical value for maximum deflection is:

$$\delta_{max} = \ 0.0419\, mm$$

### Critical Speed

The rig is actuated by four motors controlling the angular velocity of each wheel. Critical speed is the velocity which matches the resonant frequency of the shaft. It is important to verify that the operating velocities are outside of the critical speed range. By applying the previously calculated deflection, the critical speed is calculated by evaluating the point on the axle where the maximum deflection occurs, with Equation (3.38). Then, the range the shaft velocity has to avoid is defined as; $0.8 < n_{cr} < 1.3$. The results are a critical speed of 4620 rpm for the shaft.

$$n_{cr} = \frac{30\sqrt{g}}{\pi\sqrt{\delta}} \tag{3.38}$$

where:

| Symbol: | | Description: | Value: | Unit: |
|---|---|---|---|---|
| $g$ | - | Gravity | 9.81 | $^m/_{s^2}$ |
| $\delta$ | - | Deflection of shaft | 0.0419 | $mm$ |
| $n_{cr}$ | - | Critical speed for shaft | 4620 | $rpm$ |

**Fatigue**

Approximately 80% - 95% of all fractures in machine components are due to fatigue [1]. The fractures typically appear in areas with high stress concentrations. The geometry of the designed shaft does not have notches for alignment and fitting of components. Thus, there are no points where stress concentrations commonly occur. However, the areas where the bending moments from the weight of the parking rig and Loomos is distributed is of concern, due to the high load. In this section the safety against shaft fatigue is analyzed. Smith diagram were used in order to calculate the safety factor. The majority of the equations and all tables used in this section are from the literature compendium [1]. The safety against yield is calculated with Equation (3.39), which is the factor of which the equivalent stress deviates from the allowed stress.

$$SF_{fatigue} = \frac{\sigma_{AN(red)}}{\sigma_{ea}} \tag{3.39}$$

To calculate the amplitude equivalent stress, Von Mises equation for equivalent stress is used. With alternating bending, in two planes, and a resting torsion, $\tau_a = 0$, the equivalent stress is calculated with Equation (3.40). The torsion is considered to be resting, since it is not going to be a continuous cyclic load like the bending moment. Even though the Loomo rig is going to accelerate and decelerate. This assumption was verified with Prof. Kjell Gunnar Robersmyr.

$$\sigma_{ea} = \sqrt{(K_{fb} \cdot \sigma_a)^2} \tag{3.40}$$

By using Figure (3.22), Equation (3.41) can be derived, which shows that the amplitude stress is equal to the maximum bending stress from the two planes. The maximum bending at the concerned locations are calculated with Equation (3.42)

$$\sigma_a = \frac{\sigma_{max} - \sigma_{min}}{2} = \frac{(\sigma_N + \sigma_b) - (\sigma_N - \sigma_b)}{2} = \sigma_b \tag{3.41}$$

$$\sigma_b = \frac{M_b \cdot y}{I} \quad \text{where: } y = \frac{d}{2}, \quad I = \frac{\pi d^4}{64} \tag{3.42}$$

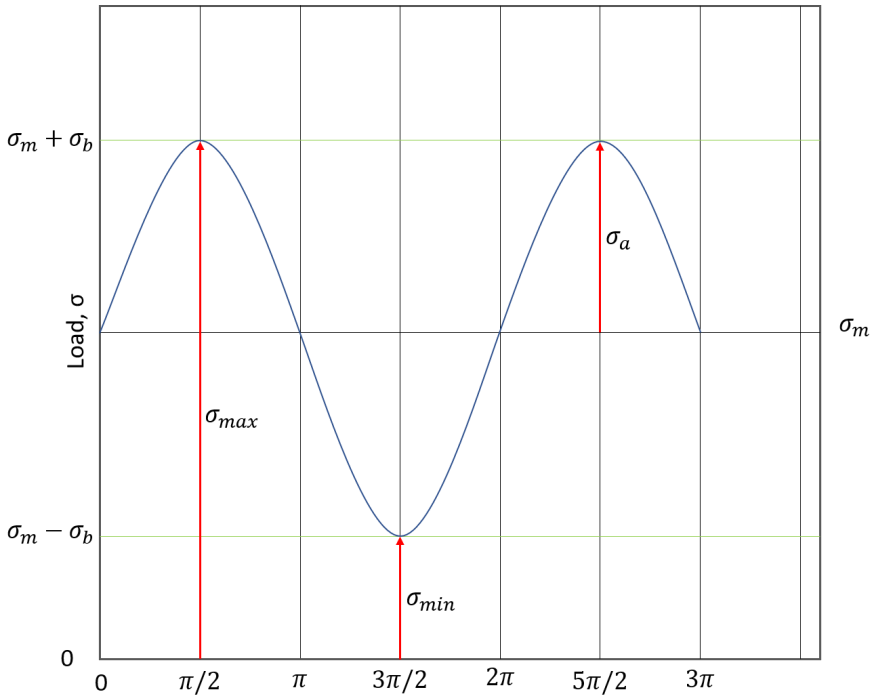



Figure 3.22: Amplitude of stress

Due to the geometry and holonomic properties of the wheels, forces are acting in multiple planes, which has been covered in section (3.3). Taking this into account the equivalent bending moment is going to be calculated with Equation (3.43), which is illustrated in Figure (3.23):

$$M_b = \sqrt{M_{b_{xy}}^2 + M_{b_{zy}}^2} \tag{3.43}$$

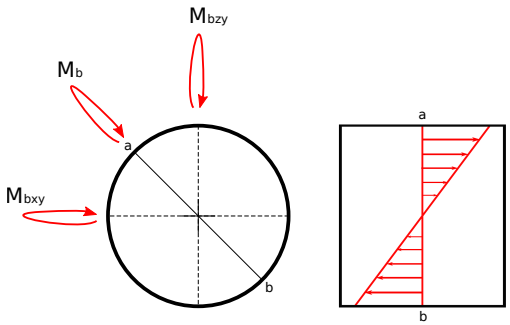



Figure 3.23: Equivalent bending moment

For shafts without change in cross section at the hub location, the notch factor due to bending is; $K_{fb} = 2.0 - 2.6$. Further on, since the hubs and inner ring of the bearings has a notch, specifically fillets or chamfers, the notch factor has to be reduced as shown in Equation (3.44). $K$ can be determined from the table shown in Figure (3.24).

$$K_{fb_{red}} = KK_{fb} \tag{3.44}$$

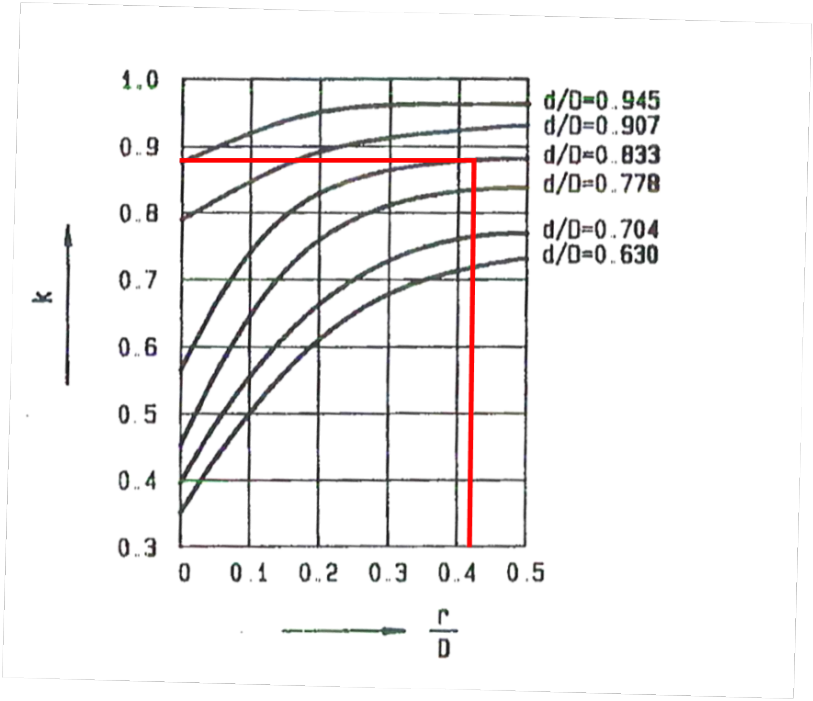



Figure 3.24: K factor [1]

where:

| Symbol: | | Description: | Value: | Unit: |
|---|---|---|---|---|
| $SF_{fatigue}$ | - | Fatigue safety factor | $\sim$ | $-$ |
| $\sigma_{AN(red)}$ | - | Adjusted stress amplitude | $\sim$ | $N/mm^2$ |
| $\sigma_{ea}$ | - | The equivalent stress | $\sim$ | $N/mm^2$ |
| $\sigma_a$ | - | The amplitude stress | $\sim$ | $N/mm^2$ |
| $\sigma_b$ | - | The maximum bending stress | $\sim$ | $N/mm^2$ |
| $M_b$ | - | The maximum bending moment at each notch location | $\sim$ | $Nm$ |
| $I$ | - | Moment of inertia | 7854 | $mm^4$ |
| $y$ | - | Distance from center of circle to most outer fibre | 10 | $mm$ |
| $d$ | - | Diameter to shaft | 20 | $mm$ |
| $K_{fb}$ | - | Notch factor due to bending | 2.6 | $-$ |
| $K_{fv}$ | - | Notch factor due to torsion | 1.7 | $-$ |
| $K$ | - | Relief factor | 0.88 | $-$ |
| $\eta$ | - | Notch sensitivity | $\sim$ | $-$ |

The shaft material is S355 steel, as stated in Section (3.3). Since the main concern of fatigue is from bending, the appropriate fatigue data is used, which is obtained from table 3.1 in the fatigue compendium and is listed in Equation (3.45) [1].

$$\sigma_{Db} = \begin{cases} 0 \pm 280 & (\sigma_m = 0, \quad \sigma_{Ab0} = \pm 280) \\ 225 \pm 225 & (\sigma_m = 225, \quad \sigma_{Ab} = \pm 225) \end{cases} \tag{3.45}$$

where:

| Symbol: | | Description: | Value: | Unit: |
|---|---|---|---|---|
| $\sigma_{Db}$ | - | The bending fatigue | $\sim$ | $N/mm^2$ |
| $\sigma_m$ | - | The mean stress | $\sim$ | $N/mm^2$ |
| $\sigma_{Ab0}$ | - | The fatigue amplitude when $\sigma_m = 0$ | $\sim$ | $N/mm^2$ |
| $\sigma_{Ab}$ | - | The fatigue amplitude when $\sigma_m = 240$ | $\sim$ | $N/mm^2$ |

The fatigue data is used to draw a Smith-diagram. However, the values given in Equation (3.45) only applies for specimen rods with a diameter of 10 mm which have a polished surface. The fatigue amplitudes have to be adjusted in order to apply for the designed shaft. This is done by taking the dimension-factor and surface-factor into account, as shown in Equation (3.46) and Equation (3.47).

$$\sigma_{Ab0(red)} = \sigma_{Ab0} \cdot b_1 \cdot b_2 \tag{3.46}$$

$$\sigma_{Ab(red)} = \sigma_{Ab} \cdot b_1 \cdot b_2 \tag{3.47}$$

where:

| Symbol: | | Description: | Value: | Unit: |
|---|---|---|---|---|
| $b_1$ | - | The dimension-factor | 0.95 | – |
| $b_2$ | - | The surface-factor | 0.92 | – |

The factors are found from Figure (3.25) and Figure (3.26). All shafts are considered to be fine lathed (corresponding to "Findreid" in the latter table). The ultimate tensile strength for S355 is set to 470 MPa, which is the lower value that fits the designed shafts. The lower limit is selected in order to calculate conservatively. The mechanical properties to S355 can be found in Appendix (C.2).

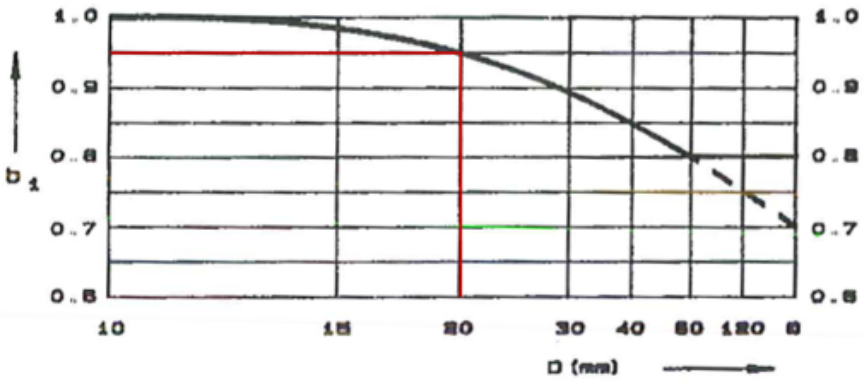



Figure 3.25: Dimension factor table - $b_1$ [1]

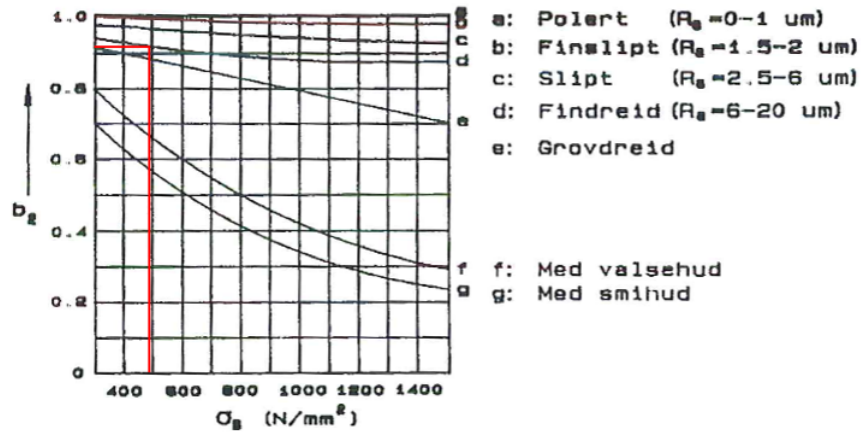



Figure 3.26: Surface factor table - $b_2$ [1]

The Smith-diagram is made with the adjusted values. $\sigma_{AN(red)}$ is obtained at $\sigma_{em}$, as shown in the Smith diagram in Figure (3.27), where the red line represents the adjusted fatigue data.

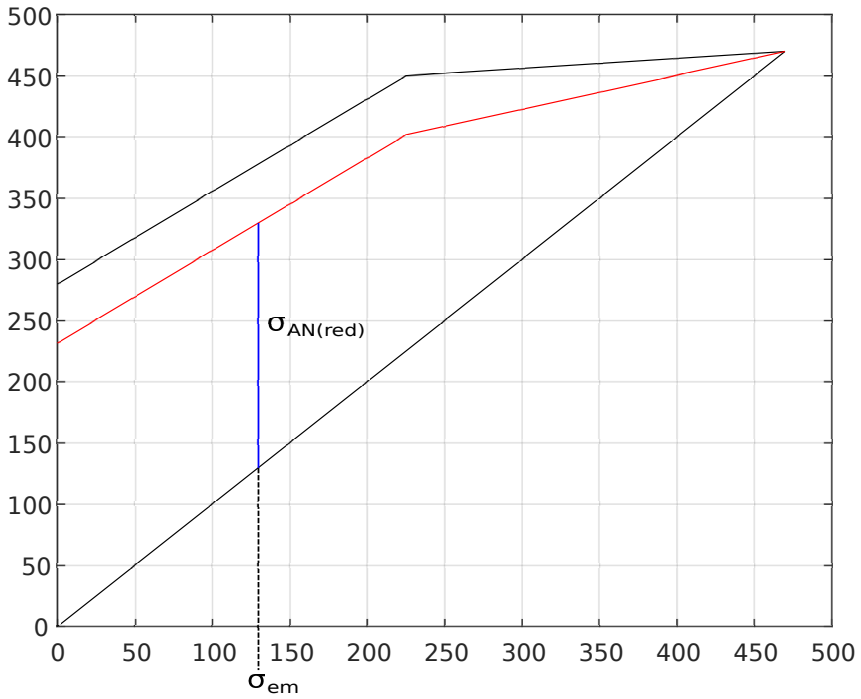



Figure 3.27: Smith Diagram example

$\sigma_{em}$ is calculated using Equation (3.48).

$$\sigma_{em} = \sigma_m \tag{3.48}$$

The value used for $\sigma_m$ is derived in Equation (3.49), by applying Figure (3.22). Further, the axial stress is calculated with Equation (3.50).

$$\sigma_m = \frac{\sigma_{max} + \sigma_{min}}{2} = \frac{(\sigma_N + \sigma_b) + (\sigma_N - \sigma_b)}{2} = \sigma_N \tag{3.49}$$

$$\sigma_N = \frac{N}{A} \tag{3.50}$$

where:

| Symbol: | | Description: | Value: | Unit: |
|---|---|---|---|---|
| $\sigma_{em}$ | - | The equivalent mean stress | $\sim$ | $^N/_{mm^2}$ |
| $\sigma_N$ | - | The axial stress | $\sim$ | $^N/_{mm^2}$ |
| $N$ | - | The axial force | $\sim$ | $N$ |
| $A$ | - | The Cross-sectional area | $\sim$ | $mm^2$ |

Typical critical fatigue locations are; on the outer surface where the bending moment is at its maximum, where the torque is present, and where stress concentrations exist [31]. However, because the shaft does not have variations in diameter where the load is supported, the location of which the maximum combination of axial and two plane bending moments is

used. If the equivalent stress level is within the limits at this location, then the same must be valid for all locations. The axial force and bending moments are gathered from Section (3.3) to calculate $\sigma_{em}$ and $\sigma_{ea}$, which uses the case of forward movement of the rig.

| | Axial force $[N]$ | $M_{b_{yz}}$ $[Nm]$ | $M_{b_{xy}}$ $[Nm]$ |
|---|---|---|---|
| Maximum location | 250 | 25.23 | 4.44 |

Table 3.1: The axial forces and bending moments

Table (3.2) shows the stresses and safety against fatigue for the maximum load location. As can be seen, the axles are suitable, since the safety against fatigue is larger than one at the maximum load location.

| $\sigma_{\mathbf{em}}$ $[N/mm^2]$ | $\sigma_{\mathbf{ea}}$ $[N/mm^2]$ | $\sigma_{AN(red)}[N/mm^2]$ | $SF_{Fatigue}$ |
|---|---|---|---|
| 0.8 | 74.6 | 232.2 | 3.1 |

Table 3.2: Stresses at maximum load location

## 3.4 Bearings

In order to use the mecanum wheels with the developed Loomo parking rig, the shaft is supported by a bearing on each side to reduce friction and enable motion. The ball bearings are part of a housing which is mainly designed to support radial load, but can also handle some axial load. From the previously discussed shaft analysis, the radial and axial forces can be utilized to calculate the expected lifetime of the bearings. The bearings installed are equivalent to SKF UCP204, which are designed for a 20 mm shaft diameter.

#### Verification

To ensure that each set of bearings can hold the weight of the parking rig when fully loaded with Loomos, the potential force acting on each bearing is extracted from the shaft analysis in Section (3.3). As previously stated in the mechanical calculation, the force distribution in mecanum wheels is equal for axial and radial direction. Since the wheels are placed equally distanced from each bearing, the force is distribute thereafter. In Equation (3.52) the expected numbers of million revolutions for the given bearing can be calculated with a 90% reliability [32].

The dynamic load rating, $C$, is provided by the manufacturer [33]. Further on, the equivalent load $P$ can be calculated with Equation (3.51). The equation uses the relationship between the axial force and the manufacturer's static load component $C_0$, to determine the coefficients presented in Table (3.3) for the Equation (3.51). [18]

$$P = X \cdot F_r + Y \cdot F_a \tag{3.51}$$

where:

| Symbol: | | Description: | Value: | Unit: |
|---|---|---|---|---|
| $F_a/C_0$ | - | Relation between bearing limit and axial force | 0.0746 | – |
| $F_r$ | - | Radial load on bearing | 500 | $N$ |
| $F_a$ | - | Axial load on bearing | 500 | $N$ |
| $C_0$ | - | Static component from manufacturer | 6700 | $N$ |
| $X$ | - | Radial load factor from table (3.3) | 0.56 | – |
| $Y$ | - | Axial load factor from table (3.3) | 1.6 | – |

| $F_a/C_0$ | e | X | Y |
|---|---|---|---|
| 0.025 | 0.22 | 0.56 | 2.0 |
| 0.04 | 0.24 | 0.56 | 1.8 |
| 0.07 | 0.27 | 0.56 | 1.6 |
| 0.13 | 0.31 | 0.56 | 1.4 |
| 0.25 | 0.37 | 0.56 | 1.2 |
| 0.50 | 0.44 | 0.56 | 1.0 |

Table 3.3: Factor to determine force in bearings [18]

$$L_{10} = \left(\frac{C}{P}\right)^p \tag{3.52}$$

From the calculations in Equation (3.52), each bearing can be expected to last 3697 mill.revs with 90% reliability, before being replace.
where:

| Symbol: | | Description: | Value: | Unit: |
|---|---|---|---|---|
| $L_{10}$ | - | Lifetime bearing | 3697 | mill. $rev$ |
| $C$ | - | Dynamic number which gives bearing lifetime of 1 mill revs with 90% reliability [33] | 12700 | $N$ |
| $P$ | - | Equivalent load each Bearing | 1080 | $N$ |
| $p$ | - | Exponent: | $10/3$ | – |
| | - | Ball bearings | $10/3$ | – |
| | - | Roller bearings | 3 | – |

**Lifespan**

From the result obtained with Equation (3.52), the expected number of revolutions can be converted to operational hours with Equation (3.53). When the number of hours the bearings are likely to last is known, the estimated time of operation for the rig can be calculated when assuming operating conditions and environment. The results are a minimum of 13693 H operating time, given the motor conditions.

$$L_{10h} = \frac{L_{10}10^6}{n60} \tag{3.53}$$

where:

| Symbol: | | Description: | Value: | Unit: |
|---|---|---|---|---|
| $L_{10h}$ | - | Lifetime of bearing in hours | 13693 | $H$ |
| $L_{10}$ | - | Lifetime of bearing in mill revs | 3697 | mill. $rev$ |
| $n_{max}$ | - | Maximum motor RPM during operation | 4500 | $RPM$ |

### Lubrication

In order to reduce wear on the bearings, lubrication is added to the housing. The housing features a grease fitting which serves to feed lubrication to the bearing. It is difficult to maintain an oil lubricant for self-sealed bearings, therefore it is preferable to use grease. The interval for refilling grease is based on the diameter of shaft in mm and operational rotational speed in RPM. As presented in Figure (3.28), the sealed ball bearing requires a refill of grease every 6500 operational hours to keep bearings at maximum performance. [2]

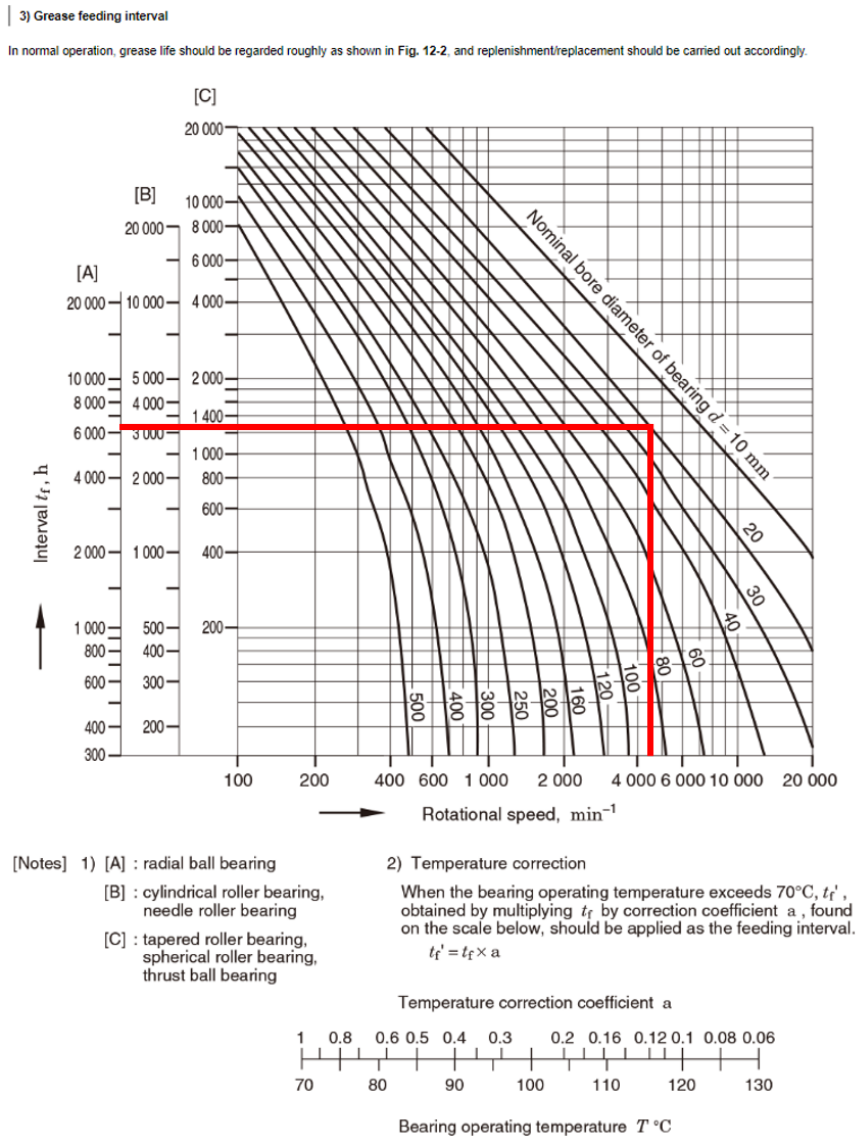



Figure 3.28: Grease refilling interval [2]

When knowing how often the grease requires refill, the expected lifetime of the grease can be calculated. The lifetime depends on the operational conditions and dimension of the shaft. The results can be obtained from Equation (3.54). [2]

$$\log L = 6.10 - 4.40 \cdot 10^{-6} \cdot d_m \cdot n - 3.125 \cdot \left(\frac{P_r}{C_r} - 0.04\right) - (0.021 - 1.80 \cdot 10^{-8} \cdot d_m \cdot n) \cdot T \quad (3.54)$$

where:

| Symbol: | | Description: | Value: | Unit: |
|---|---|---|---|---|
| $L$ | - | Grease life | 31496 | $H$ |
| $d_m$ | - | $^{(D+d)}/_{2}$ | 37.5 | $mm$ |
| $D$ | - | Outside diameter | 55 | $mm$ |
| $d$ | - | Bore diameter | 20 | $mm$ |
| $n$ | - | Rotational speed | 4500 | $min^{-1}$ |
| $P_r$ | - | Dynamic equivalent load | 1080 | $N$ |
| $C_r$ | - | Dynamic load rating manufacturer | 12700 | $N$ |
| $T$ | - | Operating temperature of bearing | 40 | $°C$ |

From Equation (3.54) the expected lifetime of grease is about 31500 hours. This is larger than the expected refilling interval of the sealed bearing and for that reason, refilling of grease should be performed according to the refilling intervals previously calculated. Additionally, the system may have possible leakage during operations, which will influence the lifetime and it is therefore reasonable to use the lubrication interval. A type of grease that is suitable for the Loomo parking rig operation is SR Grease, based on rotational speed, operating temperature and type of operation.[34]

## 3.5 Set Screws

Using set screws as a substitute for key connections is a common approach because it is more straightforward with respect to manufacturing and machining. It will however, give a larger possibility for slip and a lower torque resistance. A set screw can work with three different conditions, torsional resistance, axial resistance, and vibrations. For mecanum wheels, each of the conditions previously mentions occurs. The hub connection from each mecanum wheel to the motorized shaft is fastened by the use of a set screw. The shaft is manufactured with a flattened surface for the set screw to get a greater rigid connection with the shaft. The set screws used are plain cups with a hexagonal socket [35]. This section evaluates the axial force and torque the set screw requires to generate a proper bond to the axle by using Equation (3.55) [36].

$$T_{HP} = k \cdot r \cdot A_{HP} \tag{3.55}$$

where:

| Symbol: | | Description: | Value: | Unit: |
|---|---|---|---|---|
| $r$ | - | Radius of shaft | $\sim$ | $mm$ |
| $A_{HP}$ | - | Axial Holding Power [37] | $\sim$ | $N$ |
| $k$ | - | Factor for number of set screws: | $\sim$ | $-$ |
| | | - For one set screw: | 1 | $-$ |
| | | - For two set screws: | 1.3 - 2.0 | $-$ |
| $T_{HP}$ | - | Torsional Holding Power | $\sim$ | $Nm$ |

**Hub**

Both hubs on the wheel configuration has a set screw installed. A M6 plain cup with an internal hexagonal socket was used. Recommended tightening torques to achieve proper connection can be found in the designated tables [36]. For a M6 set screw, 7.8 Nm of tightening torque is recommended. With the recommended tightening torque, an estimated

axial holding force of the connection is provided [37]. Presented in Figure (3.29) are the axial holding power and Figure (3.30) displays how the torsional holding power between hub and shaft is generated.

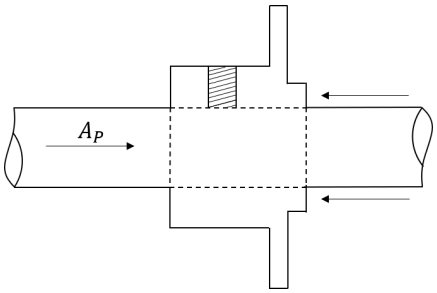



Figure 3.29: Axial Holding Power Hub

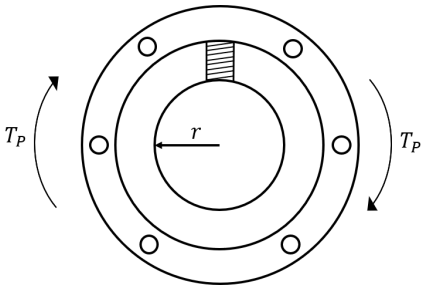



Figure 3.30: Torsional Holding Power Hub

### Bearings

Similar to the hubs, the bearings are delivered with M6 knurled cup set screws, as shown in Figure (3.31). Knurled cup setscrews are manufactured with excessive counterclockwise knurls to prevent loosening, and they are designed to withstand larger vibrations. The bearing connection consists of two screws, with an 120 degree angle gap, $\gamma_b$. When using two screws, the holding power is not necessarily double. With the use of two set screws the nominal axial holding power can be multiplied with a factor, $k$, between 1.3 and 2.0 deepening on angle between the installed set screw [36]. From tables, a 120 degree angle has $k = 1.5$ [37].

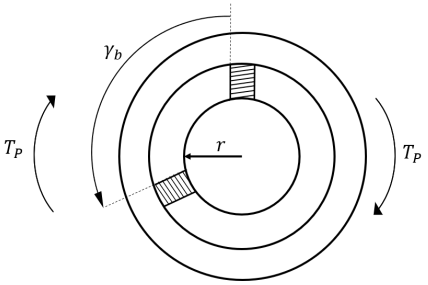



Figure 3.31: THP Bearing

### Gear

Similar to the hub, the gear is fastened to the axle by use of a set screws, as shown i Figure (3.32). In order to transfer torque with a belt connection, two M5 knurled cup set screw is used to keep the gear in position on the axle, with a thightning torque of 4.6 Nm. Because the gear is mounted on the axle where the radius is reduced from 10 mm to 8 mm, the maximum torque transfer limit less compred to hub and bearings. With two set screw placed 180 degrees apart, $k$ factor is 1.3.

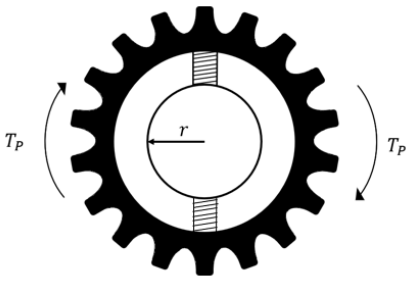



Figure 3.32: THP Gear

Table 3.4 present the resulting holding power for each component locked by the use of set screws.

| | $k$ | $A_{HP}$ [N] | $T_{HP}$ [Nm] |
|---|---|---|---|
| Hub | 1 | 4200 | 42 |
| Bearings | 1.5 | 6300 | 63 |
| Gear | 1.3 | 2500 | 26 |

Table 3.4: Load capacity set screws

## 3.6 Weld

After finalizing the concept design, it is important to verify that the stress levels in the welds do not exceed the yield strength. If the stresses exceed yield, it would be necessary to modify the design with a supporting bracket to relieve stress. A crucial area to consider is the bracket which the Loomos are stored upon. These welds are only supported on one side and the load is applied at the end, creating a significant bending moment.

### Strength Verification

Figure (3.33) presents a design with a weld positioned at the upper and lower side of the beam which hold the Loomo. The weld is a fillet weld with a single seam around the entire square profile with an effective weld length, $L_{eff}$, as shown in Figure (3.34). From the weld, the normal and shear stress as a result of bending is calculated, and stress due to torsion is neglected since the force is centered on the beam.

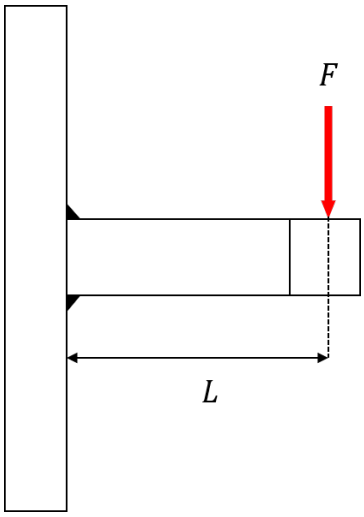



Figure 3.33: Illustration of weld placement

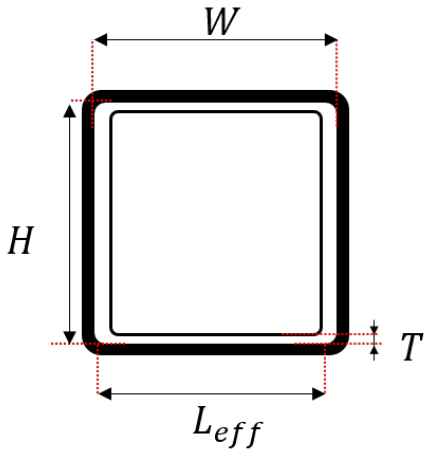



Figure 3.34: CSA of beam with weld

$$F = (m_{loomo} + m_{bracket})g \tag{3.56}$$

$$M = F \cdot L \tag{3.57}$$

$$Z_w = 3(H \cdot (W + 2a) + \frac{H^2}{3}) \tag{3.58}$$

$$\sigma = \frac{6M}{Z_w} \tag{3.59}$$

$$CSA = 2 \cdot a \cdot l_{eff} \tag{3.60}$$

$$\tau = \frac{F}{CSA} \tag{3.61}$$

$$\sigma_{eq} = \sqrt{\sigma^2 + 3 \cdot \tau^2} \tag{3.62}$$

Criterion to be met:

$$\sigma_{eq} \leq \frac{\sigma_{yield}}{\gamma_m} \tag{3.63}$$

The resulting stresses obtained from Equation (3.56) - (3.63) are presented in the table below, denoted $\sigma$ and $\tau$. The stresses that may occur in the welds are far lower than the allowable stresses, $\sigma_{allowable}$. This indicates that it is safe to store the Loomos on the bracket. where:

| Symbol: | | Description: | Value: | Unit: |
|---|---|---|---|---|
| $L$ | - | Distance to load, from weld | 190 | $mm$ |
| $W$ | - | Width of beam | 40 | $mm$ |
| $H$ | - | Height of beam | 40 | $mm$ |
| $m_{Loomo}$ | - | Weight of Loomo | 19.5 | $kg$ |
| $m_{Bracket}$ | - | Weight of Loomo bracket | 2.5 | $kg$ |
| $F$ | - | Load magnitude | 216 | $N$ |
| $M$ | - | Bending moment | 41040 | $Nmm$ |
| $CSA$ | - | Cross sectional area weld | 204 | $mm^2$ |
| $Z_w$ | - | Modulus of weld section [38] | 7120 | $mm^3$ |
| $l_{eff}$ | - | Effective weld length | 34 | $mm$ |
| $\sigma$ | - | Normal stress due to bending | 34.58 | $N/mm^2$ |
| $\tau$ | - | Shear stress due to bending | 1.05 | $N/mm^2$ |
| $\sigma_{eq}$ | - | Bending stress | 34.63 | $N/mm^2$ |
| $\sigma_{allowable}$ | - | Allowable bending stress | 284 | $N/mm^2$ |
| $a$ | - | Throat length weld | 3 | $mm$ |
| $\sigma_{yield}$ | - | Yield strength S355J2 | 355 | $N/mm^2$ |
| $\gamma_m$ | - | Material factor S355J2 steel | 1.25 | $-$ |

Further on, a scenario where only one side is loaded with Loomos was examined. The evaluation concerned the welds that connect the three vertical beams to the wheel suspension as shown in Figure (3.35). The load is assumed to be equally distributed between the welds. The stresses are calculated with Equation (3.64)-(3.68).

$$F_N = F \cdot \frac{5}{3} \tag{3.64}$$

$$M = F_N \cdot L \tag{3.65}$$

$$\sigma = \frac{M \cdot L}{I_{wx}} \tag{3.66}$$

$$I = \frac{a \cdot H^3}{6} + \frac{W \cdot ((H+2a)^3 - H^3}{12} \tag{3.67}$$

$$\tau = \frac{F_N}{CSA} \tag{3.68}$$

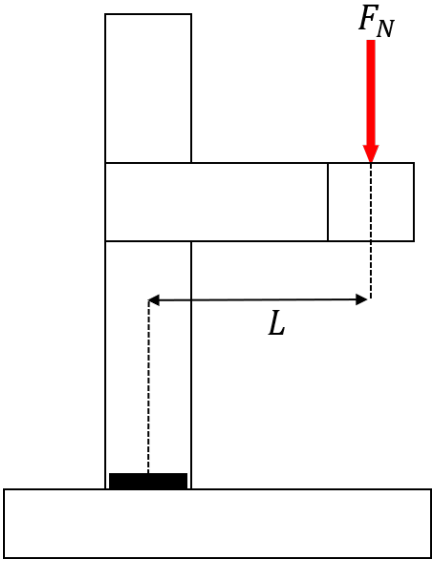



Figure 3.35: Illustration of weld position

With the same criterion as the bracket weld analysis, the yielded results are more extensive than the individual bracket, but still within the limits of allowable stress in the fillet welds when only one side is loaded. Once verified, the design could be weld in compliance with technical drawings presented in Appendix (D). Additionally, supports bracket were introduce at the bottom of the vertical beams in order to relief the load stresses and to handle additional unforeseen loads.
where:

| Symbol: | | Description: | Value: | Unit: |
|---|---|---|---|---|
| $L$ | - | Distance to load from weld | 210 | $mm$ |
| $F_N$ | - | Load magnitude | 360 | $N$ |
| $M$ | - | Bending moment | 75600 | $Nmm$ |
| $I$ | - | Moment of Inertia about tipping angle [38] | 143120 | $mm^4$ |
| $\sigma$ | - | Normal stress due to bending | 110.93 | $^N/_{mm^2}$ |
| $\tau$ | - | Shear stress due to bending | 1.76 | $^N/_{mm^2}$ |
| $\sigma_{eq}$ | - | Equivalent bending stress form applied load | 110.97 | $^N/_{mm^2}$ |
| $\sigma_{allowable}$ | - | Allowable stresses | 284 | $^N/_{mm^2}$ |

## 3.7 Stability of Loomo Rig

The parking rig is designed to store up to ten Loomos, with five units placed one each side. The stability of the rig is dependant on its baseline and the center of mass. The baseline is the distance between the wheels in lateral direction. The distance is measured from one contact point to another, where the contact point is defined as the point where the wheel and the ground intersect. The center of mass is varying dependant on the storing configuration. To ensure that the rig is stable for all possible load configurations, evenly and unevenly, different scenarios were evaluated in this section.

The center of mass is calculated with Equations (3.69), for both horizontal and vertical position, as shown in Figure (3.36). Loomos, the rig, and other components are included when calculating the center of mass. If the center of mass is shifted from the rigs center, Equation (3.70) is used to determine the distance from the center of mass to the closest wheel. When this distance is known, and the distance from ground to the total center of mass is known, the critical angle where the rig will fall over can be calculated with Equation (3.71). The critical scenarios which are evaluated are; fully stored, single side fully stored, single side with only two at top, two on each side at top, fully stored with person stepping on rig and single side fully stored with person stepping onto rig. When a person is assumed to stand on the rig, the center of mass to the person is assumed to be vertically over the edge of the rig.

$$x_j = \sum_{i=1}^{N} \frac{x_i \cdot m_i}{M} \ , \quad y_j = \sum_{i=1}^{N} \frac{y_i \cdot m_i}{M} \ , \quad z_j = \sum_{i=1}^{N} \frac{z_i \cdot m_i}{M} \tag{3.69}$$

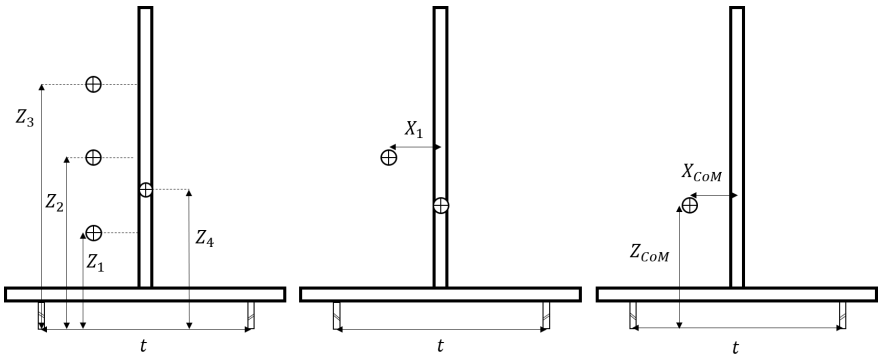



Figure 3.36: Calculation of Center of Mass

$$t = t_{tot} - 2 \cdot X_{CoM} \tag{3.70}$$

$$\theta_c = \tan^{-1}\left(\frac{t}{2h}\right) \tag{3.71}$$

where:

| Symbol: | | Description: | Value: | Unit: |
|---|---|---|---|---|
| $x_j$ | - | Distance to CoM in horizontal direction to i object | $\sim$ | $mm$ |
| $y_j$ | - | Distance to CoM in inwards direction to i object | $\sim$ | $mm$ |
| $z_j$ | - | Distance to CoM in vertical direction to i object | $\sim$ | $mm$ |
| $m_i$ | - | Mass of i object | $\sim$ | $kg$ |
| $M$ | - | Total weight of all objects in evaluation | $\sim$ | $kg$ |
| $h$ | - | Height to CoM from ground equal to $Z_{CoM}$ | $\sim$ | $mm$ |
| $t_{tot}$ | - | Baseline of rig between wheels | 560 | $mm$ |
| $t$ | - | Reduced wheel distance | $\sim$ | $mm$ |
| $\theta_c$ | - | Critical angle which initiate tipping | $\sim$ | $^\circ$ |

In Table (3.5) the results of the stability evaluations is presented. As the Loomo parking rig is not intended to be used at inclined environments it can be exposed to about 10 degrees before toppling, independent of storing configuration.

| Storing configuration: | $Z_{CoM}$ [mm] | t [mm] | $\theta_c$ [°] |
|---|---|---|---|
| Two upper on one side | 554 | 463 | 22,7 |
| Full on one side | 583 | 380 | 18,1 |
| Full on one side and person | 701 | 234 | 9,5 |
| Four upper | 652 | 560 | 23,2 |
| Full rig | 657 | 560 | 23,1 |
| Full rig with person | 732 | 414 | 15,8 |

Table 3.5: Results of stability evaluation

# 4. Electrical Design

The electrical design chapter concentrate on the required components to motorize the rig. The additional electrical components used are also presented. The selection and functionality of the speed controller and the selection of a brushless DC motor are covered together with the drive train sizing. Verification of the power source and overall power consumption of the electronic hardware component is performed.

## 4.1 Motor and Gear Sizing

Because the intended speed of the rig is equivalent to the speed of a person walking, the mecanum wheels do not need to rotate fast. What is more important is that the motor can produce the required shaft torques to accelerate up to the velocity demands. The university has two available of the shelf motor options. A high speed - low torque option and a low speed - high torque option, both brushless dc motors. The purpose of the Loomo parking rig is to safely and efficiently transport ten Loomo units with various maneuverability modes. Whether the Loomo rig is manually controlled, following a person or autonomously driven, the vehicle will move with a low velocity demand to ensure safety with respect to the rigs surroundings. Considering these factors, the available low speed - high torque option was chosen. Additionally, the selection will need a smaller gearing ratio to achieve the torque requirements. This also present the possibility of utilizing belt drive in oppose to a gearbox. By exploiting the advantages and accessibility of belt drive, the Loomo parking rigs actuation system will be compact, simple and easy to install.

### 4.1.1 Torque Requirement From Tests

From previous calculations, the torque demand in order to initialize forward motion yielded a result of 4.0 Nm. However, there are a lot of uncertainties in the calculations as the mecanum frame is assumed to be ideal. In addition, theses calculations do not account for the roller friction, bearing friction, and other internal resistances. Moreover, the rolling resistance used for the calculations most likely differs from the contact surface at the university.

To eliminate the uncertainties in the calculation, physical tests were performed with the Loomo rig. This was done by loading it with a load of 200 kg, equivalent to ten Loomos, then measuring the force required to initialize movement. The measurements were obtained by using a handheld scale. Multiple test-sets were created. The sets can be seen in Table (4.1). The table contains the measured data for overcoming the static friction in forward and sideways direction. In addition, the force required to yield a satisfactory acceleration in forward direction is listed.

| Forward [kg] | Sideways [kg] | Accelerating Forward [kg] |
|---|---|---|
| 3.6 | 32 | 26 |
| 3.1 | 29 | 27 |
| 3.5 | 31 | 22 |
| 4.4 | 33 | 24 |
| 3.7 | 36 | 28 |
| 4.1 | 31 | 20 |
| 4.8 | 37 | 27 |
| 4.8 | 34 | 27 |
| 3.4 | 40 | 29 |
| 3.7 | 39 | 18 |
| 3.6 | 38 | 25 |
| Average [N] | | |
| 38 | 339 | 243 |

Table 4.1: Loomo rig tests

As can be seen from Table (4.1) the forces vary from the theoretical calculations, which is to be expected. Most likely, the theoretically variation, at least in forward movement, is due to the fact that the rolling coefficient for the Loomo rig is different. However, based on the calculations, the force in order to move sideways should not deviate this much compared to moving forward. A reason could be that the tests were made with rubber plates added in between the bearings and the bottom frame in an attempt to increase the traction of the rig. These also allow for rotation about the axis perpendicular to the shaft, and there is the possibility that this creates a downward force, thus increasing the rolling resistance of the mecanum wheel.

It is also important that the Loomo rig is able to accelerate close to that of a person to achieve a smooth pursuit. The average expected walking speed of community walker, is about $1.1\,{}^{m}/_{s}$ [39]. Then, by setting the time that the Loomo rig has to achieve this velocity to 0.75 s, specifically an acceleration of $1.5\,{}^{m}/_{s^2}$, the torque demand for this case is calculated. The torque is calculated with Equation (4.1), where the average total resistive forward forces from the tests are applied.

$$\sum F_x = m_{tot}a = F_t - F_{r,tot} \tag{4.1}$$
$$\Downarrow$$
$$F_{t,tot} = m_{tot}a + F_{r,tot}$$

The calculations indicates that the total output force to accelerate forward is about 507 N. As this is the largest of the calculated forces, it is used as the force demand in the calculation of the required gear ratio. Then, the torque demand each drive shaft has to deliver is calculated to be about 12.9 Nm with Equation (4.2).

$$T_{demand} = \frac{F_{demand}}{4}r \tag{4.2}$$

where:

| Symbol: | | Description: | Value: | Unit: |
|---|---|---|---|---|
| $F_{demand}$ | - | Demanded force from all four wheels | 507 | $N$ |
| $T_{demand}$ | - | Demanded torque from one drive shaft | 12.9 | $Nm$ |
| $r$ | - | Mecanum wheel radius | 0.1015 | $m$ |
| $m_{tot}$ | - | Loom rig mass | 400 | $kg$ |

### 4.1.2 Motor Torque

Similar to all other electrical motors, a brushless dc motors convert electrical power to mechanical power. Unlike brushed motors, brushless motors does not rely on a mechanical connection between the rotating parts, which means they of do not wear out over time as easily. This type of motor is therefore suitable for this application, which preferres low noise, reliability and long life-time.

#### Relation between $k_V$ and $k_T$

For brushless motors, $k_V$ is a parameter that is commonly stated in the product specifications, and is the case for the selected motor. $k_V$ is known as the motor velocity constant, and is the velocity the motor has to rotate, in rpm, to yield a 1 V back electromotive force (back EMF). This value can be related to the motor torque constant, $k_T$, which can be used in order to calculate the theoretical motor torque.

Electrical effect can be described with Equation (4.3), and electromechanical effect with Equation (4.4).

$$P_e = UI \tag{4.3}$$

$$P_m = T_m \omega_m \tag{4.4}$$

Taking the overall efficiency into account, from the electrical input to the mechanical output of the brushless dc motor, Equation (4.5) applies.

$$UI = \frac{1}{\eta_{tot}} T_{em} \omega_m \tag{4.5}$$

Since $k_V$ is the velocity the motor has to rotate, in rpm, to yield a 1 V back EMF, Equation (4.6) can be derived to described the motor velocity.

$$\omega_m = k_V U \tag{4.6}$$

Substituting Equation (4.6) into Equation (4.5) yields Equation (4.7). Further, the efficiency is set to one.

$$UI = T_{em} k_V U \tag{4.7}$$

Then, solving for $T_{em}$, Equation (4.8) describes the relation between the speed constant and the produced motor torque.

$$T_{em} = \frac{I}{k_V} \tag{4.8}$$

In addition, the electromagnetic torque can also be described with Equation (4.9).

$$T_{em} = k_T I \tag{4.9}$$

Finally, by substituting Equation (4.8) into Equation (4.9), the relation between $k_V$ and $k_t$ is shown. It should be noted in theses calculations it is assumed that the motor is not within field weakening. Also the unit of the constants are not equal, however this relation is commonly accepted.

$$k_T = \frac{1}{k_V} \tag{4.10}$$

where:

| Symbol: | | Description: | Value: | Unit: |
|---|---|---|---|---|
| $U$ | - | Voltage | $\sim$ | $V$ |
| $I$ | - | Current | $\sim$ | $I$ |
| $T$ | - | Torque | $\sim$ | $Nm$ |
| $P_{em}$ | - | Mechanical effect | $\sim$ | $W$ |
| $P_e$ | - | Electrical effect | $\sim$ | $W$ |
| $\omega_m$ | - | motor velocity | $\sim$ | $^{rad}/_{s}$ |
| $k_V$ | - | Motor velocity constant | $\sim$ | $^{rad}/_{(sV)}$ |
| $k_T$ | - | Motor torque constant | $\sim$ | $^{Nm}/_{A}$ |

**Motor specifications**

The selected motor is a MTO5065-70-HA from MayTech. Its specification are listed in Table (4.2)[6].

| | | | |
|---|---|---|---|
| Idle Current: | 1.2 A | $k_V$: | 70 rpm |
| Max Current: | 50 A | Max Output Watt: | 1815 W |
| Rated Current: | 42.5A | Input volt: | 7.4-37 V |
| Max Pull: | 6700 g | Motor Weight: | 435 g |
| Shaft: | 8 mm | Output Shaft Length: | 26 mm |
| Internal Resistance: | 0.0361 Ω | Motor diameter: | 49.5 mm |

Table 4.2: MTO5065-70-HA specifications

The motor rated and maximum torque is calculated with Equation (4.11) to be 5.8 Nm and 6.8 Nm respectively. Since the value of $k_V$ is presented in the unit $rpm$ in Table (4.2), it was converted to $^{rad}/_{s}$ with Equation (4.12), in order to it match the the units in Equation (4.11).

$$T_{rated} = \frac{1}{k_V} I_{rated} \tag{4.11}$$

$$k_V \left[{}^{rad}/_{s}\right] = k_V \frac{2 \cdot pi}{60} \left[rpm\right] \tag{4.12}$$

where:

| Symbol: | | Description: | Value: | Unit: |
|---|---|---|---|---|
| $T_{rated}$ | - | Rated motor torque | 5.8 | $Nm$ |
| $T_{max}$ | - | Maximum motor torque | 6.8 | $Nm$ |
| $I_{rated}$ | - | Rated current | 42.5 | $A$ |
| $I_{max}$ | - | Maximum current | 50 | $A$ |
| $k_V$ | - | Motor velocity constant | 7.3 | $^{rad}/_{(Vs)}$ |

### 4.1.3 Drivetrain

With knowing both the torque demand for each wheel and the rated torque the selected motor, the required gearing ratio of the drivetrain can be calculated with Equation (4.13).

$$i = \frac{T_{demand}}{T_{rated}} \tag{4.13}$$

where:

| Symbol: | | Description: | Value: | Unit: |
|---|---|---|---|---|
| $T_{Demand}$ | - | Torque demand | 12.9 | Nm |
| $T_{Rated}$ | - | Rated torque from motor | 5.8 | Nm |
| i | - | Minimum gear ratio | 2.2 | - |

Before a gear ratio is actually set, the gear ratio which causes the motor inertia to match the load inertia is considered. In theory, the ratio between the load and the motor inertia are ideal if they are equal. The ratio determines how well the motor is going to be able to control the load during acceleration and deceleration.

The selected motor does not have the motor inertia listed in the supplier's datasheet. Thus, the motor inertia was calculated. The selected motor is an out runner motor. For such motors, it is the outer shell which contains the magnets and acts as the rotor. Further, the assumption that the rotor and shaft account for the majority of the motors mass is made. The ratio between the area of the rotor and area of the shaft is calculated with Equation (4.14), which is used in order to appropriately divided the mass relation in Equation (4.15). In addition, an estimate of the rotor ring width is made. Then, by applying the assumption in conjunction with the motor specifications, the motor inertia is calculated with Equation (4.16). Figure (4.1) shows how the motor moment of inertia is evaluated.

$$k_i = \frac{A_s}{A_r} = \frac{\left(\frac{d}{2}\right)^2 \pi}{[r^2 - (r-w)^2]\pi} \tag{4.14}$$

$$m_r = 0.85 m_m (1 - k_i) \quad \text{and} \tag{4.15}$$
$$m_s = 0.85 m_m (k_i)$$

$$J_m = J_r + J_s \tag{4.16}$$

$$= \frac{1}{2} m_r [r^2 + (r-w)^2] + \frac{1}{2} m_s r_s^2$$

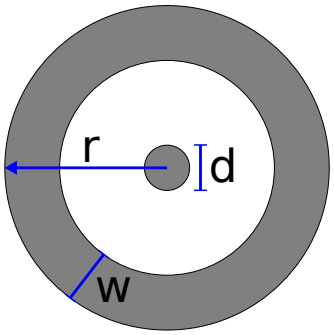



Figure 4.1: Motor inertia illustration

where:

| Symbol: | | Description: | Value: | Unit: |
|---|---|---|---|---|
| $r$ | - | Rotor radius | 24.9 | $mm$ |
| $w$ | - | Rotor width | 5 | $mm$ |
| $d$ | - | Shaft diameter | 8 | $mm$ |
| $A_r$ | - | Rotor area | 0.0012 | $m^2$ |
| $A_s$ | - | Shaft area | $5.0 \cdot 10^{-5}$ | $m^2$ |
| $k_i$ | - | Ratio between shaft and rotor | 0.0405 | $-$ |
| $m_m$ | - | Motor mass | 435 | $g$ |
| $m_r$ | - | Rotor mass | 343 | $g$ |
| $m_s$ | - | Shaft mass | 27 | $g$ |
| $J_m$ | - | Motor inertia | $1.72 \cdot 10^{-4}$ | $kgm^2$ |
| $J_r$ | - | Rotor inertia | $1.72 \cdot 10^{-4}$ | $kgm^2$ |
| $J_s$ | - | Shaft inertia | $2.13 \cdot 10^{-7}$ | $kgm^2$ |

In order to calculate the equivalent load inertia acting on the drive shaft, the conversion of energy approach is applied in Equation (4.17). The equation uses Figure (4.2) as reference, where the Loomo rig has been simplified to consider only one wheel with one fourth of the load.

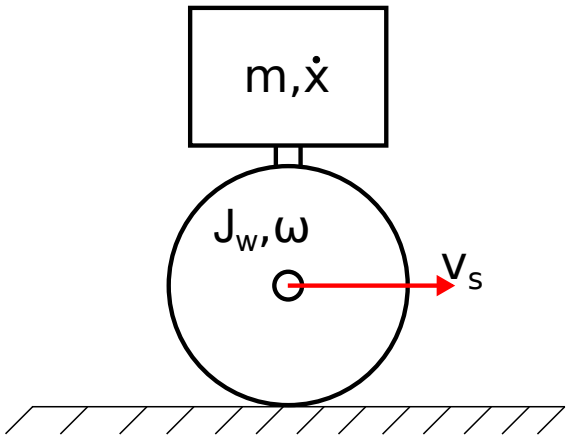



Figure 4.2: Simplified model of Loomo rig

$$\frac{1}{2}J_L\omega^2 = \frac{1}{2}J_w\omega^2 + \frac{1}{2}m\dot{x}^2 \tag{4.17}$$

For a wheel subjected to both rotation and translation, assuming no slip, it can be proven that the center velocity of the wheel is equal to the tangential velocity from the rotational part. Hence, Equation (4.18) applies.

$$v_s = \omega r \tag{4.18}$$

Equation (4.18) is used to substitute the transnational velocity into Equation (4.17), which yields Equation (4.19).

$$J_L = J_w + mr^2 \tag{4.19}$$

The moment of inertia for the mecanum wheel was gathered from the step file provided by the supplier. The data was gathered by importing the file into SOLIDWORKS, then the wheel was given the appropriate material properties. A conservative assumption is made by applying steel to all of the wheel parts except for the rollers, which is given rubber skin, nylon core and brass axle, as can be seen in Figure (3.4).

With both of the required load inertias known, the optimum gear ratio is calculated with Equation (4.20).

$$i_{opt} = \sqrt{\frac{J_L}{J_m}} \tag{4.20}$$

where:

| Symbol: | | Description: | Value: | Unit: |
|---|---|---|---|---|
| $J_L$ | - | Load mass moment of inertia | 1.0436 | $kgm^2$ |
| $J_w$ | - | Mecanum wheel mass moment of inertia | 0.0134 | $kgm^2$ |
| $m$ | - | Mass acting on one mecanum wheel | 100 | $kg$ |
| $v_s$ | - | Velocity of wheel center of rolling wheel | $\sim$ | $^m/_s$ |
| $r$ | - | Wheel radius | 0.0405 | $m$ |
| $i_{opt}$ | - | Optimum gear ratio | 78 | $m$ |

The calculations indicate that the optimum, theoretical, gearing ratio is about 78. It is however not always the best ratio in reality. Due to the space limitations, the motor is going to be connected to the shaft with the use of a belt drive. Specifically, by a timing belt. When using a timing belt, only 6 teeth is required to be in contact to develop full-rated capacity [40]. From accessibility and cost, a gear ratio of five is thus selected. It is achieved with a 12-60 pulley combination. This combination was evaluated in SOLIDWORKS to be a close maximum which still yielded a sufficient amount of teeth in contact. The belt drive setup is shown in Figure (4.3).

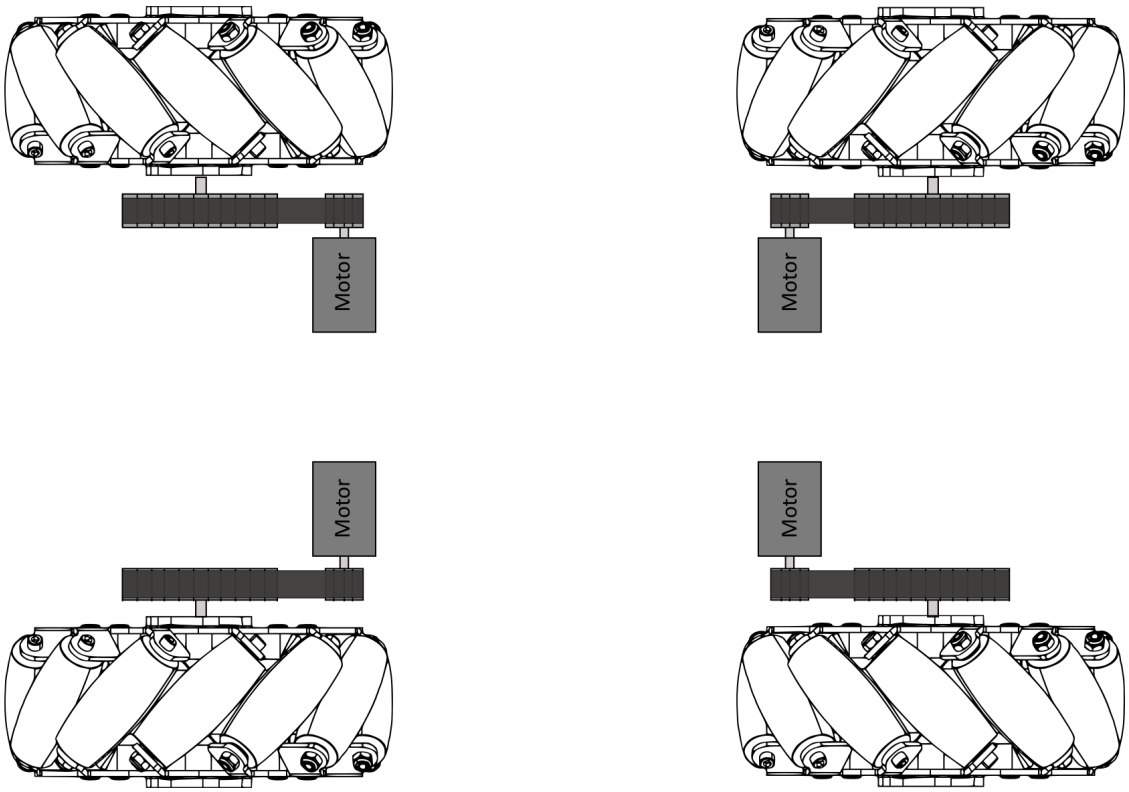



Figure 4.3: Wheels with motor and belt-drive

With the selected gear ratio, the shaft will never reach velocities of the critical speed range from Section (3.3). The reduction allows the motor to use its full range without the risk of the shaft entering critical speed.

### Timing belt

The peak torque the belt drive needs to transmit is defined as the brushless DC motor's maximum torque, and is the effective tension. Figure (4.4) illustrates the slack and tight side which occurs in belt transmissions. The required slack and tight side tension can be calculated with Equation (4.21), where counter clockwise is defined as positive direction of rotation.

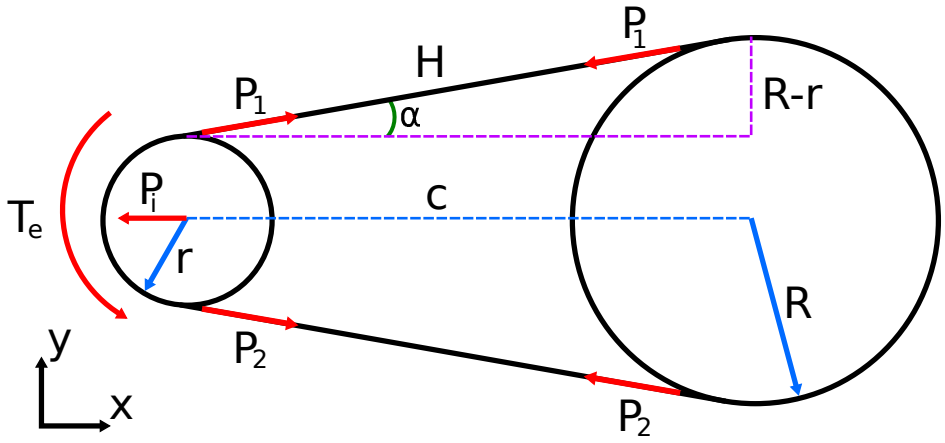



Figure 4.4: Timing belt drive

$$\sum \overset{+}{\circlearrowleft} T = 0 = T_e + (-P_1 + P_2)r \tag{4.21}$$

A timing belt performs at its best when the magnitude of the slack side tension is between 10-30% of the effective tension [41]. Here the middle value is going to be used, $P_2 = 0.20T_e$, and is substitute into Equation (4.21), yielding Equation (4.22) which is used to calculate the required tight side tension.

$$P_1 = \frac{T_e}{r} - P_2 = T_e\left(\frac{1}{r} - 0.2\right) \tag{4.22}$$

In order to find the force components of $P_1$ and $P_2$ along the same axis as $P_i$, the angle that occurs due to the size difference of the gear has to be calculated. This angle can be calculated with Equation (4.23).

$$\alpha = \tan^{-1}\left(\frac{R-r}{C}\right) \tag{4.23}$$

The center distance between the pulley and the gear is calculated with Equation (4.24). Here it is assumed that the belt is long enough to cover half of the pulley.

$$C = \sqrt{H^2 - (R-r)^2} \tag{4.24}$$

$$H = \frac{S - \pi(R+r)}{2}$$

The required initial belt tension is dependant of the elastic characteristics of the belt, however it is usually sufficient to assume that Equation (4.25) is applicable to calculate it [42]. The required pretension in order to achieve prime performance is then calculated to be about 31.5 N

$$P_i = \frac{P_1 \cos(\alpha) + P_2 \cos(\alpha)}{2} \tag{4.25}$$

where:

| Symbol: | | Description: | Value: | Unit: |
|---|---|---|---|---|
| $T_e$ | - | Effective tension | 6.8 | $Nm$ |
| $P_1$ | - | Tight side tension | 65.8 | $N$ |
| $P_2$ | - | Slack side tension | 1.4 | $N$ |
| $P_i$ | - | Pre-tension | 31.5 | $N$ |
| $r$ | - | Pulley radius | 0.009 | $m$ |
| $R$ | - | Gear radius | 0.047 | $m$ |
| $S$ | - | Belt length | 0.395 | $m$ |
| $C$ | - | Center distance | 0.103 | $m$ |

## 4.2 Electronic Speed controller

Electronic speed controller (ESC) is a power regulation device which control the motor throttle. Additionally, it may support reversing and dynamic breaking. Brushed and brushless are two types of controllers and can only be used with their respective motors. Since a brushless DC motor was selected, a compatible brushless ESC has to be selected.

The ESC utilized is a Vedder developed ESC for brushless motors. The three phase current signals are manipulated with metal oxide semiconductor field effect transistors (MOSFET), and the back EMF can be used to determined rotation, as an alternative to the use of hall sensors [43].

To control the torque of the motors, the field oriented control (FOC) option is used on the VESC. The option is seleced, since for any brushless dc motor, FOC will be the most efficient way of motor control. For FOC, the current representation will be sinusoidal, which will give more space vectors compared to six static space vectors from traditional three phase commutation. The increase of space vectors results in significantly more magnetic field orientations that the rotor can follow. The number of possible field orientations eliminates the pulsating torque problem from six vector commutation. Further, by utilizing pulse with modulation, it is possible to control the strength of magnetic field when the rotor is turning.

## 4.3 Hardware

In this section, all the of the hardware components used are listed. Among them are the components in order to establish communication between the motor controller units and the processing unit. As well as the other hardware components, such as the battery and the ESC used.

### Teensy 3.6

Teensy 3.6 is a microcontroller development system that is programmable with the C languages. It is compatible with Arduino functions, libraries, and IDE through an additional package. Due to a broad platform of open-source material, the Teensy is well suited for communication with the ESCs. By using UART communication between the Teensy and the processor, and CAN-Bus communication through an adapter from the Teensy to the motors, the rig can be controlled in manual and autonomous mode. For manual mode, the analog in(AI) pins are used for reading joystick values. Additionally, the Teensy is equipped with an SD-card reader which can be used to log data for post-processing. Figure (4.5) shows the pinouts of the Teensy 3.6, in addition to the pin assignments.

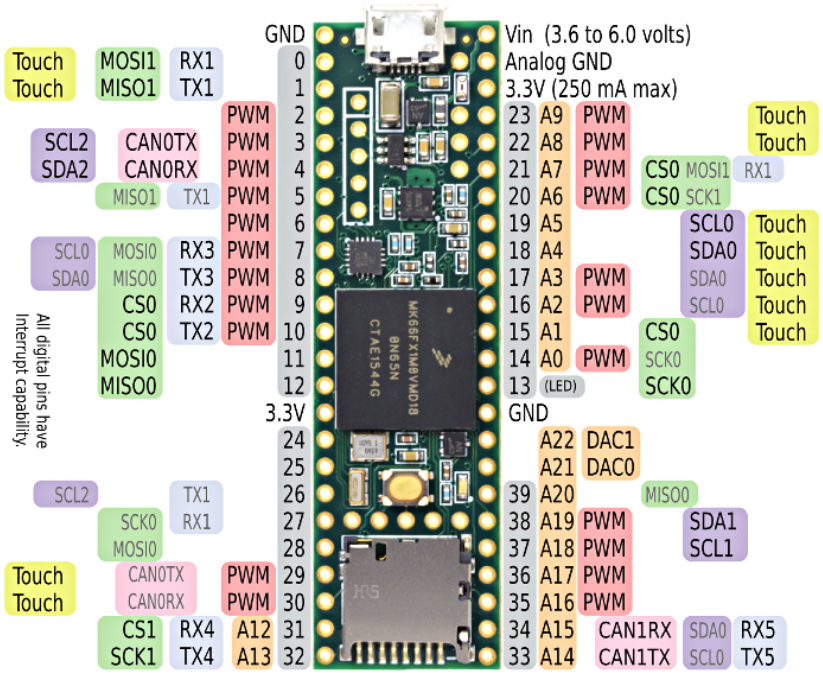



Figure 4.5: Teensy 3.6 pinouts [3]

### CAN-Bus adapter

In order to send and receive messages from a CAN-network onto the Teensy, a CAN-Bus adapter is required. The selected CAN-bus adapter is specialized for fitting onto a Teensy 3.5 or 3.6, and its shape will still keep the other pins available if soldered directly onto the Teensy. The adapter has an internal 120 Ω termination resistor for connection between high and low on the CAN-Bus line.

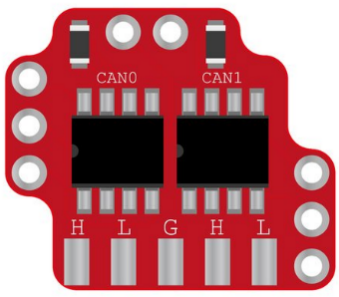



Figure 4.6: Dual Can-bus adapter [4]

### VESC

The MayTech Vedder developed electronic speed controller (VESC) is used to control each of the brushless motors via programmed CAN-Bus signals. The open-source controller, together with the ROS developed packages, is it suitable for robotic applications such as the Loomo parking rig.

Figure 4.7: SuperFOC6.8 - VESC6 [5]

### MTO 5065 BLDC Motor

The developed Loomo parking rig is equipped with four MayTech brushless motors for actuation. This motor is suited for robot applications due to its compact design and high torque characteristics. The motor is of type open cover outrunner and is compatible with the selected VESCs. Figure (4.8) shows the discussed motor and in Section (4.1.2) the motor specifications are presented.

Figure 4.8: MTO 5065 HA 70 Kv [6]

### NVIDIA Jetson AGX Xavier Developer Kit

A Jetson AGX Xavier Developer Kit is installed to processes raw data from the equipped cameras. The Xavier is tailor made for robots, drones and other autonomous machines. It is an embedded module with very high processing power, which makes it ideal for this application. [7].


Figure 4.9: Jetson AGX Xavier [7]

### PCI-E Interface card 7x-USB

A Interface card for USB extension is add to the Jetson AGX Xaviers PCI slot. The card has seven USB 3.0 ports that will allow the camera data from four Azure Kinects to be processed by the Jetson module.

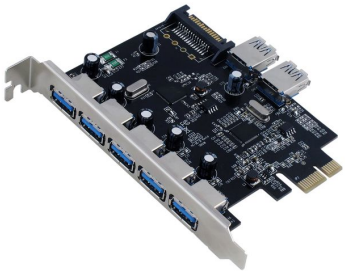


Figure 4.10: Exsys Interface Card [8]

**Battery Vision CP12200**

The developed Loomo parking rig is equipped with two, six celled 12 V lead acid batteries. Each battery has a capacity of 20 Ah and equivalent to the battery presented in Figure (4.11). It is desired to wire them in series to increase the voltage.


Figure 4.11: Battery Vision CP12200 [9]

**Joystick**

In order to implement manual control for the Loomo parking rig, two joysticks with dual axis are attached to a hand controller. The joystick consists of two potentiometers and a switch. The potentiometers sends analog values based on their orientation. The switch is closed by pressing down on the stick. The values can be read by the microcontroller, which then can be appropriately convert in to velocity references.

Figure 4.12: Dual Axis Joystick [10]

## 4.4 System Overall Power Consumption

To verify that the batteries used are able to power the rig during operation, the power consumption of each component is evaluated. In Table (4.3) each components power requirement is presented. With Equation (4.26) power consumption is converter to watt, with the Teensy 3.6 and Azure Kinects running on 5 V converters, while the batteries delivery of 24 V.

| Device: | Power consumption: | Unit: |
|---|---|---|
| Teensy 3.6 | 80 | mA |
| Nvidia Xavier | 50 | W |
| Azure Kinect | 6 | W |
| Kinect external power | 2.5 | A |
| MTO5065 70 HA | 42.5 | A |
| PCI-e Card | 10 | W |

Table 4.3: Power Consumption Components

$$\Delta W = \sum_{i=1}^{n} W_n \quad , \quad \text{Where: } W = VI \tag{4.26}$$

where:

| Symbol: | | Description: | Value: | Unit: |
|---|---|---|---|---|
| W | - | Watt | $\sim$ | W |
| I | - | Current | $\sim$ | A |
| V | - | Voltage | $\sim$ | V |
| $\Delta W$ | - | Sum watt | 4214.4 | W |

The maximum power consumption if all the motors are run at full speed and all the sensors are operative is calculated to be 4214.4 W.

## 4.5 Power Supply

In order to keep the the functions on the rig working in manual and autonomous modes, the rig must be equipped with a battery pack. The battery pack should be able to deliver the current requirements calculated in the previous section. As of now, the battery pack consist of two 12 V lead acid batteries wired in series providing the system with with a 24 V and 20 Ah. The batteries have a capacity of 480 Wh which is too small for this application, but may be used for testing purposes. At absolute maximum, the system requires 4214.4W for optimum functionality and the batteries can supply this for 7 minutes. Although, due to losses when discharging by Peukert's Law, the system the will have a reduced operational time compared to theoretical operational time[44]. The lead acid battery pack is used as a temporary solution.

## 4.6 Wiring

The system is temporally supplied by two 12v 20Ah batteries connected in series centred on the parking rig to achieve uniform weight distribution. A 200 ampere fuse is added close to the battery which will melt and break the circuit in the event of excessive current demands from the motors, e.g steep inclination. The use of a 200 ampere fuse is chosen because each motors can draw a maximum of 50 ampere. A power switch is placed between the fuse and a power distributor to allow powering on and off the system. The distributor disperse the power to the front and back of the rig. At each end of the rig, another power distributor is placed to divide the power between two speed controllers. An overview of the wiring can be seen in Figure (4.13).

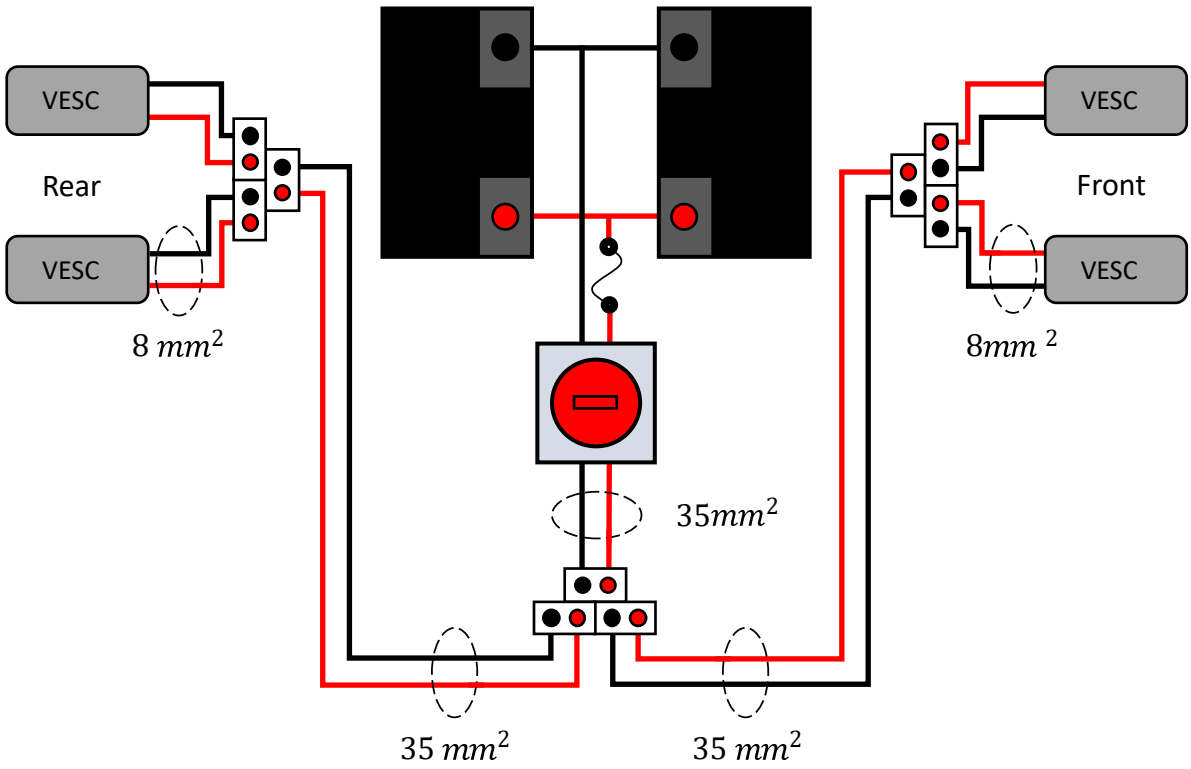



Figure 4.13: Power distribution

The current drawn from the batteries can become large, and to reduce the losses and excessive heating, the sizing of the wires is considered. A cable of 35 $mm^2$, equivalent to American Wiring Gauge (AWG) 2.0 [45], is chosen from the batteries to the second distributor. From the second distributor to each VESC, the current will be considerably less with a maximum of 50 ampere. Because of this, wire size 8 $mm^2$ or AWG 8.0 are utilized [45]. The losses is then calculated with Equation (4.27) [46]. Any length of wire powering the system does not exceed 0.50 m, which will be used in calculations from here on.

$$\Delta U = R_{Wire} I L \tag{4.27}$$

where:

| Symbol: | | Description: | Value: | Unit: |
|---|---|---|---|---|
| $R_{Wire}$ | - | Wire Resistance | $\sim$ | $^{m\Omega}/_{m}$ |
| | - | Wire Resistance AWG 2.0 [47] | 0.5127 | $^{m\Omega}/_{m}$ |
| | - | Wire Resistance AWG 8.0 [47] | 2.061 | $^{m\Omega}/_{m}$ |
| $I$ | - | Current | $\sim$ | $A$ |
| $L$ | - | Wire length | 0.50 | $m$ |
| $\Delta U$ | - | Voltage loss wiring | $\sim$ | $V$ |
| | - | Max Voltage loss AWG 2.0 | 0.0512 | $V$ |
| | - | Max Voltage loss AWG 8.0 | 0.0515 | $V$ |

$$\Delta U_{Total} = 2U_{AWG2} + U_{AWG8} \tag{4.28}$$

From Equation (4.28) the total voltage loss from the batteries to each motor controller is calculated. The loss are 0.64% of the nominal 24 V provide by the batteries and are consider acceptable for the system performance.

Further the change in heat in the wiring when driving the parking rig is evaluated. The wiring previously mention is used when calculating the temperature change with Equation (4.29) with a 40 A over 30 seconds [48].

$$\Delta t = \frac{Q}{cm} = \frac{VC}{cm} = \frac{I^2 R_{Wire} Ls}{cm} \tag{4.29}$$

where:

| Symbol: | | Description: | Value: | Unit: |
|---|---|---|---|---|
| Q | - | Heat added from battery | $\sim$ | $Joule$ |
| c | - | Heat capacity copper [49] | 390 | $^{J}/_{kgK}$ |
| m | - | Mass of copper in wire [50] | $\sim$ | $kg$ |
| V | - | Voltage | $\sim$ | $V$ |
| C | - | Charge | $\sim$ | $C$ |
| s | - | Time | 30 | $s$ |
| $\Delta t$ | - | Change in temperature | $\sim$ | $K$ |
| | - | Change in temperature AWG 2.0 | 0.42 | $K$ |
| | - | Change in temperature AWG 8.0 | 7.04 | $K$ |

The results from Equation (4.29) present a small increase in temperature for the wiring. There is however no danger of melting the PVC insulation which can withstand 140°C, or the 3D printed plastic covers wich has a 65°C structure changing point [51] [52].

# 5. Modeling

In the modeling chapter, the kinematic constraints of a mecanum wheeled robot is derived. Further on, the process of creating a simulation model of the plant is documented. Also, the software used to develop the simulation model and the framework for the overall robot system is introduced in this chapter.

## 5.1 Mecanum Wheel Kinematics

In order to send the appropriate velocity reference to the respective mecanum wheel, in addition to calculates estimates of the robot's pose, kinematic models are required. In this section the inverse and forward kinematic models to a mecanum wheeled robot are derived.

### 5.1.1 Inverse kinematics

In the deriving of the inverse kinematic model, a bottom view of the mecanum robot is used. First the constraints to one single mecanum wheel is considered, which is assumed to be subjected to pure rolling. This means that the contact velocity is assumed to be zero and the wheel does not slip. Thus, Equation (5.1) applies.

$$v_r = r\omega \tag{5.1}$$

where:

| Symbol: | | Description: | Value: | Unit: |
|---|---|---|---|---|
| $v_r$ | - | Wheel velocity along rolling direction | $\sim$ | $^m/_s$ |
| $r$ | - | Wheel radius | 0.1015 | $m$ |
| $\omega$ | - | Wheel angular velocity | $\sim$ | $^{rad}/_s$ |

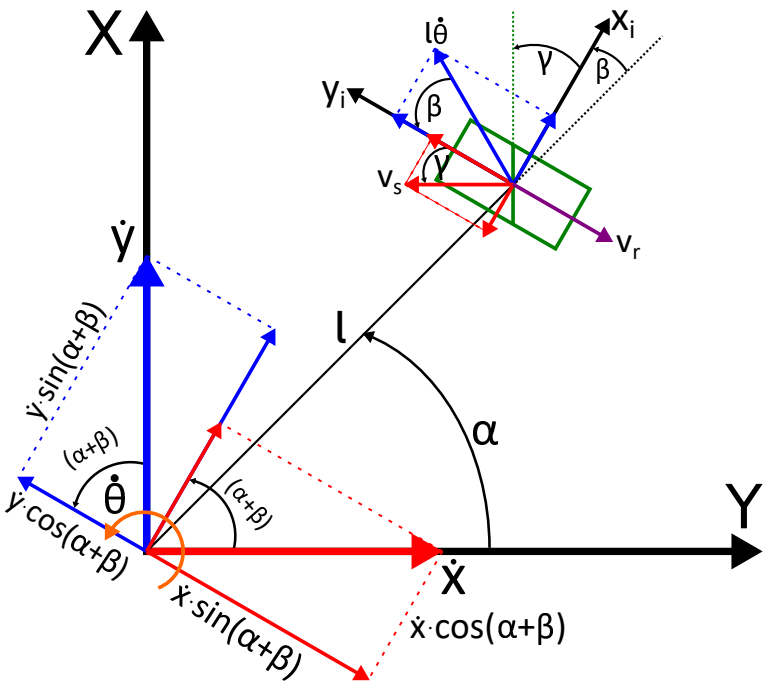



Figure 5.1: Single mecanum wheel

In Figure (5.1), the movement vectors of a single mecanum wheel, with its local coordinate system, $i$, and the local coordinate system to an example robot frame is shown. When deriving the kinematic constraints for a single kinematic wheel, the robot coordinate system is considered to be the global frame. The rolling and sliding constraints are derived by summarizing the appropriate vectors associated to each of the coordinate systems. Specifically, the vectors attached to the robot frame, and the wheel frame are collected on each side of the respective equation. The rolling Equation (5.2) consists of all vectors parallel to the rolling direction, $v_r$, and the sliding Equation (5.3) consists of all vectors perpendicular to the rolling direction.

$$-v_r + v_s \cos(\gamma) = -\dot{x} \sin(\alpha + \beta) + \dot{y} \cos(\alpha + \beta) + l\dot{\theta} \cos(\beta) \tag{5.2}$$

$$-v_s \sin(\gamma) = \dot{x} \cos(\alpha + \beta) + \dot{y} \sin(\alpha + \beta) + l\dot{\theta} \sin(\beta) \tag{5.3}$$

Since $v_s$ is not controllable, the kinematic constraint Equation (5.2) and Equation (5.3) are solved for this variable in order to eliminate it, which yields Equation (5.4).

$$\frac{-\dot{x} \sin(\alpha + \beta) + \dot{y} \cos(\alpha + \beta) + l\dot{\theta} \cos(\beta) + v_r}{\cos(\gamma)} = \frac{-\dot{x} \cos(\alpha + \beta) - \dot{y} \sin(\alpha + \beta) - l\dot{\theta} \sin(\beta)}{\sin(\gamma)} \tag{5.4}$$

Then, by applying the trigonometric relations presented in Equations (5.5) and Equation (5.6), Equation (5.4) is simplified. The simplification calculations can be found in Appendix (B).

$$\sin(\alpha \pm \beta) = \sin(\alpha) \cos(\beta) \pm \cos(\alpha) \sin(\beta) \tag{5.5}$$

$$\cos(\alpha \pm \beta) = \cos(\alpha) \cos(\beta) \mp \sin(\alpha) \sin(\beta) \tag{5.6}$$

Equation (5.7) describes the simplified kinematic constraint of a single mecanum wheel, arbitrarily placed in reference to a robot frame.

$$\dot{x} \cos(\alpha + \beta + \gamma) + \dot{y} \sin(\alpha + \beta + \gamma) + l\dot{\theta} \sin(\beta + \gamma) = -v_r \sin(\gamma) \tag{5.7}$$

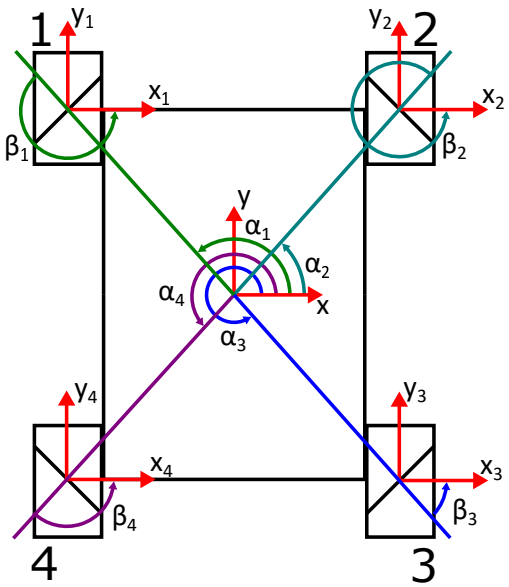



Figure 5.2: Angles with used wheel configuration (Bottom view)

In Figure (5.2), the wheel configuration of the parking rig, viewed from the bottom, is shown. With this specific configuration and due to the behaviour of the trigonometric expression, it can be seen that Equation (5.8) is fulfilled.

$$\alpha + \beta = 0 \tag{5.8}$$

This relation is substituted into Equation (5.7), which yields the reduced kinematic Equation (5.9).

$$\dot{x}\cos(\gamma) + \dot{y}\sin(\gamma) + l\dot{\theta}\sin(-\alpha + \gamma) = -v_r \sin(\gamma) \tag{5.9}$$

Equation (5.9) is then rearranged and the trigonometric relation in Equation (5.10) and Equation (5.11) are applied, which yields Equation (5.12).

$$\sin(-a) = -\sin(a) \tag{5.10}$$

$$\cos(-a) = \cos(a) \tag{5.11}$$

$$\frac{\dot{x}}{\tan(\gamma)} + \dot{y} + l\dot{\theta}\left(\cos(\alpha) - \frac{\sin(\alpha)}{\tan(\gamma)}\right) = -v_d \tag{5.12}$$

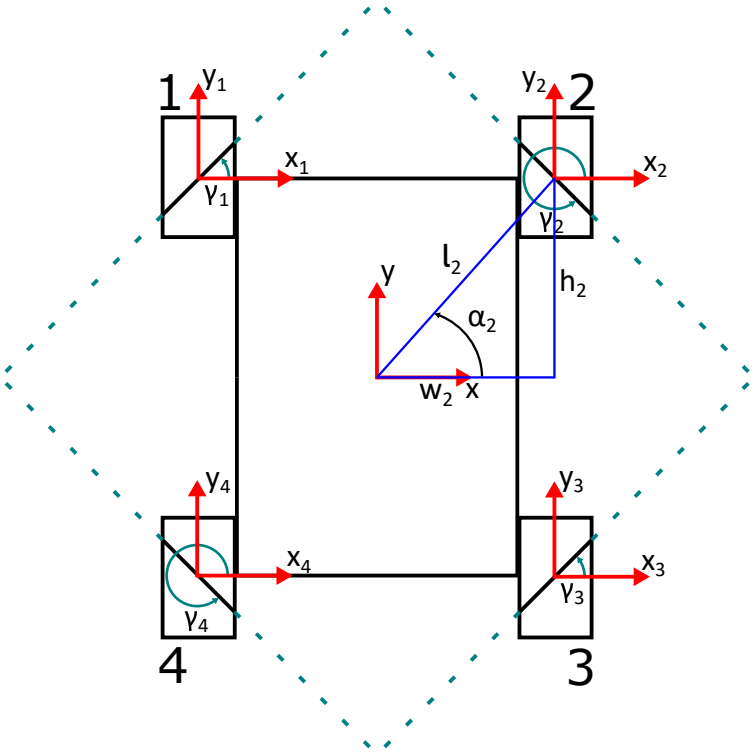



Figure 5.3: Roller angles (Bottom view)

Figure (5.3) shows the length from the center of the robot frame to the center of an arbitrarily selected mecanum wheel. It should be noted that for this configuration, the following relation in Equation (5.13) - (5.15) are true, where the lengths are the measured values of the rig prototype.

$$l = l_1 = l_2 = l_3 = l_4 \tag{5.13}$$

$$h = h_1 = h_2 = h_3 = h_4 \tag{5.14}$$

$$w = w_1 = w_2 = w_3 = w_4 \tag{5.15}$$

where:

| Symbol: | | Description: | Value: | Unit: |
|---|---|---|---|---|
| $l$ | - | Length from center of robot to center of wheel | 700 | $mm$ |
| $h$ | - | Height from center of robot to center of wheel | 635 | $mm$ |
| $w$ | - | Width from center of robot to center of wheel | 285 | $mm$ |

In addition, Figure (5.3) can be used in order to derive Equation (5.16) and Equation (5.17), which are substituted into Equation (5.12). The latter, in conjunction with Equation (5.1), yields the final Equation (5.18). The final equation is applicable for all four wheels connected to the robot frame.

$$w = l\cos(\alpha) \tag{5.16}$$

$$h = l\sin(\alpha) \tag{5.17}$$

$$\begin{bmatrix} \frac{1}{\tan(\gamma)} & 1 & w - \frac{h}{\tan(\gamma)} \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = -r\omega \tag{5.18}$$

Lastly, Figure (5.3) shows how the angle $\gamma$ is defined on the frame configuration. Essentially, it is the angle between the $x$-axis and the closest roller line, and since the roller angle to a mecanum wheel is 45°, Equation (5.19) and Equation (5.20) applies.

$$\gamma_1 = \gamma_3 = 45^\circ \tag{5.19}$$

$$\gamma_2 = \gamma_4 = 360^\circ - 45^\circ = 315^\circ \tag{5.20}$$

In conjunction with Equation (5.16) - (5.18), Table (5.1) was made to substituted with the appropriate angles for each wheel on the construction. The lengths and the appropriate signs yields the final inverse kinematic model of the mecanum wheeled robot shown in Equation (5.21).

| Wheel no. | $\alpha$ | $\gamma$ | Sign $\cos(\alpha)$ | Sign $\sin(\alpha)$ | Sign $\sin(\gamma)$ |
|---|---|---|---|---|---|
| 1 | 114.2° | 45° | - | + | + |
| 2 | 65.8° | 315° | + | + | - |
| 3 | 294.2° | 45° | + | - | + |
| 4 | 245.8° | 315° | - | - | - |

Table 5.1: Kinematic sign table

$$\begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix} = \frac{-1}{r} \begin{bmatrix} 1 & 1 & -w-h \\ -1 & 1 & w+h \\ 1 & 1 & w+h \\ -1 & 1 & -w-h \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} \tag{5.21}$$

Equation (5.21) is derived based on a bottom view perspective of the mecanum wheel configuration. Because the operator will perceive the Loomo rig from the top, it is beneficial to change the inverse kinematic model to correspond with this view. When changing the view from bottom to top, the x- and y-directions behave the same. This change will only affect the angle in the sense that an object that rotates counter clockwise viewed from the bottom, rotates clockwise when viewed from the top. This is easily changed in the inverse kinematic model by changing the direction of $\dot{\theta}$. This yields the inverse kinematic model viewed from the top in Equation (5.22), with counter clockwise as the positive direction of rotation.

$$\begin{bmatrix}\omega_1\\ \omega_2\\ \omega_3\\ \omega_4\end{bmatrix} = \frac{-1}{r}\begin{bmatrix}1 & 1 & w+h\\ -1 & 1 & -w-h\\ 1 & 1 & -w-h\\ -1 & 1 & w+h\end{bmatrix}\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}\end{bmatrix} \tag{5.22}$$

### 5.1.2 Forward Kinematics

In navigation applications it is common to have what is called an odometry model. Odometry is the use of motion sensors, such as encoders, hall sensors or IMU, in order to estimates a moving robot's change in position over time. For indoors environments, localization with odometry boils down to estimating and tracking its pose, meaning the robot's position and orientation. Odometry is widely used because it can be implemented independently without external information outward, which makes it simple, inexpensive and easy to accomplish in real time.[53] However, as an example, the wheels used to move the robot's base have the possibility to slip or skid. This makes the estimated change in position prone to errors. In addition, errors will continuously accumulate during operation. Due to this, it is common to use odometry to quickly estimate the robot's change in position, then correct deviations with external sensors, such as cameras or laser scanners.

In order to convert the data from the hall sensors into useful information, a forwards kinematics model of the rig can be used. This model was derived by using the already derived inverse kinematics model. The inverse kinematic model shown in Equation (5.21) is a system of equations that can also be described by Equation (5.23).

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \tag{5.23}$$

where:

$$\boldsymbol{A} = \frac{-1}{r}\begin{bmatrix}1 & 1 & w+h\\ -1 & 1 & -w-h\\ 1 & 1 & -w-h\\ -1 & 1 & w+h\end{bmatrix} \qquad \boldsymbol{x} = \begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}\end{bmatrix} \qquad \boldsymbol{b} = \begin{bmatrix}\omega_1\\ \omega_2\\ \omega_3\\ \omega_4\end{bmatrix}$$

In order to yield the forward kinematic model, Equation (5.23) has to be solved with respect to $\boldsymbol{x}$. This is done by eliminating the matrix $\boldsymbol{A}$ by multiplying with its inverse. However, as can be seen, the $\boldsymbol{A}$ matrix is not square. This means that inverse can not be directly calculated, but its pseudo inverse might be possible to obtain.

First, the rank of $\boldsymbol{A}$ is calculated. This is done by reducing the matrix into Echelon Form, which is shown in Equation (5.24). As can be seen, the pivot diagonal contains three ones. The matrix is of full column rank, since the rank $= n = 3$.

$$\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\\ 0 & 0 & 0\end{bmatrix} \tag{5.24}$$

The validation is performed to check the dependency of the matrix. If the rank of a matrix with $m$ rows and $n$ columns is not of full column rank, where full rank being; rank $= n$, the matrix columns are linearly dependent. If this is true, the matrix will become singular in the upcoming operations and will not be possible to invert.

Since the matrix $\boldsymbol{A}$ has been verified to be column linearly independent, the transpose of $\boldsymbol{A}$ is multiplied by Equation (5.23), which yields Equation (5.25).

$$\boldsymbol{A^T A x} = \boldsymbol{A^T b} \tag{5.25}$$

$\boldsymbol{A^T}$ is a $m \times n$ matrix and $\boldsymbol{A}$ is a $n \times m$ matrix, thus their matrix product will yield a $n \times n$ matrix. In addition, it is non-singular and possible to invert, proven by the previous validation.
Now Equation (5.23) can be solved for $\boldsymbol{x}$ by multiplying by $(\boldsymbol{A^T A})^{-1}$ on both sides. This yields Equation (5.26)

$$\boldsymbol{x} = (\boldsymbol{A^T A})^{-1} \boldsymbol{A^T b} \tag{5.26}$$

In Equation (5.26), the terms containing the variants of the $\boldsymbol{A}$ matrix is what is referred to as the pseudo inverse. Typically this has the superscript of a cross or a dagger. The pseudo inverse is highlighted in Equation (5.27)

$$\boldsymbol{A}^\dagger = (\boldsymbol{A^T A})^{-1} \boldsymbol{A^T} \tag{5.27}$$

By performing the steps to calculate the pseudo inverse, the forward kinematic model is derived, which is shown in Equation (5.28) and Equation (5.29).

$$\boldsymbol{x} = \boldsymbol{A}^\dagger \boldsymbol{b} \tag{5.28}$$

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \frac{-r}{4} \begin{bmatrix} 1 & -1 & 1 & -1 \\ 1 & 1 & 1 & 1 \\ \frac{1}{w+h} & -\frac{1}{w+h} & -\frac{1}{w+h} & \frac{1}{w+h} \end{bmatrix} \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix} \tag{5.29}$$

In order to reference the forward kinematics model from the robot frame to a global world frame, the velocities $\dot{x}$, $\dot{y}$ and $\dot{\theta}$ has to be appropriately transformed. The transformation matrix is going to be derived by using Figure (5.4).

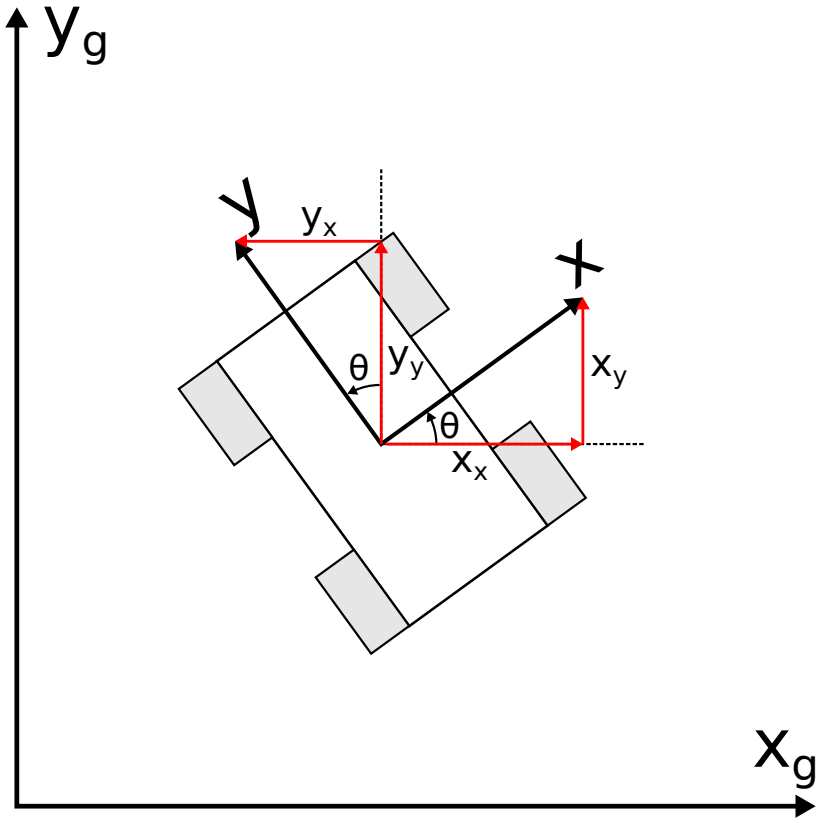



Figure 5.4: Robot inside global reference frame

When the robot rotates, it is about the z-axis. This means that the angle is only affected by itself, and is the same when viewed from the robot frame and global frame. This is expressed with Equation (5.30). As for the sum of change in the x- and y-direction viewed from the global frame, it can be expressed with Equation (5.31) and Equation (5.32), respectively.

$$\Delta\theta = \Delta\theta \tag{5.30}$$

$$\Delta x_g = \Delta x \cos(\theta) - \Delta y \sin(\theta) \tag{5.31}$$

$$\Delta y_g = \Delta x \sin(\theta) + \Delta y \cos(\theta) \tag{5.32}$$

The transformation matrix, $\boldsymbol{R}$, is then derived by merging Equation (5.30)-(5.32) into a system of Equations (5.33).

$$\begin{bmatrix} x_g \\ y_g \\ \theta \end{bmatrix} = \underbrace{\begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{R}} \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \tag{5.33}$$

To calculate an estimate of the robot's pose in the global reference frame, the robot frame velocities are first transformed to global reference with Equation (5.34).

$$\begin{bmatrix} \dot{x}_g \\ \dot{y}_g \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} \tag{5.34}$$

Then the velocities are integrated. Since the velocity data is obtained from the sampled data, the integration is done numerically with Equation (5.35).

$$\begin{bmatrix} x_g[k+1] \\ y_g[k+1] \\ \theta[k+1] \end{bmatrix} = \begin{bmatrix} x_0 + x_g[k] \\ y_0 + y_g[k] \\ \theta_0 + \theta_g[k] \end{bmatrix} + \begin{bmatrix} \dot{x}_g[k+1] \\ \dot{y}_g[k+1] \\ \dot{\theta}_g[k+1] \end{bmatrix} dt \tag{5.35}$$

where:

| Symbol: | | Description: | Value: | Unit: |
|---|---|---|---|---|
| $x_g, y_g, \theta[k+1]$ | - | Updated position | $\sim$ | $m$ |
| $x_g, y_g, \theta[k]$ | - | Previous position | $\sim$ | $m$ |
| $x_0, y_0, \theta_0$ | - | Initial position | $\sim$ | $m$ |
| $\dot{x}_g, \dot{y}_g, \dot{\theta}[k+1]$ | - | Sampled velocity | $\sim$ | $^m/_s$ |
| $dt$ | - | Sampling time | $\sim$ | $s$ |

## 5.2 Robot Operating Systems

Many of the advanced techniques used in this project are implemented through Robot Operating System, more commonly referred to as ROS. In order to understand the flow of the system, an introduction to ROS is presented. ROS is an open-source framework that provides libraries and tools for creating robot applications. It provides support for hardware, drivers, simulation, visualization, and more. The founders of ROS had the impression that the field of robotics evolved slowly because developers were focusing their efforts on completely different fields. Therefore, they aimed to combine people's work and made it their primary goal to develop more advanced robots based on code reuse. ROS is designed to be modular, which enables the user to utilize as much or as little of ROS as desired. The framework has gained massive support in the robotics community and proven to be an effective tool for creating advanced robotic applications.[54]

ROS is constructed as a coupled system consisting of multiple nodes connected together. A node is a computational process and each node should be responsible for one task. A ROS master enables individual nodes to locate one another, and a system can only have one master node. Once these nodes have located each other they can communicate peer-to-peer. Organizing the system in this way provides the modular design and reduces the complexity of each script. Additionally, it makes the debugging process more systematic because it is simple to isolate the problem to a single node.

The couplings between the nodes are called topics and is a communication bus for exchanging messages between the nodes. A topic has a unique name and correlates with a specific data type. The data types in ROS are called messages and follow the SI unit standard. When a node wants to make information available to other nodes, it publishes the information to a topic. When a publisher is registered with the master, other nodes can access the data from the topic by subscribing to it. Whether a subscriber is in the same node, in a different node on the same machine, or a different node on a different machine, the message will be put in a subscriber node's callback queue for processing. Message reception is independent of the message's source and location. The ROS framework is very flexible because it supports multiple programming languages, and the messaging technique makes it possible for nodes that are written in different languages to communicate with each other. As long as the node maintain the standardized message types, nodes written in different languages will understand each other. A simple ROS network is shown in Figure (5.5), where a node is publishing to a topic with a specific message type and two other nodes are subscribing to it.

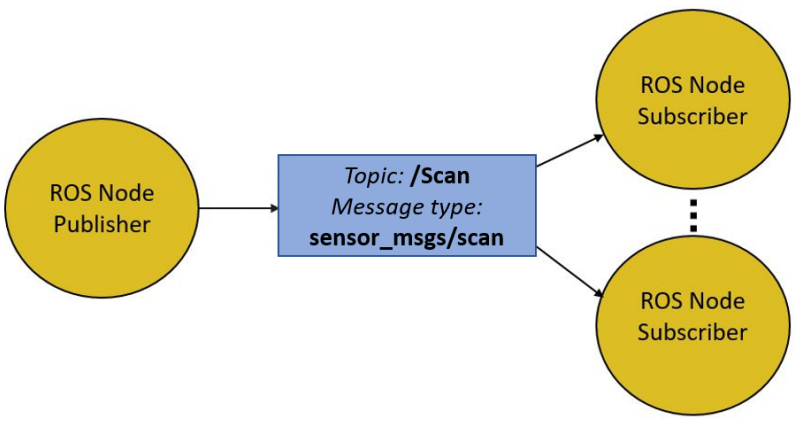



Figure 5.5: A simple ROS network

Nodes can be initialized through *.launch* files. A launch file is written in the XML format and is used to start one or multiple nodes together with their respective configuration files. Inside the file, the location of the package directory, configuration files, as well as the type of node can be specified.

In ROS, nodes and all their corresponding configuration files required to fulfill a feature or application are referred to as packages. Further on, a stack is a collection of packages which are created to accomplish a common goal. For instance, this thesis is based on a navigation stack, which serves to localize and navigate a robot in a 2D environment.

## 5.3 Simulation Model

Simulation is a widely used approach when developing and testing complex systems. When working with a mobile robot, it can be very time consuming if the system has to be tested on the physical model every time a change is made. A common way to make the process more efficient is to describe the physical system as a model on a computer. This is convenient when the real plant is inaccessible or when the system is to be tested in an environment which is not possible to manipulate. Simulation grants the contingency to design a control system in parallel with the development of the physical plant, resulting in increased efficiency and reduced costs. Simulating the process can also increase the safety during development and help avoid accidents due to unexpected errors in the system. If the simulation model is adequate, the fine-tuned system can be directly attached to the real plant as soon as the plant is accessible [55].

The simulation software used in this project was GAZEBO, which is an open source physics engine that is designed to accurately and efficiently simulate robots in an environment that resembles the real world. GAZEBO is a standalone software that supports ROS integration through a dedicated package, the gazebo_ros_pkg package [56]. This package provide the necessary interfaces to simulate a robot using ROS messages. GAZEBO plugins are used to simulate sensors and integrate actuators [57]. To display physical and virtual sensor data, RVIZ was used. RVIZ is an open source 3D visualization tool that is dedicated to the ROS framework. This software is extensively used to see what the robot sees, which is a great tool for debugging processes. It provides a great way to identify problems such as sensor misalignments or robot model inaccuracies [58].

### 5.3.1 Robot Model

A robot model is a collection of links, joints, and plugins. Links are the rigid bodies of the robot, whereas joints are the movable components of the robot that cause relative motion between adjacent links. GAZEBO requires that the robot model is described as Simulation Description Format (SDF), while RVIZ utilizes Unified Robot Description Format (URDF). The robot in this project was described as a URDF file because it is the standardized format in ROS and because the gazebo_ros_pkg includes a URDF to SDF converter. This conversion is necessary because URDF can only specify kinematics and dynamic properties of a robot in isolation, but can not specify the pose of the robot within a world. To define a robot model, the location and orientation of all the links and joints has to be specified. The pose of all the frames were found from the Loomos rigs SOLIDWORKS assembly. The robot consist of a base link, four wheel links and four camera links. Revlolute joints were

assigned between the base link and the wheels, whereas the joint between the base link and the cameras was set to fixed. Each link is divided into multiple properties. [59]

### Visual Property

The visual property of a link describes the shape of the object for visualization purposes only. The shape can be a simple geometrical element or a complex geometry described by a mesh file. The visual property can also define the material color of the object. The visual property of the robot is shown in Figure (5.6). [59]

Figure 5.6: Visual property of the URDF

### Collision Property

All links are also described with a collision element. In oppose to the visual element, a collision property contains physical attributes and is required to detect physical collision of the model. The collision element should be a simplified representation of the visual geometry. If the collision model is very complicated, the collision calculations are going to be very heavy. The base link was therefor assigned as a simple cuboid, whereas all the wheels were specified as cylinders. The red volume in Figure (5.7) represent the collision geometry. [59]

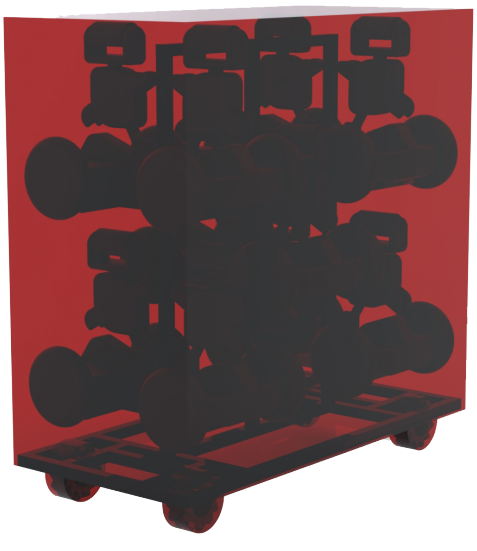


Figure 5.7: Collision property of the URDF

### Inertia Property

The inertia property depends on both the mass and the distribution of mass of the object. It is important that the inertia origin is located at the center of gravity. The inertia data extracted from SOLIDWORKS resulted in the model collapsing in GAZEBO. For that reason, the inertia for each link was calculated based on simplified shapes. The base link was assigned an inertia of a box, while the wheels were replaced by inertia from cylinders. [59]

### Transmission

For each joint that will be controlled by an actuator, a transmission property must be defined. The transmission property contains information about what joint to manipulate, the type of actuator and mechanical reduction. In this project, the wheels will be controlled individually by four motors which requires a transmission property each. [59]

### Plugins

Plugins are used to give the URDF models greater functionality by interpreting ROS messages from sensor outputs and motor inputs. Plugins are specified in the URDF file and wrapped with the $< gazebo >$ tag to indicate that the information should be passed to GAZEBO.

To move the model around in the simulation environment, a planar move plugin was used. This plugin can move arbitrary objects along a horizontal plane and accepts linear velocities $(\dot{x}, \dot{y})$ and angular velocities $(\dot{\theta})$. This means that this plugin is not intended for use in slopes, as it can only move the object in a 2D plane. For this simulation, the planar mover is linked to the base link, which moves the robot as a whole. In a realistic scenario, the wheels would be driving the motion of the base, however this requires an accurate description of the mecanum wheels, which is not available. For this reason, the readily available planar move plugin was used to allow the development of a navigation system to continue.

To make the simulation visually correct, four effort controllers are used to actuate the wheels. The controllers accept velocity as input and outputs effort. Because the wheels only serve as visualization, the friction between the wheels and the contact surface is set to zero to avoid interference with the planar mover. Although this not an accurate representation of the real plant, the model provides a sufficient simulation model and enables further development of the robot.[60]

Plugins were also used to simulate the sensor data from a depth camera. The parameters were set according to the available data for the physical camera sensor. Parameters include operating range, resolution and refresh rate. [61]

### Gazebo Control Node

To make the simulation visually correct, a node was created to convert the linear and angular velocities of the base into wheel velocities. The node subscribes to the same velocity commands as the planar mover. The information from the velocity commands is used together with the kinematics deduced in Section (5.1) to calculate the velocity of each wheel. Every time a message is publishes to the topic, a callback function is called. Inside the callback function the inverse kinematics are handled and the wheel velocities are published to the respective topic. Each motor has its own topic. This means that the velocity calculations are only processes every time a change in motion is made, which will reduce the computational power. Each controller listens to their own topics and the node publishes each velocity to the appropriate controller. By doing this, the wheels are able to turn freely based on the same velocity commands that are sent to the planar mover.

### 5.3.2 World Model

A section of the university was created according to the available maze map, displayed in Figure (5.8). The maze map does not contain dimensions of the floor plan, however, because some lengths were known, the rest could be scaled proportionately and a sufficient representation could be obtained. The world model was created in the Gazebo model creator and used to simulate the robot model in a representative environment. The developed model is shown in Figure (5.9).

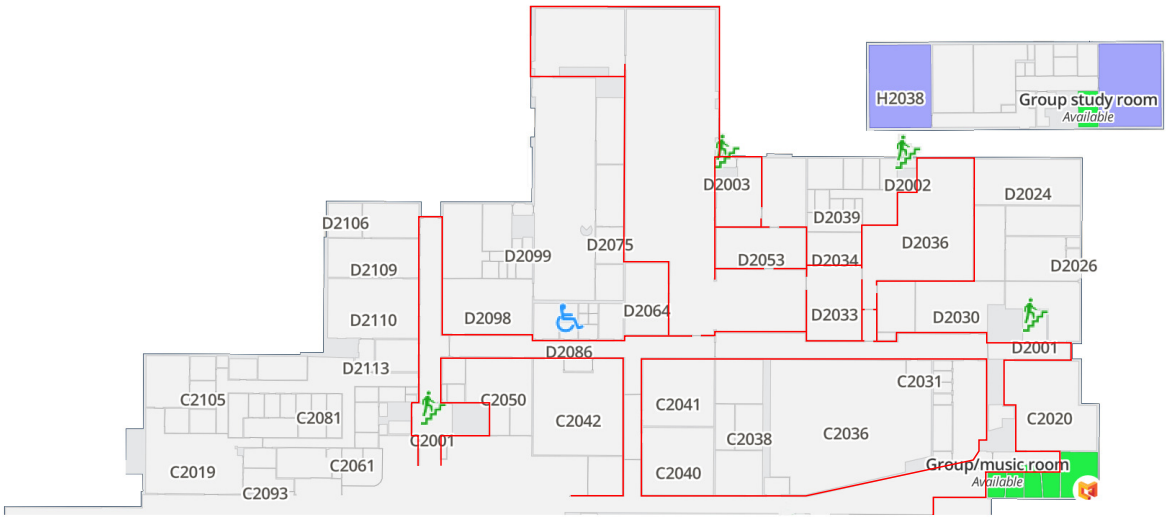



Figure 5.8: Floor plan

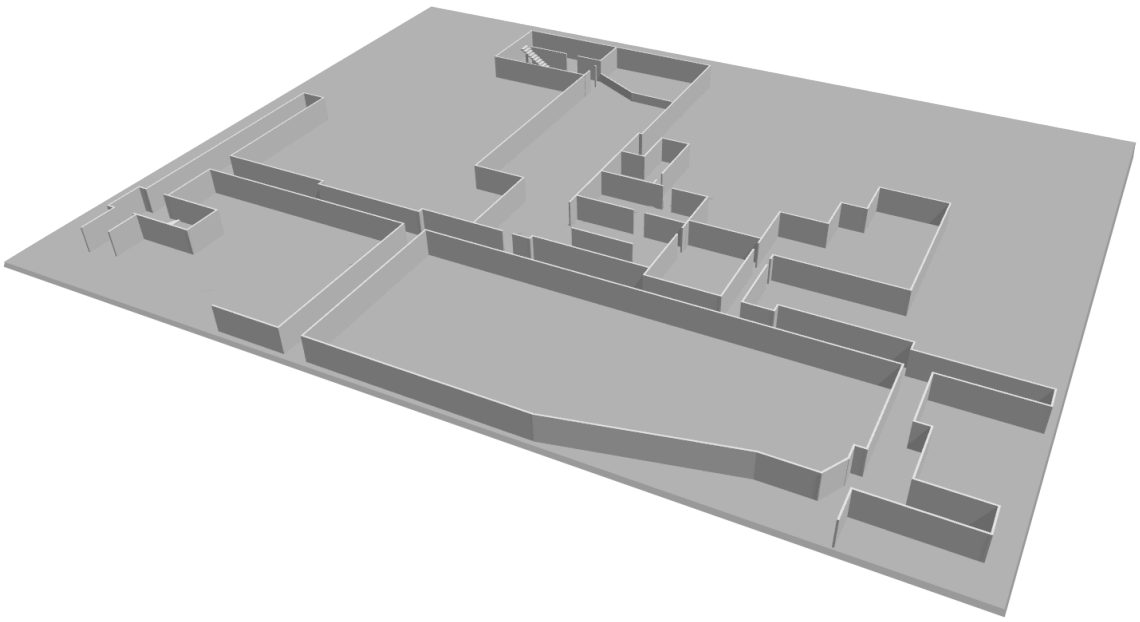


Figure 5.9: World model

# 6. System Architecture

In the system architecture chapter, the communication interface created between the hardware components are covered. This involves communication between the processing unit and the base controller, as well as the communication from the base controller to the motors. The techniques used are UART communication and CAN-bus communication, and both are described. Further, the methods concerning the perception of the environment are presented. This deals with the selection and placement of the perception sensors, as well as the processing of the perceived data and the space transforms in order to relate it correctly.

## 6.1 Hardware Communication

In this section the software used and programs created in order to establish the appropriate communication between the selected hardware components are presented. The developed programs are going to be referenced to individually in order to better isolate what is necessary to achieve specific tasks.

### 6.1.1 CAN BUS Communication

Controller Area Network (CAN) is a serial communication protocol. It was originally invented for the automotive industry by the company Bosch, and has later become quite popular in the automation industry. The main strength of the communication protocol is that it allows communication between multiple electronic control units, ECUs, independently on a single common network, which eliminate the need of complex point to point wiring between all the control units[62]. This makes CAN-bus suitable for the application. Figure (6.1) shows a typical CAN-bus line and how the ECUs, referred to as nodes in the network, are connected. A resistor value of 120 Ω is commonly used, since it is then in accordance with ISO 11898, Road vehicles — Controller area network [63].

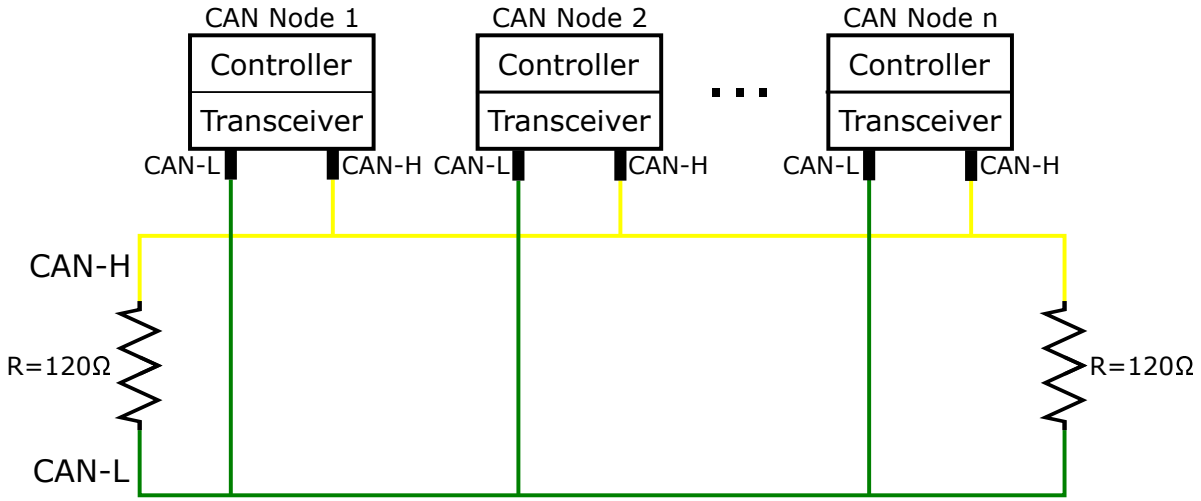



Figure 6.1: CAN Bus network

### CAN messages IDs

The VESC has defined all the addresses containing the data it collect. The only change required to enable CAN-bus, is to activate the communication and give the VESC an ID. This was achieved in the VESC-tool software. In addition, the CAN-bus frequency, baud rate, and status message mode was set. The message mode defines the extent of the information in the sent message, with a number from one to five. The numbering follows Table (6.1). The table was derived from the VESC CAN communication source code [64], which shows the different message objects specified in the statuses.

In order to write and request the correct information, it has to be determined which CAN ID corresponds to which message. In order to determine theses messages, the VESCs were connected to the CAN-network and examined. This was done by utilizing a PCAN-USB adapter connected to a PC through USB [65]. The device listens to all the messages sent over the bus network, and gives information about the message's IDs, the data it contains, and the data length. Figure (6.2) shows the information gathered from PCAN-View with the four VESCs connected to the network. PCAN-View is the additional software used to visually display the information passing in the CAN-bus network. As can be seen from Figure (6.2), each data length contains two hex characters which equals one byte, 8 bits, since one hexadecimal character represents 0-15, 4 bits.

| Status message mode no. | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| ERPM | X | X | X | X | X |
| Current draw | X | X | X | X | X |
| Duty cycle | X | X | X | X | X |
| Amp hours consumed | | X | X | X | X |
| Amp hours charged | | X | X | X | X |
| Watt hours consumed | | | X | X | X |
| Watt hours charged | | | X | X | X |
| Mosfet temperature | | | | X | X |
| Motor temperature | | | | X | X |
| Total current in | | | | X | X |
| Total current in | | | | X | X |
| PID position | | | | X | X |
| Tachometer value | | | | | X |
| Input voltage | | | | | X |

Table 6.1: CAN Status Message mode

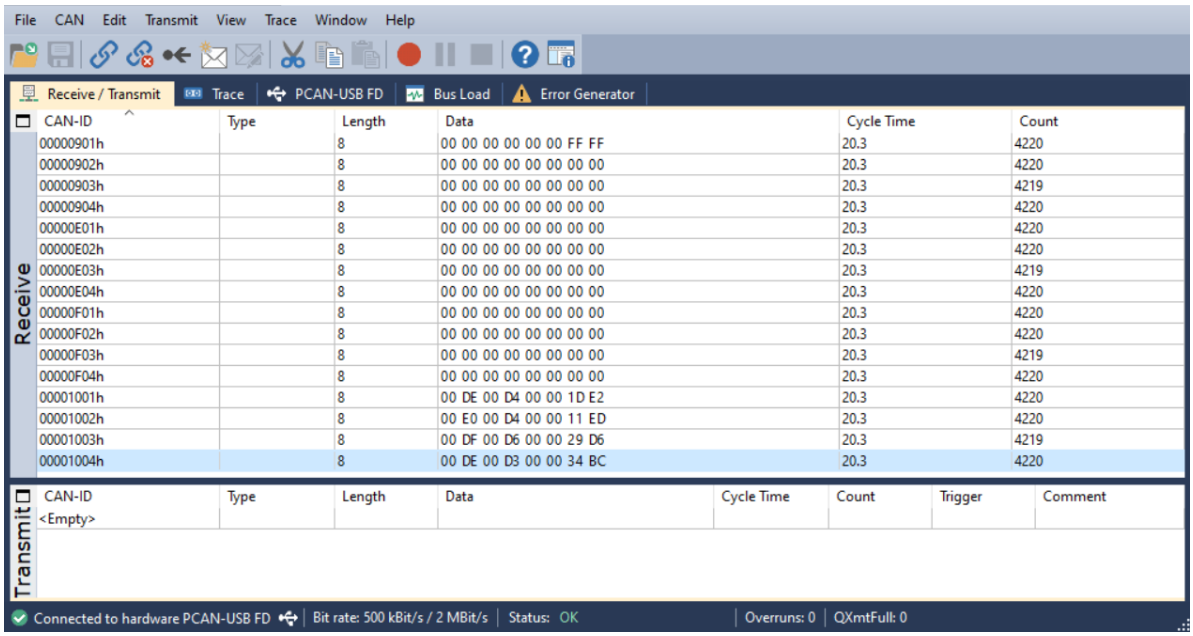



Figure 6.2: PCAN-View

In the VESC datatypes head source code, the CAN commands are defined in an enumerator, where the first command, is set to zero [66]. This means that the consecutive command IDs are one value greater. Then, by utilizing the PCAN-USB the network messages are cross-referenced with the VESC datatypes header source code, the data associated with its ID is deciphered. Here "TA" is the node ID represented by one byte. To specify, this means that if the ID is 1, TA=01. Only the messages considered of importance have their full message structure deciphered. These are the commands that affects the control of the motors and the status packets.

**0x000000TA** is the CAN-ID used in order to set the duty cycle of which the motor runs at, and is sent as a message of 4 bytes. The message is scaled in such a way that the requested duty cycle, in percent, has to be multiplied with 100 000.

**0x000001TA** is the CAN-ID used in order to set the wanted current draw from the VESC to the brushless dc motor. The value the VESC interprets is in mA and is 4 bytes in length.

**0x000002TA** is the CAN-ID used in order to set the wanted breaking current draw from the VESC to the brushless dc motor. The value the VESC interprets is in mA and is 4 bytes in length.

**0x000003TA** is the CAN-ID used in order to set the wanted ERPM, electrical RPM, and is 4 bytes in length. In order to transform RPM into ERPM Equation (6.1) is applied.

$$\text{ERPM} = p\text{RPM} \tag{6.1}$$

| Symbol: | | Description: | Value: | Unit: |
|---|---|---|---|---|
| ERPM | - | Electrical revolution per minute | $\sim$ | $^{rev}/_{min}$ |
| RPM | - | Mechanical revolution per minute | $\sim$ | $^{rev}/_{min}$ |
| $p$ | - | BLDC number of pole pairs | 7 | $-$ |

**0x000004TA** is the CAN-ID used in order to set a shaft position and is 4 bytes in length. The sent request value has to be multiplied with 1 000 000 in order to yield radians.

**0x000009TA** is the CAN-ID which contains the data concerning ERPM, total current consumed and duty cycle, and is the essential odometry information which the VESC measures or estimates. The message is 8 bytes in length, where the different elements of the message are distributed as shown in Table (6.2). In order to reference according to how the data message is handled, the first byte is denoted byte 0.

| Byte 0 | Byte 1 | Byte 2 | Byte 3 | Byte 4 | Byte 5 | Byte 6 | Byte 7 |
|---|---|---|---|---|---|---|---|
| ERPM | | | | Current | | Duty cycle | |

Table 6.2: ERPM, current and duty cycle message structure

### 6.1.2 Writing and Reading CAN Messages

The selected microcontroller, Teensy 3.6, supports dual CAN-controllers. In order to handle the CAN-bus protocol on the Teensy, the FLEXCAN library was used [67]. It is a serial communication drive for the CAN peripheral built into the Teensy CPUs. Most importantly, it is a library that is compatible with the used dual CAN-adapter.

Figure (6.3) shows how the CAN-network is wired. The second CAN-line, CAN1 was used. The CAN1 $R_s$ pin is connected to the Teensy ground (LOW) in order to select high speed mode. To select the low power mode, the $R_s$ pin has to be set high.

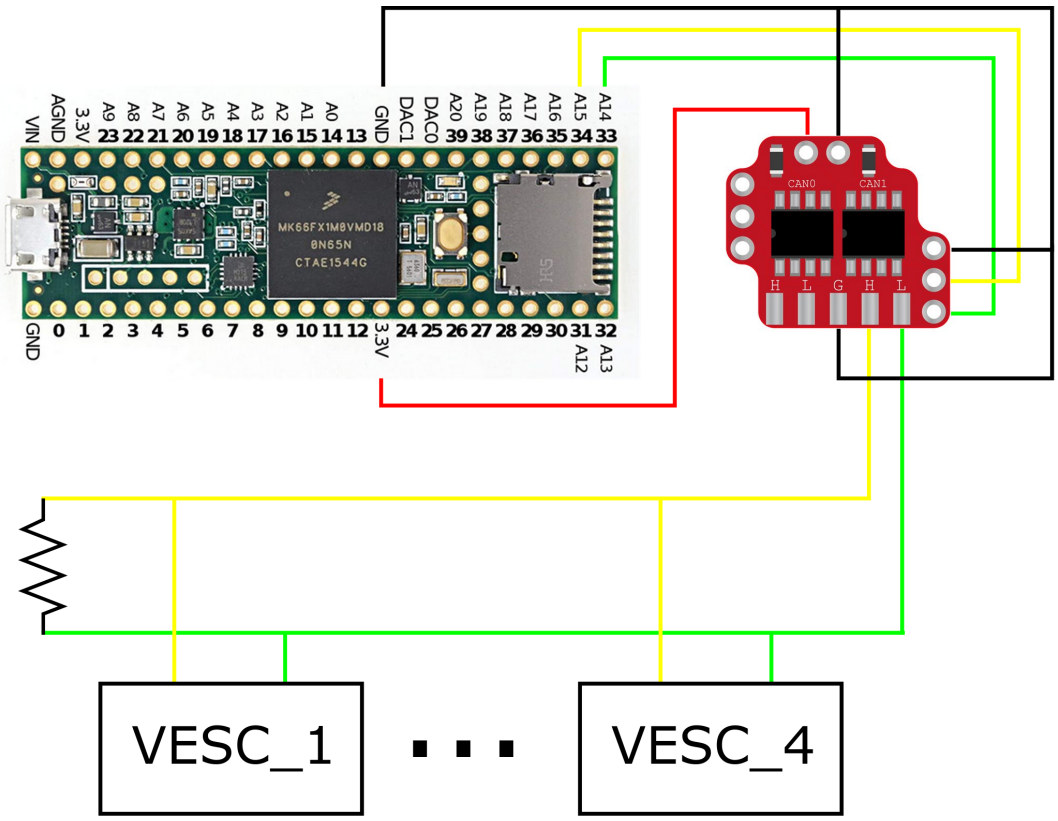



Figure 6.3: CAN components connection

### Writing Messages

The FLEXCAN library was included in order to handle the CAN-bus protocol. The microcontroller has two CAN-controllers and in order to select which to initialize, it has to be defined in the begin-function included in the FLEXCAN library. Further, the pins which receive and transmit data from the CAN-adapter has to be specified. The pins are set to 34 and 33 respectively. Additionally, the function takes two more arguments. The first concern the baud rate, which is set to 250 000bps. This has to correspond with the value set in the VESC-tool. The last argument is the filter mask, which is set to allow through all traffic.

Appendix (A.8) shows how the FLEXCAN library is utilized in order to write messages to the CAN-network. Additional functions that the library provides are also used, such as the write message function. The write function is displayed in Listing (6.1). To select which CAN-controller the message is sent through, the $i$ in Can$i$ is set to either 0 or 1. The function only requires one argument, which is the message to be sent.

```
Cani.write(message)
```

Listing 6.1: CAN begin function

In order to effectively add new messages, the function shown in Listing (6.2) was created. The function requires three arguments. The first argument is the desired name of the CAN-message, where the name of the message is arbitrary. The second argument is the ID of the message to be written. Here both the message command ID, and the ID of the unit

is specified. Lastly, the value to be sent is specified. The function allows to give the third argument, value, in decimals. This is made possible by the use of pointers, which is a variable that holds the memory address. Essentially, this brings the value to a level where its number type does not matter [68].

```
write_can_message(CAN_message_t msg, uint32_t can_id, int32_t value)
```

Listing 6.2: New message function

#### Reading Messages

In order to read messages from the CAN-network the read-function shown in Listing (6.3) is called. To select which of the controllers that reads, $i$ is either set to 0 or 1.

```
Cani.read(inMessage)
```

Listing 6.3: Read network messages

The read function will read all of the message traffic on the network. Thus, in order to be able to get the information from a specific message ID, such as the wheel velocity, multiple functions were created. How these are used can be found in Appendix (A.9). All of the functions used to read are created identically with the exception of the function used to read the ERPM. The difference is that this message consist of 4 bytes instead of 2. A representation of the read functions is displayed in Listing (6.4). The functions has two arguments. The first is the ID of only the targeted unit, and does not contain the command ID in addition. This is done since the command ID is left shifted in order to merge it into a completed CAN-ID. This allows re-use of a function created for a specific messages for multiple units. The second argument requires a variable which is used to store the read value in order to avoid clearing data if the message are read at an unfavorable frequency.

```
read_specific_message(uint16_t can_node_id, int stored_read_value)
```

Listing 6.4: Read speciffic network message

### 6.1.3 UART Communication

UART stands for Universal Asynchronous Receiver/Transmitter and is commonly referred to as serial communication. The UART is a physical circuit in a microctonroller, or a stand-alone Integrated Circuit (IC). This means that the UART itself is not a communication protocol [69]. The Jetson Xavier needs to communicates with the Teensy, and because UART is simple to implement, it was chosen as the communication method.

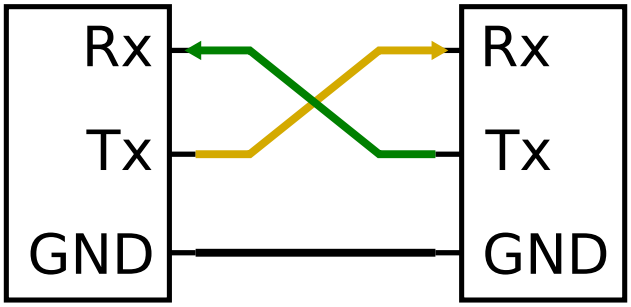



Figure 6.4: UART devices connection

A UART mainly consists of three parts; the controller unit, transmitter and receiver. In order to transmit data between two UART devices, two wires are required. Each device has a transmitting port known as Tx and a receiving port known as Rx, where the appropriate ports are cross-connected as shown in Figure (6.4). In addition, the figure shows the connection of a common ground. Since the Tx and Rx pins measure the voltage difference between the respective pin and ground, it is crucial to have a common zero volt reference in order to avoid loss of data or receiving corrupt data.

### UART Between Jetson Xavier and Teensy

Robot Operating System is going to be used for the navigation and localization of the robot. Thus, it is decided to use the ROS-package *rosserial*. It is a protocol for wrapping standard ROS serialized messages, and allows for multiplexing of multiple topics and services over a character device such as a serial port or network socket [70]. This allows to use the Teensy as a node in a ROS-network, which gives the microcontroller the possibility to publish data to topics and subscribing to other topics. If multiple microcontrollers are to be used as nodes, they all have to be able to communicate directly with the device running the master node.

The Teensy can achieve the serial communication by connecting via USB. However, the hardware serial (UART) was used to enable the possibility of compiling new code through the USB connection. By using the hardware serial, the main USB host stays available whilst communicating with the Jetson Xavier, allowing external sources to update the code on the Teensy during operation. By enabling the hardware serial on the Teensy the GPIO expansion header pins on the Jetson Xavier development kit carrier board has to be used. The appropriate UART pins to use are found from the pinout schematic in Appendix (C.1) [71]. Then, by applying the Teensy pin assignments from Figure (4.5), the correct wiring is deduced. The connection between the Xavier and Teensy is shown in Figure (6.5).

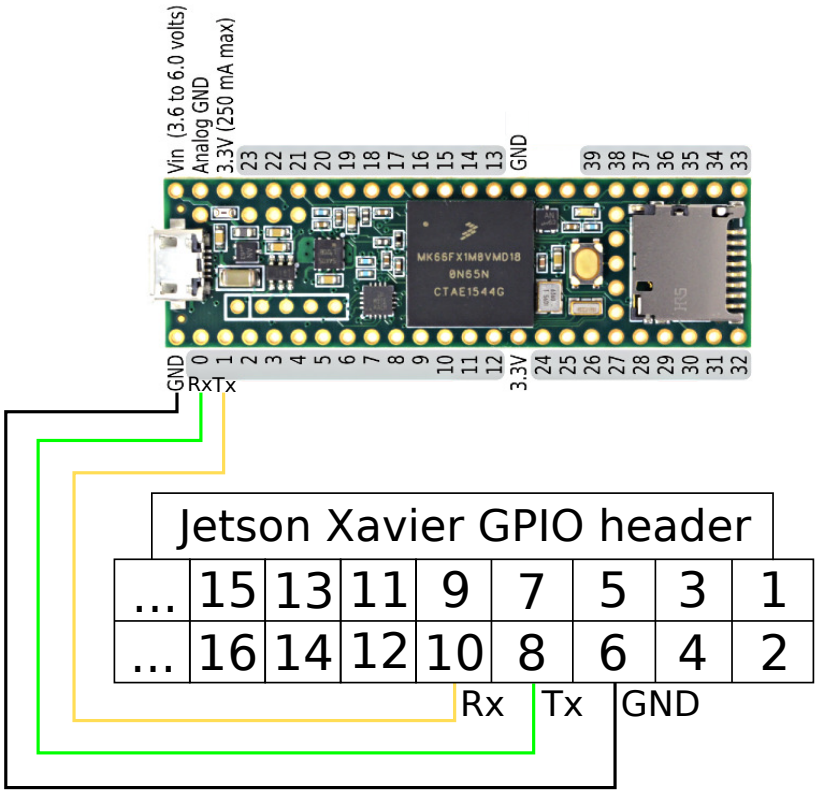



Figure 6.5: Teensy to Xavier UART wiring

In the *rosserial* node, which is establishing the communication, the two most essential parameters that can be specified are the baud rate and the bus name. The baud rate must be specified at the rate that the devices operates at. If they are not synchronized, the communication will brake. Figure (6.6) shows the bus names assigned to all of the UARTs on the Jetson Xavier. Since it is the UART_1{TX,RX,RTS,CTS}, or UARTA, which the Teensy is connected to, the bus name which has to be specified is; /dev/ttyTHS0

| UART Instance | Base Address | Ball Name | Pin on 699 Connector |
|---|---|---|---|
| UARTA | 0x03100000 | UART1_{TX,RX,RTS,CTS} | UART1 (K53,K54,L51,H54) |
| UARTB | 0x03110000 | UART2_{TX,RX,RTS,CTS} | UART2 (C58,C56,G58,A57) |
| UARTC | 0x0c280000 | UART3_{TX,RX} | UART3 (H62,K60) |
| UARTD | 0x03130000 | UART4_{TX,RX,RTS,CTS} | UART4 (L5,L48,L4,L49) |
| UARTE | 0x03140000 | UART5_{TX,RX,RTS,CTS} | UART5 (J58,H58,K58,H57) |
| UARTF | 0x03150000 | DP_AUX_CH0_{P,N} | DP0_AUX_CH (F52,F51) |
| UARTG | 0x0c290000 | SPI2_{SCK,MISO,MOSI,CS0} | SPI2 (E61,D62,F60,D60) |
| UARTH | 0x03170000 | USB1_{DN,DP} or USB0_{DN,DP} | USB1(C10,C11) or USB0(F13,F12) |

| Usage on Jetson AGX Xavier | Pin Type | Bus Name | DTS node | DTS status | DTS alias |
|---|---|---|---|---|---|
| Use for UART connector 40 pin connector | CMOS – 1.8V | ttyTHS0 | serial@3100000 | OK | serial0 |
| UART debug chip FT4232 | CMOS – 1.8V | ttyTHS1 | serial@3110000 | OK | serial1 |
| UART debug chip FT4232 | CMOS – 1.8V | ttyTHS2 | serial@c280000 | Disabled | serial2 |
| Use for normal GPIO for CAM | CMOS – 1.8V | ttyTHS3 | serial@3130000 | Disabled | serial3 |
| Use for the M.2 KEY E connector | CMOS – 1.8V | ttyTHS4 | serial@3140000 | OK | serial4 |
| Use for Display chip | CMOS – 1.8V if use to UART | ttyTHS5 | serial@3150000 | Disabled | serial5 |
| Use for PCIe x16 connector | CMOS – 1.8V if use to UART | ttyTHS6 | serial@c290000 | Disabled | serial6 |
| Use for USB connector | CMOS – 1.8V if use to UART | ttyTHS7 | serial@3170000 | Disabled | serial7 |

Figure 6.6: Bus name assignment [11]

Typically the permission for accessing the exposed serial ports is denied, resulting in no way of establishing the connection or transferring data between the units. This can however be changed by utilizing the CHMOD command from the terminal on the Xavier. Figure (6.7) shows how the specific changes are made [72][73]. The command can be used with alphabetical or numerical arguments. The difference between them is that the numerical changes persists through a reboot of the system whilst alphabetical does not. To use the alphabetical arguments, the desired target is specified. Then, with any combination of the permissions letters wanted. In order to use the numerical argument, the sum of the wanted permissions are added into the correct target. Since it is enough to allow read and write permission to the group members, in this case the Teensy, the permission is set as in Listing (6.5).

```
sudo chmod g+rw /dev/ttyTHS0 // Alphabetical
sudo chmod 060 /dev/ttyTHS0 // Numerical
```

Listing 6.5: Permission for communication

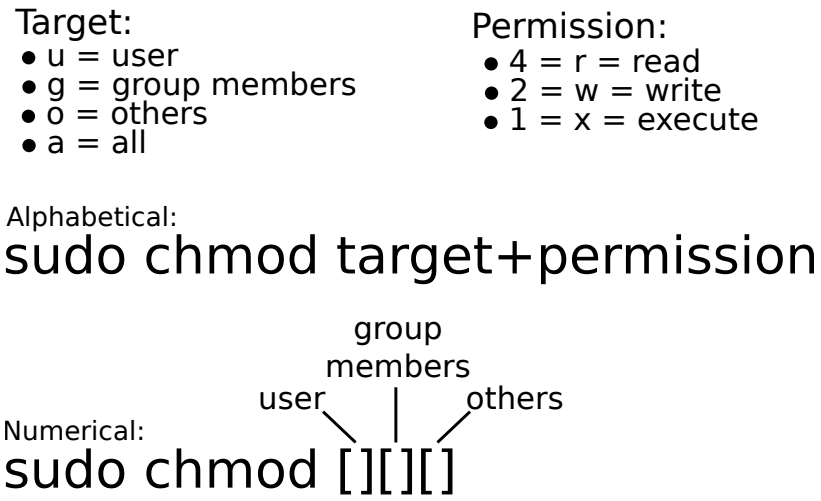



Figure 6.7: Use of CHMOD

### 6.1.4 Publishing and Subscribing over Teensy

In order to enable the hardware serial on the Teensy, it has to specified. This is done by adding the line in Listing (6.6). The line has to to be added before the ROS header is included in the Teensy program.

```
#define USE_TEENSY_HW_SERIAL
```

Listing 6.6: Enabling Hardware Serial

Then, as Figure (4.5) shows, the Teensy has six hardware serials [74]. In order to specify the one that has been wired to the Xavier, a class defining it was created. This was done as shown in Listing (6.7). Serial1 is used because it is linked to the hardware serial in use.

```
class NewHardware : public ArduinoHardware{
   public:
      // Specifying the hardware serial port and baudrate
      NewHardware():ArduinoHardware(&Serial1, 57600){};
}
```

Listing 6.7: Serial class

With the necessary perquisite, the subscribing and publishing nodes were created separately. Theses nodes only handle the information about the wheel velocities in ROS messages, and can be respectively found in Appendix (A.4) and Appendix (A.3). The message handling is done similar to any other node created with C++. The publishing node publishes the measured wheel velocities in the topic named wheel_omega_all. Since the appended buffer is of the type 32 bit integer, and it is beneficial to send all the wheel velocities in a single message, the standard ROS message Int32MultiArray was selected. The Int32MultiArray message allows to publish an array of data [75]. The same message structure is selected for the subscribing node, which subscribes to the omega_ref topic.

After the subscribing and publishing nodes were tested individually, they were merged together into a single node which both subscribes and publishes. The combination node can be found in Appendix (A.2). Figure (6.8) shows the data being transferred from Xavier to the CAN-network through the Teensy, and back.

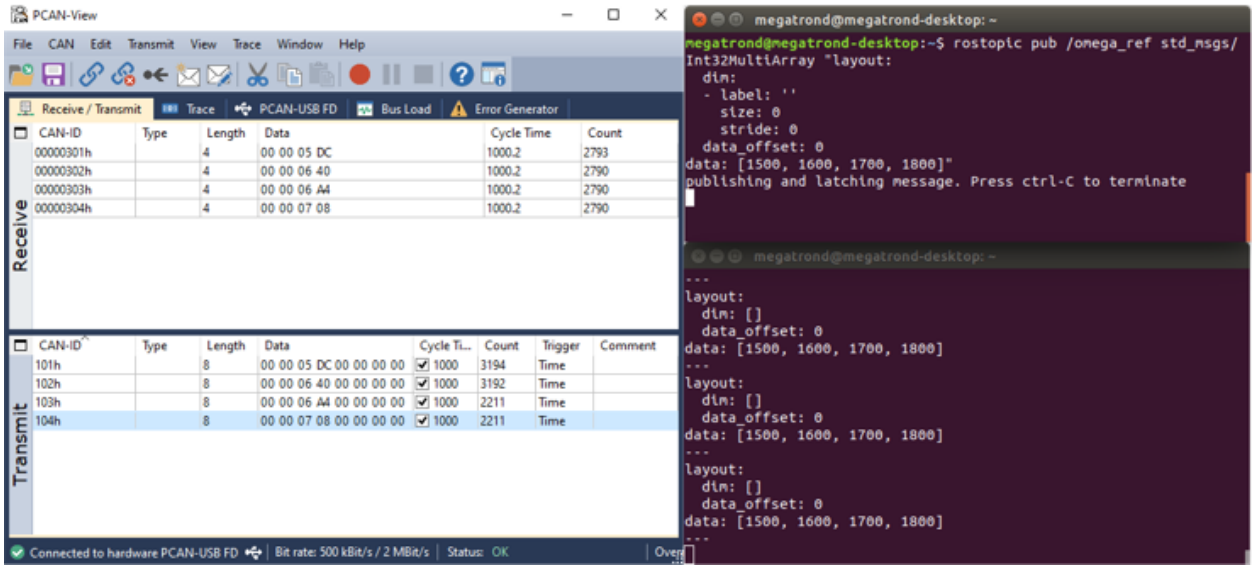



Figure 6.8: Two way data transfer

## 6.2 Manual Control

Before developing the automation application of the Loomo rig, a low level method of controlling the vehicle was created. It enables easy control of the rig, but the controller also served as a tool for verify the inverse kinematic equations from Section (5.1).
In order to control the Loomo rig manually, a hand controller is created. It consists of two joysticks.

The manual program uses the analog signal from the potentiometers that the Teensy reads. The analog value span is $[0-1023]$, and this is mapped from $[(-4)-4]\,{}^{m}/_{s}$. This value was found to be satisfactory from initial tests. Next, the reference velocity is applied to calculate the individual wheel velocities required by applying the inverse kinematic model derived in Section (5.1). As a last step, before the reference value is sent over the CAN-network, the value is converted from raidans per seconds into ERPM by Equation (6.2), since this is unit the VESC interprets it as. The hand controller program can be found in Appendix (A.7).

Figure (6.9) shows how the joysticks are used in order to change the velocity reference, resulting in the change in pose of the Loomo rig. The Loomo rig is viewed from top down in Figure (6.9).

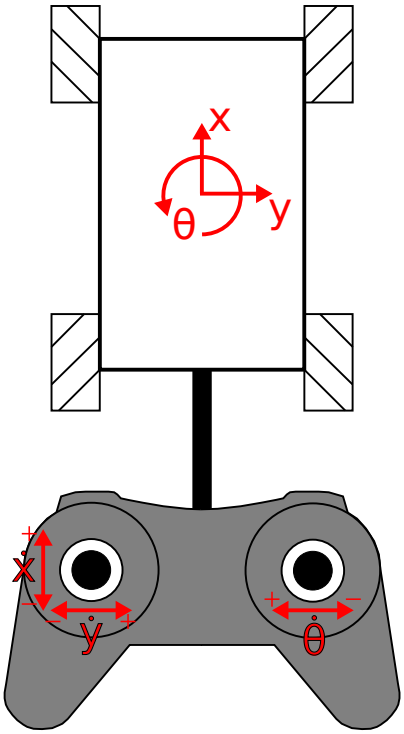



Figure 6.9: Hand controller guide

$$ERPM = \frac{60p\omega}{2\pi} \tag{6.2}$$

where:

| Symbol: | | Description: | Value: | Unit: |
|---|---|---|---|---|
| $ERPM$ | - | Electrical revolutions per minute | $\sim$ | $^{rev}/_{min}$ |
| $p$ | - | Number of BLDC poles | 7 | $-$ |
| $\omega$ | - | Wheel velocity | $\sim$ | $^{rad}/_{s}$ |

## 6.3 Visual Perception

Computer vision is a rapidly growing technology that is being increasingly implemented in many modern applications, such as process control and robotics. Perception is the process by which the robot uses its sensors to obtain information about the state of its environment, also referred to as a measurement or observation. This project looks at the possibility of using multiple cameras for obtaining a 360 degree visual perception of the environment. An alternative would be to use a rotational LiDAR to scan the surroundings. Opposed to a depth camera, the range scanner can only scan a single 2D plane. This prevents the sensor from detecting obstacles located below or above the scanning plane. As a result, a robot based solely on a 2D laser scanner can not ensure a collision-free navigation. Additionally, due to the size and geometry of the robot, it would be hard to place a single rotating LiDAR at a location which allow 360 degree perception. A depth camera on the other hand, can create a 3D representation of the environment and will be able to detect obstacles in all heights, if positioned correctly. A depth sensor also provides more information, thus it demands more post processing, which again means it is more computational expensive. Due to its versatility, the information from a camera can replace one or more sensor units and can be used for various purposes, for instance, localization, mapping, obstacle-avoidance, and object recognition. Depth camera scores high, compared to LiDAR, due to 3D reconstruction. However, the depth camera also has drawbacks, such as typically having a significantly lower range and the field of view is generally narrow.

Two different depth sensors were available at the university, Microsoft's Azure Kinect and Intel's Realsense D435i. Table (6.3) shows a comparison between the two hardware specifications.

| | Azure Kinect | Realsense D435i |
|---|---|---|
| | Depth camera | |
| Resolution | $512 \times 512$ | $1280 \times 720$ |
| Field of view | $120° \times 120°$ | $87° \times 58°$ |
| Max frame rate | 30 $Hz$ | 90 $Hz$ |
| Operating range | $0.25 - 2.21$ $m$ | $0.1 - 10$ $m$ |
| | Color camera | |
| Resolution | $1280 \times 720$ | $1920 \times 1080$ |
| Field of view | $90° \times 59°$ | $69.4° \times 42.5°$ |
| Max frame rate | 30 $Hz$ | 30 $Hz$ |
| | Additional Information | |
| Physical size | $125 \times 103 \times 39$ $mm$ | $90 \times 25 \times 25$ $mm$ |
| Embedded IMU | Yes | Yes |

Table 6.3: Sensor comparison

### 6.3.1 Camera Placements

In order to obtain a 360° perception with the least amount of depth cameras, they must be placed in a specific manner. Due to the length of the robot, the most crucial factor for this project is the field of view. To evaluate the field of view at different positions on the robot, the projection pyramids for each camera were visualized and compared in SOLIDWORKS. All images illustrate the field of view at a range of 1 m from the cameras.

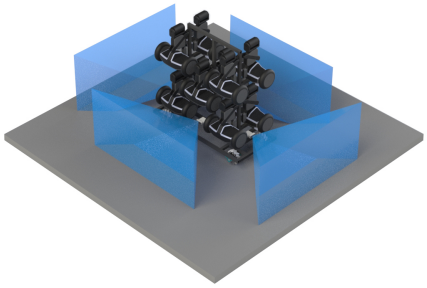


Figure 6.10: Realsense Parallel to Ground

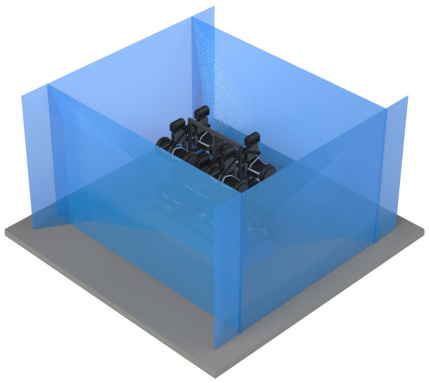


Figure 6.11: Kinect Parallel to Ground

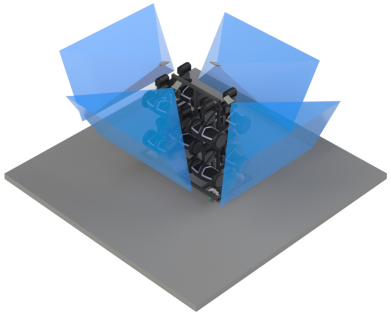


Figure 6.12: Realsense Tilted Downwards

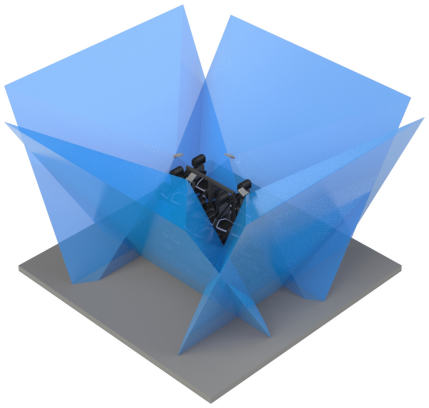


Figure 6.13: Kinect Tilted Downwards

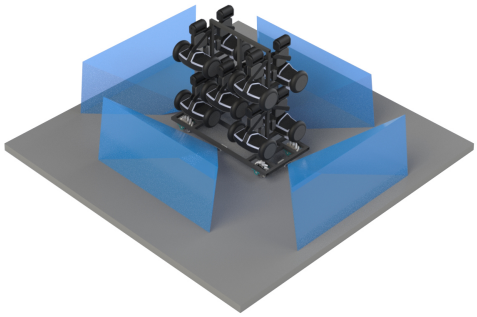


Figure 6.14: Realsense Titled Upwards

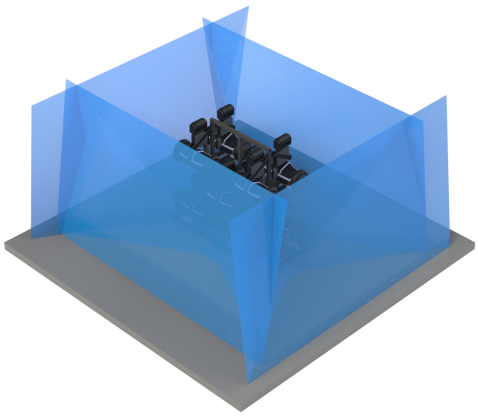


Figure 6.15: Kinect Titled Upwards

For the first position, the cameras are placed approximately 350 mm above the floor and the orientation is parallel with the ground. The result can be seen in Figure (6.10) and Figure (6.11). The next position is on top of the robot, with the camera tilted downwards, as can be seen in Figure (6.12) and Figure (6.13). For the last position, the camera is attached to the underside of the frame, and the camera is tilted upwards. Figure (6.14) and Figure (6.15) illustrate this configuration.

From the images above, it is clear to see that the field of view from the Realsense is not sufficient and that the Kinect is superior. Contrarily, the Kinect does not match the range properties of the Realsense, but the field of view is considered as the most important factor. For this reason the Azure Kinect depth camera is chosen. The position in (6.11) was selected because it enables the camera to be parallel to the ground, which means that the tilt angle will not cause distortions and does not have to be compensated for.

### 6.3.2 Azure Kinect Depth Camera

For this project, four Azure Kinects were chosen as the perception sensors. The Azure Kinect is a multisensor where the two main components are a depth camera and a RGB camera. An overview of the sensor is illustrated in Figure (6.16).

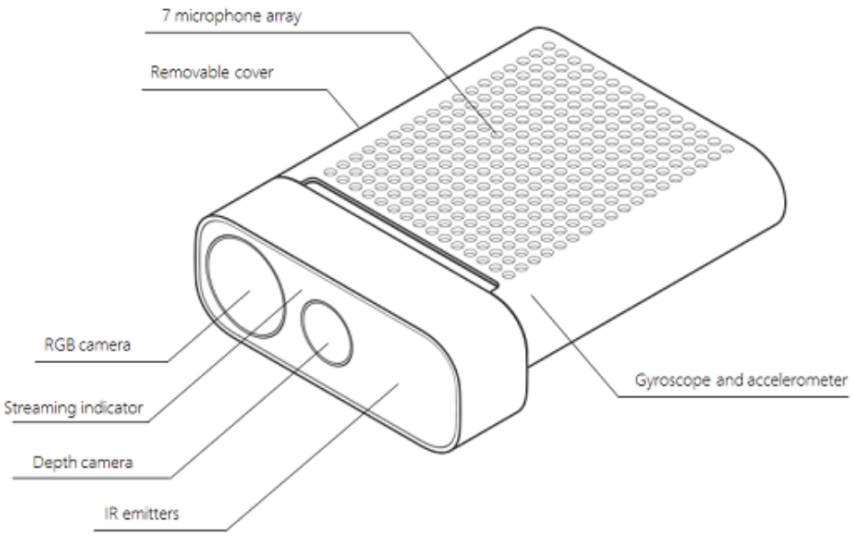



Figure 6.16: Azure Kinect [12]

The depth feature of the sensor consist of a IR pattern projector and a Time-of-Flight image sensor. The depth is determined by measuring the time emitted light takes from the camera to the object and back. This is done by calculating the phase shift between the modulated wave of the emitted infrared light and the returning light. Further on, this is used to calculate the depth at each pixel of the RGB camera. These sensors combined make the Kinect a well-suited component for localization and object detection. The Kinect is also equipped with an 6DOF Inertial Measurement Unit (IMU), which features an accelerometer and a gyroscope. The Azure Kinect is calibrated at the factory and the calibration parameters for visual and inertial sensors may be accessed through the Sensor SDK. [12]

### 6.3.3 Depth Image to Laser Scan

A Kinect may produce more than 9 millions points a second, which makes it very computational heavy with regard to processing and memory [12]. Fortunately, a full image is not necessary for representing the surrounding environment. A 2D image is sufficient and can be created from only the important parts of a 3D image. *laserscan_kinect* is a package available in ROS which is used to transform spatial data from a Kinect to a 2D plane. The input to the package is a depth image with $n$ rows and $m$ columns where the image center is located in $(c_x, c_y)$. The output is a scan array containing the distances for consecutive scanning angles. A prerequisite for using this package is that a transform configuration is publishing the correct transformation between the base frame and the sensor frame. A sensor unit must also make a depth image available by publishing it to a topic. [13]

The conversion algorithm utilized in *laserscan_kinect* assumes that the important information for collision avoidance belong to the location of the closest obstacle. For this reason, the lowest distance values are extracted from each column of the depth image. Equation (6.3) describes this where $z_{j,i}$ denotes a distance, $i$ argue the column and $j$ denotes the row in the image. [13]

$$z_i = min(z_{0,i}, z_{1,i}, ..., z_{j,i}) \tag{6.3}$$

Once the closest point in each column is detected, the distance relative to the optical center can be determined by simple geometry. The package is based on the pinhole model as described in Figure (6.17), and Equation (6.4)-(6.6). [13]

$$\delta = \theta \frac{j_{min} - c_y - \frac{1}{2}}{n - 1} \tag{6.4}$$

$$d = l \cdot \sin\left(\frac{\pi}{2} - \alpha - \delta\right) \tag{6.5}$$

$$= z \cdot \frac{\sin(\frac{\pi}{2} - \alpha - \delta)}{\sin(\frac{\pi}{2} - \delta)} \tag{6.6}$$

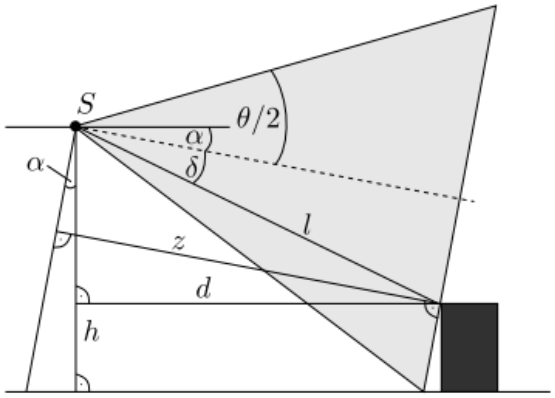



Figure 6.17: Pinhole model [13]

where:

| Symbol: | | Description: | Value: | Unit: |
|---|---|---|---|---|
| $\delta$ | - | Angle to the obstacle in vertical plane | $\sim$ | $rad$ |
| $\theta$ | - | Field of view | $\sim$ | $rad$ |
| $j_{min}$ | - | The row location of the depth image | $\sim$ | $pixels$ |
| $c_y$ | - | Camera offset in vertical direction | $\sim$ | $pixels$ |
| $n$ | - | Number of pixels in vertical direction | $\sim$ | $pixels$ |
| $d$ | - | Rectified normal distance to closest object | $\sim$ | $m$ |
| $l$ | - | Distance from camera to closest object | $\sim$ | $m$ |
| $z$ | - | Normal distance from camera to object | $\sim$ | $m$ |
| $\alpha$ | - | Camera tilt angle | $\sim$ | $rad$ |

Considering that the algorithm is based on finding the nearest obstacle in the image, it is necessary to remove the floor to prevent the ground from appearing as the closest obstacle. To do this, the package uses a ground removal algorithm. If the tilt angle and the height of the sensor relative to ground is known, it is possible to determining which pixel is occupied by the ground. In this algorithm, the ground is determined based on a tolerance $\varepsilon_g$. The ground is removed based on Equation (6.7) and Figure (6.18). If the value is categorized as ground, the range value of that pixel is set to infinite, and will not be included in the converted scan. [13]

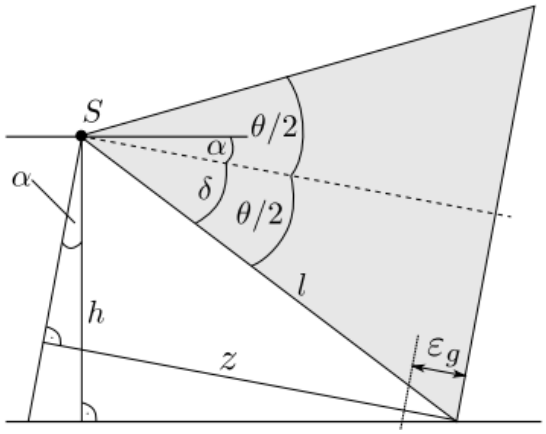



Figure 6.18: Ground removal [13]

$$P_g = \begin{cases} z = h \cdot \frac{\sin(\frac{\pi}{2}-\delta)}{\cos(\frac{\pi}{2}-\delta-\alpha)} - \varepsilon_g & if \;\; 0 \leq z_t^k < z_{max} \\ 0 & else \end{cases} \tag{6.7}$$

Figure (6.19) shows the depth image generated by the Kinect from the environment depicted in Figure (6.20). The resulting laser scan converted from the depth image is shown in Figure (6.21). This was done for all four Kinects which creates four individual laser scans.

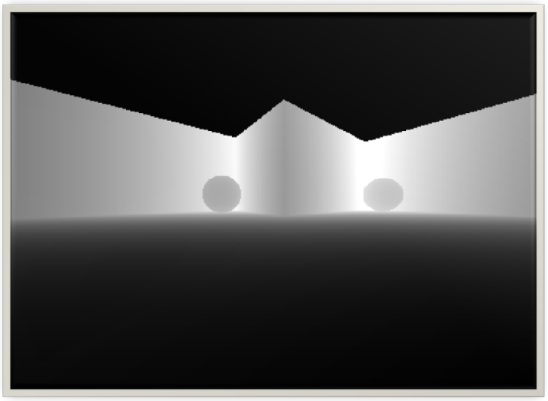


Figure 6.19: Depth image

Figure (6.20) displays the Loomo rig in the GAZEBO environment. The spheres and wall in front of it represents the same field of view as the front camera perceiving the wall.

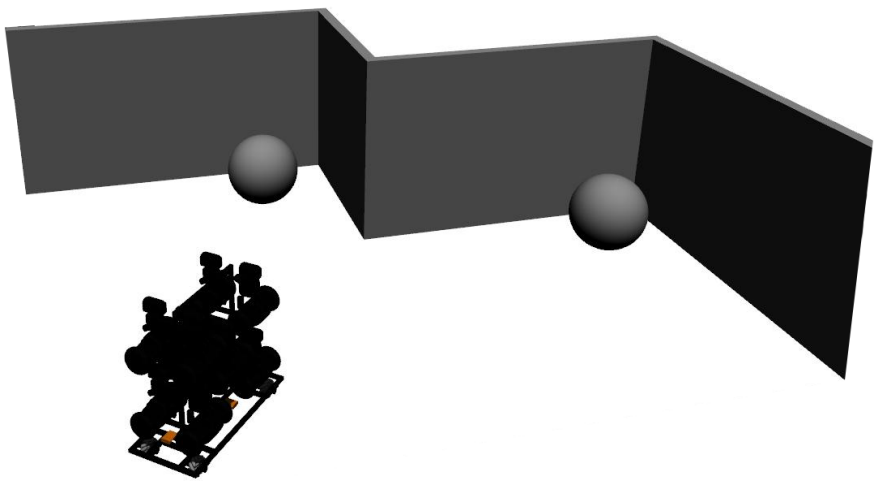


Figure 6.20: Environment in GAZEBO

Figure (6.21) visualizes the resulting laser scan from the depth camera in RVIZ as the red line. The particles is the point cloud perceived by the depth camera located at the front.

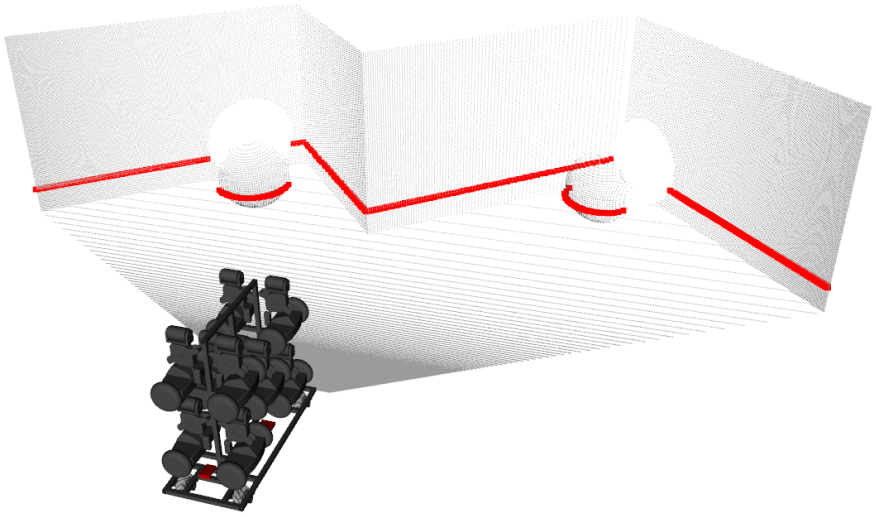


Figure 6.21: Laser scan visualized in RVIZ

### 6.3.4 Merging Laser Scans

To generate a single scan message from the four separate Kinects, the *ira_laser_merger* package was utilized. The package takes multiple laser scans as input and transforms them all to a common frame. The output of the package is a merged scan which will appear as it was generated from a single scanner. Figure (6.22) shows the test environment in GAZEBO and Figure (6.23) shows the resulting merged scan visualized in RVIZ. As can be seen, the closest points of the spheres are detected by the depth cameras as a merged laserscan.

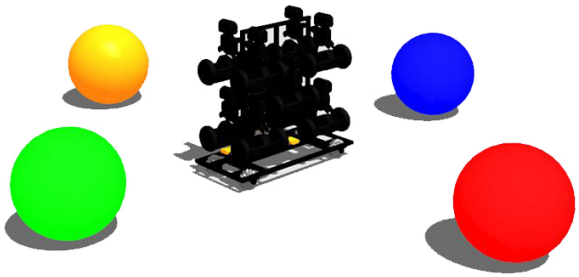


Figure 6.22: Test environment in GAZEBO

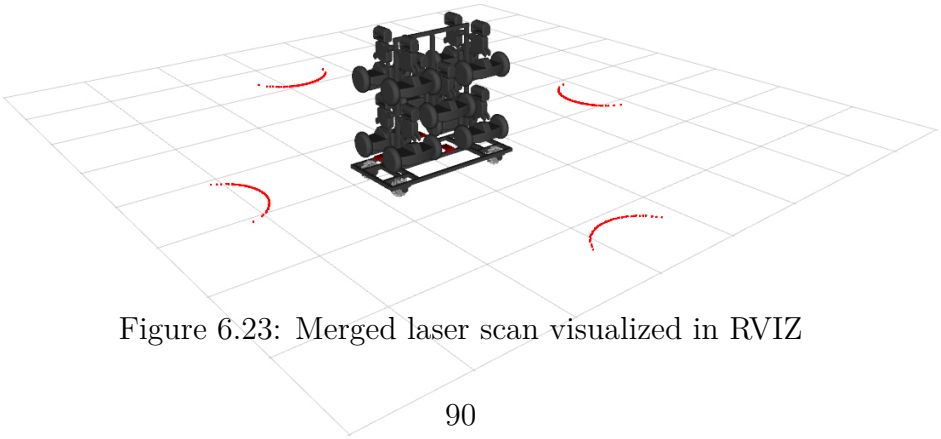


Figure 6.23: Merged laser scan visualized in RVIZ

## 6.4 Transform Configuration

The robot consist of several bodies with different location and orientations of their coordinate frames. The tf package is responsible for calculating the frames location relative to each other and the changes between them over time. tf is designed in a tree structure with parents and children, the tf tree for this robot is available in Appendix (F.6). The transformation between map and odom is dynamic. It constantly computes the robot's pose based on information from the localization component and is used as a long-term global reference. In this project, the localisation source is the combined data from the four depth cameras. Further on, the transformation between odom and the base link is also dynamic. It computes the robot's pose based on information collected from an odometry source. Due to cumulative errors of the internal sensor, the true position will drift over time. Because of this, the odometry source is used as an accurate short-term reference, and occasionally updated by information from the localization source. The odometry model for this project is based on fusion between an IMU and wheel encoders. Lastly, the transformation from the base link to the wheels, as well as the transformation between base link and the four cameras are static because the distance between them is constant. All the static transformations are gathered from the URDF file. Figure (6.24) shows the static frames and how they are linked to the base frame. [76]

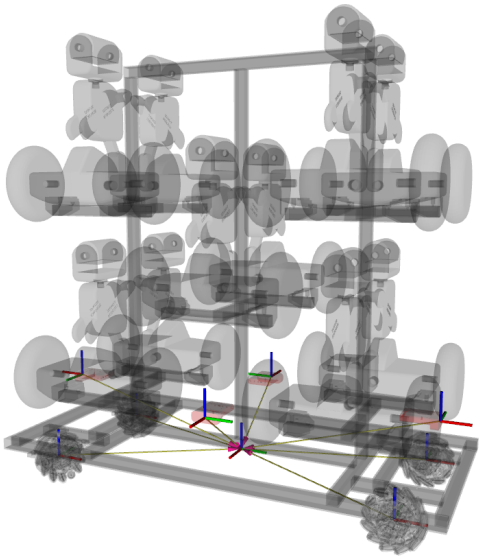


Figure 6.24: Transformation between links

# 7. Localization and Navigation

The localization and navigation chapter aims to cover the features used to achieve the overall functionality of the navigation system. Two navigation modes are presented. ArUco tracking and following, as well as a semi-autonomous navigation. For the latter mode, map generation, solving the localization problem and path planning is documented. Lastly, a method of assistive actuation by the use of hand gestures is presented.

## 7.1 Probabilistic Localization

Localization is known as the problem of determining the pose of a robot in an environment. Autonomous roaming of a robot is possible if the location of the robot and surrounding obstacles are known. However, these variations are not directly measurable, and the robot relies on sensor data as well as advanced data processing to estimate its position. When considering the global localization for this project, the robot does not know its initial pose and needs to be given one by the operator. Once the robot knows its starting position, the position tracking problem is initiated. This is the robot's ability to track its motion and maintain an estimate of its pose.

### 7.1.1 Occupancy Grid

The rig is intended to be used in a known environment, the University of Agder campus Grimstad, hence it was possible to generate a premade map of the static objects in the operating area. For this project, the actual operating area was not available, therefore the concept was developed and verified in a simulation model. The map will be used for the localization problem to complete the autonomous application. Figure (7.1) shows the map used in this project.

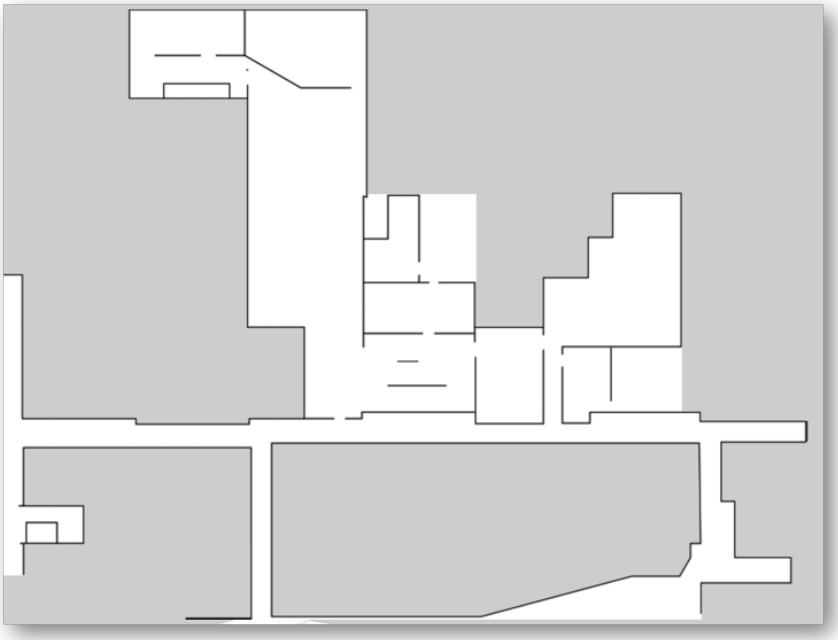


Figure 7.1: Map of UiA section

The map in Figure (7.1) is characterized as an occupancy grid. An occupancy grid represents the environment as an equally spaced grid where the color of the cell determines the state of the pixel. If the cell is white, it is perceived as free. If the cell is black, it is interpreted as occupied, whereas other colors represent the unknown territory. [77]

A dedicated node is used to provide map data as a ROS service. The package that contains the node is called *map_server* and is a part of the navigation stack. *map_server* stores the generated map and makes it available for other nodes. This node accepts an image as an input and converts it into an occupancy grid which is published as a *nav_msgs/OccupancyGrid* message [78].

### 7.1.2 Adaptive Monte Carlo localization

Adaptive Monte Carlo localization (AMCL) is a probabilistic localization system for mobile robots moving in a 2D environment. The algorithm uses a particle filter to track a robot's pose in a known map. A particle filter is a probabilistic filter that is used to estimate internal states in a dynamic system where the observations are influenced by random perturbations. Particle filters are non-parametric, and can represent a broad selection of distributions. The filter is a form of bayes filter, which means it is recursive and uses the belief at the previous time step $x_{t-1}$ to calculate the current belief $x_t$. A control $u_t$ is necessary to induce a transition from $x_{t-1}$ to $x_t$. The filter is based on representing the belief as a set of particles (samples) to determine the posterior distribution. The number of particles is often large (e.g 1000) and each particle is a hypothesis as to what the true state may be. When using a particle filter for solving the localization problem, each particle is a pose hypothesis of the robot state represented by x, y, and theta. Dynamic robot environments are stochastic and can not be described using a deterministic function, but rather a probability distribution. [14]

The AMCL algorithm requires two models; a motion model that computes the robot's motion and a sensor model which calculates the probability of the sensor readings. The motion model rely on control data which carry information about the change of state in the environment, while the sensor model depend on measurement data that present information about a momentary state of the environment. For this application the motion model correspond to a odometry model and its data is denoted $u$, while the sensor model correspond to a model of a laser scanner and its measurement is denoted $z$. [14]

The location of a robot is determined by sampling from the posterior belief of $x_t$ represented by a finite set of particles $S_t$. First, the algorithm draws samples from the posterior distribution, which in the AMCL case is the motion model. The motion model is also referred to as the state transition probability. The distribution is shown in Equation (7.1). [14]

$$x_t = p(x_t \mid x_{t-1}, u_t) \tag{7.1}$$

The equation indicate that the new state is dependent on the previously estimated state and the motion of the robot. Further on, a correction step is performed. The algorithm determines the importance weight of each particles by applying the sensor model, as shown in Equation (7.2). [14]

$$\omega_t = p(z_t \mid x_t, m) \tag{7.2}$$

After the correction step, the particles are resampled and the particles with a high importance weight are more likely to be resampled. The AMCL algorithm adaptively calculate the number of particles required to minimize the error between the real and sampled posterior. At start-up the algorithm needs many particles to determine the position of the robot. However, after the global localization is completed, the problem becomes a trajectory tracking problem, which require fewer particles.

By combining the state transition probability and the measurement probability as shown in Equation (7.3), the dynamical stochastic system of the robot and its environment can be determined. The state at time $t$ is dependent on the state at time $t-1$ and the motion $u_t$. The observation model uses the map together with the old pose to try to align the new and old observation to determine the updated pose of the robot. The motion model grants a rough region where to search, but ultimately it is the observation model that is dominant. [14]

$$x_t = p(z_t \mid x_t, m_{t-1})p(x_t \mid u_{t-1}, x_{t-1}) \tag{7.3}$$

where:

| Symbol: | | Description: | Value: | Unit: |
|---|---|---|---|---|
| $x_t$ | - | Robot pose | $\sim$ | – |
| $m_{t-1}$ | - | The map previously estimated map state | $\sim$ | – |
| $z_t$ | - | Current measurement | $\sim$ | – |

An example of AMCL applied to a 1D localization problem is presented in Figure (7.2). At first, the particles are initialized randomly and uniformly distributed in the environment, as illustrated in a). The black lines at the bottom of the picture represent the particles, and the fact that they are equally tall means they are uniformly weighted. The next step is to apply the observation model which calculates the importance weights. As can be seen in section b), the particles are the same set as a), however, the weight of them is non-uniform anymore. Based on the observation model, the robot knows it is close to a door, and for that reason, the particles near the doors have a higher importance weight. Next, section c) shows the particles after resampling and after applying the motion model. A new set of particles with uniform distribution is obtained, however, this time the particles are denser at the more likely places. The reason for this is that the particles with higher importance weight are more likely to be resampled. In section d), new importance weight are computed based on the new observations. Lastly, e) illustrates a new set of particles and the process continues in this cycle.[14]

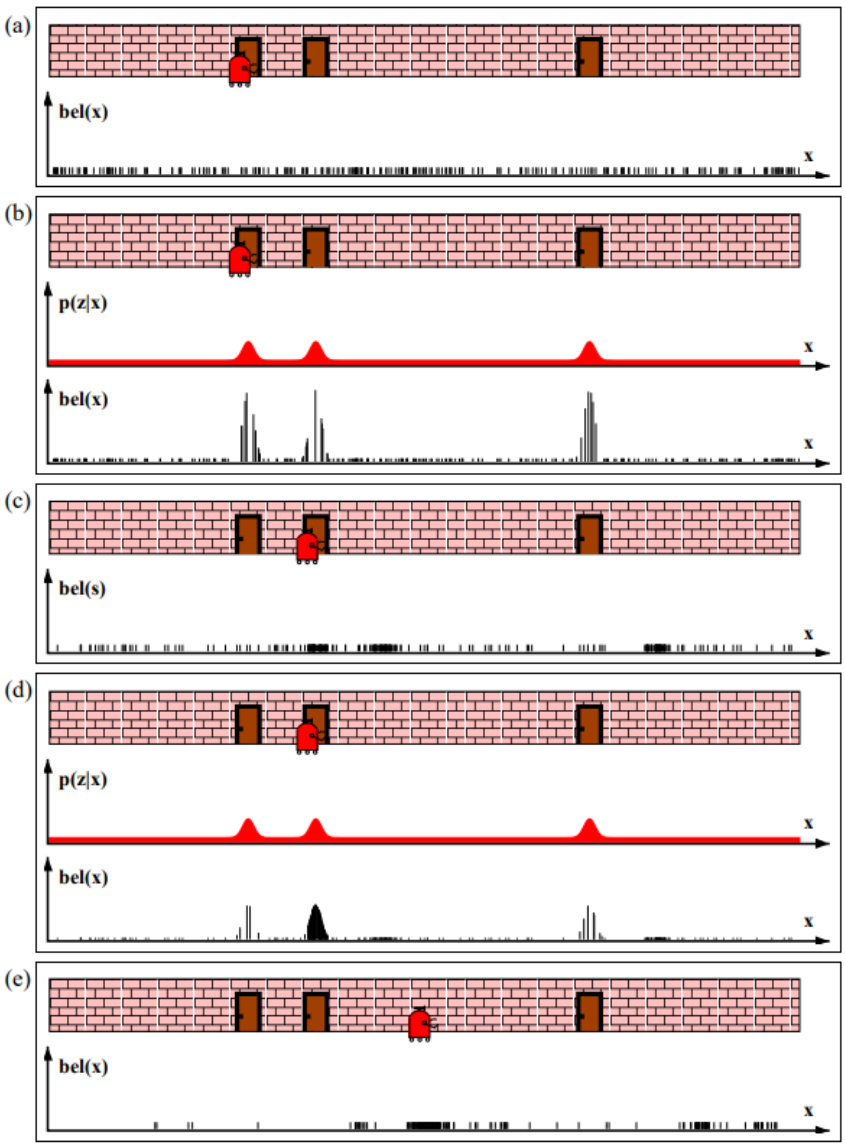
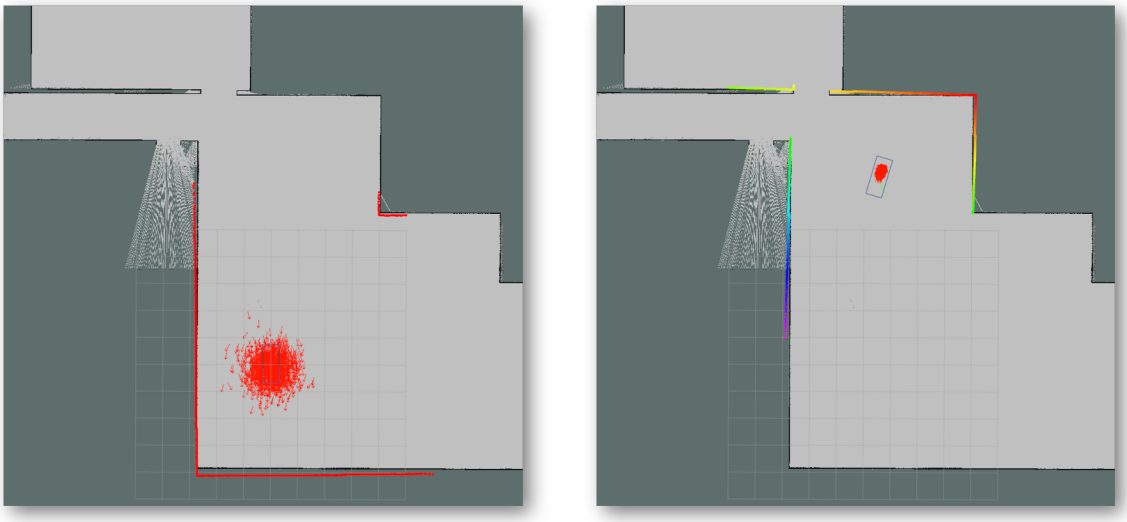



Figure 7.2: AMCL algorithm example [14]

An AMCL package is available in ROS and was implemented in this project. Prerequisites for the *amcl_node* is an occupancy map and transform configurations. The package also require an odometry models as well as a sensor model. The odometry and observation models are covered later in Section (7.1.3) and Section (7.1.4) respectively. To configure the algorithm for the Loomo rig, a configuration file was created. This file specifies important factors which needs to be tuned to comply with the specific system. Because this project rely on a simulation model, where everything is ideal, the configuration was straight forward. If this was to be implemented on the physical model, the algorithm will require additional tuning.

Figure (7.3) illustrates how the particles converge to a dense region after several iterations. When starting up the system, the robot must be given an approximate initial position which will be used as the posterior distribution. Figure (7.3(a)) is captured at start-up when the robot is given an approximate initial position, whereas Figure (7.3(b)) is captured several iterations later and shows how the robot is more certain of its position after moving. The certainty difference is emphasized by the density of the particle cloud.



(a) Likelihood field (b) Probability distribution

Figure 7.3: AMCL likelihood iteration

### 7.1.3 Motion Model

Because the algorithm is implemented into a simulation model where everything is ideal, the odometry is taken directly from the simulation software. In reality, the odometry should be based on filtered sensor readings. Because the physical model was unavailable, this was not developed in this project. A proposed solution for fusing information from an IMU and wheel encoders is however presented later in Section (9.2.1).

$$x_t = p(x_t \mid x_{t-1}, u_t) \tag{7.4}$$

where:

| Symbol: | | Description: | Value: | Unit: |
|---|---|---|---|---|
| $x_t$ | - | New state | $\sim$ | – |
| $x_{t-1}$ | - | Previous state | $\sim$ | – |
| $u_t$ | - | Motion data | $\sim$ | – |

### 7.1.4 Observation Model

As mentioned above, the algorithm requires a sensor model, which is an approximate model of a physical range finder. Range sensors are among the most used sensors within the field of robotics. The most basic model is called the beam-based sensor model. Although this model is accurate to the physics of an actual range finder, it suffers drawbacks in regards to computational complexity and lack of smoothness. Lack of smoothness presents problems in cluttered environments with many small obstacles such as chair- and table legs. Further on, evaluating each single sensor measurement, as done in the beam-based model, is very computationally expensive and requires substantial memory. The AMCL package provided in ROS uses an alternative sensor model called the likelihood field model, which overcomes these limitations. [14]

The likelihood field model, also known as the end point model, measures the correlation between the measurement and a map. The likelihood of a range measurement is a function of the distance of the beams end point to the closest obstacle in the environment. The model lacks a physical explanation and does not compute a probability in regards to the physics of a sensor. The model does however work well in practice and the posteriors are smoother in cluttered environments, and the computation is less expensive. A prerequisite for using this model is to know the transformation between the robots base and where the sensor beam $z_k$ originates. This is available from the transform configuration discussed in Section (6.4). The basic principle of likelihood fields is that the algorithm skips observations where the endpoints do not match the map for a considerable number of particles. The beams that are skipped are not used in computing the probability of the observations $p(z|m)$, which is used as the weight of each particle. Therefore particles that are correct do not get penalized because of unexpected obstacles in the environment, which is proven to help in dynamic environments. [14]

The likelihood field model assumes three types of noise and uncertainties. One of them is measurement noise. In an ideal sensor, the range finder would always return the correct range to the objects. However, in reality, noise is influencing the measurements. The measurement noise is often modeled as a Gaussian distribution with a mean equal to the "true" range of the measurement and a standard deviation. In this case, the Gaussian is denoted by $p_{hit}$ and is implemented by the distribution presented in Equation (7.5). [14]

$$p_{hit}(z_t^k \mid x_t, m) \;=\; \varepsilon_{\sigma_{hit}^2}\,(dist^2) \tag{7.5}$$

where:

| Symbol: | | Description: | Value: | Unit: |
|---|---|---|---|---|
| $z_t^k$ | - | A single measurement at time t | $\sim$ | – |
| $x_t$ | - | Robot state at time t | $\sim$ | – |
| $m$ | - | Map | $\sim$ | – |
| $\varepsilon_{\sigma_{hit}^2}$ | - | Zero-mean error with variance | $\sim$ | – |
| $dist$ | - | Distance to the closest obstacle in the map | $\sim$ | – |

The next uncertainty account for perturbations due to scan failures. Sometimes laser scanners fail to detect objects which are black, light-absorbing or when the environment includes bright light. In such instances, the range sensor tend to return the value of it's maximum range $z_{max}$. It is necessary to account for max-range errors in the measurement model and this is done through a point-mass distribution centered at $z_{max}$ as shown in Figure (7.4). [14]

$$p_{max}(z_t^k \mid x_t, m) \;=\; \begin{cases} 1 & if\ z = z_{max} \\ 0 & else \end{cases} \tag{7.6}$$

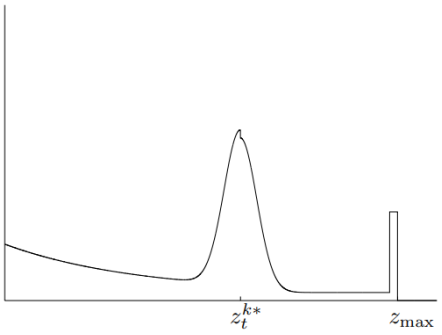



Figure 7.4: Max range error [14]

The last deviation is due to random noise in the measurements. As the term indicates, range finders sometimes return strange measurements which lacks a logical explanation. For this algorithm, these errors are modeled by a uniform distribution which is spread across the entire sensor observation range. [14]

$$p_{rand}(z_t^k \mid x_t, m) = \begin{cases} 1/z_{max} & if \;\; 0 \leq z_t^k < z_{max} \\ 0 & else \end{cases} \tag{7.7}$$

By combining the individual uncertainty models, a complete probability distribution can be obtained. This is shown in Equation (7.8). [14]

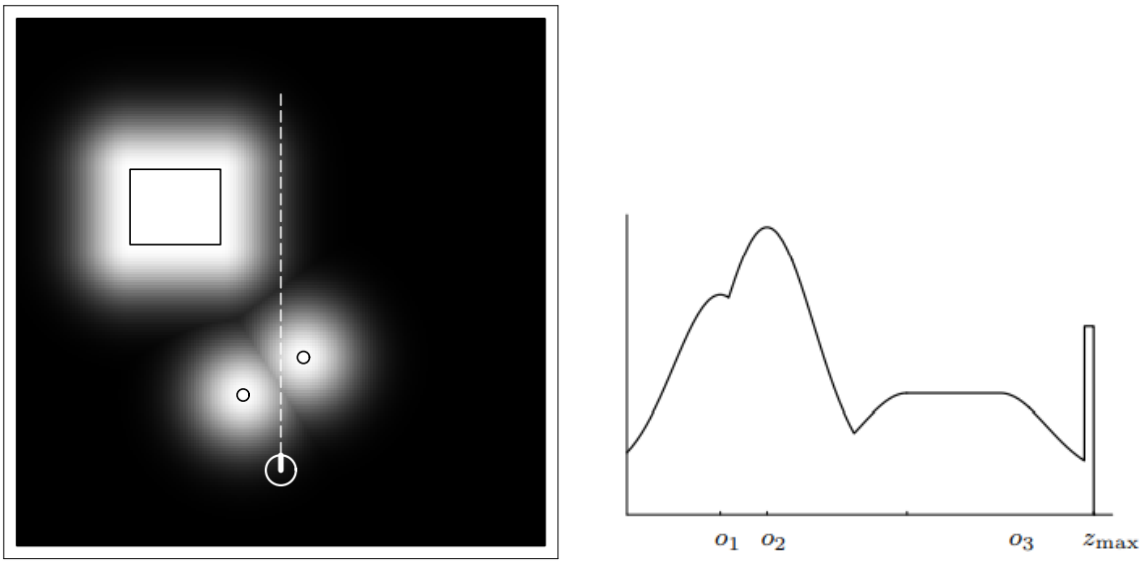
$$p(z_t^k \mid x_t, m) = z_{hit} \cdot p_{hit} + z_{rand} \cdot p_{rand} + z_{max} \cdot p_{max} \tag{7.8}$$

$$z_{hit} + z_{rand} + z_{max} = 1$$

where:

| Symbol: | | Description: | Value: | Unit: |
|---|---|---|---|---|
| $z_{hit}$ | - | Hit weighted average | $\sim$ | – |
| $p_{hit}$ | - | Hit distribution | $\sim$ | – |
| $z_{rand}$ | - | Random weighted average | $\sim$ | – |
| $p_{rand}$ | - | Random distribution | $\sim$ | – |
| $z_{max}$ | - | Max weighted average | $\sim$ | – |
| $p_{max}$ | - | Max distribution | $\sim$ | – |

The mixing parameters can be adjusted by hand to tune the systems accuracy. Figure (7.5(a)) shows a likelihood field for an environment with three obstacles, where the dashed line indicate a single measurement $z_t^k$. The brighter the region, the more likely the measurement is to hit an obstacle. Figure (7.5(b)) shows the resulting probability distribution. [14]




(a) Likelihood field

(b) Probability distribution

Figure 7.5: Likelihood example [14]

The endpoint of the measurement is projected into the global coordinate system of the map by Equation (7.9).

$$\begin{pmatrix} x_{z_t^k} \\ y_{z_t^k} \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{pmatrix} \begin{pmatrix} x_{k,sens} \\ y_{k,sens} \end{pmatrix} + z_t^k \begin{pmatrix} cos(\theta + \theta_{k,sens}) \\ sin(\theta + \theta_{k,sens}) \end{pmatrix} \tag{7.9}$$

where:

| Symbol: | | Description: | Value: | Unit: |
|---|---|---|---|---|
| $x_{z_t^k}$ | - | x coordinate of sensor beam endpoint | $\sim$ | $m$ |
| $y_{z_t^k}$ | - | y coordinate of sensor beam endpoint | $\sim$ | $m$ |
| $x$ | - | Robot x coordinate | $\sim$ | $m$ |
| $y$ | - | Robot y coordinate | $\sim$ | $m$ |
| $\theta$ | - | Robot orientation | $\sim$ | $rad$ |
| $x_{k,sens}$ | - | x coordinate of sensor relative to robot frame | $\sim$ | $m$ |
| $y_{k,sens}$ | - | x coordinate of sensor relative to robot frame | $\sim$ | $m$ |
| $\theta_{k,sens}$ | - | rotation of sensor relative to robot frame | $\sim$ | $rad$ |
| $z_t^k$ | - | A single distance measurement | $\sim$ | $rad$ |

In order to project the endpoint, it is necessary to know where the robot is in global coordinates and a map must be provided. The essence of this model is to compare the laser scan to the given map based on likelihood fields and the probability of the measurements.

## 7.2 Path Planning

Once the robot knows where it is in the environment, path planning can be implemented. The *move_base* package is a part of the navigation stack and was used for this purpose. This package is a collection of nodes which serves to navigate a mobile robot in a 2D environment. The package includes map interpretations, path planning, obstacle avoidance and recovery behaviours. The square in Figure (7.6) depicts an overview of how the package communicates.

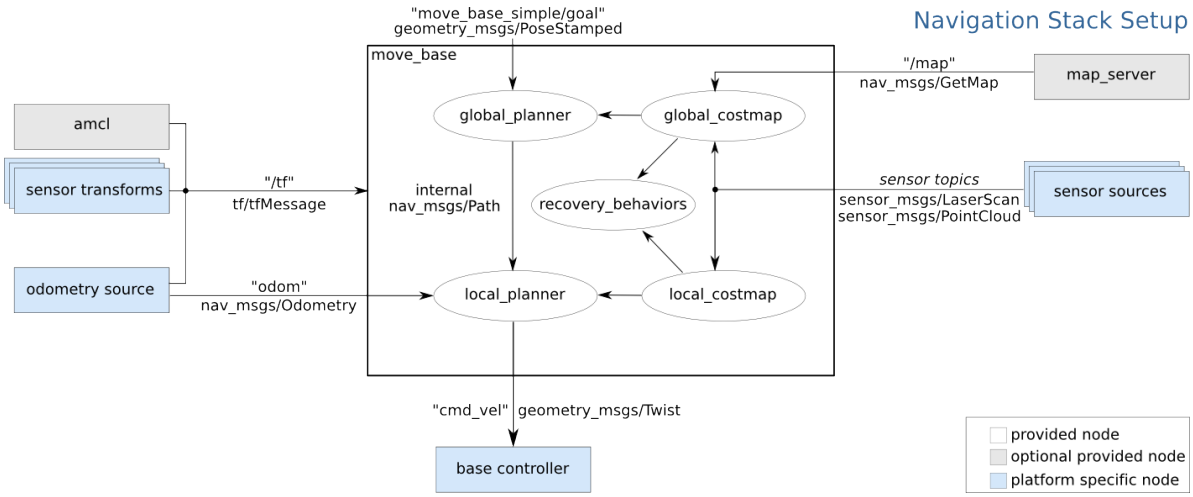



Figure 7.6: move_base overview [15]

As the figure indicates, the move_base node utilize information from localization and odometry source, in combination with a map to plan a route in the environment. This section aims to give an overview of the packages used and how they were configured to work together with the Loomo rig.

### 7.2.1 Costmap

To store information about the obstacles in the environment, a global and a local costmap is used. A costmap is a occupancy grid based map, which represents the environment and applies cost to all cells to specify how difficult it is to navigate in different areas. The global costmap is a static costmap attached to the map frame and is used for long term planning of trajectory across the entire map. The local costmap is a dynamic map updated in real time and attached to the odom frame, hence it follows the robot around. This costmap is mainly used for short term obstacle avoidance. In order for the robot to understand how big it is in relation to the obstacles, and concurrent how close it's center can reside from the obstacles, a footprint of the robot has to be declared. The footprint is the 2d representation of the mobile platform. Since the parking rig has a rectangular design, the footprint was created with the polygon configuration.[77]

A package which generates costmaps is available in ROS. The package is called *costmap_2d* and is a part of the navigation stack. A prerequisite for using *costmap_2d* is that the tf package is publishing transforms between the coordinate frames at the expected rate. In the costmap package, the functionalities are separated into layers. For this project the costmap is separated in three layers, the static map layer, obstacles layer and inflation layer. The global costmap contains all three layers, whereas the local costmap only uses obstacle layer

and inflation layer. The static map layer collects the premade map from the *map_server* and interpret it to create an appropriate global costmap, thus only the global costmap uses this layer. Next, the obstacle layer includes obstacles which are not observed in the static map, this can be dynamic obstacles such as humans roaming the environment. This layer tracks obstacles in 2D, for instance from a planar laser scanner. Based on the localization data and sensor readings, obstacles are added or removed from the map. This is required in both costmaps, because if an unexpected obstacle occurs, both the global and local plan has to be recomputed. The inflation layer inflates the obstacles to calculate cost for each costmap cell around the obstacles. The inflation radius tells the robot how close it can approach an obstacle without colliding. For the static layer, the obstacles are inflated based on the provided map, whereas in the obstacle layer, the obstacles are inflated based on the sensor readings in real time. Figure (7.7) shows a global costmap of a section of UiA, whereas Figure (7.8) depicts a local costmap. [77] [79]

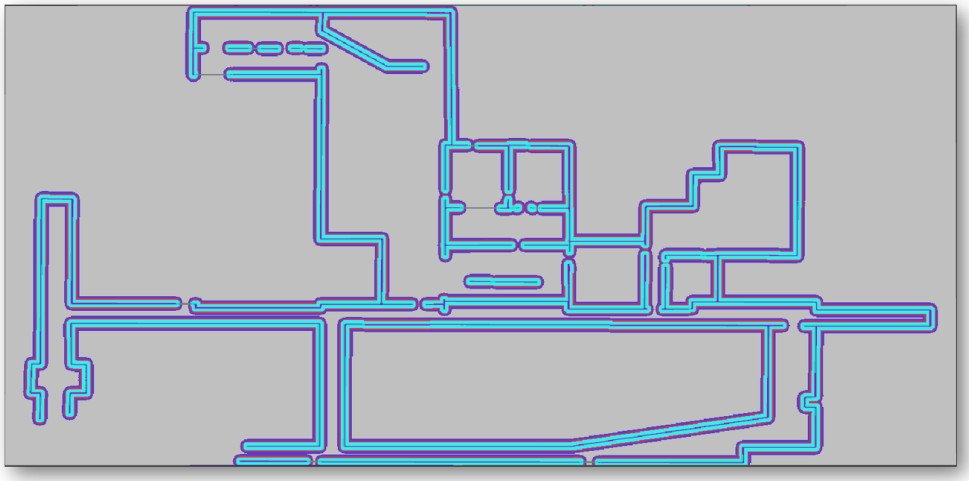


Figure 7.7: Global costmap

To use the *costmap_2d* package, some parameters had to be specified. Three configuration files were developed for this purpose. One of the files contain information which is common for both the costmaps, whereas the other two are specific configurations for each of the costmaps. The configurations are loaded together with the *move_base* node. A parameter which heavily influenced the quality of the path planning was the costmap resolutions. For large maps, a high resolution will be very computational heavy. For instance when set to 10 Hz, the map would typically be updated at 2 Hz. Further on, the inflation radius was set to 0.5 m, which indicate that its center should never come closer to an obstacle than this.

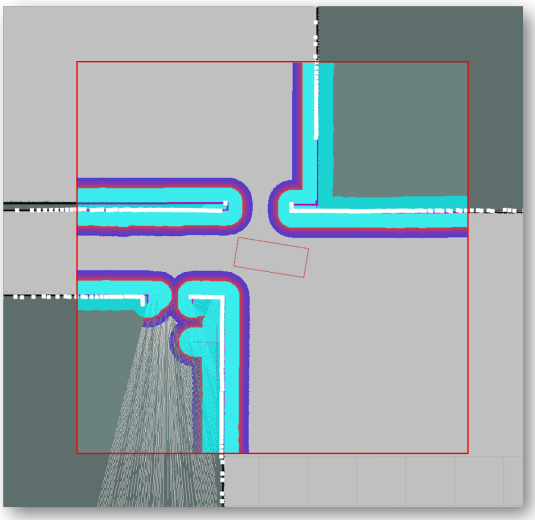


Figure 7.8: Local costmap 6x6m

### 7.2.2 Gloabl Planner - navfn

In order to set a goal position in the environment, the robot needs a global planner. The global planer operates on the global costmap and is responsible for generating a long-term plan from current position to the goal. One of the three available global planners with ROS is the *navfn* planner which was implemented to the Loomo parking rig. The objective of a global planner is to find the path with the lowest cost from start to end. Dijkstra's algorithm calculates the shortest distance between starting node end ending node in a weighted environment. By using Dijkstra's algorithm and the global costmap, the shortest feasible path with the lowest cost is obtained. The cost applied in the costmap is used as weights when applying Dijkstra's short path planning method [77]. The algorithm creates a tree between the nodes to find the shortest path among all other vertices within the map. The algorithm uses the following steps to calculate the path. [80] [81]

- Step 1: First every node is assigned a distance value. Initially the start node is set to zero and the rest to infinity to ensure the algorithm starts at the desired location.
- Step 2: Then the start node is set to the current node, and the rest of the nodes to unvisited.
- Step 3: From the current node, all unvisited neighbors are visited and their distance to the start node is calculate.
- Step 4: After all neighbors are accounted for, they are marked as visited. Then the path is continued from the node with the lowest tentative distance value. Visited nodes can never be visited again.
- Step 5: If the destination node gets marked as visited, or the node with the lowest tentative distance is infinity, the algorithm is finished.
- Step 6: Otherwise select an unvisited node and repeat from step 3.

In Figure (7.9) a system where Dijkstra's algorithm is applied in a small scale is presented. Green represents the current node and red represents the unvisited neighbours. As the algorithm proceeds yellow represents visited node. In the priority queue table, the tentative lowest distance is presented first. Finally the end node is reached represented in blue, and the shortest path is found. As shown in the figure, node $F$ is not visited because the end node is reached with the same distance value, thus the algorithm would be unable to produce a short path.

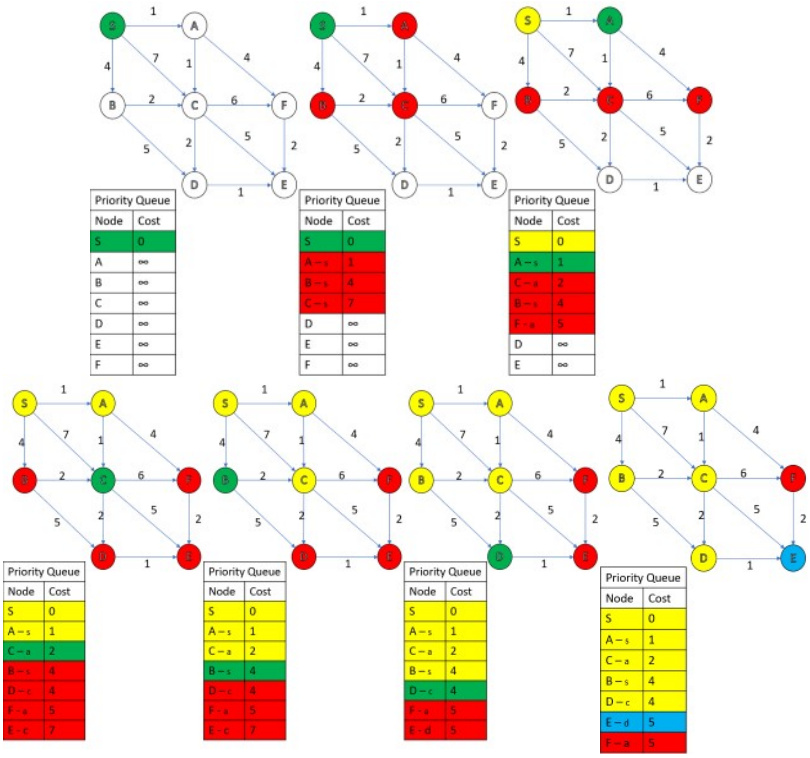



Figure 7.9: Example of Dijkstra

The cost calculation performed by *navfn* is in accordance with Equation (7.10):

$$\text{cost} = \text{cost\_neutral} + \text{cost\_factor} \cdot \text{costmap\_cost\_value} \tag{7.10}$$

Commonly used is a cost_factor = 0.8, cost_neutral = 50 and lethal_cost = 253. From the costmap used in the move_base package, the values are in the range of 0 – 252. A cost factor of 0.8 will allow the input value to be evenly distributed between neutral cost 50 and lethal cost 253, where obstacles occur. A larger cost factor may treat narrow area such as hallways or doors as undesired for path planning due to a high cost value. By tuning these parameters: neutral_cost and cost_factor, a suitable path planner can be achieved for the given purpose and accuracy demands due to map configurations. [82]

The global planner was visualized in RVIZ to verify that the node was working properly. When given a goal in the map, the global planner calculates a plan based on the global costmap. The inflation radius of the global costmap determines how close the path will be planned relative to obstacles. Figure (7.10) depicts the global path generated by the robot. If unexpected obstacles occur somewhere in the planned path, the global planner will evaluate a new path based on the global costmaps obstacle layer.

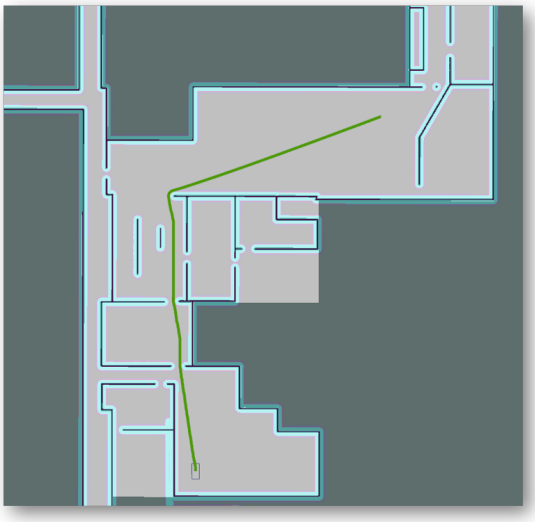


Figure 7.10: Global plan visualized in RVIZ

### 7.2.3 Local Planner - DWA

A local planner must also be specified in the move_base package. There are several different methods for local planning available in ROS. The local planner serves as a short-term planner, in oppose to the global planner. It will utilize the information from the odometry source, global planner and local costmap to output velocity commands to the cmd_vel topic. The base controller is subscribing to the cmd_vel topic and uses the received information to calculate the necessary wheel velocities that are required to obtain the desired motion stated by the local planner.[83]

Dynamic Window Approch (DWA) was chosen for this project, which is a method for obstacle avoidance which includes a controller that connects the path planner to the robot. The map is used to create circular trajectories from the robot to a local goal. Locally the planner creates a cost function that determine the velocities for x, y and theta. The DWA approach is performed in five steps [83]:

- Step 1: Discretely sample in the robot's control space (dx,dy,dtheta).
- Step 2: For each sampled velocity, perform forward simulation from the robot's current state to predict what would happen if the sampled velocity were applied for some (short) period of time [84].
- Step 3: Evaluate (score) each trajectory resulting from the forward simulation, using a metric that incorporates characteristics such as: proximity to obstacles, proximity to the goal, proximity to the global path, and speed. Discard illegal trajectories (those that collide with obstacles) [84].
- Step 4: Pick the highest-scoring trajectory and send the associated velocity to the mobile base.
- Step 5: Reset scoring and repeat.

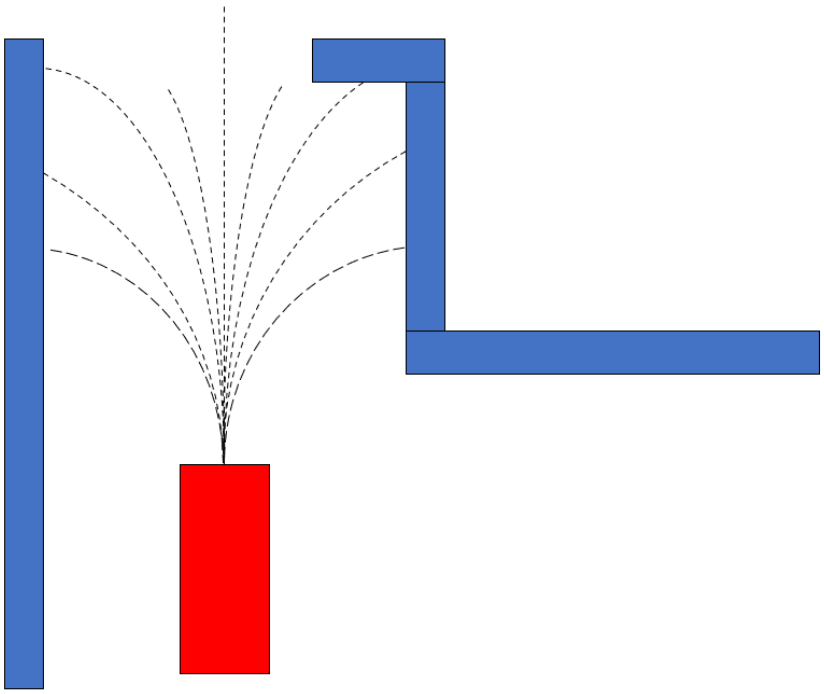


Figure 7.11: DWA trajectory example

DWA depends on the local costmap which presented obstacles in the local map to calculate the velocity pairs. By tuning the parameters of the local costmap the optimal results from the DWA planner will be achieved. [85] [84]

Forward simulation is the second step of the DWA planner. The planner uses velocity samples from the robot's control space and eliminated bad choices of circular trajectory samples (i.e. those intersecting with obstacles or edges). Then each sample is simulated forward for a duration depending on the *sim_time(s)* parameter. By tuning the *sim_time(s)* parameter the results may be improved. Additionally, the number of samples for x, y and theta may be tuned to improve the path and available computational power. Lastly, the sampling rate between points on the trajectory, *sim_granularity*, will influence the performance. The parameter may increase the accuracy of obstacle avoidance at a computational cost. [77]

Third step of the DWA planner in ROS, are trajectory scoring for optimizing the local path by velocity pairs which is dependent on three parameters: progress to goal, clearance from obstacles and forward velocity. The objective function, creating an overall cost value will be calculated with the following Equation (7.11). [77]

$$\begin{aligned} cost\_dwa = {} & \text{path\_distance\_ bias}\cdot(\text{distance(m) to path from the endpoint of the trajectory}) \\ & + \text{ goal\_distance\_bias} \cdot (\text{distance(m) to local goal from the endpoint of the trajectory}) \\ & + \text{ occdist\_scale} \cdot (\text{maximum obstacle cost along the trajectory in obstacle cost (0-254)}) \end{aligned} \tag{7.11}$$

The goal is to keep the cost as low as possible. *path_distance_bias* is referred to as how strictly the local planner should follow the global planner path. The higher the value, the more of the global trajectory will be preferred. *goal_distance_bias* is a weight increasing the attempt of the robot to reach local goals, independent of path choices. *occdist_scale* is the weight for object avoidance. A high value can result in an indecisive robot getting stuck. [77]

The local plan was verified by visualizing it in RVIZ. Figure (7.12) illustrates the local plan in red and the global plan in purple.

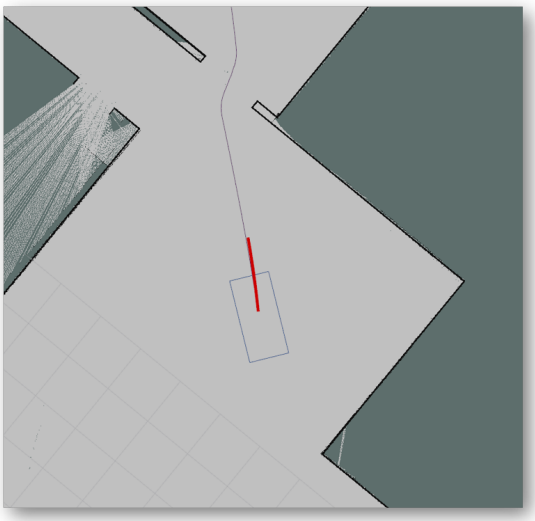


Figure 7.12: Local plan visualized in RVIZ

### 7.2.4 Recover Behaviors

If the robot finds itself in a situation where it is stuck, it will perform recover behaviors. For instance if the robot is surrounded with obstacles and can not find a way to its goal. This can typically happen in dynamic environments where humans are surrounding the robot. There are two types of recovery behavior that are default for the move_base node, clear costmap recovery and rotate recovery. If the robot perceives itself as stuck, it will start by performing a conservative reset where it refreshes the costmap within a specified range around itself. Next, if possible, the robot will rotate around itself. If this operation succeed, the robot will continue to the navigation goal. However, if this does not work, the robot will perform a aggressive reset which clears a bigger area of the costmap and then rotate. If this fails, the navigation goal will be aborted. Figure (7.13) shows the flow of the recovery behavior.

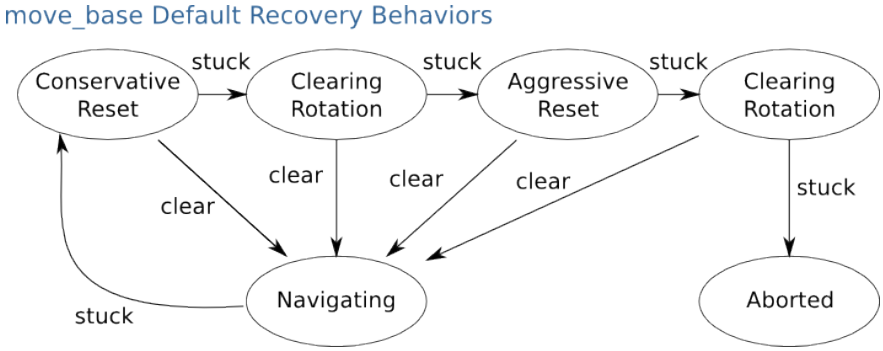



Figure 7.13: Recovery behavior flow

## 7.3 ArUco Tracking

One of the objectives were to let the Loomo parking rig track and follow and person or a Loomo. It was opted to explore the abilities of tracking with Augmented reality markers. Augmented reality markers are commonly used in robot localization and navigation, because they are easy to recognise and can be used as landmarks to estimate the pose of a robot. The marker detection use a process based on corresponding the points between a 2D projection and real environment projection. Augmented reality markers can be designed with multiple libraries and in this project the ArUco library is utilized. [16]

The ArUco marker library is designed as a binary square fiducial marker. The benefit is that each of these markers will provide enough correspondence points for camera pose estimation. In addition, the inner binary codifications are designed in a robust manner. A marker is composed with an outer black border around the binary matrix. The binary matrix determines the identity of a marker. The ArUco library contains various marker sizes as displayed in Figure (7.14). Commonly used are the $4 \times 4$ and $6 \times 6$ markers. A $4 \times 4$ marker was utilized for tracking with the designed Loomo parking rig.[16]

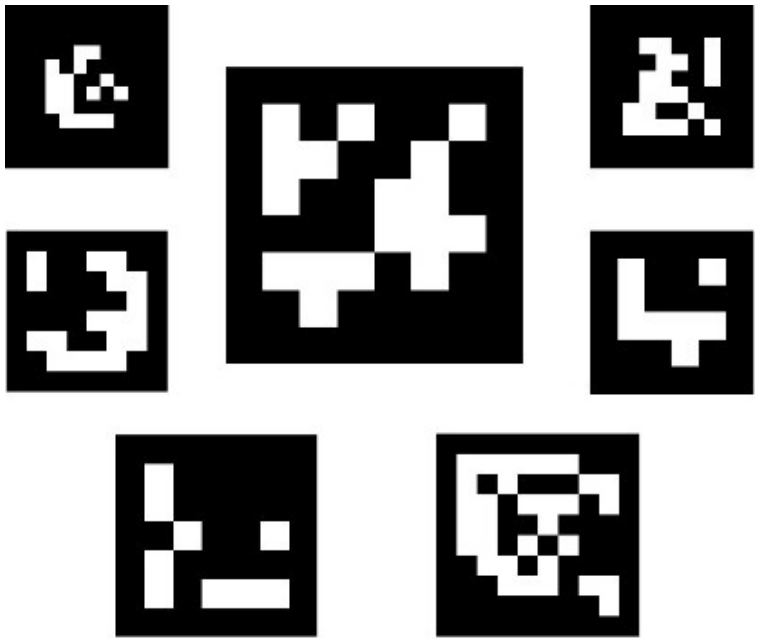


Figure 7.14: Example of standard ArUco markers [16]

**ArUco Detection**

An image sensor may capture multiple ArUco markers at a time. It is then important that the detection algorithms are able to identify each marker, in order to differentiate between them. The data returned from the algorithm should list each of the detected markers with the related four corners and an id. The detection of markers is performed in two stages. The first stage consists of an analysis to detect black squares that might be markers. The second stage analyze the detected squares in order to determine possible marker identity. [16]

- Stage 1: Image processing; looking for square shaped objects that could possibly be a marker. The process starts with the use of an adaptive threshold, in order to segment markers. Then contours are extracted from the thresholded image and shapes that can not represent squares are discard. Final part of the stage is filtering away contours that are too small or big and too close to each other [16].

- Stage 2: After all marker candidates are found, an analysis of their inner codification is performed. To achieve this, a perspective transformation is applied to get the marker in canonical form. Further, Otsu is used on the canonical image to separate black and white bits. This will divide the image into cells according to specified marker and border size. Then the black and white bits are counted to determine the cells value. Finally, the bits are analyzed to determine if the marker is represented in the specified ArUco directory. [16].

In order to increase the autonomy of the Loomo parking rig, it is developed an algorithm that makes the rig track and follow a an ArUco marker. Further on, the algorithm used to detect ArUco markers and send velocity commands, as navigation goals, will be elaborated. Prerequisites for using the algorithm is to have calibrated all cameras used for tracking.

First a subscription to the image stream is initiated providing raw RGB images from the camera topic in ROS. The RGB image is grayscaled and the algorithm start searching for markers. The marker detection is performed with the openCV ArUco module detectmarker(). The detectmarker() function depends on five arguments and returns three values: corners, ids and rejected points.[16]

- First argument is the input image which is the gray scaled RGB image
- Second argument is the ArUco dictionary used to find markers is specified.
- Third argument is twofold, markerCornes and markerIds
  - markerCorners list each of the corner of each of the detected markers, clockwise from top left.
  - markerIds list the ID of each marker detected from markerCorners
- Fourth argument is DetectionParameters. These parameters are often customized prior to the detection
- Fifth argument is arbitrary, and is used to store, in a list, the rejected candidates that did not contain a valid codification.

### ArUco Pose Estimation

After detecting ArUco markers it is possible to obtain a marker pose by using the detected corners together with the intrinsic camera matrix, distortion coefficient vector and marker size. By utilizing these parameters in the ArUco function estimatePoseSingleMarkers() a rotational and translation vector is achieved relative to the camera. The function estimatePoseSingleMarkers() is dependent on four arguments. [16]

- First argument is the four corners of the detected ArUco marker.
- Second argument is the size of the ArUco marker given in meters
- Third argument is the 3x3 intrinsic camera matrix.
- Fourth argument is the distortion vector.

### ArUco Tracking

In order to perform the detection of ArUco markers, the fiducial ROS package was used [86]. Specifically, the aruco_detect node, within the package. The node is specifically developed to determine the pose of detected fiducial markers [87]. The node will identify observed ArUco markers by the specified camera stream and determine the pose of the marker relative to the camera.
In Figure (7.15) the information about an observer ArUco marker, detected by the aruco_detect node, is shown. A launch file to use the node must include the camera stream and the camera information to locate markers. Additionally, the dictionary to look for the marker and marker size in meters to more accurately determine the pose is specified within this file. Dictionary value 3, which is used in the launch file which represents dictionary DICT_4X4_1000. The launch file is presented in Appendix (F.4). [88]

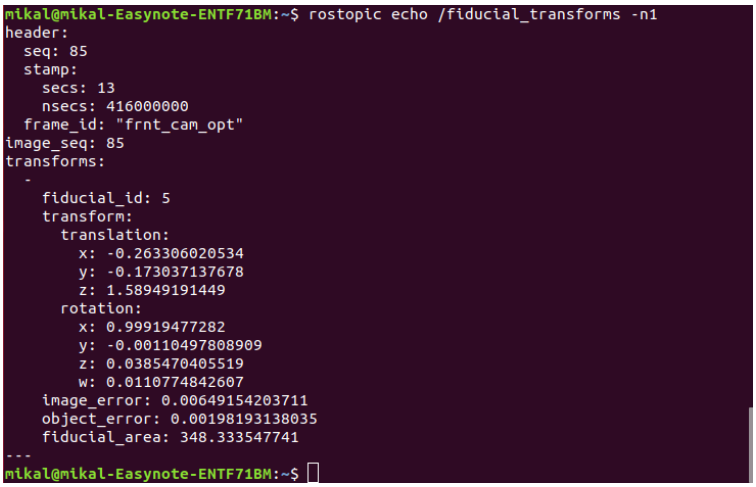



Figure 7.15: fiducial_transforms message

In order to follow an ArUco marker within the map, an algorithm is developed to send navigation goals. Navigation goals in ROS will initiate velocity commands that moves the rig towards the location where an ArUco marker has been observed. The algorithm subscribes to ROS topics which provides the necessary information to execute the velocity commands. The algorithm will be elaborated in more detail.

Initializing the algorithm as a ROS node was done prior to subscribing to several topics. The algorithm subscribes to three topics; odom, fiducial_transforms and fiducial_vertices. The subscription is received in a standard format with ROS library for python, rospy. By using rospy, the subscription is made with three arguments: name of ROS topic as a string, format of message and an information callback.

From each callback, the required information is retrieved. From fiducial_callback; frame_id, the translation and rotation of a detected marker by the aruco_detect node is received. Further, the received pose is set as global variable within the algorithm. Parameters are gathered by using a for loop in order to iterate through the nested transform information. In order to process the vertices data of the detected ArUco, the nested vertices message is retrieved inside vertices_callback, and also iterated through. From the odom topic, the odom_callback function retrieves the pose of the rig in x and y position and rotation in the map, where the initial pose has to be considered.

When all the information from each topic is received, processing can begin if an ArUco is detected. The process is executed if the ArUco translation in z and vertex x0 is not zero. If there are no detected ArUcos there are no valid information received and the rig will not move. From a created translation vector, the distance from the rig to the ArUco marker is calculated with Equation (7.12). Then, by using the ArUco's corner pixel positions, the horizontal center of the marker is calculated and used to determine the marker placement within the camera frame. The deviation from image center, in meters, is used to estimate orientation. The conversion is performed by calculating a pixel ratio based on the size of the marker. From the rotation around z-axis, the goal pose can be transformed from Euler angles to Quaternions, since the orientation message format uses this representation. Quaternions is a way of representing orientation in four dimensional space. ROS uses it in order to avoid Gimbal lock. Specifically, the loss of one degree of freedom, i.e. two axis are equal, which the use of Euler angles can yield.

$$l = \sqrt{t_x^2 + t_y^2 + t_z^2} \tag{7.12}$$

where:

| Symbol: | | Description: | Value: | Unit: |
|---|---|---|---|---|
| $l$ | - | Length to ArUco from camera | $\sim$ | $m$ |
| $t_x$ | - | Translation ArUco X-direction | $\sim$ | $m$ |
| $t_y$ | - | Translation ArUco Y-direction | $\sim$ | $m$ |
| $t_z$ | - | Translation ArUco Z-direction | $\sim$ | $m$ |

From the calculated angle of the ArUco and retrieved orientation from odom_callback, the quadrant the ArUco is located in, relative to the rig, is determined. From the pose received from the odom_callback combined with the location of the ArUco marker, the relation between the Loomo rig and the marker in the global map is determined. By knowing the position the rig has to travel to reach the marker, a navigation goal can be set.

Previous calculations is used with a function movebase_client to send a navigation goal with four arguments: x, y and quaternion z and w. Within the movebase_client function, there is a client server move_base based on MoveBaseAction. In order for the function to listen for navigation goals, a start server criterion must be meet. Further on, the goal message can be design with an id, a time stamp, and the navigation goal, in the MoveBaseGoal design format. A goal with x = 1 and y = 1 tells the robot to move to (1,1) in the map, then in order to set an new goal, (1,1) must be used as a starting point to allow continuous movement and not let origin be used as a center for each goal. This node will run continuously as the rig is in follow mode, action will only be sent if a marker is observed. The code can be found in Appendix(F.3).

## 7.4 Assistive Actuation

A request was to look into the possibility of utilizing the leap motion controller, from Ultraleap, in order to control the Loomo rig with hand gestures. The sensor, combined with the appropriate software, tracks the position, orientation and velocity of the palm and fingers of multiple hands. Thus, it can be used in order to send reference signals by the use of the position and orientation hand data. Figure (7.16) displays the leap motion sensor in addition to a visualisation of the processed tracking data.

Figure 7.16: Leap Motion Hand Tracking [17]

The Motion Leap consists of three infrared LEDs and two cameras with a wide field of view. The Cameras track the infrared light from the LEDs with a wavelength of 850 nanometers. The wide view angle to the cameras results in a field of view of 150° × 120°. Figure (7.17) displays the layout of its components. The sensor data is read into the device's local memory through its USB controller. Here any necessary resolution adjustments are performed. Then, the data is streamed through USB to the unit with the Leap Motion tracking software. The data takes the form of a grayscale stereo image of the near infrared light spectrum. As an alternative, the raw data can be gathered directly with the use of the image API. The raw data sent over the USB is separated into the gathered data from the left and right camera. [89]

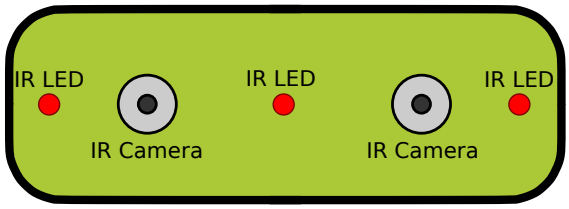



Figure 7.17: Leap Motion active components placement

Once the image data is received from the Leap Motion sensor, the images are processed. Though stereo imaging is used, the image processing does not generate a depth map. Instead advanced algorithms developed by Ultraleap are applied to the raw sensor data. [89]

After the images are processed, any unwanted background objects, such as heads, and ambient environmental lighting are compensated for. Then, the images are analyzed in order to reconstruct a 3D representation of what the device sees. A tracking layer is then used for matching of the data in order to extract the tracking information. The Leap Motion tracking algorithms interprets the 3D data and deduces the positions of the unwanted objects which are to be obstructed. [89]

Lastly, before the results are expressed, filtering techniques are applied. The applied techniques smooth temporal coherence of the data. The results are represented as a series of frames or snapshots which are containing all of the tracking data. [89]

### Utilizing the Source Development Kit and ROS leap_motion

In order to utilized the developed techniques, the Source Development Kit (SDK) was used. With the Leap Motion SDK installed on Linux, the sensor is initialized by running the lines in Listing (7.1) in a teminal. The first line in the listing is only required if the sensor does not start by starting the control panel, which was a problem that occurred in the setup used. Further, an official ROS package exists. The package is a wrapper for interfacing the Leap Motion 3D sensor. The packages was used since ROS is utilized in order to branch the communication between the Xavier and the electrical speed controllers. [90]

```
leapd
LeapControlPanel
```

Listing 7.1: Leap Motion start commands

The perquisites of using the Leap Motion ROS package is that the SDK provided by Ultraleap is installed and that, by default, the leapSDK folder is present inside the ROS workspace source folder. With the Leap Motion package installed, the demo launch was started. This starts a visual display of the processed sensor data in RVIZ. The topics that are used in the demo was then further examined, and it was discovered that the topic with the name /leap_motion/leap_filtered publishes the hand tracking data. A launch file was then created by editing the demo launch file, where any unnecessary nodes were removed, such as the visualisation in RVIZ and the creation of a depth image purely used for visualization purposes. The launch file can be found in Appendix (F.2).

By running the edited launch file, the /leap_motion/leap_filtered topic was further examined. The messages is of the type leap_motion/Human. This is a custom message, which contains multiple nested messages. The Loomo rig was decided be controlled in Gazebo by the orientation of a hand tracked by the Leap Motion sensor. This information is found in the nested /leap_motion/Hand message. It was discovered that if no hands are detected in the frame, the previous sampled data is looped. It is beneficial to order the rig to stop if it ever loses track of the controlling hand. The nested message /leap_motion/nr_of_hands handles the data concerning the number of hands in the frame. The information from its related topic, nr_of_hands, was used to check if the rig should stop.

The move_base package is used in order to handle the control of the rig. The package uses velocity references which has to be published to a topic named cmd_vel, since this is the topic the move_base subscribes to. With the required data available, a node to control the Loomo rig was created. The created node is made to check if there is a hand in the frame. Then send a converted interpreted of the hand orientation in order to send rotation

references. To move the rig forwards or backwards, the palm position in relation to the Leap Motion sensor was used. This information is handled by the /leap_motion/Hand message. The sensor measures, while running the launch file, the distance span 0.05-0.55 m. This span was locked to 0.1-0.3 m in the node, which further on is mapped to -0.1 - 0.1 m by applying Equation (7.13).

$$X_{out} = \frac{(X_{in} - in_{min})(out_{max} - out_{min})}{(in_{max} - in_{min})} + out_{min} \tag{7.13}$$

where:

| Symbol: | | Description: | Value: | Unit: |
|---|---|---|---|---|
| $X_{out}$ | - | Mapped value of input | $\sim$ | $m$ |
| $X_{in}$ | - | Input value | $\sim$ | $m$ |
| $in_{min}$ | - | Minimum in value | 0.1 | $m$ |
| $in_{max}$ | - | Maximum in value | 0.3 | $m$ |
| $out_{min}$ | - | Minimum out value | $-0.1$ | $m$ |
| $out_{max}$ | - | Maximum out value | 0.1 | $m$ |

Lastly, in order to get a controlled way of stopping the Loomo rig, a check of the grip strength of the hand is introduced to the node. This information is also handled by the /leap_motion/Hand message. The grip strength was used in such a way that an open hand sets the rig into drive state, and a closed hand sets the state to stop. The node created to achieve this can be found in Appendix (F.1).

# 8. Results

This chapter present the results of the mechanical and electrical design, the derived kinematic for the robot, and the results concerning the three operating modes. The majority of the results presented in this chapter are obtained from the robot simulation and do not directly comply with the physical model.

### Design

Concept 5 was chosen as the final design. The mechanical construction was built by the group members and the final result can be seen in Figure (8.1). The rig is designed to carry ten units of Segway Robotics Loomo, which is equivalent to 200 kg. The frame itself weighs 85 kg and additional components are assumed to add an extra 35 kg. For this reason, the rig is designed with a capacity of 400 kg, this ensure a safety factor of 1.25. It has also been designed to be stable for all load cases. The current system includes the power system as well as the electronic speed controllers, motors and CAN-bus communication line. The Xavier and Kinect cameras are not yet implemented onto the physical model.

Figure 8.1: Final design - side view

### Kinematics

The inverse and forward kinematic models that were derived for the Loomo rig are shown in Equation (8.1) and Equation (8.2) respectively. Theses equations are applicable to any width and height as long as the corresponding distances in the models are equal. In addition, the wheel configuration has to be the same as shown in Figure (8.2) with the same reference to each individual mecanum wheel.

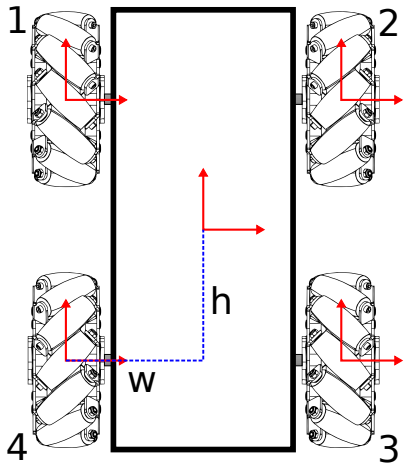



Figure 8.2: Top view of configuration

$$\begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix} = \frac{-1}{r} \begin{bmatrix} 1 & 1 & w+h \\ -1 & 1 & -w-h \\ 1 & 1 & -w-h \\ -1 & 1 & w+h \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} \tag{8.1}$$

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \frac{-r}{4} \begin{bmatrix} 1 & -1 & 1 & -1 \\ 1 & 1 & 1 & 1 \\ \frac{1}{w+h} & -\frac{1}{w+h} & -\frac{1}{w+h} & \frac{1}{w+h} \end{bmatrix} \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix} \tag{8.2}$$

### Robot model

The robot model created for this project is not an hyper realistic representation of the real plant, however it is sufficient model which was suitable to develop a navigation system for the virtual Loomo rig. It contains the same holonomic ability as the actual plant, and was used to verify the kinematics of the robot. The model also includes equivalent localization sources, only they are virtual. The system has been designed in such a way that the virtual sensors used in the simulation model, should be easy to replace with a physical sensor. The resulting navigation system will however require further tuning when implemented on the physical system.

### Visual perception

Multi-camera perception served as a good technique in this project. The Azure Kinect has a large field of view and provided information about obstacles in all heights of the environment. The robot selects only the important information from the 3D image and converts it into 2D to ease the processing demand. Further on, merging of four 2D images were obtained and the result was used for localization. The computational demand for processing the 2D representation was manageable.

### Localization

Adaptive monte carlo localization (AMCL) proved to be a robust localisation technique. The algorithm was verified in the simulation model, but not on the physical model. Figure (7.2) shows the particle cloud from the AMCL at initial position, and after moving. The green arrow represent the true pose of the robot, whereas the red arrows are pose hypothesis calculated by the AMCL. After iterating, the robot is very certain of its pose.


Figure 8.3: Localization visualized in RVIZ

### Path planning

The path planning performed by the robot is robust, also in dynamic environments. Dijkstra's algorithm was chosen as the global planner. The global path produced is efficient in the sense that the route is planned close to the inflation limit. Figure (8.4) compares the global plan versus the chosen path. The green line represent the global plan and the red line show the route which the robot actually executed.

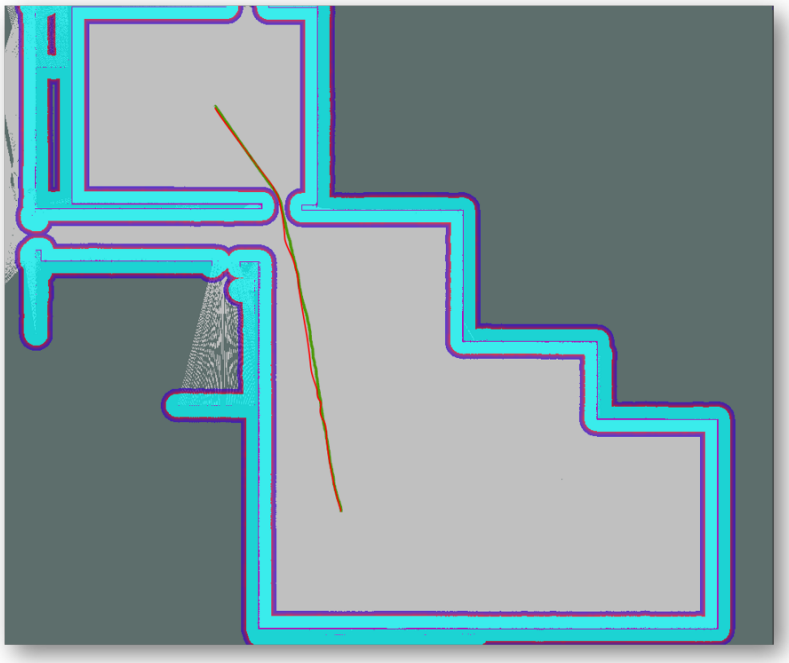


Figure 8.4: Global plan versus actual position

The Dynamic Window Approch(DWA) was chosen as the local planner. The local plan tracks the global plan well and avoids colliding with obstacles. Figure (8.5) illustrates the DWA local planner iterated over a course of 2 seconds. The green path represent the global plan, whereas each color of the local planner represent forward simulations by the DWA planner. The figure prove that the local planner is able to track the global plan very well.

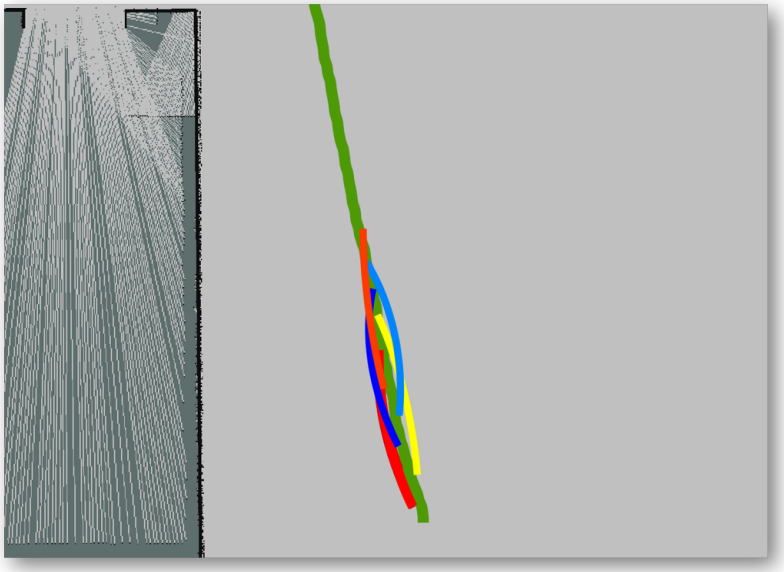


Figure 8.5: DWA local planner

Additionally, if unexpected obstacles occur, the algorithm is quick to reroute around it and reach the goal. Figure (8.6) shows the original planned path compared to the path created once an obstacle was detected.

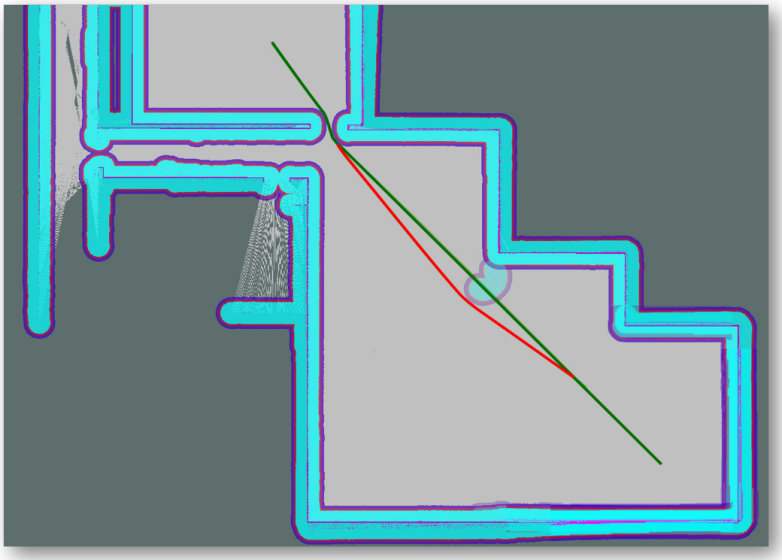


Figure 8.6: Reroute due to obstacle

The node graph which explain the communication between individual nodes during localization and navigation can be found in Appendix (F.7). The transform configuration tree which describes the transformation between all the coordinate frames can be found in Appendix (F.6).

### ArUco

By using the *aruco_detect* node from ROS, the robot was able to find the virtual $20 \times 20\,\text{cm}$ ArUco marker at short to medium ranges. An ArUco marker object was design in the ROS environment for testing purposes. Both the position in the 2D plane and pose relative to the robot was calculated correctly. The marker was moved around to in GAZEBO and intended to represent the motion of a person, which the rig should follow. However, the algorithm developed to follow an ArUco marker was not successful. From Figure (8.7) it can be seen which camera that detected the marker with information related to the marker orientation. Whereas, Figure (8.8) presents a ArUco detected by the camera.

```
mikal@mikal-Easynote-ENTF71BM:~$ rostopic echo /fiducial_transforms -n1
header:
  seq: 85
  stamp:
    secs: 13
    nsecs: 416000000
  frame_id: "frnt_cam_opt"
image_seq: 85
transforms:
  -
    fiducial_id: 5
    transform:
      translation:
        x: -0.263306020534
        y: -0.173037137678
        z: 1.58949191449
      rotation:
        x: 0.99919477282
        y: -0.00110497808909
        z: 0.0385470405519
        w: 0.0110774842607
    image_error: 0.00649154203711
    object_error: 0.00198193138035
    fiducial_area: 348.333547741
---
mikal@mikal-Easynote-ENTF71BM:~$
```

Figure 8.7: ArUco marker detection result

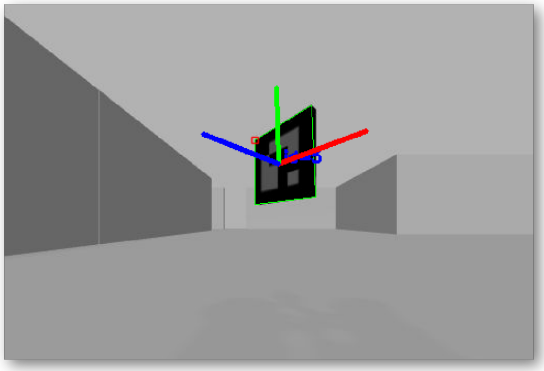


Figure 8.8: ArUco marker detected in Gazebo

**Assistive actuation**

By utilizing the Leap motion SDK with the respective ROS package, assistive actuation was successfully achieved in the GAZEBO environment. The filtered data is stable, which makes the sensor suitable for controlling the rig by interpreting the pose of a hand.

# 9. Discussion

### Mechanical

When the Loomo rig was developed, the design was a result of an evaluation of several mechanical components. Each evaluation was based on conservative design equations that ensured unlimited lifetime considering the working conditions and environment.

Even though the rig is never going to accumulate a high travel distance and the rotating parts become oversized, the conservative calculations yielded a drive shaft close to standardized material dimensions. By choosing to slightly increase the diameter from the conservative calculations to 20 mm, material cost increase. However, compared to the reduction in labor cost for machining, the increase in material cost is negligible. Especially for manual labor of few components. In addition to machining difficulty being reduced, due to the increased stiffens considering the size.

It was experienced that the first instinct to people when interacted with the Loomo rig, was to stand on the rig. Further, emphasising the selection of conservative calculations and the additional increase in drive shaft diameter.

The rig is prone to slip. The design was welded together which can cause tension between frame parts. Additionally, the wheels have a very low tolerance to irregularities in the frame structure causing slip. To reduce the chance of slip during operation it was tested with rubber plates between the frame and the bearings to form a suspension on an uneven surface. It had the most effect on a fully load rig, which resulted in the purposed suspension in Section (9.1.2).

### Hardware Setup

By the use of a hand controller, the programs to send and receive CAN-messages was verified. Further, by using the the hand controller, the inverse kinematic model was also verified. The communication involving the Xavier has only been tested separately detached from the rig. Even though the information is being sent from the Xavier to the VESCs and back, if changes to the message frequencies has to be made still remains.

### ArUco

When testing there were some difficulties around sharp turns entering narrow hallways where the change of rotation by the simulated person were large. The algorithm had difficulties setting new navigation goals following the person due to not detecting the marker. All tests were performed by simulation in ROS, where the rig had difficulties navigating due to continuously overwriting the navigation goal. This resulted in difficulties finding the ArUco maker, because this resulted in the Loomo rig rotating about it own axis. Additionally, the robot would occasionally perceive the ArUco marker as an obstacle. As a consequence, the robot was not able to generate a navigation goal to the marker. The script which converts the ArUco pose to navigation goals contains flaws and requires further investigation and improvements.

**Assistive actuation**

In the GAZEBO environment, the Leap Motion sensor was used to move rig forwards or backwards depending on the hand distance from the sensor. In order to rotate the rig, the hand has to be rotated appropriately. These control features requires that the leap motion sensor is oriented horizontal, on a table, with the cameras pointing upwards. If the sensors is to be attached to the physical rig, it might be more beneficial to attach it in a different orientation, and the program has to be changed accordingly. Further, additional safety features might have to be added, that the GAZEBO environment did not introduce.

## 9.1 Improvements

The section presents two improvements for the Loomo parking rig as a result of experimental testing, and safety features that are not yet implemented. The improvements are considered to be crucial additions prior to utilizing the rig in the University setting.

### 9.1.1 Safety

The university of Agder, campus Grimstad, is the intended environment for the Loomo rig. At the university there will likely be humans present during the operation of the rig. It is important to ensure safe operation of the rig and to avoid potentially dangerous situations. Due to the large mass of the construction, the rig may cause severe injures in cause of an accident. The only safety for the current system is the obstacle avoidance feature for the autonomous mode, however this is software based and it would be beneficial to implement a mechanical safety as well. Figure (9.1) emphasize the need for additional safety. As the figure illustrates, the cameras field of view do not intersect exactly at the periphery of the robot frame. If someone was to enter the blind zone, the robot would not see them and could potentially cause a collision.

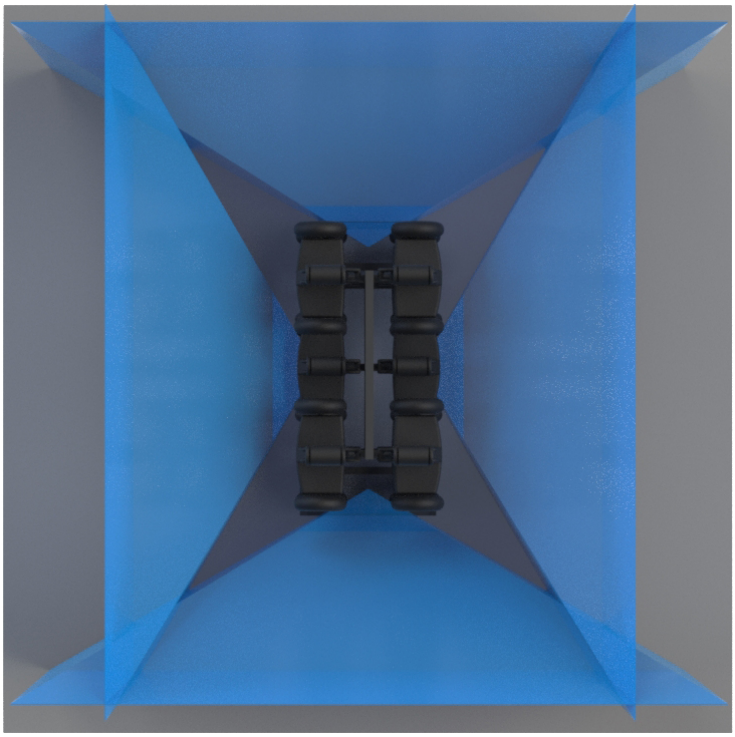


Figure 9.1: Bird view of FOV

A proposal is to mount bumper plates around the construction, which can be used to detect collision. In the event of a collision, the bumpers could signalize to a relay that the power for the motors should be cut. This would incorporate additional safety with respect to humans roaming in the same environment as the Loomo rig. Another approach could be to maximize the breaking current when a bumper register a collision, in order to minimize the impact force.

Additionally, the rig is not yet equipped with an emergency button and the only way to power off the rig is to cut the supply via the power switch. An emergency button is a necessary addition to the vehicle. It is recommended to include one attached to the rig itself, and one on the remote hand controller. The button should be placed on a location which is easy to see and access in case of emergency.

### 9.1.2 Traction

During testing of the physical rig, it was observed that the traction was poor due to irregularities in the floor. This is mostly because the rig is very stiff and a small distortion in the floor can lead to one or two wheels losing contact with the floor and cause slippage of the wheels. When this happens, it strongly influences the control of the rig. To improve this, a suspension system is proposed. The proposal consist of a spring-damper which ensures flexibility so that all wheels are in contact with the floor at all times. Figure (9.2) and Figure (9.3) illustrates the proposed suspension system for the Loomo rig.

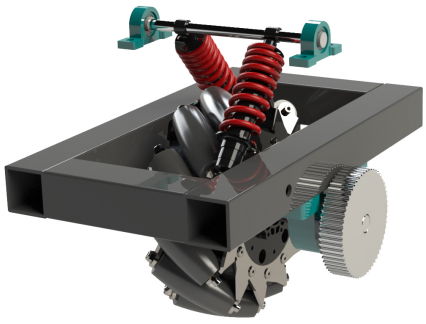


Figure 9.2: Suspension detail

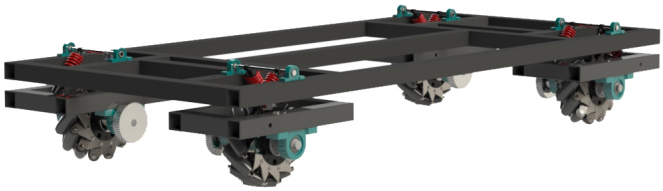


Figure 9.3: Suspension proposal

## 9.2 Further Work

The system documented in this report is not complete, and further work is necessary before the localization and navigation features presented can be implemented in reality. Several additions has to be further developed, and these are considered in this section. This includes a proposal of required models for measured data, and some physical adjustments for the Loomo rig.

### 9.2.1 Physical Odomoetry Model

Because the physical model was unavailable, a complete odometry model was not developed. The hardware selected for the rig does however have the prerequisites for creating an odometry model.

The motor is equipped with hall sensors which allows it to send feedback concerning the rotors position. This can be used together with the derived forward kinematics to estimate how far the robot has traveled. However, basing the motion of the robot solely on encoders is not sufficient because they are subject to cumulative errors. Additionally, errors can occur due to slippage or uneven terrain. The method can be quite accurate for a short distances, however over time the results will crucially deviate from the actual motion.

Additionally, the rig is equipped with four Kinects, which includes an embedded inertial measurement unit (IMU). The embedded 6DOF IMU consist of and accelerometer and a gyroscope. An accelerometer measures the forces it is subjected to and uses the gravitational force vector to determine the relative tilt angle. The value is directly measurable and is therefore characterized as long term-stable. The drawback of accelerometers is that they are very sensitive and therefore prone to external disturbances such as vibrations. Because of this, they are often referred to as short-term unstable. The gyroscope on the other hand is not prone to disturbances. It measures the rate of rotation, which means it has to be integrated over time to determine the tilt angle. When the integration is performed over a longer duration, the deviation from the actual angle becomes large. This means a gyroscope tends to drift over time and is characterized as short-term stable.

A common approach to obtain reliable measurements is to combine the best features of various sensors by fusing the information. A package which enables fusion of an arbitrary number of sensors is available in ROS. By incorporating multiple sensors, the estimate of where the robot is will be more certain compared to a single information source. The package is called *robot_localization* and features a node consisting of an extended kalman filter to combine information from different sensors. This package would be suitable for creating an odometry model for the physical Loomo rig by fusing the information from the hall sensors and the information from the IMU.

### 9.2.2 Conversion from Virtual to Physical Model

If the virtual navigation setup was to be implemented on the physical model, some important factors have to be considered. Because the ROS framework is based on sending and receiving standardized message types, the nodes does not care where the information is coming from. It only cares that the message type is correct and in theory, the model can not know if the information is coming from a physical or virtual sensor. Hence, it should be relatively easy to replace virtual sensors with physical sensors, as long as they are sending the same message type.

To setup the Azure Kinect together with ROS, a dedicated wrapper must be used. Fortunately, Microsoft has developed a ROS package for the sensor which enables it to publish information to the respective topics. This means that this package will transmit the same information as the virtual sensor used in the simulation model.[91]

The simulation model sends velocity commands to the gazebo planar move plugin. On the physical model, the Teensy would serve as the base controller. The program created for the Teensy does however subscribe to the same topic as the gazebo planar plugin, and for that reason it should be straight forward to swap the current base controller with the Teensy controller.

### 9.2.3 Initial Position

As previously stated, the AMCL node requires an initial position. When the node is started, the particles are spread uniformly around the environment, where all the particles have the same likelihood. By setting an approximate initial pose, the particles will converge to a smaller area and determine an accurate position faster. As for now, the initial position is set manually in RVIZ. A possible solution for automatically setting the initial position could be to save the latest position when the system is shut down, and load this the next time the system is powered on. There is however a flaw related to this approach. If the rig is manually moved while powered off, the robot will not be able to register the change in position and the last position will be invalid.

### 9.2.4 Mode Settings

A solution which lets the operator chose between the different modes have not been developed yet. However, the functionalities of the modes have been verified separately. A suitable approach could be to implement a switch on the hand controller, where the position of the switch determine the mode.

In manual mode, the low-level controller will bypass the navigation system, which means that it will not include any sort of obstacle avoidance. This lets the operator manually navigation into areas that is considered unavailable by the navigation system. The assisted actuation mode also does not implement obstacle avoidance, because the operator manually manipulate the velocity of the vehicle and act as the feedback for the system. For both the autonomous and ArUco tracking mode, obstacle avoidance is implemented. The robot is considered as semi-autonomous because it relays on instructions from the operator, such as an initial position at start-up and the desired goal position in the map.

### 9.2.5 Battery Pack

The rig is equipped with four brushless motors that have a maximum current draw of 50 ampere each. It is beneficial to develop a battery pack that can supply the full current demand instead of the utilized lead acid batteries. An improved lead acid battery would significantly increase the weight on the rig in order to meet current requirements, which can compromise rig controllability. The use of a lithium battery pack will reduce the extra weight to effect ratio, compared to lead acid batteries. However, this will introduce a greater cost. Another option, is to develop a custom battery pack with i.e Li-ion cells to meet large current demands, which will improve the rig performance. In addition, the power storage can be specialized to operating time demand.

### 9.2.6 Mounting of Cameras

The cameras are not yet mounted on the rig, however, the simulations have proven that the chosen mounting location is satisfactory. A way to mount the cameras is presented in Figure (9.4). If doing it this way, it is important to properly fix the beam to the vertical frame to minimize the osculations and vibration, which could potentially produce noise in the images.

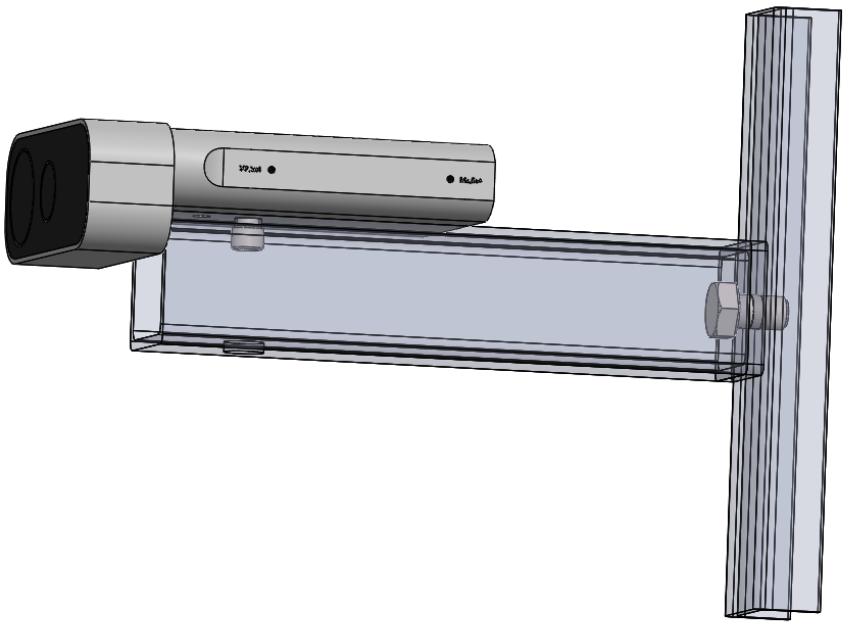


Figure 9.4: Azure mounting proposal

**Robot Operating Systems**

All of the ROS files associated with the project is available in the GitHub repository:

https://github.com/didrif/megatronds

In Appendix (H), the perquisite required after downloading the project is described in order to set up the workspace.

# 10. Conclusion

This thesis covers the development of a holonomic load-carrying rig with a rated capacity of 400 kg. The rig is intended to simplify the process of transporting ten units of Segway Robotics Loomo. To accomplish the goal, a mechanical and electrical design had to be established, in addition to a navigation system.

The mechanical part of the report documents the design of a load bearing axle, bearing calculations and weld verification. Further on, the rig was equipped with a holonomic locomotion technique. The omnidirectional motion of the rig is produced by four mecanum wheels, driven by four brushless DC motors. In order to gain adequate control over the rig, the kinematics for mecanum wheels had to be derived. The kinematics were verified both on the physical and simulated model. The physical rig was concluded to be stable and able to hold the designed weight.

The developed system rely on communication between the components. The main processing unit communicates with a microcontroller unit via UART communication. Further on, the messages between the microcontroller and the electronic motor controllers were establishment using CAN-bus communication. The communication method was verified and was able to run the motors with a low level steering mechanism.

The perception source for this project was multiple depth cameras. Four Azure Kinects, with 120° each, provide 360° perception around the robot. A 3D image contains an excessive amount of data and requires a lot of computational power to process. To ease the processing demand, only the important information from the 3D image was extracted and transformed into a 2D representation. This was done for all four cameras, and ultimately, the four 2D reconstructed images were merged into a single 2D representation. This showed to be a good approach in simulation, because all obstacles were registered and the processing demand was manageable. It was however not possible to verify the concept on the physical model.

Three modes for controlling the motion of the rig are evaluated in this report; manual mode, assistive actuation mode and autonomous mode. The simplest of them is manual control which is based on controlling the rig by a joystick based hand controller. This was implemented and verified on the physical model. The remaining two modes were developed and tested in a simulation model, but not the physical model. The next mode considers assistive actuation. The idea was to use a tracking camera to interpret the pose of a hand. This enables the rig to be pushed around in the environment without actually touching the object. This proved to be a feasible and efficient method for controlling the robot. The autonomous mode is based on the ROS navigation stack. Because the intended operation area of the robot was known, a map based localization was chosen. The Adaptive Monte Carlo Localization algorithm was utilized to localize the rig in a pre-made map of the environment. This proved to be a robust localization method, also in a dynamic environment. The Dijkstra's shortest path algorithm was chosen as the global planner, whereas the dynamic window approach was chosen as the local planner. Together, they proved to create and execute efficient and feasible paths based on the provided map and the obstacle in the environment. An additional feature in the autonomous mode was a human following concept. Unfortunately, this mode was not successfully implemented, but it was determined to be achievable nonetheless. The overall results of the simulation model verify that most of the modes are feasible, and if tuned properly the robot can navigate in an accurate and rational manner.

The methods used are not specific for this product, and are applicable for other industrial mobile robots which rely on human interactions. The concepts discussed in this paper has been deemed feasible by the simulation model. However, in order to obtain good results on the physical model, some improvements must be made. The locomotion technique related to mecanum wheels is prone to slippage, and this was observed on the physical model. To reduce the slippage, a proposal is made to add a suspension system which ensures that all wheels are in contact with the ground at all times. Additionally, the odometry used in the simulation model is ideal, however in a realistic scenario, the odometry is affected by noise and uncertainty. The robot is equipped with motors that have built in hall sensors, and cameras with embedded IMUs. This facilitates the development of a odometry model by fusing the information together. A robot intended to share an environment together with humans must ensure a high level of safety, something which the current rig lacks. Some further work regarding safety was discussed and should be accounted for if the prototype is to be developed further. To conclude, the product developed in this thesis is not complete, but it serves as a good foundation for further development of a load carrying platform.

# Bibliography


[1] K. G. Robbersmyr, *MAS-102 3. Utmatting.* Fronter, 2016.

[2] KOYO. Purpose and method of lubrication. [Online]. Available: https://koyo.jtekt.co.jp/en/support/bearing-knowledge/12-1000.html

[3] M. Teensy 3.6 development board. [Online]. Available: https://microcontrollerslab.com/teensy-3-6-development-board-pinout/

[4] F. Tech. Dual can-bus adapter for teensy 3.5, 3.6. [Online]. Available: https://www.tindie.com/products/Fusion/dual-can-bus-adapter-for-teensy-35-36/?utm_source=twitter&utm_medium=twitter&utm_campaign=product_back_in_stock_tweets

[5] Maytech. Maytech superfoc6.8 50a vesc6-based speed controller compatible to vesctool programmable for esk8/ebike. [Online]. Available: https://maytech.cn/collections/all-speed-controllers/products/maytech-superfoc6-8-with-dissipation-case-50a-speed-controller-based-on-vesc6

[6] MayTech. Maytech brushless 5065 70/220kv open cover outrunner sensored motor for esk8/e-bike/robotics. [Online]. Available: https://maytech.cn/products/brushless-hall-sensor-motor-mto5065-220-ha

[7] Nvidia. Jetson agx xavier developer kit. [Online]. Available: https://developer.nvidia.com/embedded/jetson-agx-xavier-developer-kit

[8] Exsys. Ex-11087 - interface card 7x usb 3.0 pci-e x1, exsys. [Online]. Available: https://www.elfadistrelec.no/no/interface-card-7x-usb-pci-x1-exsys-ex-11087/p/11033532?q=pci+usb&pos=26&origPos=22&origPageSize=100&track=true

[9] Amazon. Vision cp12200/12 v 20ah agm lead battery. [Online]. Available: https://www.amazon.co.uk/Vision-CP12200-20Ah-AGM-Battery/dp/B007GY6AK2

[10] AliExpress. High quality dual-axis xy joystick module ps2 joystick control lever sensor ky-023 for arduino diy kit. [Online]. Available: https://www.aliexpress.com/item/1954188480.html

[11] elinux.org. Jetson/agx xavier misc interfaces. [Online]. Available: https://elinux.org/Jetson/AGX_Xavier_Misc_Interfaces

[12] Microsoft. Azure kinect hardware specifications. [Online]. Available: https://docs.microsoft.com/nb-no/azure/Kinect-dk/hardware-specification

[13] M. Drwięga and J. Jakubiak, "A set of depth sensor processing ros tools for wheeled mobile robot navigation," *Journal of Automation Mobile Robotics and Intelligent Systems*, vol. 11, 2017.

[14] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics*. MIT press Cambridge, 2000, vol. 1.

[15] E. Marder-Eppstein. move_base. [Online]. Available: http://wiki.ros.org/move_base

[16] OpenCV. Detection of aruco markers. [Online]. Available: https://docs.opencv.org/trunk/d5/dae/tutorial_aruco_detection.html

[17] N. Level. Leap motion controller. [Online]. Available: https://www.nlevel.ru/katalog/umnye-gadzhety/datchik-dvizheniia-leap-motion-controller/

[18] K. G. Robbersmyr, *MAS-102 Lager*. Fronter, 2016.

[19] Segway. Loomo. [Online]. Available: https://store.segway.com/segway-loomo-mini-transporter-robot-sidekick

[20] M. Abdelrahman, I. Zeidis, O. Bondarev, B. Adamov, F. Becker, and K. Zimmermann, "A description of the dynamics of a four wheel mecanum mobile system as a basis for a platform concept for special purpose vehicles for disabled persons," in *58-th Ilmenau Scientific Colloquium*, 2014.

[21] AndyMark. 8 in. mk mecanum wheels. [Online]. Available: https://www.andymark.com/products/8-in-mk-mecanum-wheel-set-options?via=Z2lkOi8vYW5keW1hcmsvV29ya2FyZWE6OkNhdGFsb2c6OkNhdGVnb3J5LzVhZjhlMjIyYmM2Z

[22] J. Sali, A. Adom, and S. Yaacob. A design of omni-directional for mobile robot. [Online]. Available: https://www.researchgate.net/publication/268326364_A_Design_Of_Omni-Directional_For_Mobile_Robot

[23] L. Yunwang, S. Dai, L. Zhao, and Et-al, "Topological design methods for mecanum wheel configurations of an omnidirectional mobile robot," *https://www.mdpi.com/2073-8994/11/10/1268/htm*, vol. 3, 2019.

[24] N. Sonawane, "An experimental method to calculate coefficient of friction in mecanum wheel rollers and cost analysis using dfma techniques," 2015.

[25] Hamilton. Rolling resistance and industrial wheels. [Online]. Available: https://www.mhi.org/media/members/14220/130101690137732025.pdf

[26] E. ToolBox. Rolling resistance. [Online]. Available: https://www.engineeringtoolbox.com/rolling-friction-resistance-d_1303.html

[27] S.-L. Wang. Motion control and the skidding of mecanum-wheel vehicles. [Online]. Available: http://ijiset.com/vol5/v5s5/IJISET_V5_I05_10.pdf

[28] Z. HENDZEL and L. RYKALA. Modelling of dynamics of a wheeled mobile robot with mecanum wheels with the use of lagrange equations of the second kind. [Online]. Available: https://www.researchgate.net/publication/315058609_Modelling_of_Dynamics_of_a_Wheeled_Mobile_Robot_with_Mecanum_Wheels_with_the_use_of_Lagrange_Equations_of_the_Second_Kind

[29] K. G. Robbersmyr, *MAS-102 Aksler*. Fronter, 2016.

[30] J. M. Gere and B. J. Goodno, "Mechanics of materials eight edition," 2009.

[31] S. K. Armah. Preliminary design of a power transmission shaft under fatigue loading using asme code. [Online]. Available: https://thescipub.com/pdf/10.3844/ajeassp.2018.227.244

[32] R. C. Juvinall and K. M. Marshek, *The Fundamentals of Machine Component Design*. pages 600-615: John Wiley & Sons, INC., 2012.

[33] SKF. Pillow block ball bearing units, ucp204. [Online]. Available: https://www.skf.com/group/products/mounted-bearings/ball-bearing-units/pillow-block-ball-bearing-units/productid-Y%2FUCP%20204%2FH?system=metric

[34] KOYO. Lubricant. [Online]. Available: https://koyo.jtekt.co.jp/en/support/bearing-knowledge/12-2000.html

[35] UNBRAKO. Unbrako enginnering guide. [Online]. Available: http://www.unbrako.com/images/downloads/engguide.pdf

[36] S. socket. Set screw tightening torque. [Online]. Available: http://www.safetysocket.com/sites/www.safetysocket.com/files/parts/bd/Products/setscrewtorque.htm

[37] UNILOK. Socket set screw. [Online]. Available: http://www.harjivandashathibhai.com/Unilok/PDF/SSS.pdf

[38] MITcalc. Weld connections. [Online]. Available: http://www.mitcalc.com/doc/welding/help/en/welding.htm

[39] A. Middleton, S. Fritz, and M. Lusardi. Walking speed: The functional vital sign. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4254896/

[40] R. C. Juvinall and K. M. Marshek, *The Fundamentals of Machine Component Design*. page 789: John Wiley & Sons, INC., 2012.

[41] S. Mraz. Tension in timing-belt drives. [Online]. Available: https://www.machinedesign.com/archive/article/21812296/tension-in-timingbelt-drives

[42] R. C. Juvinall and K. M. Marshek, *The Fundamentals of Machine Component Design*. pages 782-789: John Wiley & Sons, INC., 2012.

[43] Modelflight. What is an electronic speed controller and how does it differ from brushed to brushless motors? [Online]. Available: https://www.modelflight.com.au/blog/electronic-speed-controllers

[44] Wikipedia. Peukert's law. [Online]. Available: https://en.wikipedia.org/wiki/Peukert%27s_law

[45] MultiCable. Cross reference awg to mm2. [Online]. Available: https://www.multicable.com/resources/reference-data/cross-reference-awg-to-mm2/

[46] Metroid. How to calculate voltage drop. [Online]. Available: https://www.metroid.net.au/engineering/calculate-voltage-drop/

[47] Wikipedia. American wire gauge. [Online]. Available: https://en.wikipedia.org/wiki/American_wire_gauge

[48] C. Gillespie. How to calculate coulombs. [Online]. Available: https://sciencing.com/calculate-coulombs-2645.html

[49] E. toolbox. Specific heat of some metals. [Online]. Available: https://www.engineeringtoolbox.com/specific-heat-metals-d_152.html

[50] E. Edge. Awg copper wire table size and data. [Online]. Available: https://www.engineersedge.com/copper_wire.htm

[51] myElctrical Engineering. Cable insulation properties. [Online]. Available: https://myelectrical.com/notes/entryid/178/cable-insulation-properties

[52] Wikipedia. Polylactic acid. [Online]. Available: https://en.wikipedia.org/wiki/Polylactic_acid

[53] T. Zhang and W. Chen. An indoor mobile robot navigation technique using odometry and electronic compass. [Online]. Available: https://journals.sagepub.com/doi/full/10.1177/1729881417711643

[54] O. Robotics. Is ros for me? [Online]. Available: https://www.ros.org/is-ros-for-me/

[55] ——. Ros: Core components. [Online]. Available: https://www.ros.org/core-components/

[56] J. Hsu, N. Koenig, and D. Coleman. gazebo_ros_pkgs. [Online]. Available: http://wiki.ros.org/gazebo_ros_pkgs

[57] O. S. R. Foundation. Ros integration overview. [Online]. Available: http://gazebosim.org/tutorials?tut=ros_overview

[58] D. Hershberger, D. Gossow, and J. Faust. rviz. [Online]. Available: http://wiki.ros.org/rviz

[59] I. Sucan and J. Kay. urdf. [Online]. Available: http://wiki.ros.org/urdf

[60] W. Meeussen. ros_controllers. [Online]. Available: http://wiki.ros.org/ros_controllers

[61] O. S. R. Foundation. Using gazebo plugins with ros: Depth camera. [Online]. Available: http://gazebosim.org/tutorials?tut=ros_gzplugins#DepthCamera

[62] E. CSS. Can bus explained - a simple intro (2020). [Online]. Available: https://www.csselectronics.com/screen/page/simple-intro-to-can-bus/language/en

[63] N. Instruments. Can physical layer and termination guide. [Online]. Available: https://www.ni.com/en-no/innovations/white-papers/09/can-physical-layer-and-termination-guide.html

[64] B. Vedder. comm_can.c. [Online]. Available: https://github.com/vedderb/bldc/blob/master/comm_can.c#L1159

[65] P.-s. . Pcan-usb fd. [Online]. Available: https://www.peak-system.com/PCAN-USB-FD.365.0.html?&L=1

[66] B. Vedder. datatypes.h. [Online]. Available: https://github.com/vedderb/bldc/blob/master/datatypes.h#L766

[67] C. Kidder. Flexcan_library. [Online]. Available: https://github.com/collin80/FlexCAN_Library

[68] c. . Pointers. [Online]. Available: http://www.cplusplus.com/doc/tutorial/pointers/

[69] C. Basics. Basics of uart communication. [Online]. Available: https://www.circuitbasics.com/basics-uart-communication/

[70] S. Adam. rosserial. [Online]. Available: http://wiki.ros.org/rosserial_arduino

[71] J. Hacks. Nvidia jetson agx xavier gpio header pinout. [Online]. Available: https://www.jetsonhacks.com/nvidia-jetson-agx-xavier-gpio-header-pinout/

[72] J. Swaby. Changing file permissions with chmod. [Online]. Available: http://mindhive.mit.edu/node/1315

[73] V. Gite. Chmod numeric permissions notation unix / linux command. [Online]. Available: https://www.cyberciti.biz/faq/unix-linux-bsd-chmod-numeric-permissions-notation-command/

[74] PJRC. Using the hardware serial ports. [Online]. Available: https://www.pjrc.com/teensy/td_uart.html

[75] ROS. std_msgs/int32multiarray message. [Online]. Available: http://docs.ros.org/melodic/api/std_msgs/html/msg/Int32MultiArray.html

[76] T. Foote, E. Marder-Eppstein, and W. Meeussen. tf2. [Online]. Available: http://wiki.ros.org/tf2

[77] K. Zheng. Ros navigation tuning guide. [Online]. Available: http://kaiyuzheng.me/documents/navguide.pdf

[78] B. Gerkey and T. Pratkanis. Map server. [Online]. Available: http://wiki.ros.org/map_server

[79] E. Marder-Eppstein and D. V. Lu. costmap_2d. [Online]. Available: https://github.com/ros-planning/navigation/tree/melodic-devel/costmap_2d

[80] Computerphile. Dijkstra's algorithm. [Online]. Available: https://www.youtube.com/watch?v=GazC3A4OQTE

[81] M. Ottestad, "Path planning," 2019.

[82] E. Marder-Eppstein and K. Konolige. Navigation navfn. [Online]. Available: https://github.com/ros-planning/navigation/blob/indigo-devel/navfn/include/navfn/navfn.h

[83] E. Marder-Eppstein. dwa_local_planner. [Online]. Available: http://wiki.ros.org/dwa_local_planner

[84] D. Fox, W. Burgard, and S. Thrun, "Dynamic window approach to collision avoidance," *IEEE Robotics & Automation Magazine 4(1):23 - 33*, vol. 4, 1997.

[85] O. Brock and O. Khatib, "High-speed navigation using the global dynamic window approach," *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No. 99CH36288C), pages 341-346*, vol. 1, 1999.

[86] J. Vaughan. fiducials. [Online]. Available: http://wiki.ros.org/fiducials?distro=melodic

[87] R. M. Salinas and B. Magyar. aruco. [Online]. Available: http://wiki.ros.org/aruco

[88] J. Vaughan. aruco_detect. [Online]. Available: http://wiki.ros.org/aruco_detect?distro=melodic

[89] A. Colgan. How does the leap motion controller work? [Online]. Available: http://blog.leapmotion.com/hardware-to-software-how-does-the-leap-motion-controller-work/

[90] F. Lier, M. Shah, and I. Saito. leap_motion. [Online]. Available: http://wiki.ros.org/leap_motion

[91] Microsoft. Azure_kinect_ros_driver. [Online]. Available: https://github.com/microsoft/Azure_Kinect_ROS_Driver

[92] P. . Udev rules for teensy boards, http://www.pjrc.com/teensy/. [Online]. Available: https://www.pjrc.com/teensy/49-teensy.rules

# Appendices

# A. Teensy 3.6 Scripts

## A.1 Loomo Parking Rig Teensy Program

```
   // Defining the Teensy 3.6 UART hardware serials
#define USE_TEENSY_HW_SERIAL
#include <ros.h>
#include <std_msgs/Float32MultiArray.h>
#include <std_msgs/Int32MultiArray.h>
#include <FlexCAN.h>
#ifndef __MK66FX1M0__
  #error "Teensy 3.6 with CAN bus is required"
#endif
// Initilaziing Kinematics variables
#define r 0.1015
#define w 0.2850
#define h 0.6350
#define joystick_x 14
#define joystick_y 15
#define joystick_theta 16
#define manual_mode_pin 10
#define emergency_stop_pin 11
bool manual_mode = false;
bool emergency_stop = false;
//-----------------------------------------------------------------
float ReadJoystick(int joystick, int joystick_val) {
  joystick_val = map(analogRead(joystick), 0, 1023, -7, 7);
  return joystick_val;
}
//------------------------------//
float x_dot = 0;
float y_dot = 0;
float theta_dot = 0;
//-----------------------------------------------------------------------
float pi = 3.14159;
int32_t no_poles = 7;
float state_ref[3] = {-3.5,0,0}; // Used for subscribing refrence velocities
    sent from Xavier
//float omega[4] = {0,0,0,0}; //Initializing the "vector" with x_dot,y_dot and
     theta_d
int32_t omega[4] = {0,0,0,0};
int32_t omega_ref[4] = {0,0,0,0};
//float e_rpm_float[4] = {0,0,0,0};
//-----------------------------------------------------------------------

// Creating the calss to define the serial port to use
class NewHardware : public ArduinoHardware
{
  public:
  NewHardware():ArduinoHardware(&Serial1){}; // Specify port
  // NewHardware():ArduinoHardware(&Serial1, 57600){}; // Specify port
};

// Creating Nodehandle
ros::NodeHandle nh;
```

```
//--------- PUBLISHING ----------------------------------//
//Instantiating the message
std_msgs::Int32MultiArray msg_wheel_omega;
//Defining Publisher
ros::Publisher pub_omega("wheel_omega_all", &msg_wheel_omega);
//--------- PUBLISHING END ------------------------------//

//--------- SUBSCRIBABLE --------------------------------//
//Defining publishing topic function
void state_ref_array(const std_msgs::Float32MultiArray& state_ref_msg){
  state_ref[0] = state_ref_msg.data[0];
  state_ref[1] = state_ref_msg.data[1];
  state_ref[2] = state_ref_msg.data[2];
}

//Defining subscribing topic
ros::Subscriber<std_msgs::Float32MultiArray> sub_state("state_ref",
   state_ref_array);
//--------- SUBSCRIBABLE END ----------------------------//

static struct CAN_filter_t defaultMask;
static CAN_message_t msg_can_vesc_1;
static CAN_message_t msg_can_vesc_2;
static CAN_message_t msg_can_vesc_3;
static CAN_message_t msg_can_vesc_4;
static CAN_message_t inMsg;

int32_t e_rpm_1, e_rpm_2, e_rpm_3, e_rpm_4;
int32_t duty_cycle_1, duty_cycle_2, duty_cycle_3, duty_cycle_4;
int32_t current_1, current_2, current_3, current_4;
uint16_t Node_ID_1 = 0x01;
uint16_t Node_ID_2 = 0x02;
uint16_t Node_ID_3 = 0x03;
uint16_t Node_ID_4 = 0x04;

// ------------------------------------------------------------
void buffer_append_int32(uint8_t *buffer, int32_t number, int32_t *index) {
 // Function to append the value used in write_can_message function
 buffer[(*index)++] = number >> 24;
 buffer[(*index)++] = number >> 16;
 buffer[(*index)++] = number >> 8;
 buffer[(*index)++] = number;
}
//------------------------------------------------------------
void write_can_message(CAN_message_t msg, uint32_t can_id, int32_t value){
 // Function to fill in necessary information to write a CAN message
 msg.ext = 1; // If the id is extended or not (0=11bit ID, 1=29bit ID)
 msg.id = can_id; //Message id. This contains the command and which unit it
    is ment for
 msg.len = 4; // Message length (2= 0x0000, 4 = 0x00000000, 8 = 0
    x0000000000000000, etc...)
 uint8_t buffer [sizeof(msg.len)]; //Create an empty buffer to write message
    in
 int32_t send_index = 0; // Selecting which index to start the appending
 buffer_append_int32(buffer , value, &send_index); // Appends "value" to the
    buffer
 memcpy(msg.buf, buffer, msg.len*sizeof(uint32_t)); // Copies the buffer
    information into the message (msg.buf)
 Can1.write(msg); // Write CANBus command
}
// ------------------------------------------------------------
```

```
int16_t read_buffer_int16(const uint8_t *buffer, int32_t *index){
  int16_t get_buffer_value = ((uint16_t) buffer[*index]) << 8 |
                        ((uint16_t)buffer[*index+1]);
  return get_buffer_value;
}
// ------------------------------------------------------------
int16_t get_dec_value (uint8_t *bytePtr, int index){
  int32_t send_index = index;
  int16_t value_decimal = read_buffer_int16(bytePtr, &send_index);
  return value_decimal;
}
// ------------------------------------------------------------
  int32_t read_erpm(uint16_t can_node_id, int stored_e_rpm){
  // Function to read the ERPM sent from the VESC
  int32_t e_rpm_read;
  if(inMsg.id == (0x01 << 8 | can_node_id)){
    //if(inMsg.id == (0x901)){
    // In order to merge the 4 bytes, the data length 0 and 1 are left shifted
       16 bits
    // 0x0000ABCD << 16 -> 0xABCD0000, 0xABCD0000 | 0x0000abcd -> 0xABCDabcd
    e_rpm_read = get_dec_value(inMsg.buf,0) << 16 | get_dec_value(inMsg.buf,2);
  }
  else {
    /*
    If "dead-time" happens with signal conflict, resulting in no new data,
    the previous value is used instead of using a zero value. This is done
    to avoid jittering
    */
    e_rpm_read = stored_e_rpm;
  }
  return e_rpm_read;
}
// ------------------------------------------------------------------
  int16_t read_current(uint16_t can_node_id, int stored_current){
  // Function to read the ERPM sent from the VESC
  int16_t current_read;
  if(inMsg.id == (0x02 << 8 | can_node_id)){
    // Current is stored in byte 4 and 5
    current_read = get_dec_value(inMsg.buf,4);
  }
  else {
    current_read = stored_current;
  }
  return current_read;
}
// ------------------------------------------------------------------
  int16_t read_duty_cycle(uint16_t can_node_id, int stored_duty_cycle){
  // Function to read the ERPM sent from the VESC
  int16_t duty_cycle_read;
  if(inMsg.id == (0x03 << 8 | can_node_id)){
    // Duty cycle is stored in byte 6 and 7
    duty_cycle_read = get_dec_value(inMsg.buf,6);
  }
  else {
    duty_cycle_read = stored_duty_cycle;
  }
  return duty_cycle_read;
}
//------------------------------------------------------------------
void setup(void){
   //Setting baudrate
```

```
  nh.getHardware()->setBaud(57600);
  //Initializing node
  nh.initNode();
  //Start subscribing topic
  nh.subscribe(sub_state);
  //Start advertising topic (publishing)
  nh.advertise(pub_omega);
  delay(1000);
  Serial.println(F("Teensy 3.6 dual connected to second CAN is initialized."))
      ;
  Can1.begin(500000,defaultMask,0,0);
  pinMode(35, OUTPUT);
  pinMode(manual_mode_pin, INPUT);
  pinMode(emergency_stop_pin, INPUT);
  digitalWrite(35, LOW);
}
// -------------------------------------------------------------
void loop(void){
  /*manual_mode = digitalRead(manual_mode_pin);
  emergency_stop = digitalRead(emergency_stop);
  if (manual_mode == true && emergency_stop == false){
    x_dot = ReadJoystick(joystick_x,0);
    y_dot = ReadJoystick(joystick_y,0);
    theta_dot = ReadJoystick(joystick_theta,0);
  }
  else if (manual_mode == false && emergency_stop == false){
    x_dot = state_ref[0];
    y_dot = state_ref[1];
    theta_dot = state_ref[2];
  }
  else{
    x_dot = 0;
    y_dot = 0;
    theta_dot = 0;
  }*/
     x_dot = state_ref[0];
    y_dot = state_ref[1];
    theta_dot = state_ref[2];
 // CHECK IF WHEEL NO 3 and 4 has to change sign. ie -1/r --> 1/r
  omega_ref[0] = -1/r*(x_dot+y_dot+(-w-h)*theta_dot);
  omega_ref[1] = -1/r*(-x_dot+y_dot+(w+h)*theta_dot);
  omega_ref[2] = -1/r*(x_dot+y_dot+(w+h)*theta_dot);
  omega_ref[3] = -1/r*(-x_dot+y_dot+(-w-h)*theta_dot);
  write_can_message(msg_can_vesc_1, 0x301, omega_ref[0]*no_poles*60/2/pi); //
     0x03(set RPM) | 0xID
  write_can_message(msg_can_vesc_2, 0x302, omega_ref[1]*no_poles*60/2/pi);
  write_can_message(msg_can_vesc_3, 0x303, omega_ref[2]*no_poles*60/2/pi);
  write_can_message(msg_can_vesc_4, 0x304, omega_ref[3]*no_poles*60/2/pi);
  while (Can1.available())
  {
    Can1.read(inMsg);
    e_rpm_1 = read_erpm(Node_ID_1, e_rpm_1);
    e_rpm_2 = read_erpm(Node_ID_2, e_rpm_2);
    e_rpm_3 = read_erpm(Node_ID_3, e_rpm_3);
    e_rpm_4 = read_erpm(Node_ID_4, e_rpm_4);
    current_1 = read_current(Node_ID_1,current_1);
    current_2 = read_current(Node_ID_2,current_2);
    duty_cycle_1 = read_duty_cycle(Node_ID_1,duty_cycle_1);
    duty_cycle_2 = read_duty_cycle(Node_ID_2,duty_cycle_2);
  }
 /* e_rpm_float[0] = (float)e_rpm_1;
```

```
  e_rpm_float[1] = (float)e_rpm_2;
  e_rpm_float[2] = (float)e_rpm_3;
  e_rpm_float[3] = (float)e_rpm_4;
  omega[0] = e_rpm_float[0]///no_poles*2*pi/60;
  omega[1] = e_rpm_float[1]///no_poles*2*pi/60;
  omega[2] = e_rpm_float[2]///no_poles*2*pi/60;
  omega[3] = e_rpm_float[3]///no_poles*2*pi/60; */
  omega[0] = e_rpm_1/no_poles*2*pi/60;
  omega[1] = e_rpm_2/no_poles*2*pi/60;
  omega[2] = e_rpm_3/no_poles*2*pi/60;
  omega[3] = e_rpm_4/no_poles*2*pi/60;
  msg_wheel_omega.data = omega;
  msg_wheel_omega.data_length =4;
  pub_omega.publish(&msg_wheel_omega);
  delay(250);
  nh.spinOnce();
}
```

## A.2 Teensy and Xavier two way communication

```
// Defining the Teensy 3.6 UART hardware serials
#define USE_TEENSY_HW_SERIAL
#include <ros.h>
#include <std_msgs/Int32MultiArray.h>
#include <FlexCAN.h>
#ifndef __MK66FX1M0__
  #error "Teensy 3.6 with CAN bus is required"
#endif

//-----------------------------------------------------------------------
int32_t no_poles = 7;
int32_t value_ref[4] = {0,0,0,0}; // Used for subscribing refrence velocities
    sent from Xavier
              //[ wheel1; wheel2; wheel3; wheel4]
int32_t value[4] = {0,0,0,0}; //Initializing the "vector" with the measured
    wheel velocities
//-----------------------------------------------------------------------

// Creating the calss to define the serial port to use
class NewHardware : public ArduinoHardware
{
  public:
  NewHardware():ArduinoHardware(&Serial1){}; // Specify port
  // NewHardware():ArduinoHardware(&Serial1, 57600){}; // Specify port
};

// Creating Nodehandle
ros::NodeHandle nh;


//Instantiating the message
std_msgs::Int32MultiArray msg_wheel_omega;
//Defining Publisher
ros::Publisher pub_omega("wheel_omega_all", &msg_wheel_omega);

//Defining topic function
void omega_ref_array(const std_msgs::Int32MultiArray& omega_ref_msg){
```

```
  value_ref[0] = omega_ref_msg.data[0];
  value_ref[1] = omega_ref_msg.data[1];
  value_ref[2] = omega_ref_msg.data[2];
  value_ref[3] = omega_ref_msg.data[3];
}

//Defining subscribing topic
ros::Subscriber<std_msgs::Int32MultiArray> sub_omega("omega_ref",
   omega_ref_array);


static struct CAN_filter_t defaultMask;
static CAN_message_t msg_can_vesc_1;
static CAN_message_t msg_can_vesc_2;
static CAN_message_t msg_can_vesc_3;
static CAN_message_t msg_can_vesc_4;
static CAN_message_t inMsg;

int32_t e_rpm_1, e_rpm_2, e_rpm_3, e_rpm_4;
int32_t duty_cycle_1, duty_cycle_2, duty_cycle_3, duty_cycle_4;
int32_t current_1, current_2, current_3, current_4;
uint16_t Node_ID_1 = 0x01;
uint16_t Node_ID_2 = 0x02;
uint16_t Node_ID_3 = 0x03;
uint16_t Node_ID_4 = 0x04;

// -------------------------------------------------------------
void buffer_append_int32(uint8_t *buffer, int32_t number, int32_t *index) {
 // Function to append the value used in write_can_message function
 buffer[(*index)++] = number >> 24;
 buffer[(*index)++] = number >> 16;
 buffer[(*index)++] = number >> 8;
 buffer[(*index)++] = number;
}
//-------------------------------------------------------------
void write_can_message(CAN_message_t msg, uint32_t can_id, int32_t value){
 // Function to fill in necessary information to write a CAN message
 msg.ext = 1; // If the id is extended or not (0=11bit ID, 1=29bit ID)
 msg.id = can_id; //Message id. This contains the command and which unit it
    is ment for
 msg.len = 4; // Message length (2= 0x0000, 4 = 0x00000000, 8 = 0
    x0000000000000000, etc...)
 uint8_t buffer [sizeof(msg.len)]; //Create an empty buffer to write message
    in
 int32_t send_index = 0; // Selecting which index to start the appending
 buffer_append_int32(buffer , value, &send_index); // Appends "value" to the
    buffer
 memcpy(msg.buf, buffer, msg.len*sizeof(uint32_t)); // Copies the buffer
    information into the message (msg.buf)
 Can1.write(msg); // Write CANBus command
}
// -------------------------------------------------------------
int16_t read_buffer_int16(const uint8_t *buffer, int32_t *index){
 int16_t get_buffer_value = ((uint16_t) buffer[*index]) << 8 |
                      ((uint16_t)buffer[*index+1]);
 return get_buffer_value;
}
// -------------------------------------------------------------
int16_t get_dec_value (uint8_t *bytePtr, int index){
 int32_t send_index = index;
 int16_t value_decimal = read_buffer_int16(bytePtr, &send_index);
```

```
 return value_decimal;
}
// ---------------------------------------------------------------
 int32_t read_erpm(uint16_t can_node_id, int stored_e_rpm){
 // Function to read the ERPM sent from the VESC
 int32_t e_rpm_read;
 if(inMsg.id == (0x01 << 8 | can_node_id)){
  //if(inMsg.id == (0x901)){
  // In order to merge the 4 bytes, the data length 0 and 1 are left shifted
     16 bits
  // 0x0000ABCD << 16 -> 0xABCD0000, 0xABCD0000 | 0x0000abcd -> 0xABCDabcd
  e_rpm_read = get_dec_value(inMsg.buf,0) << 16 | get_dec_value(inMsg.buf,2);
 }
 else {
  /*
  If "dead-time" happens with signal conflict, resulting in no new data,
  the previous value is used instead of using a zero value. This is done
  to avoid jittering
  */
  e_rpm_read = stored_e_rpm;
 }
 return e_rpm_read;
}
// ---------------------------------------------------------------
 int16_t read_current(uint16_t can_node_id, int stored_current){
 // Function to read the ERPM sent from the VESC
 int16_t current_read;
 if(inMsg.id == (0x02 << 8 | can_node_id)){
  // Current is stored in byte 4 and 5
  current_read = get_dec_value(inMsg.buf,4);
 }
 else {
  current_read = stored_current;
 }
 return current_read;
}
// ---------------------------------------------------------------
 int16_t read_duty_cycle(uint16_t can_node_id, int stored_duty_cycle){
 // Function to read the ERPM sent from the VESC
 int16_t duty_cycle_read;
 if(inMsg.id == (0x03 << 8 | can_node_id)){
  // Duty cycle is stored in byte 6 and 7
  duty_cycle_read = get_dec_value(inMsg.buf,6);
 }
 else {
  duty_cycle_read = stored_duty_cycle;
 }
 return duty_cycle_read;
}
//---------------------------------------------------------------
void setup(void){
   //Setting baudrate
 nh.getHardware()->setBaud(57600);
 //Initializing node
 nh.initNode();
 //Start subscribing topic
 nh.subscribe(sub_omega);
 //Start advertising topic (publishing)
 nh.advertise(pub_omega);
 delay(1000);
```

```
  Serial.println(F("Teensy 3.6 dual connected to second CAN is initialized."))
      ;
  Can1.begin(500000,defaultMask,0,0);
  pinMode(35, OUTPUT);
  digitalWrite(35, LOW);
}
// -------------------------------------------------------------
void loop(void){
  write_can_message(msg_can_vesc_1, 0x301 , value_ref[0]); // 0x03(set RPM) |
      0xID
  write_can_message(msg_can_vesc_2, 0x302 , value_ref[1]);
  write_can_message(msg_can_vesc_3, 0x303 , value_ref[2]);
  write_can_message(msg_can_vesc_4, 0x304 , value_ref[3]);
  while (Can1.available())
  {
    Can1.read(inMsg);
    e_rpm_1 = read_erpm(Node_ID_1, e_rpm_1);
    e_rpm_2 = read_erpm(Node_ID_2, e_rpm_2);
    e_rpm_3 = read_erpm(Node_ID_3, e_rpm_3);
    e_rpm_4 = read_erpm(Node_ID_4, e_rpm_4);
    current_1 = read_current(Node_ID_1,current_1);
    current_2 = read_current(Node_ID_2,current_2);
    duty_cycle_1 = read_duty_cycle(Node_ID_1,duty_cycle_1);
    duty_cycle_2 = read_duty_cycle(Node_ID_2,duty_cycle_2);
  }
  value[0] = e_rpm_1/7;
  value[1] = e_rpm_2/7;
  value[2] = e_rpm_3/7;
  value[3] = e_rpm_4/7;
  msg_wheel_omega.data = value;
  msg_wheel_omega.data_length =4;
  pub_omega.publish(&msg_wheel_omega);
  delay(250);
  nh.spinOnce();
}
```

## A.3 Publish data from CAN to ROS

```
// Defining the Teensy 3.6 UART hardware serials
#define USE_TEENSY_HW_SERIAL
#include <ros.h>
#include <std_msgs/Int32MultiArray.h>
#include <FlexCAN.h>
#ifndef __MK66FX1M0__
  #error "Teensy 3.6 with dual CAN bus is required"
#endif


// Creating the calss to define the serial port to use
class NewHardware : public ArduinoHardware
{
  public:
  NewHardware():ArduinoHardware(&Serial1){}; // Specify port
  // NewHardware():ArduinoHardware(&Serial1, 57600){}; // Specify port
};

// Creating Nodehandle
```

```
ros::NodeHandle nh;

//Instantiating the message
std_msgs::Int32MultiArray msg_wheel_omega;


//Defining Publisher
ros::Publisher wheel_omega("wheel_omega_all", &msg_wheel_omega);


static struct CAN_filter_t defaultMask;
static CAN_message_t inMsg;

int32_t e_rpm_1, e_rpm_2;
int32_t duty_cycle_1, duty_cycle_2;
int32_t current_1, current_2;
uint16_t Node_ID_1 = 0x01;
uint16_t Node_ID_2 = 0x02;

// ---------------------------------------------------------------
int16_t read_buffer_int16(const uint8_t *buffer, int32_t *index){
  int16_t get_buffer_value = ((uint16_t) buffer[*index]) << 8 | ((uint16_t)
     buffer[*index+1]);
  return get_buffer_value;
}
// ---------------------------------------------------------------
int16_t get_dec_value (uint8_t *bytePtr, int index){
  int32_t send_index = index;
  int16_t value_decimal = read_buffer_int16(bytePtr, &send_index);
  return value_decimal;
}
// ---------------------------------------------------------------
  int32_t read_erpm(uint16_t can_node_id, int stored_e_rpm){
  // Function to read the ERPM sent from the VESC
  int32_t e_rpm_read;
  if(inMsg.id == (0x01 << 8 | can_node_id)){
    //if(inMsg.id == (0x901)){
    // In order to merge the 4 bytes, the data length 0 and 1 are left shifted
       16 bits
    // 0x0000ABCD << 16 -> 0xABCD0000, 0xABCD0000 | 0x0000abcd -> 0xABCDabcd
    e_rpm_read = get_dec_value(inMsg.buf,0) << 16 | get_dec_value(inMsg.buf,2);
  }
  else {
    /*
    If "dead-time" happens with signal conflict, resulting in no new data,
    the previous value is used instead of using a zero value. This is done
    to avoid jittering
    */
    e_rpm_read = stored_e_rpm;
  }
  return e_rpm_read;
}
// ---------------------------------------------------------------
  int16_t read_current(uint16_t can_node_id, int stored_current){
  // Function to read the ERPM sent from the VESC
  int16_t current_read;
  if(inMsg.id == (0x02 << 8 | can_node_id)){
    // Current is stored in byte 4 and 5
    current_read = get_dec_value(inMsg.buf,4);
  }
  else {
```

```
    current_read = stored_current;
  }
  return current_read;
}
// ---------------------------------------------------------------
  int16_t read_duty_cycle(uint16_t can_node_id, int stored_duty_cycle){
  // Function to read the ERPM sent from the VESC
  int16_t duty_cycle_read;
  if(inMsg.id == (0x03 << 8 | can_node_id)){
    // Duty cycle is stored in byte 6 and 7
    duty_cycle_read = get_dec_value(inMsg.buf,6);
  }
  else {
    duty_cycle_read = stored_duty_cycle;
  }
  return duty_cycle_read;
}

int32_t omega_all[4] = {0,0,0,0};
                //[ wheel1; wheel2; wheel3; wheel4]
int32_t value[4] = {omega_all[0],omega_all[1],omega_all[2],omega_all[3]};

void setup(){
  //Setting baudrate
  nh.getHardware()->setBaud(57600);
  //Initializing node
  nh.initNode();
  //Start advertising
  nh.advertise(wheel_omega);

  delay(1000);
  Serial.println(F("Hello Teensy 3.6 dual CAN Test."));
  Can1.begin(500000,defaultMask,0,0);
  pinMode(35, OUTPUT);
  digitalWrite(35, LOW);

}
void loop(){
  while (Can1.available())
  {
    //CAN_message_t inMsg;
    Can1.read(inMsg);
    e_rpm_1 = read_erpm(Node_ID_1, e_rpm_1);
    e_rpm_2 = read_erpm(Node_ID_2, e_rpm_2);
    current_1 = read_current(Node_ID_1,current_1);
    current_2 = read_current(Node_ID_2,current_2);
    duty_cycle_1 = read_duty_cycle(Node_ID_1,duty_cycle_1);
    duty_cycle_2 = read_duty_cycle(Node_ID_2,duty_cycle_2);
  }
  value[1] = e_rpm_1/7;
  value[2] = e_rpm_2/7;
  msg_wheel_omega.data = value;
  msg_wheel_omega.data_length =4;
  wheel_omega.publish(&msg_wheel_omega);

  nh.spinOnce();
  delay(250);
}
```

## A.4 Subscribe data from ROS to CAN

```
// Defining the Teensy 3.6 UART hardware serials
#define USE_TEENSY_HW_SERIAL
#include <ros.h>
#include <std_msgs/Int32MultiArray.h>
#include <FlexCAN.h>
#ifndef __MK66FX1M0__
  #error "Teensy 3.6 with CAN bus is required"
#endif


// Creating the calss to define the serial port to use
class NewHardware : public ArduinoHardware
{
  public:
  NewHardware():ArduinoHardware(&Serial1){}; // Specify port
  // NewHardware():ArduinoHardware(&Serial1, 57600){}; // Specify port
};

// Creating Nodehandle
ros::NodeHandle nh;

int32_t value_ref[4] = {0,0,0,0}; // Used for subscribing refrence value sent
    from Xavier
//Defining topic function
void omega_ref_array(const std_msgs::Int32MultiArray& omega_ref_msg){
   value_ref[0] = omega_ref_msg.data[1];
   value_ref[1] = omega_ref_msg.data[2];
   value_ref[2] = omega_ref_msg.data[3];
   value_ref[3] = omega_ref_msg.data[4];
}

//Defining subscribing topic
ros::Subscriber<std_msgs::Int32MultiArray> sub_omega("omega_ref",
    omega_ref_array);


static struct CAN_filter_t defaultMask;
static CAN_message_t msg_can_vesc;

// -------------------------------------------------------------
void buffer_append_int32(uint8_t *buffer, int32_t number, int32_t *index) {
  // Function to append the value used in write_can_message function
  buffer[(*index)++] = number >> 24;
  buffer[(*index)++] = number >> 16;
  buffer[(*index)++] = number >> 8;
  buffer[(*index)++] = number;
}
//-------------------------------------------------------------
void write_can_message(CAN_message_t msg, uint32_t can_id, int32_t value){
  // Function to fill in necessary information to write a CAN message
  msg.ext = 1; // If the id is extended or not (0=11bit ID, 1=29bit ID)
  msg.id = can_id; //Message id. This contains the command and which unit it
     is ment for
  msg.len = 4; // Message length (2= 0x0000, 4 = 0x00000000, 8 = 0
     x0000000000000000, etc...)
  uint8_t buffer [sizeof(msg.len)]; //Create an empty buffer to write message
     in
  int32_t send_index = 0; // Selecting which index to start the appending
```

```
  buffer_append_int32(buffer , value, &send_index); // Appends "value" to the
      buffer
  memcpy(msg.buf, buffer, msg.len*sizeof(uint32_t)); // Copies the buffer
      information into the message (msg.buf)
  Can1.write(msg); // Write CANBus command
}
//-------------------------------------------------------------
void setup(void){
   //Setting baudrate
  nh.getHardware()->setBaud(57600);
  //Initializing node
  nh.initNode();
  nh.subscribe(sub_omega);
  delay(1000);
  Serial.println(F("Hello Teensy 3.6 dual CAN Test."));
  Can1.begin(500000,defaultMask,0,0);
  pinMode(35, OUTPUT);
  digitalWrite(35, LOW);
}
// -------------------------------------------------------------
void loop(void){
   write_can_message(msg_can_vesc, 0x301 , value_ref[1]);
   delay(1000);
   nh.spinOnce();
}
```

## A.5 Publish multi array from Teensy to Xavier (ROS)

```
// Defining the Teensy 3.6 UART hardware serials
#define USE_TEENSY_HW_SERIAL
#include <ros.h>
//#include <std_msgs/Int32.h>
#include <std_msgs/Int32MultiArray.h>

//Creating test values
int32_t omega_all[4] = {-1000, 2000, -1000, 1000};

// Creating the calss to define the serial port to use
class NewHardware : public ArduinoHardware
{
  public:
  NewHardware():ArduinoHardware(&Serial1){}; // Specify port
};

// Creating Nodehandle
ros::NodeHandle nh;

//Instantiating the message
std_msgs::Int32MultiArray msg_wheel_omega;
              //[ wheel1; wheel2; wheel3; wheel4]
int32_t value[4] = {omega_all[0],omega_all[1],omega_all[2],omega_all[3]};

//Defining Publisher
ros::Publisher wheel_omega("wheel_omega_all", &msg_wheel_omega);

void setup(){
  //Setting baudrate
```

```
 nh.getHardware()->setBaud(57600);
 //Initializing node
 nh.initNode();
 //Start advertising
 nh.advertise(wheel_omega);
}

void loop(){

 msg_wheel_omega.data = value;
 msg_wheel_omega.data_length =4;
 wheel_omega.publish(&msg_wheel_omega);

 nh.spinOnce();
 delay(1000);
}
```

## A.6 Subsrcibe multi array from Xavier (ROS) to Teensy

```
// Defining the Teensy 3.6 UART hardware serials
#define USE_TEENSY_HW_SERIAL
#include <ros.h>
#include <std_msgs/Int32MultiArray.h>

// Creating the calss to define the serial port to use
class NewHardware : public ArduinoHardware
{
 public:
 NewHardware():ArduinoHardware(&Serial1){}; // Specify port
 // NewHardware():ArduinoHardware(&Serial1, 57600){};
};

// Creating Nodehandle
ros::NodeHandle nh;

int32_t value_ref[4] = {0,0,0,0}; // Used for subscribing refrence value sent
    from Xavier

//Defining topic function
void omega_ref_array(const std_msgs::Int32MultiArray& omega_ref_msg){
  value_ref[0] = omega_ref_msg.data[0];
  value_ref[1] = omega_ref_msg.data[1];
  value_ref[2] = omega_ref_msg.data[2];
  value_ref[3] = omega_ref_msg.data[3];
}

//Defining subscribing topic
ros::Subscriber<std_msgs::Int32MultiArray> sub_omega("omega_ref",
    omega_ref_array);


//-----------------------------------------------------------------
void setup(void){
 //Setting baudrate
 nh.getHardware()->setBaud(57600);
 //Initializing node
 nh.initNode();
```

```
 nh.subscribe(sub_omega);
}
// ------------------------------------------------------------
void loop(void){
  Serial.print("omega_ref_1= ");
  Serial.print(value_ref[0]);
  Serial.print(" | omega_ref_2= ");
  Serial.print(value_ref[1]);
  Serial.print(" | omega_ref_3= ");
  Serial.print(value_ref[2]);
  Serial.print(" | omega_ref_4= ");
  Serial.print(value_ref[3]);
  Serial.println(" | Boi");
  delay(1000);
  nh.spinOnce();
}
```

## A.7 Hand-Controller only Program

```
/*
 This program contains which is necessary to control the rig, with the use
 kinematic equations, a hand-controller and sending of the appriopriate CAN
    messages
 */
#include <FlexCAN.h>

//------------------------------------------------------------
#ifndef __MK66FX1M0__
  #error "Teensy 3.6 with dual CAN bus is required to run"
#endif
static struct CAN_filter_t defaultMask;
static CAN_message_t msg_can_vesc_01;
static CAN_message_t msg_can_vesc_02;
static CAN_message_t msg_can_vesc_03;
static CAN_message_t msg_can_vesc_04;

int led = 13;
int no_poles = 7; //The number of poles to the BLDC
#define PI_val 3.14159

// Initilaziing Kinematics variables
#define r 0.1015
#define w 0.2850
#define h 0.6350
#define joystick_x 14
#define joystick_y 15
#define joystick_theta 16
//-----------------------------//
float omega_1 = 0;
float omega_2 = 0;
float omega_3 = 0;
float omega_4 = 0;
float x_dot = 0;
float y_dot = 0;
float theta_dot = 0;
float x_test = 0;
float y_test = 0;
```

```
float z_test = 0;
//--------------------------------------------------------------
float ReadJoystick(int joystick, int joystick_val) {
  joystick_val = map(analogRead(joystick), 0, 1023, -4, 4);
  return joystick_val;
}
//--------------------------------------------------------------
void buffer_append_int32(uint8_t* buffer, int32_t number, int32_t *index) {
  // Function to append the value
  buffer[(*index)++] = number >> 24;
  buffer[(*index)++] = number >> 16;
  buffer[(*index)++] = number >> 8;
  buffer[(*index)++] = number;
}
//--------------------------------------------------------------
void write_can_message(CAN_message_t msg, uint32_t can_id, int32_t value){
  // Function to fill in necessary infromation to write a CAN message
  msg.ext = 1;
  msg.id = can_id;
  msg.len = 4;
  uint8_t buffer [sizeof(msg.len)];
  int32_t send_index = 0;
  buffer_append_int32(buffer , value, &send_index);
  memcpy(msg.buf, buffer, msg.len*sizeof(uint32_t));
  Can1.write(msg); // Write CANBus command
}
//--------------------------------------------------------------
void setup(void){
  delay(1000);
  Serial.println(F("Hello Teensy 3.6 dual CAN Test."));
  Can1.begin(500000,defaultMask,0,0);
  //if using enable pins on a transceiver they need to be set on
  pinMode(35, OUTPUT);
  digitalWrite(35, LOW);
  Serial.begin(9600);
  pinMode(led, OUTPUT);
  digitalWrite(led,HIGH);
}
// --------------------------------------------------------------
void loop(void){
  x_dot = ReadJoystick(joystick_x,0);
  y_dot = ReadJoystick(joystick_y,0);
  theta_dot = ReadJoystick(joystick_theta,0);
  //x_dot = 0;
  //y_dot = 0;
  //theta_dot = 0;
  x_test = analogRead(14);
  y_test = analogRead(15);
  z_test = analogRead(16);
  omega_1 = -1/r*(x_dot+y_dot+(-w-h)*theta_dot)*no_poles*30/PI_val;
  omega_2 = -1/r*(-x_dot+y_dot+(w+h)*theta_dot)*no_poles*30/PI_val;
  omega_3 = -1/r*(x_dot+y_dot+(w+h)*theta_dot)*no_poles*30/PI_val;
  omega_4 = -1/r*(-x_dot+y_dot+(-w-h)*theta_dot)*7*7;
  write_can_message(msg_can_vesc_01, 0x301, omega_1);
  write_can_message(msg_can_vesc_02, 0x302, omega_2);
  write_can_message(msg_can_vesc_03, 0x303, omega_3);
  write_can_message(msg_can_vesc_04, 0x304, omega_4);

  // Serial prints for debugging
  /*
  Serial.print(x_test);
```

```
 Serial.print(" | ");
 Serial.print(y_test);
 Serial.print(" | ");
 Serial.print(z_test);
 Serial.println("");
 */
 /*
 Serial.print("|x_dot= ");
 Serial.print(x_dot);
 Serial.print("|y_dot= ");
 Serial.print(y_dot);
 Serial.print("|theta_dot");
 Serial.print(theta_dot);
 Serial.print("|omega_1= ");
 Serial.print(omega_1);
 Serial.print(" |omega_2= ");
 Serial.print(omega_2);
 Serial.print(" |omega_3= ");
 Serial.print(omega_3);
 Serial.print(" |omega_4= ");
 Serial.print(omega_4);
 Serial.println(" | ");*/
 delay(20);
}
```

## A.8 Write CAN message

```
#include <FlexCAN.h>
//-------------------------------------------------------------
#ifndef __MK66FX1M0__
  #error "Teensy 3.6 with dual CAN bus is required to run"
#endif
static struct CAN_filter_t defaultMask;
static CAN_message_t msg_can_vesc;
// -------------------------------------------------------------
void buffer_append_int32(uint8_t *buffer, int32_t number, int32_t *index) {
 // Function to append the value used in write_can_message function
 buffer[(*index)++] = number >> 24;
 buffer[(*index)++] = number >> 16;
 buffer[(*index)++] = number >> 8;
 buffer[(*index)++] = number;
}
//-------------------------------------------------------------
void write_can_message(CAN_message_t msg, uint32_t can_id, int32_t value){
 // Function to fill in necessary information to write a CAN message
 msg.ext = 1; // If the id is extended or not (0=11bit ID, 1=29bit ID)
 msg.id = can_id; //Message id. This contains the command and which unit it
     is ment for
 msg.len = 4; // Message length (2= 0x0000, 4 = 0x00000000, 8 = 0
     x0000000000000000, etc...)
 uint8_t buffer [sizeof(msg.len)]; //Create an empty buffer to write message
     in
 int32_t send_index = 0; // Selecting which index to start the appending
 buffer_append_int32(buffer , value, &send_index); // Appends "value" to the
     buffer
 memcpy(msg.buf, buffer, msg.len*sizeof(uint32_t)); // Copies the buffer
     information into the message (msg.buf)
```

```
 Can1.write(msg); // Write CANBus command
}
//-------------------------------------------------------------
void setup(void){
 delay(1000);
 Serial.println(F("Hello Teensy 3.6 dual CAN Test."));
 Can1.begin(500000,defaultMask,0,0);
 pinMode(35, OUTPUT);
 digitalWrite(35, LOW);
}
// -------------------------------------------------------------
void loop(void){
  write_can_message(msg_can_vesc, 0x301 , 1200);
  delay(1000);
}
```

## A.9 Reading CAN messages

```
#include <FlexCAN.h>
//#include <string.h>
#ifndef __MK66FX1M0__
 #error "Teensy 3.6 with dual CAN bus is required"
#endif
static struct CAN_filter_t defaultMask;
static CAN_message_t inMsg;

int32_t e_rpm_1, e_rpm_2;
int32_t duty_cycle_1, duty_cycle_2;
int32_t current_1, current_2;
uint16_t Node_ID_1 = 0x01;
uint16_t Node_ID_2 = 0x02;

// -------------------------------------------------------------
int16_t read_buffer_int16(const uint8_t *buffer, int32_t *index){
 int16_t get_buffer_value = ((uint16_t) buffer[*index]) << 8 | ((uint16_t)
    buffer[*index+1]);
 //*index += 2;
 return get_buffer_value;
}
// -------------------------------------------------------------
int16_t get_dec_value (uint8_t *bytePtr, int index){
 int32_t send_index = index;
 int16_t value_decimal = read_buffer_int16(bytePtr, &send_index);
 return value_decimal;
}
// -------------------------------------------------------------
 int32_t read_erpm(uint16_t can_node_id, int stored_e_rpm){
 // Function to read the ERPM sent from the VESC
 int32_t e_rpm_read;
 if(inMsg.id == (0x01 << 8 | can_node_id)){
  //if(inMsg.id == (0x901)){
  // In order to merge the 4 bytes, the data length 0 and 1 are left shifted
     16 bits
  // 0x0000ABCD << 16 -> 0xABCD0000, 0xABCD0000 | 0x0000abcd -> 0xABCDabcd
  e_rpm_read = get_dec_value(inMsg.buf,0) << 16 | get_dec_value(inMsg.buf,2);
 }
```

```
  else {
    /*
    If "dead-time" happens with signal conflict, resulting in no new data,
    the previous value is used instead of using a zero value. This is done
    to avoid jittering
    */
    e_rpm_read = stored_e_rpm;
  }
  return e_rpm_read;
}
// ------------------------------------------------------------
  int16_t read_current(uint16_t can_node_id, int stored_current){
  // Function to read the CURRENT sent from the VESC
  int16_t current_read;
  if(inMsg.id == (0x02 << 8 | can_node_id)){
    // Current is stored in byte 4 and 5
    current_read = get_dec_value(inMsg.buf,4);
  }
  else {
    /*
    If "dead-time" happens with signal conflict, resulting in no new data,
    the previous value is used instead of using a zero value. This is done
    to avoid jittering
    */
    current_read = stored_current;
  }
  return current_read;
}
// ------------------------------------------------------------
  int16_t read_duty_cycle(uint16_t can_node_id, int stored_duty_cycle){
  // Function to read the DUTY CYCLE sent from the VESC
  int16_t duty_cycle_read;
  if(inMsg.id == (0x03 << 8 | can_node_id)){
    // Duty cycle is stored in byte 6 and 7
    duty_cycle_read = get_dec_value(inMsg.buf,6);
  }
  else {
    /*
    If "dead-time" happens with signal conflict, resulting in no new data,
    the previous value is used instead of using a zero value. This is done
    to avoid jittering
    */
    duty_cycle_read = stored_duty_cycle;
  }
  return duty_cycle_read;
}
void setup(void)
{
  delay(1000);
  Serial.println(F("Hello Teensy 3.6 dual CAN Test."));
  Can1.begin(500000,defaultMask,0,0);
  pinMode(35, OUTPUT);
  digitalWrite(35, LOW);
}
// ------------------------------------------------------------
void loop(void)
{

  while (Can1.available())
  {
  //CAN_message_t inMsg;
```

```
 Can1.read(inMsg);
 e_rpm_1 = read_erpm(Node_ID_1, e_rpm_1);
 e_rpm_2 = read_erpm(Node_ID_2, e_rpm_2);
 current_1 = read_current(Node_ID_1,current_1);
 current_2 = read_current(Node_ID_2,current_2);
 duty_cycle_1 = read_duty_cycle(Node_ID_1,duty_cycle_1);
 duty_cycle_2 = read_duty_cycle(Node_ID_2,duty_cycle_2);
 }
 Serial.print("Data ID:");
 Serial.print(Node_ID_1,HEX);
 Serial.print(" | e_rpm = ");
 Serial.print(e_rpm_1,HEX);
 Serial.print(" | current = ");
 Serial.print(current_1,HEX);
 Serial.print(" | duty cycle = ");
 Serial.println(duty_cycle_1,HEX);

 Serial.print("Data ID:");
 Serial.print(Node_ID_2,HEX);
 Serial.print(" | e_rpm = ");
 Serial.print(e_rpm_2,HEX);
 Serial.print(" | current = ");
 Serial.print(current_2,HEX);
 Serial.print(" | duty cycle = ");
 Serial.println(duty_cycle_2,HEX);
 delay(1000);
}
```

# B. Inverse kinematic simplifications

$$-v_d + v_s\cos(\gamma) = -\dot{x}\sin(\alpha+\beta) + \dot{y}\cos(\alpha+\beta) + l\dot{\theta}\cos(\beta) \tag{B.1}$$

$$-v_s\sin(\gamma) = \dot{x}\cos(\alpha+\beta) + \dot{y}\sin(\alpha+\beta) + l\dot{\theta}\sin(\beta) \tag{B.2}$$

$$v_s = \frac{-\dot{x}\sin(\alpha+\beta) + \dot{y}\cos(\alpha+\beta) + l\dot{\theta}\cos(\beta) + v_d}{\cos(\gamma)} \tag{B.3}$$

$$v_s = \frac{-\dot{x}\cos(\alpha+\beta) - \dot{y}\sin(\alpha+\beta) - l\dot{\theta}\sin(\beta)}{\sin(\gamma)} \tag{B.4}$$

$$\frac{-\dot{x}\sin(\alpha+\beta) + \dot{y}\cos(\alpha+\beta) + l\dot{\theta}\cos(\beta) + v_d}{\cos(\gamma)} = \frac{-\dot{x}\cos(\alpha+\beta) - \dot{y}\sin(\alpha+\beta) - l\dot{\theta}\sin(\beta)}{\sin(\gamma)} \tag{B.5}$$

$$v_s = \frac{-\dot{x}\sin(\alpha+\beta) + \dot{y}\cos(\alpha+\beta) + l\dot{\theta}\cos(\beta) + v_d}{\cos(\gamma)} \tag{B.6}$$

$$v_s = \frac{-\dot{x}\cos(\alpha+\beta) - \dot{y}\sin(\alpha+\beta) - l\dot{\theta}\sin(\beta)}{\sin(\gamma)} \tag{B.7}$$

$$\sin(\alpha \pm \beta) = \sin(\alpha)\cos(\beta) \pm \cos(\alpha)\sin(\beta) \tag{B.8}$$

$$\cos(\alpha \pm \beta) = \cos(\alpha)\cos(\beta) \mp \sin(\alpha)\sin(\beta) \tag{B.9}$$

$$\frac{-\dot{x}\sin(\alpha+\beta) + \dot{y}\cos(\alpha+\beta) + l\dot{\theta}\cos(\beta) + v_d}{\cos(\gamma)} = \frac{-\dot{x}\cos(\alpha+\beta) - \dot{y}\sin(\alpha+\beta) - l\dot{\theta}\sin(\beta)}{\sin(\gamma)} \tag{B.10}$$

$$\begin{aligned} &-\dot{x}\sin(\alpha+\beta)\sin(\gamma) + \dot{y}\cos(\alpha+\beta)\sin(\gamma) + l\dot{\theta}\cos(\beta)\sin(\gamma) + v_d\sin(\gamma) = ... \\ &-\dot{x}\cos(\alpha+\beta)\cos(\gamma) - \dot{y}\sin(\alpha+\beta)\cos(\gamma) - l\dot{\theta}\sin(\beta)\cos(\gamma) \end{aligned} \tag{B.11}$$

$$\begin{aligned} &\dot{x}[\cos(\alpha+\beta)\sin(\gamma) - \sin(\alpha+\beta)\sin(\gamma)] \\ &+\dot{y}[\cos(\alpha+\beta)\sin(\gamma) + \sin(\alpha+\beta)\cos(\gamma)] \\ &+l\dot{\theta}[\cos(\beta)\sin(\gamma) + \sin(\beta)\cos(\gamma)] \\ &= -v_d\sin(\gamma) \end{aligned} \tag{B.12}$$

$$\dot{x}\cos(\alpha+\beta+\gamma) + \dot{y}\sin(\alpha+\beta+\gamma) + l\dot{\theta}\sin(\beta+\gamma) = -v_d\sin(\gamma) \tag{B.13}$$

$$\alpha + \beta = 0$$

$$\dot{x}\cos(\gamma) + \dot{y}\sin(\gamma) + l\dot{\theta}\sin(\beta+\gamma) = -v_d\sin(\gamma) \tag{B.14}$$

$$\dot{x}\frac{\cos(\gamma)}{\sin(\gamma)} + \dot{y}\frac{\sin(\gamma)}{\sin(\gamma)} + l\dot{\theta}\frac{\sin(\beta+\gamma)}{\sin(\gamma)} = -v_d \tag{B.15}$$

$$\dot{x}\frac{1}{\tan(\gamma)} + \dot{y} + l\dot{\theta}\frac{\sin(\beta+\gamma)}{\sin(\gamma)} = -v_d \tag{B.16}$$

$$\dot{x}\frac{\cos(\gamma)}{\sin(\gamma)} + \dot{y}\frac{\sin(\gamma)}{\sin(\gamma)} + l\dot{\theta}\frac{\sin(\beta)\cos(\gamma) + \cos(\beta)\sin(\gamma)}{\sin(\gamma)} = -v_d \tag{B.17}$$

$$\frac{\dot{x}}{\tan(\gamma)} + \dot{y} + l\dot{\theta}\left(\frac{\sin(\beta)}{\tan(\gamma)} + \cos(\beta)\right) = -v_d \tag{B.18}$$

$$\frac{\dot{x}}{\tan(\gamma)} + \dot{y} + l\dot{\theta}\left(\frac{\sin(-\alpha)}{\tan(\gamma)} + \cos(-\alpha)\right) = -v_d \tag{B.19}$$

$$\frac{\dot{x}}{\tan(\gamma)} + \dot{y} + l\dot{\theta}\left(\cos(\alpha) - \frac{\sin(\alpha)}{\tan(\gamma)}\right) = -v_d \tag{B.20}$$

$$\begin{bmatrix} \frac{1}{\tan(\gamma)} & 1 & w - \frac{h}{\tan(\gamma)} \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = -r\omega \tag{B.21}$$

# C. Data Sheets

## C.1 Jetson Xavier Development kit GPIO pinout

| Jetson AGX Xavier Expansion Header | | | | | |
|---|---|---|---|---|---|
| Sysfs GPIO | Connector Label | Pin | Pin | Connector Label | Sysfs GPIO |
| | **3.3 VDC**<br>*Power, 1A max* | 1 | 2 | **5.0 VDC**<br>*Power, 1A max* | |
| | **I2C_GP5_DAT**<br>*General I2C #5 Data*<br>*1.8/3.3V, I2C Bus 8* | 3 | 4 | **5.0 VDC**<br>*Power, 1A max* | |
| | **I2C_GP5_CLK**<br>*General I2C #5 Clock*<br>*1.8/3.3V, I2C Bus 8* | 5 | 6 | **GND** | |
| gpio422 | **MCLK05**<br>*Audio Master Clock*<br>*1.8/3.3V* | 7 | 8 | **UART1_TX**<br>*UART #1 Transmit*<br>*3.3V* | |
| | **GND** | 9 | 10 | **UART1_RX**<br>*UART #1 Receive*<br>*3.3V* | |
| gpio428 | **UART1_RTS**<br>*UART #1 Request to Send*<br>*1.8/3.3V* | 11 | 12 | **I2S2_CLK**<br>*Audio I2S #2 Clock*<br>*1.8/3.3V* | gpio351 |
| gpio424 | **PWM01**<br>*Pulse Width Modulation #1*<br>*1.8/3.3V* | 13 | 14 | **GND** | |

| | | | | | |
|---|---|---|---|---|---|
| gpio393 | **GPIO27_PWM2**<br>*GPIO/Pulse Width Modulation #2*<br>*1.8/3.3V* | 15 | 16 | **GPIO8_AO_DMIC_IN_DAT**<br>*Digital Mic Input*<br>*3.3V* | gpio256 |
| | **3.3 VDC**<br>*Power, 1A max* | 17 | 18 | **GPIO35_PWM3**<br>*GPIO/Pulse Width Modulation #3*<br>*1.8/3.3V* | gpio344 |
| gpio493 | **SPI1_MOSI**<br>*SPI #1 Master Out/Slave In*<br>*1.8/3.3V* | 19 | 20 | **GND** | |
| gpio492 | **SPI1_MISO**<br>*SPI #1 Master In/Slave Out*<br>*1.8/3.3V* | 21 | 22 | **GPIO17_40HEADER**<br>*GPIO*<br>*1.8/3.3V* | gpio417 |
| gpio491 | **SPI1_SCLK**<br>*SPI #1 Shift Clock*<br>*1.8/3.3V* | 23 | 24 | **SPI1_CS0**<br>*SPI #1 Chip Select #0*<br>*1.8/3.3V* | gpio494 |
| | **GND** | 25 | 26 | **SPI1_CS1**<br>*SPI #1 Chip Select #1*<br>*1.8/3.3V* | gpio495 |
| | **I2C_GP2_DAT**<br>*General I2C #2 Data*<br>*1.8/3.3V, I2C Bus 1* | 27 | 28 | **I2C_GP2_CLK**<br>*General I2C #2 Clock*<br>*1.8/3.3V, I2C Bus 1* | |
| gpio251 | **CAN0_DIN**<br>*CAN #0 Data In*<br>*3.3V* | 29 | 30 | **GND** | |
| gpio250 | **CAN0_DOUT**<br>*CAN #0 Data Out*<br>*3.3V* | 31 | 32 | **GPIO9_CAN1_GPIO0_DMIC_CLK**<br>*Digital Mic Input Clock*<br>*3.3V* | gpio257 |

| | | | | | |
|---|---|---|---|---|---|
| gpio354 | **I2S_FS**<br>*AUDIO I2S #2*<br>*Left/Right Clock*<br>*1.8/3.3V* | 35 | 36 | **UART1_CTS**<br>*UART #1 Clear to Send*<br>*1.8/3.3V* | gpio429 |
| gpio249 | **CAN1_DIN**<br>*CAN #1 Data In*<br>*3.3V* | 37 | 38 | **I2S_SDIN**<br>*Audio I2S #2 Data In*<br>*1.8/3.3V* | gpio353 |
| | **GND** | 39 | 40 | **I2S_SDOUT**<br>*Audio I2S #2 Data Out*<br>*1.8/3.3V* | gpio352 |

***Note:*** *1.8V/3.3V Selectable by J514*

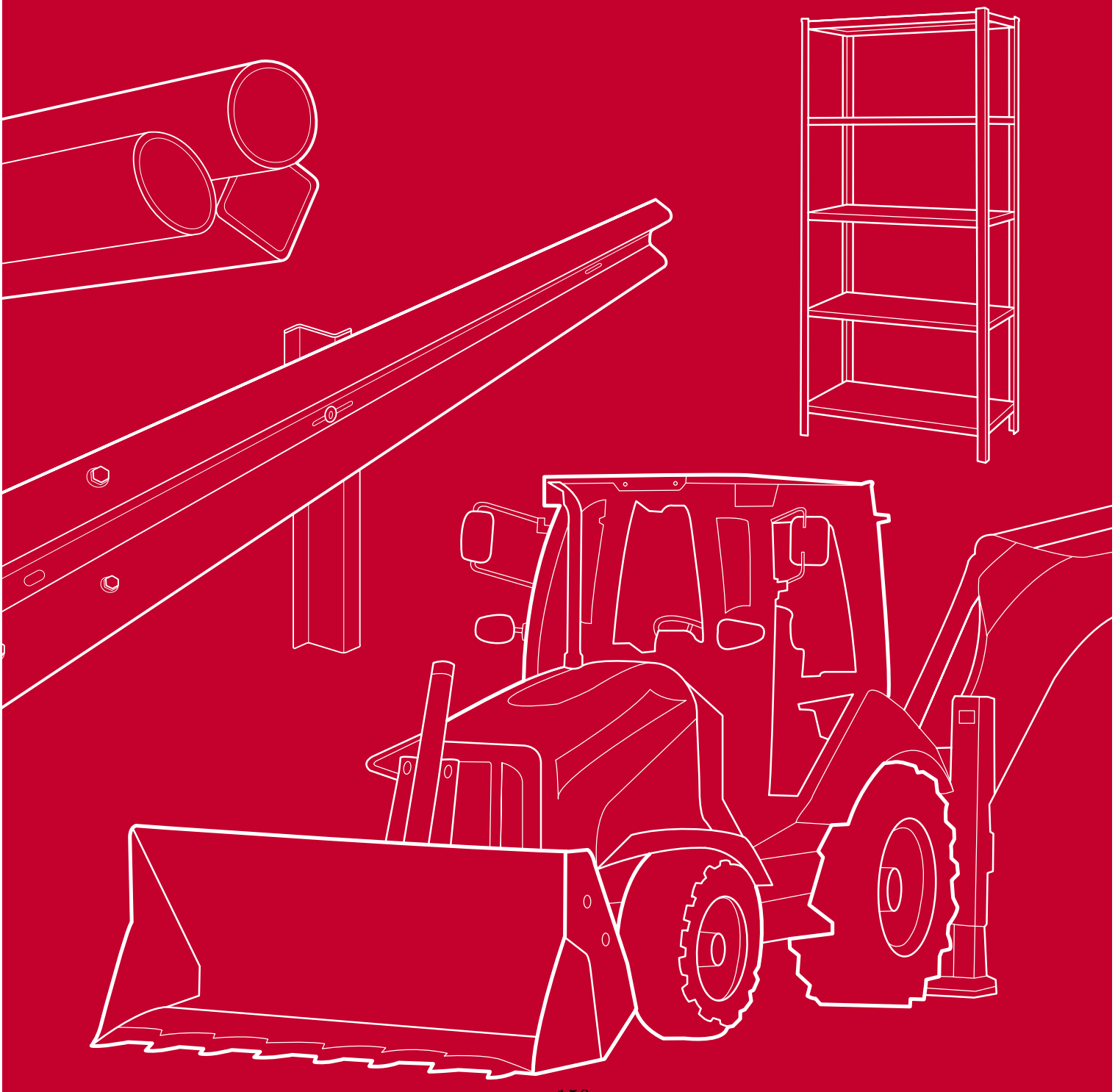

**Jetson AGX Xavier Pin 1**

# C.2 S355 material properties - EN 10025

Corus Strip Products UK

## European structural steel standard EN 10025-2 : 2004

### Grade designations, properties and nearest equivalents



## General

EN 10025 : 2004 is the new six-part European standard for hot-rolled structural steel.

The information here covers the steels in part 2 of that standard, which are grades manufactured by Corus Strip Products UK.

The new standard designates grades differently from the previous standards for these steels.

This leaflet shows the grades and their mechanical properties from the new standard together with the nearest equivalent grades from former standards. It also contains advice about specifying these steels when ordering.

Details of the dimensional range are available from Corus.

## The parts of the new standard

The standard is published in the six parts shown below. It combines what were formerly separate standards into one new standard for the majority of hot-rolled structural steel products.

Part 1 – *General technical delivery conditions*

Part 2 – *Technical delivery conditions for non-alloy structural steels*
Supersedes EN 10025 : 1993

Part 3 – *Technical delivery conditions for normalised/ normalised rolled weldable fine grain structural steels*
Supersedes EN 10113 : parts 1 & 2 : 1993

Part 4 – *Technical delivery conditions for thermomechanical rolled weldable fine grain structural steels*
Supersedes EN 10113 : parts 1 & 3 : 1993

Part 5 – *Technical delivery conditions for structural steels with improved atmospheric corrosion resistance* (also known as weathering steels)
Supersedes EN 10155 : 1993

Part 6 – *Technical delivery conditions for flat products of high yield strength structural steels in the quenched and tempered condition*
Supersedes EN 10137 : parts 1 & 2 : 1996

## Grade designation systems

The designation systems used in the new standard are similar but not identical to those used in EN 10025 : 1993. The symbols and the properties they designate are shown in table 1 below. The table includes examples that demonstrate the use of the symbols within the new designations.

## Mechanical properties and equivalent grades

Table 2 opposite shows the grades and their mechanical properties for part 2 of the new standard, together with the nearest equivalent grades from the superseded standards.

**Table 1: Symbols used in EN 10025-2 : 2004 : Non-alloy structural steels**

| Symbol | Example | Property designated |
|---|---|---|
| S | S185 | Structural steel |
| E | E295 | Engineering steel |
| 360 | E360 | Minimum yield strength ($R_{eH}$) in MPa at 16mm |
| JR | S235JR | Longitudinal Charpy V-notch impacts 27J at +20°C |
| J0 | S275J0 | Longitudinal Charpy V-notch impacts 27J at 0°C |
| J2 | S355J2 | Longitudinal Charpy V-notch impacts 27J at -20°C |
| K2 | S355K2 | Longitudinal Charpy V-notch impacts 40J at -20°C |
| +AR | S235JR+AR | Supply condition as rolled |
| +N | S275J0+N | Supply condition normalised or normalised rolled |
| Customer options | | |
| C | S235C | Grade suitable for cold forming |

**Note:** Text highlighted in red shows the symbols as they are used in examples of grades from the new standard.

## Ordering with the new standard

When ordering, please include the following information.

**The standard**
Include the part number, e.g. EN 10025-2 : 2004.

**Steel grade**
Use the new designations.

**Nominal dimensions**
Include any agreed special tolerances.

**Options**
Consult EN 10025-1 (section 13) and EN 10025-2 (section 13) for details of options.

If options are not specified when ordering, Corus will supply the basic specification.

## Advice

Advice about the new standard is available from Customer Technical Services, whose details appear on the back cover of this leaflet.

Copies of this leaflet are available from our web site at: www.corusgroup.com/stripproductsuk

**Table 2: EN 10025-2 : 2004 : Non-alloy structural steels : Grades, mechanical properties and nearest equivalent grades**

| **EN 10025-2 : 2004** | | | | | **EN 10025 : 1993** | **BS 4360 : 1990** |
|---|---|---|---|---|---|---|
| Grade | Strength at t=16mm (MPa) | | Longitudinal Charpy V-notch | | Nearest equivalent grade | Nearest equivalent grade |
| | Min yield ($R_{eH}$) | Tensile ($R_m$) | Temp (°C) | Energy (J) t=16mm | | |
| S185 | 185 | 290/510 | – | – | S185 | – |
| – [1] | – | – | – | – | S235 | 40A |
| S235JR [2] | 235 | 360/510 | 20 | 27 | S235JRG1/G2 | 40B |
| S235J0 | 235 | 360/510 | 0 | 27 | S235J0 | 40C |
| S235J2 | 235 | 360/510 | -20 | 27 | S235J2G3/G4 | 40D |
| – [1] | – | – | – | – | S275 | 43A |
| S275JR [2] | 275 | 410/560 | 20 | 27 | S275JR | 43B |
| S275J0 | 275 | 410/560 | 0 | 27 | S275J0 | 43C |
| S275J2 | 275 | 410/560 | -20 | 27 | S275J2G3/G4 | 43D |
| – [1] | – | – | – | – | S355 | 50A |
| S355JR [2] | 355 | 470/630 | 20 | 27 | S355JR | 50B |
| S355J0 | 355 | 470/630 | 0 | 27 | S355J0 | 50C |
| S355J2 | 355 | 470/630 | -20 | 27 | S355J2G3/G4 | 50D |
| S355K2 | 355 | 470/630 | -20 | 40 | S355K2G3/G4 | 50DD |
| E295 | 295 | 470/610 | – | – | E295 | – |
| E335 | 335 | 570/710 | – | – | E335 | – |
| E360 | 360 | 670/830 | – | – | E360 | – |

**Notes:**

1. This grade has been removed from the standard because it does not offer a guaranteed minimum impact performance, which is required by the EU Construction Products Directive (CPD 89/106/EC). The lowest grade offered is the JR version for each yield strength variation.
2. Corus will only verify the specified impact value for this grade if asked to do so at the time of the enquiry and the order.
3. 1 MPa=1 $N/mm^2$.

# www.corusgroup.com



Design: ELEVATOR www.elevatordesign.co.uk

**Commercial enquiries**
Corus Strip Products UK
PO Box 10
Newport
South Wales
NP19 4XN
T: +44 (0)1633 290022
F: +44 (0)1633 755104
cspuk.marketing@corusgroup.com
**www.corusgroup.com/stripproductsuk**

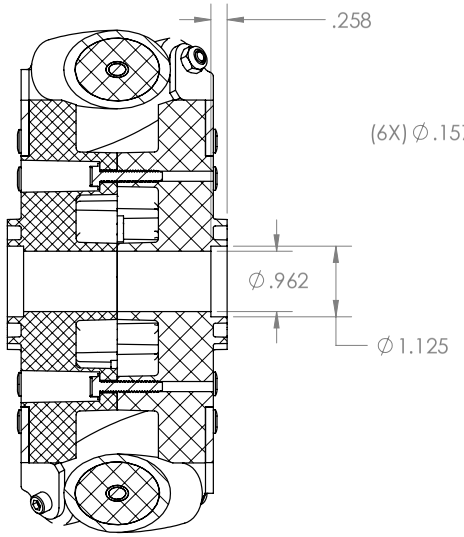
**Technical enquiries**
Customer Technical Services
Corus Strip Products UK
PO Box 10
Newport
South Wales
NP19 4XN
T: +44 (0)1633 755171
F: +44 (0)1633 755177
**strip.enquiries@corusgroup.com**

**English language version**

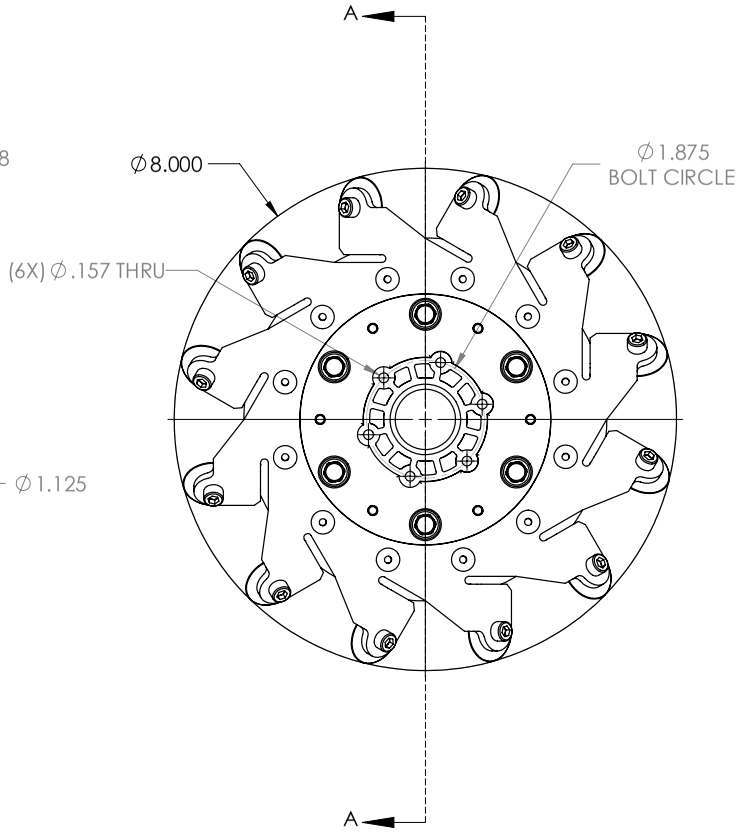
CSPUK25:2000:UK:09/2006

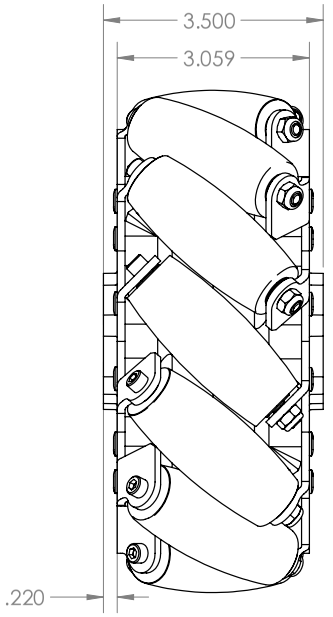
# D. Technical drawings

| REVISION HISTORY | | | |
|---|---|---|---|
| REV. | DESCRIPTION | DATE | DRAWN BY |
| 1 | ORIGINAL PRINT | 11/9/2017 | K. NEPOMUCENO |

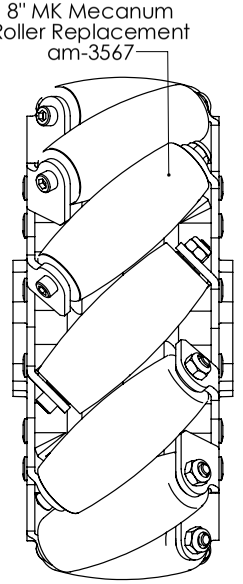



SECTION A-A
SCALE 1 : 2




LEFT WHEEL
(am-3340L)




RIGHT WHEEL
(am-3340R)

*** SEE CAD FOR ADDITIONAL TOOLPATH INFO***

| | | | |
|---|---|---|---|
| UNLESS OTHERWISE SPECIFIED: | | COMMENTS: | |
| DIMENSIONS ARE IN INCHES | | DUAL DIMENSIONS ARE IN MILLIMETERS. | |
| TOLERANCES: | | BREAK ALL SHARP EDGES. | |
| FRACTIONAL | ±1/16 | | |
| ANGULAR: MACH | ±1° | | |
| ANGULAR: BEND | ±5° | | |
| ONE PLACE DECIMAL | ±0.1 | | |
| TWO PLACE DECIMAL | ±0.01 | | |
| THREE PLACE DECIMAL | ±0.005 | | |
| FOUR PLACE DECIMAL | ±0.0005 | | |
| MATERIAL | | | |
| FINISH | | | |
| DO NOT SCALE DRAWING | | | |

| | NAME | DATE |
|---|---|---|
| CHECKED | | |
| ENG APPR. | | |
| MFG APPR. | | |
| Q.A. | | |

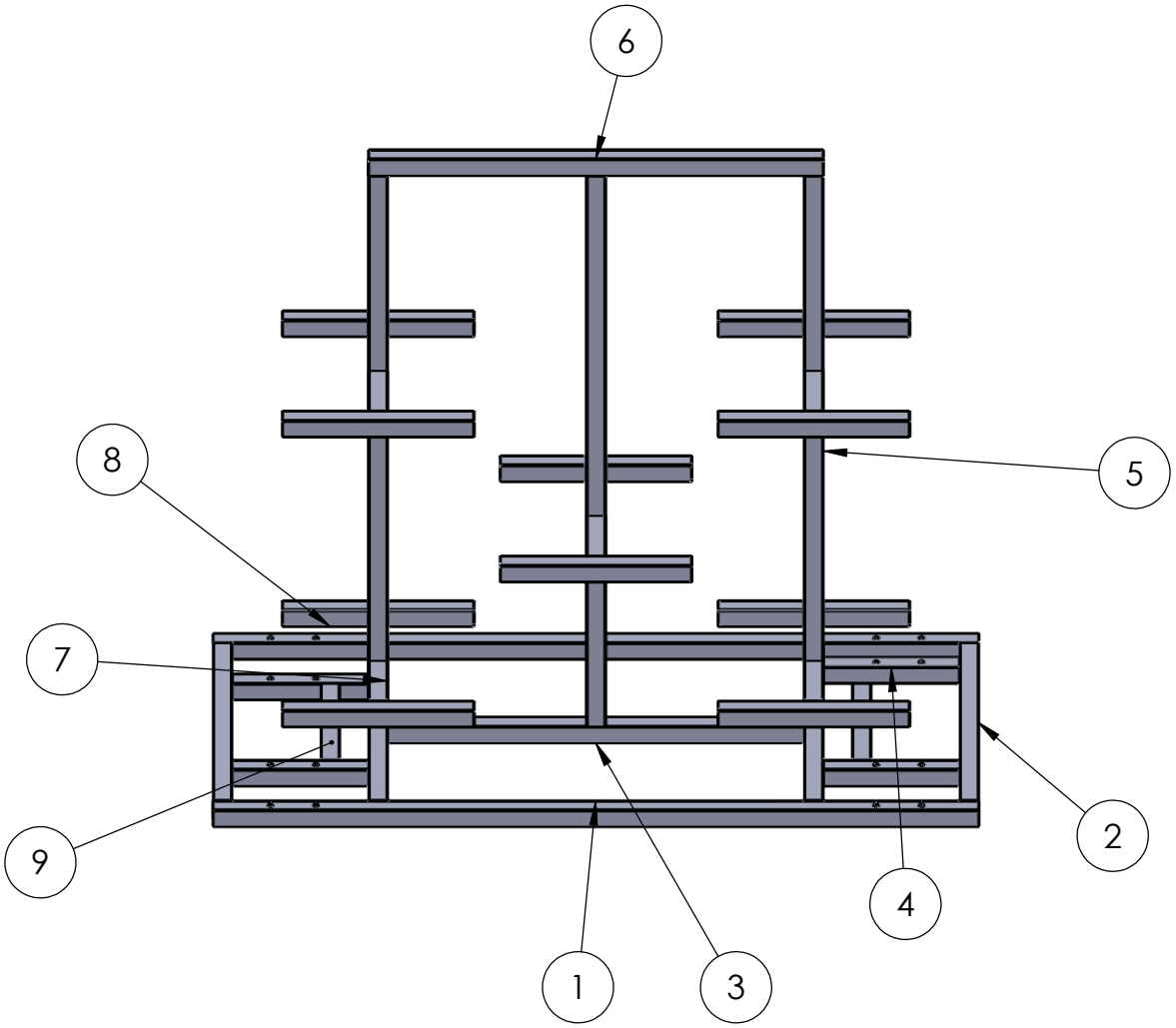
AndyMark

TITLE:
8" MK Mecanum Wheels

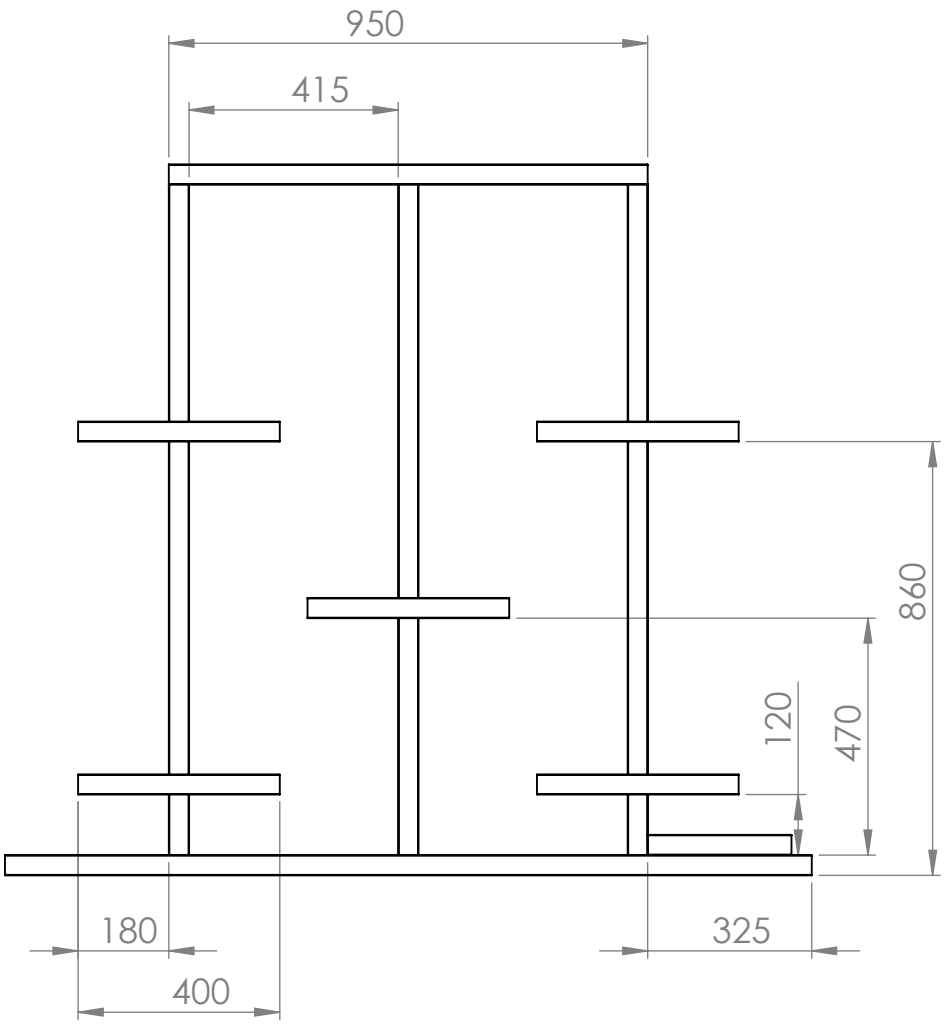
| SIZE | DWG. NO. | REV |
|---|---|---|
| B | am-3340 | 1 |
| SCALE: 1:2 | SHEET 1 OF 1 | |

# D.2 Loomo Parking Rig

| ITEM NO. | PART NUMBER | DESCRIPTION | QTY. |
|---|---|---|---|
| 1 | 1600 mm H40 | | 2 |
| 2 | 660 mm H40 | | 4 |
| 3 | 870 mm H40 | | 1 |
| 4 | 285 mm H40 | | 4 |
| 5 | 1330 mm H40 | | 3 |
| 6 | 950 mm H40 | | 1 |
| 7 | 170 mm H40 | | 10 |
| 8 | 400 mm H40 | | 10 |
| 9 | 320 mm H40 | | 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: |
| DRAWN | | | | | | | |
| CHK'D | | | | | | | |
| APPV'D | | | | | | | |
| MFG | | | | | | | |
| Q.A | | | | MATERIAL: | | DWG NO. | A4 |
| | | | | | | Loomo Rig Mastergruppe 6 | |
| | | | | WEIGHT: | | SCALE:1:50 | SHEET 1 OF 3 |




| | | |
|---|---|---|
| UNLESS OTHERWISE SPECIFIED:<br>DIMENSIONS ARE IN MILLIMETERS<br>SURFACE FINISH:<br>TOLERANCES:<br>LINEAR:<br>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:

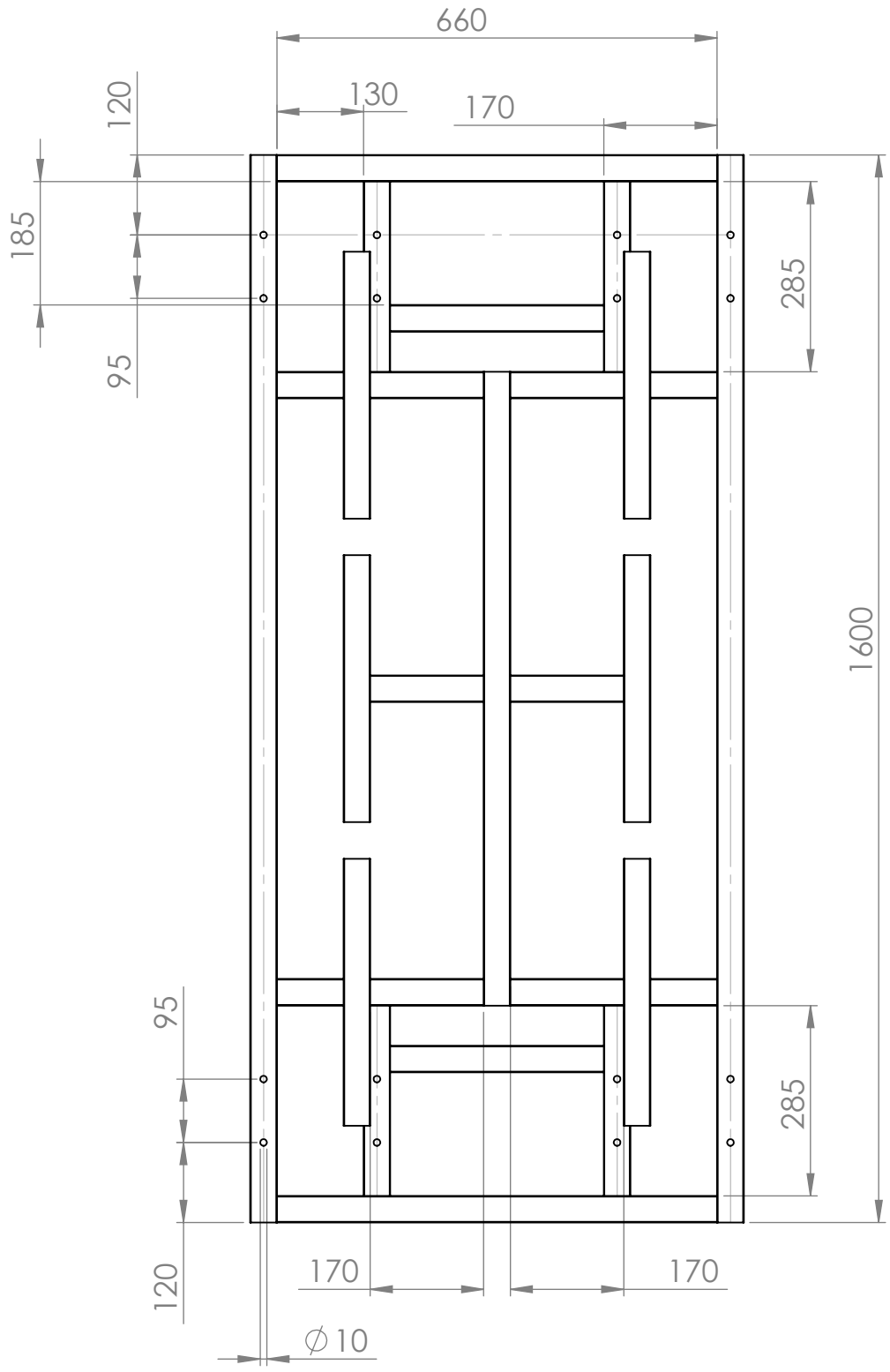
DWG NO. Loomo Rig Mastergruppe 6

A4

WEIGHT:

SCALE:1:50

SHEET 2 OF 3




| | | | |
|---|---|---|---|
| UNLESS OTHERWISE SPECIFIED:<br>DIMENSIONS ARE IN MILLIMETERS<br>SURFACE FINISH:<br>TOLERANCES:<br>LINEAR:<br>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. Loomo Rig Mastergruppe 6

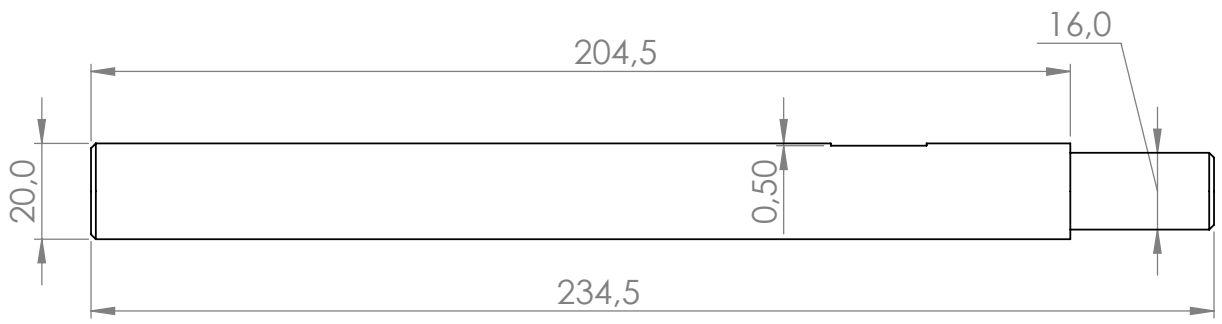
A4

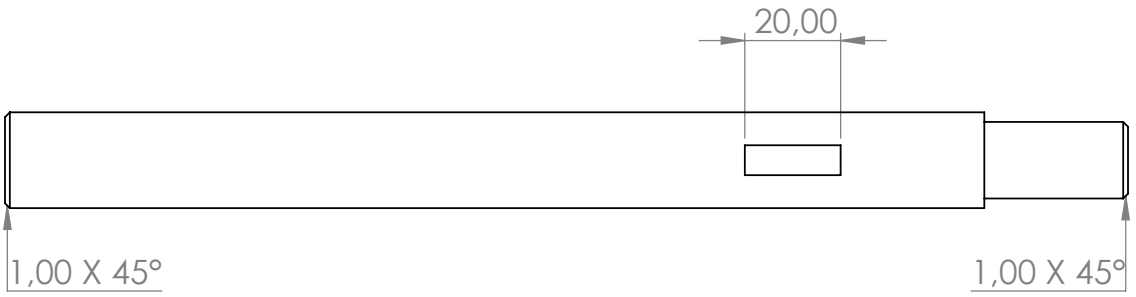
WEIGHT:

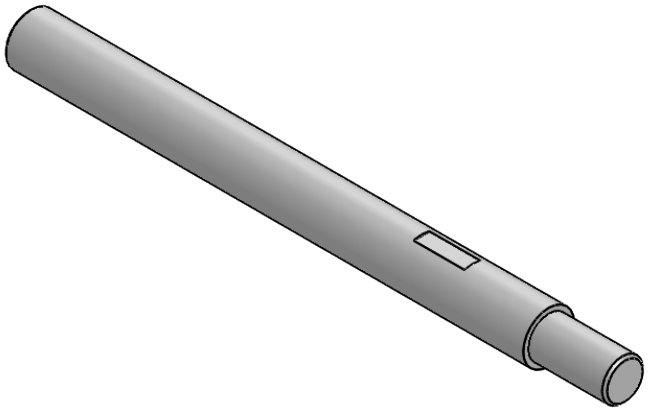
SCALE:1:50

SHEET 3 OF 3

# D.3 Axle



| | | | |
|---|---|---|---|
| UNLESS OTHERWISE SPECIFIED:<br>DIMENSIONS ARE IN MILLIMETERS<br>SURFACE FINISH:<br>TOLERANCES:<br>LINEAR:<br>ANGULAR: | FINISH: | DEBURR AND BREAK SHARP EDGES | DO NOT SCALE DRAWING / REVISION |

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

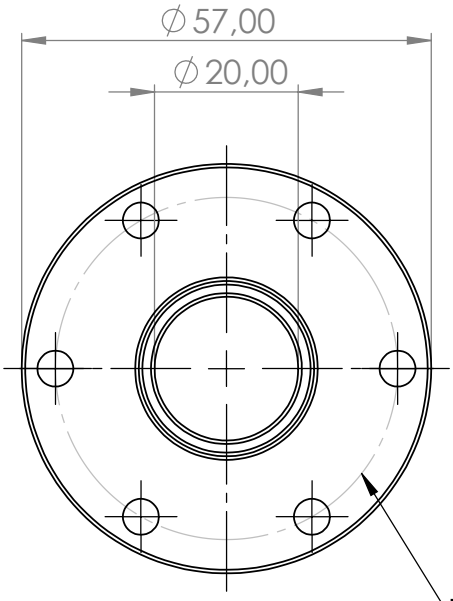
TITLE:

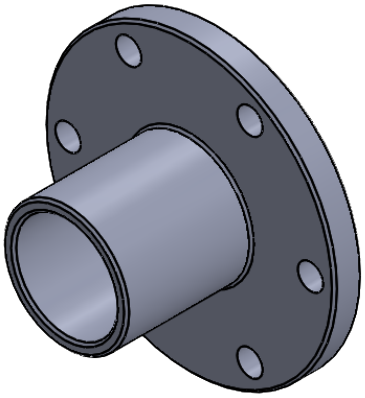
MATERIAL: S355J2

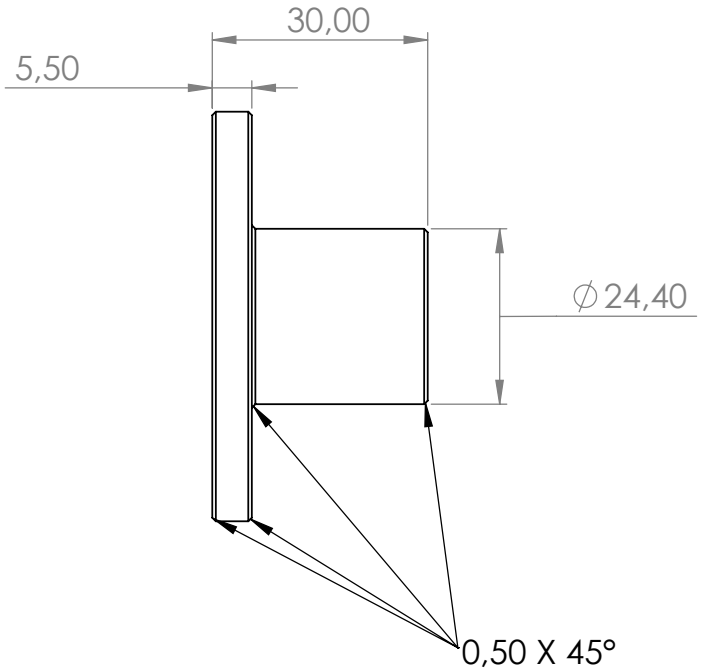
DWG NO. Axle Mastergruppe 6

WEIGHT: | SCALE:1:2 | SHEET 1 OF 1

# D.4 Outer Wheel Hub






| | | | |
|---|---|---|---|
| UNLESS OTHERWISE SPECIFIED:<br>DIMENSIONS ARE IN MILLIMETERS<br>SURFACE FINISH:<br>TOLERANCES:<br>LINEAR:<br>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: S355J2

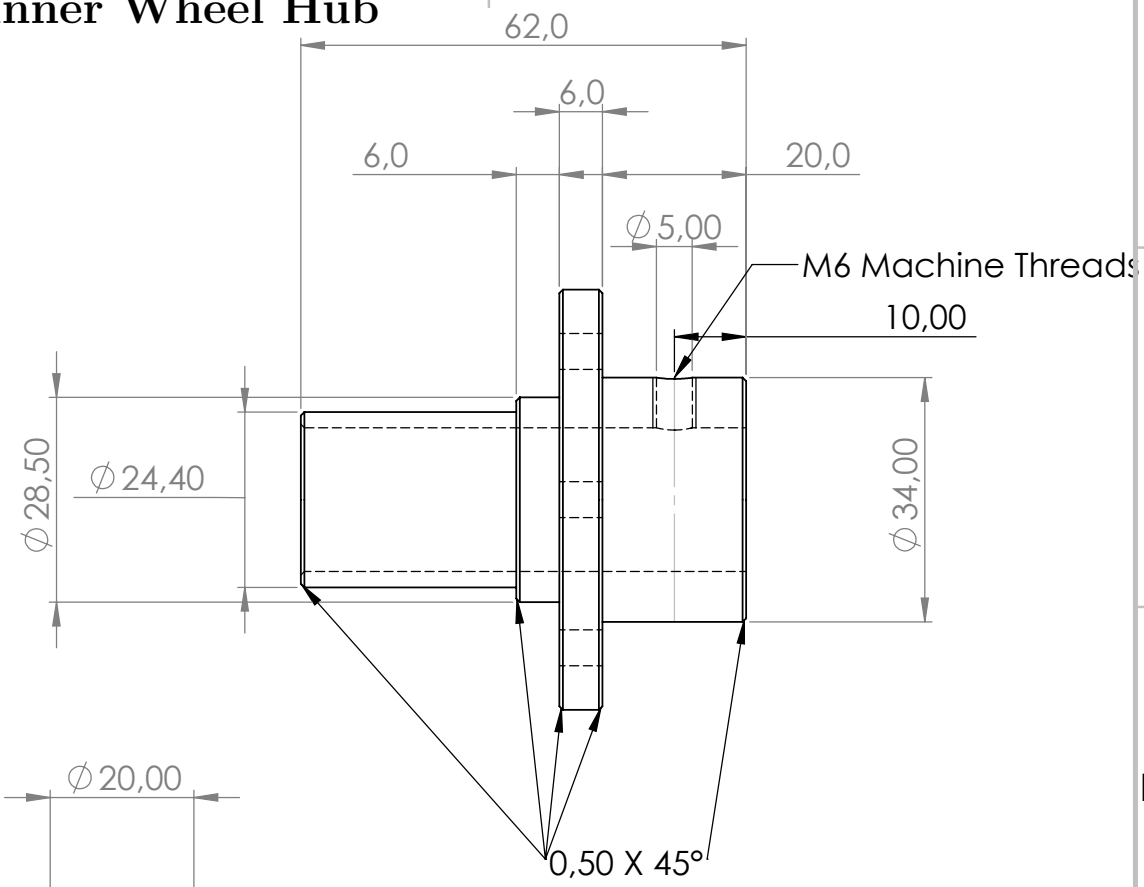
DWG NO. Hub 1 Mastergruppe 6

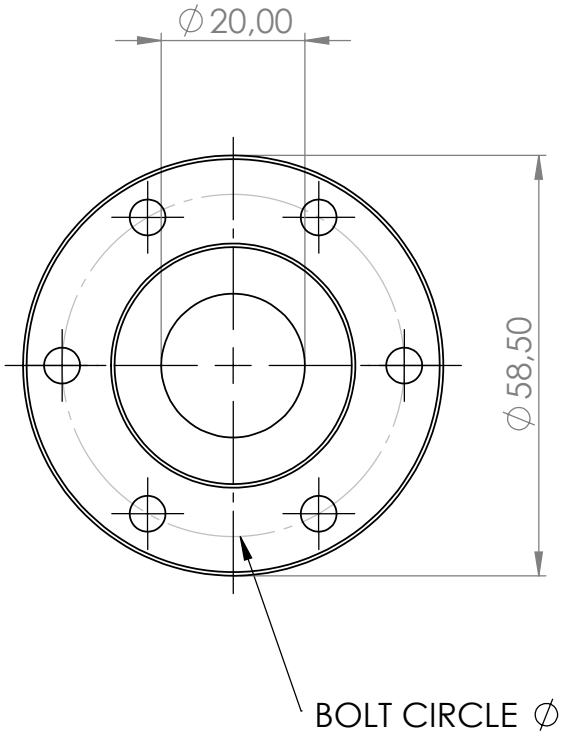
WEIGHT:

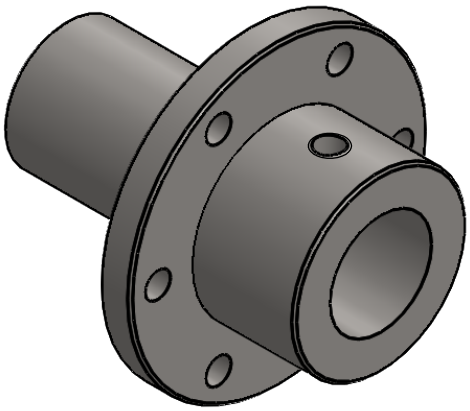
SCALE:1:1

SHEET 1 OF 1

# D.5 Inner Wheel Hub



| | | | |
|---|---|---|---|
| UNLESS OTHERWISE SPECIFIED:<br>DIMENSIONS ARE IN MILLIMETERS<br>SURFACE FINISH:<br>TOLERANCES:<br>LINEAR:<br>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: S355J2

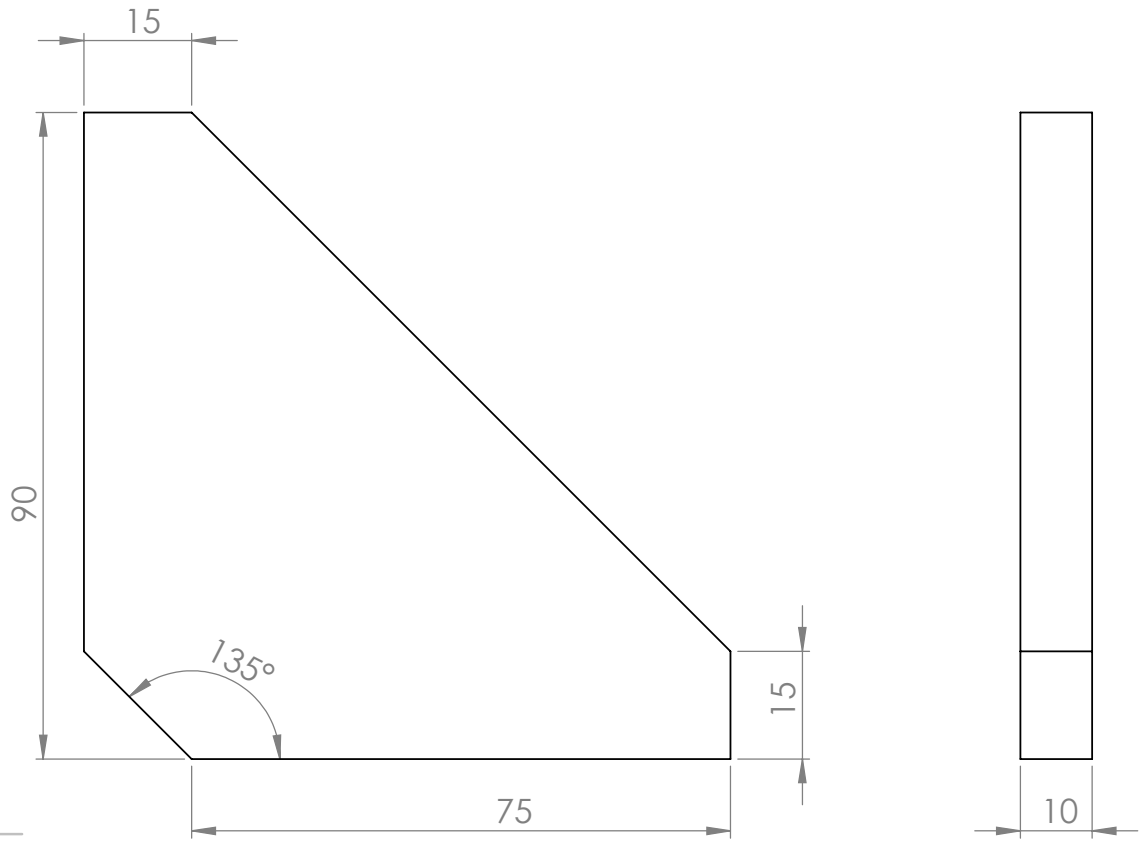
DWG NO. Hub 2 Mastergruppe 6

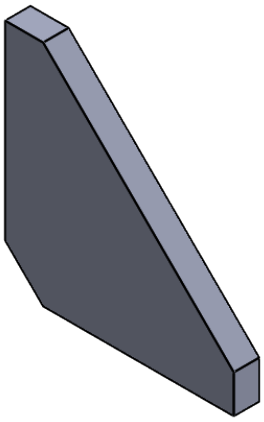
WEIGHT:

SCALE:1:2

SHEET 1 OF 1

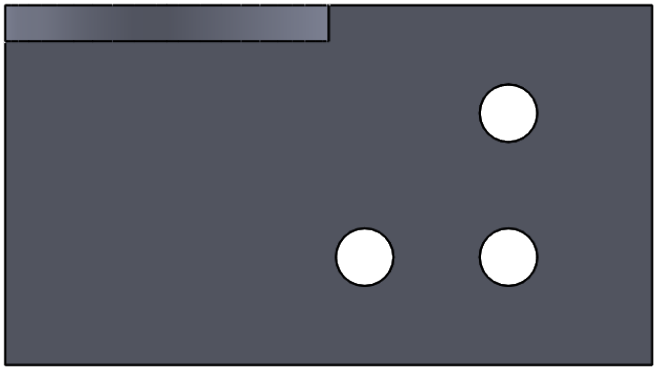
# D.6 Support Bracket

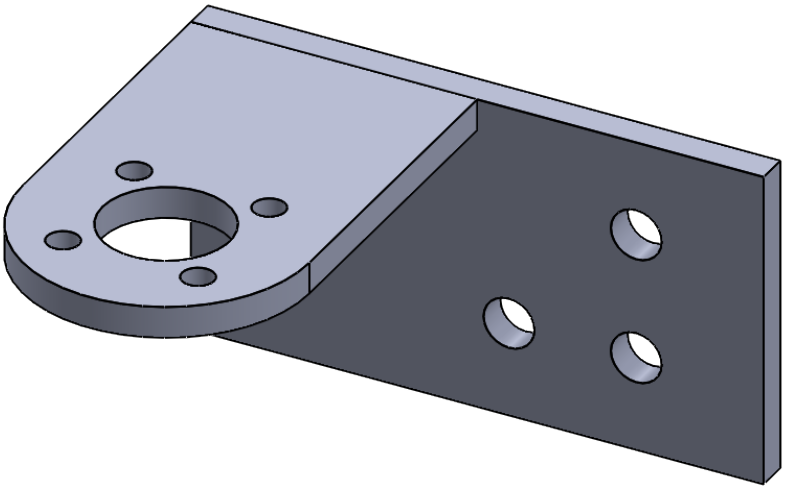


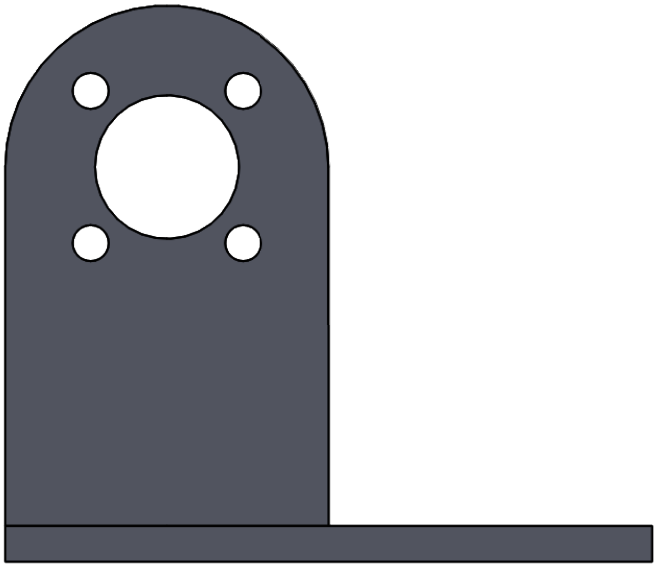



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| UNLESS OTHERWISE SPECIFIED:<br>DIMENSIONS ARE IN MILLIMETERS<br>SURFACE FINISH:<br>TOLERANCES:<br>LINEAR:<br>ANGULAR: | FINISH: | | DEBURR AND BREAK SHARP EDGES | | DO NOT SCALE DRAWING | REVISION | |
| | NAME | SIGNATURE | DATE | | | TITLE: | |
| DRAWN | | | | | | | |
| CHK'D | | | | | | | |
| APPV'D | | | | | | | |
| MFG | | | | | | | |
| Q.A | | | | MATERIAL: S355J2 | | DWG NO. Support Bracket | A4 |
| | | | | | | | |
| | | | | WEIGHT: | | SCALE:1:1 | SHEET 1 OF 1 |

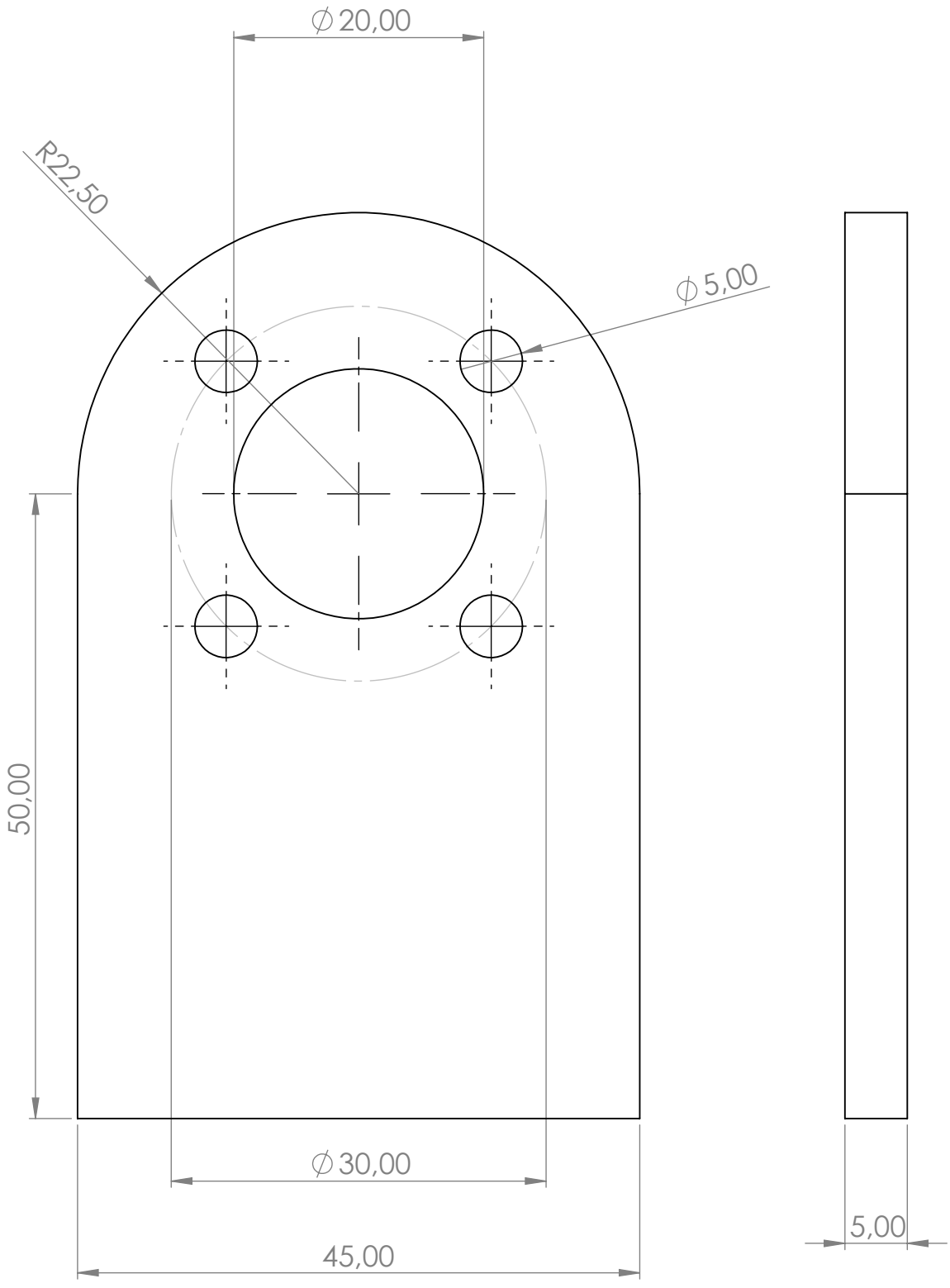
# D.7 Motor Mount







| | | | | | | | |
|---|---|---|---|---|---|---|---|
| UNLESS OTHERWISE SPECIFIED:<br>DIMENSIONS ARE IN MILLIMETERS<br>SURFACE FINISH:<br>TOLERANCES:<br>LINEAR:<br>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: S355J2 | | DWG NO. Motor Mount | A4 |
| | | | | | | | |
| | | | | WEIGHT: | | SCALE:1:2 | SHEET 1 OF 3 |




| | | | |
|---|---|---|---|
| UNLESS OTHERWISE SPECIFIED:<br>DIMENSIONS ARE IN MILLIMETERS<br>SURFACE FINISH:<br>TOLERANCES:<br>LINEAR:<br>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: S355J2

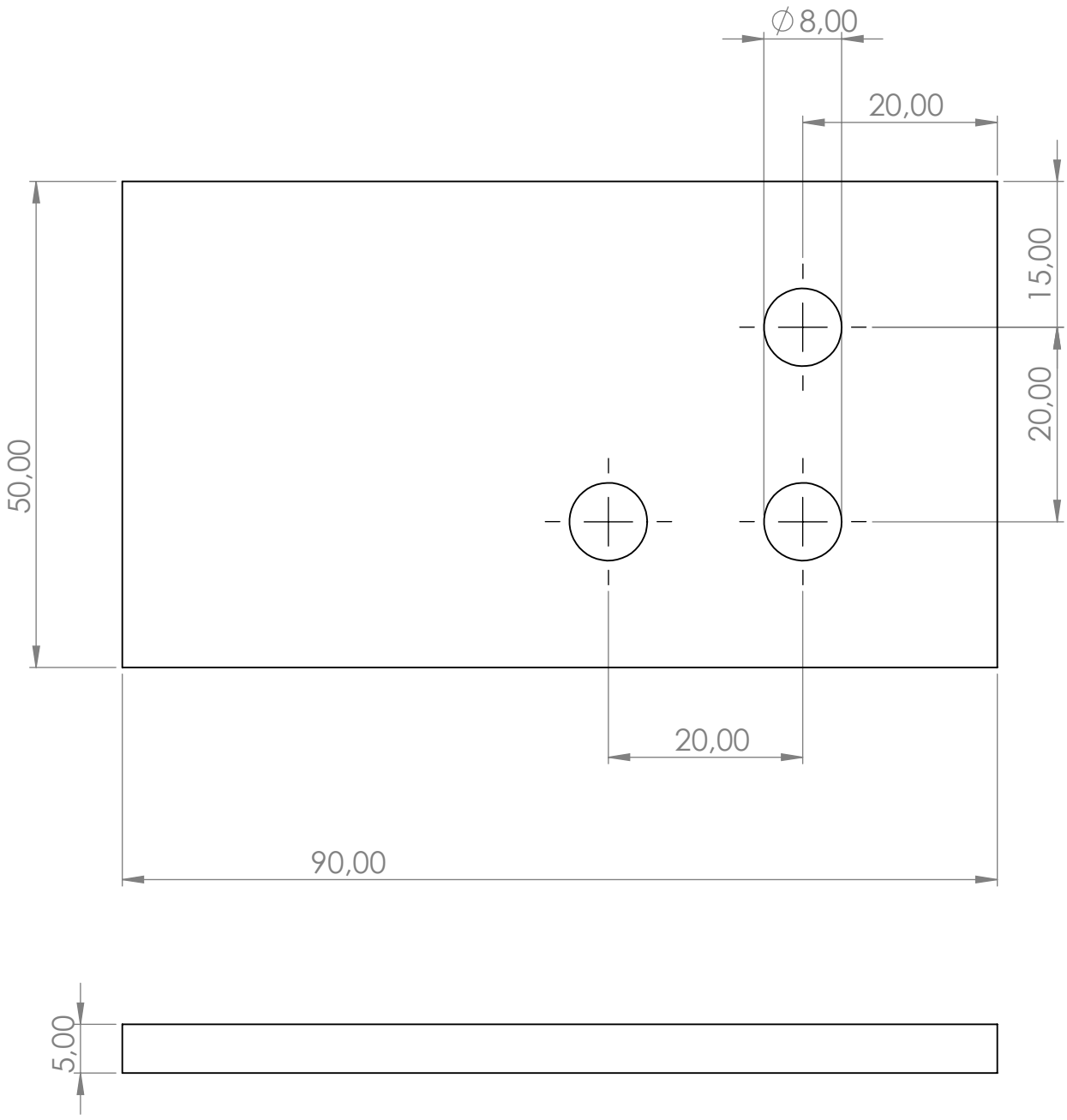
DWG NO. Motor Mount

A4

WEIGHT:

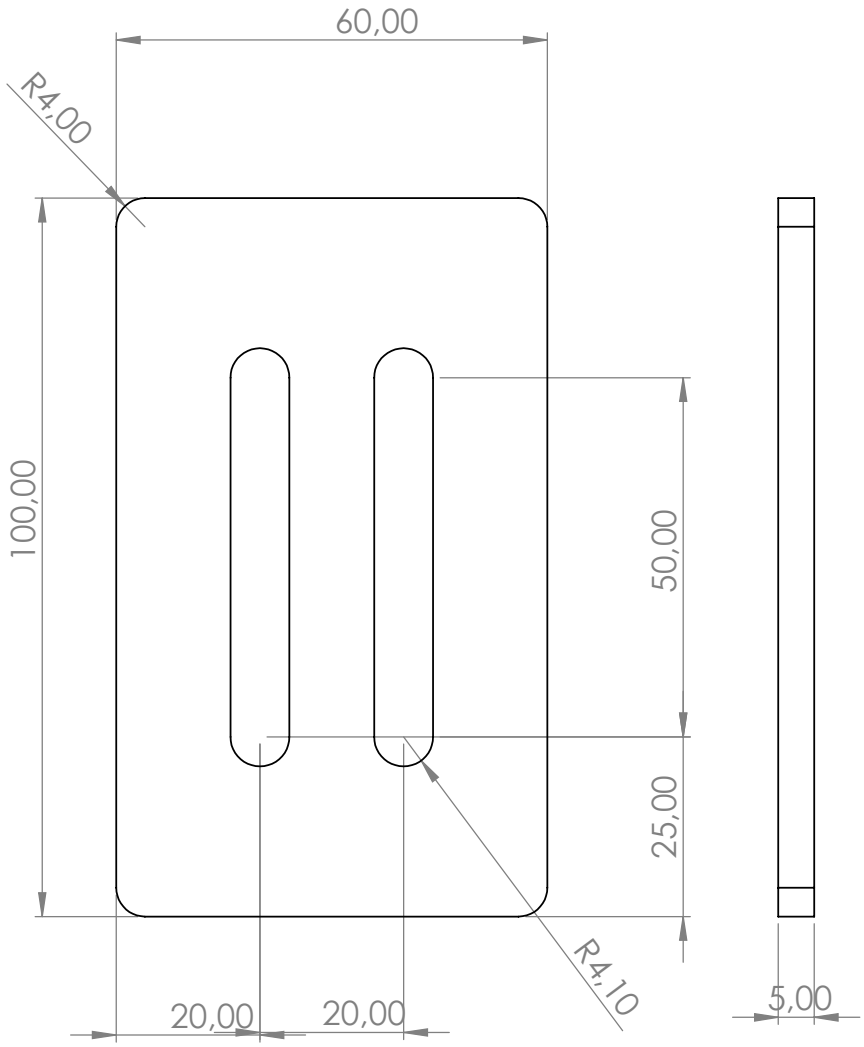
SCALE:1:2

SHEET 2 OF 3



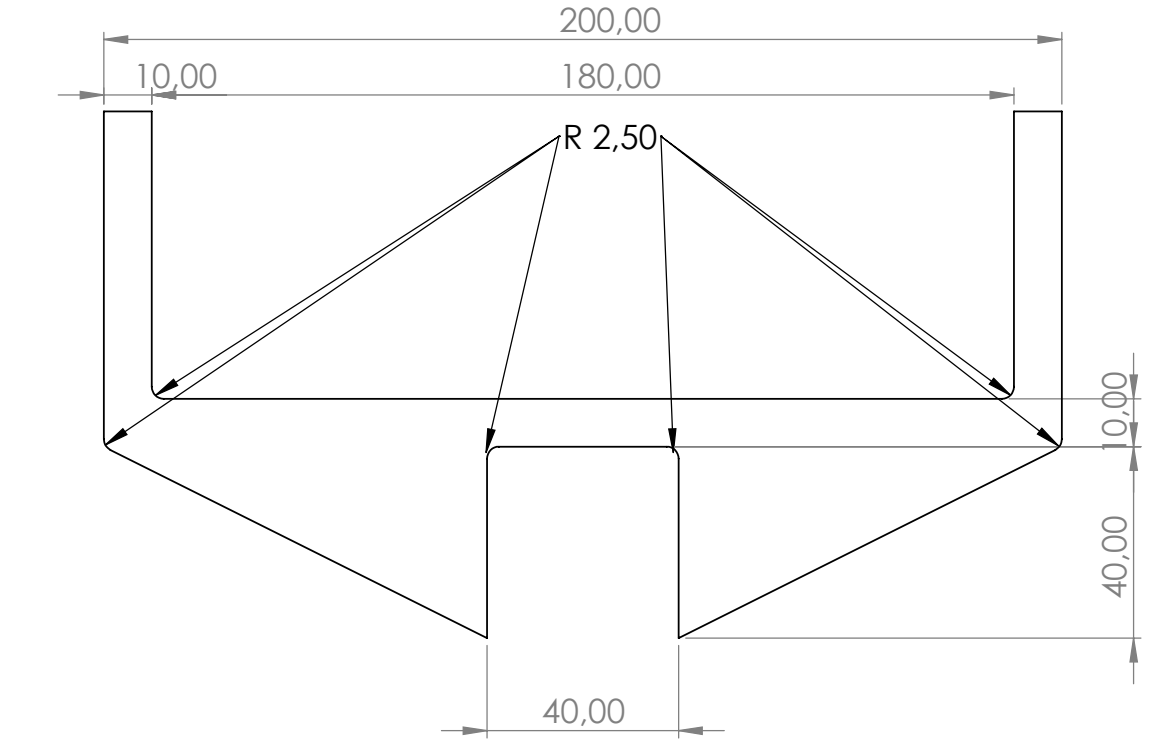

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| UNLESS OTHERWISE SPECIFIED:<br>DIMENSIONS ARE IN MILLIMETERS<br>SURFACE FINISH:<br>TOLERANCES:<br>LINEAR:<br>ANGULAR: | FINISH: | | DEBURR AND BREAK SHARP EDGES | | DO NOT SCALE DRAWING | REVISION | |
| | NAME | SIGNATURE | DATE | | | TITLE: | |
| DRAWN | | | | | | | |
| CHK'D | | | | | | | |
| APPV'D | | | | | | | |
| MFG | | | | | | | |
| Q.A | | | | MATERIAL: S355J2 | | DWG NO. Motor Mount | A4 |
| | | | | WEIGHT: | | SCALE:1:2 | SHEET 3 OF 3 |

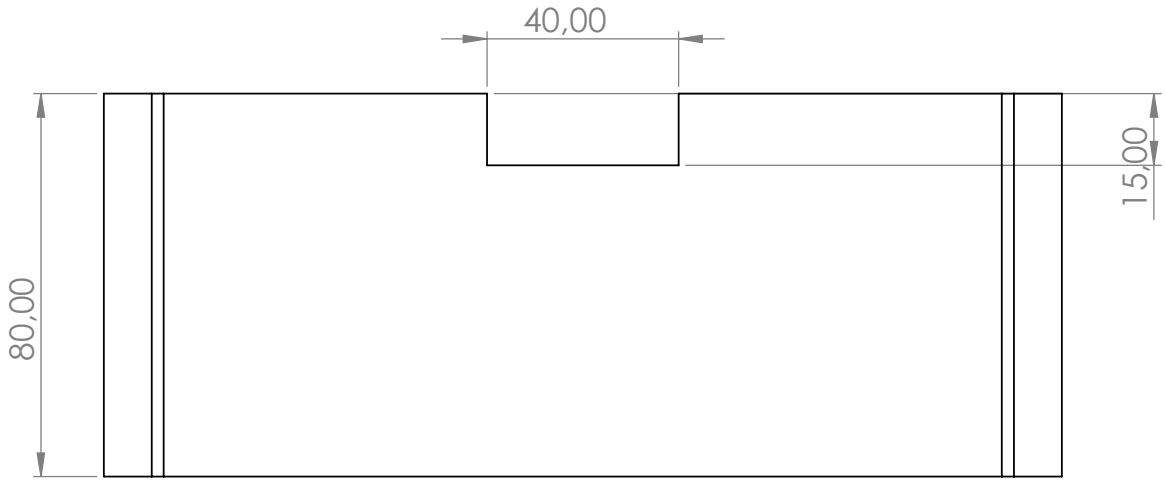
# D.8 Motor Bracket




| | | | | | | | |
|---|---|---|---|---|---|---|---|
| UNLESS OTHERWISE SPECIFIED:<br>DIMENSIONS ARE IN MILLIMETERS<br>SURFACE FINISH:<br>TOLERANCES:<br>LINEAR:<br>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: S355J2 | | DWG NO. Motor Bracket | A4 |
| | | | | | | | |
| | | | | WEIGHT: | | SCALE:1:1 | SHEET 1 OF 1 |

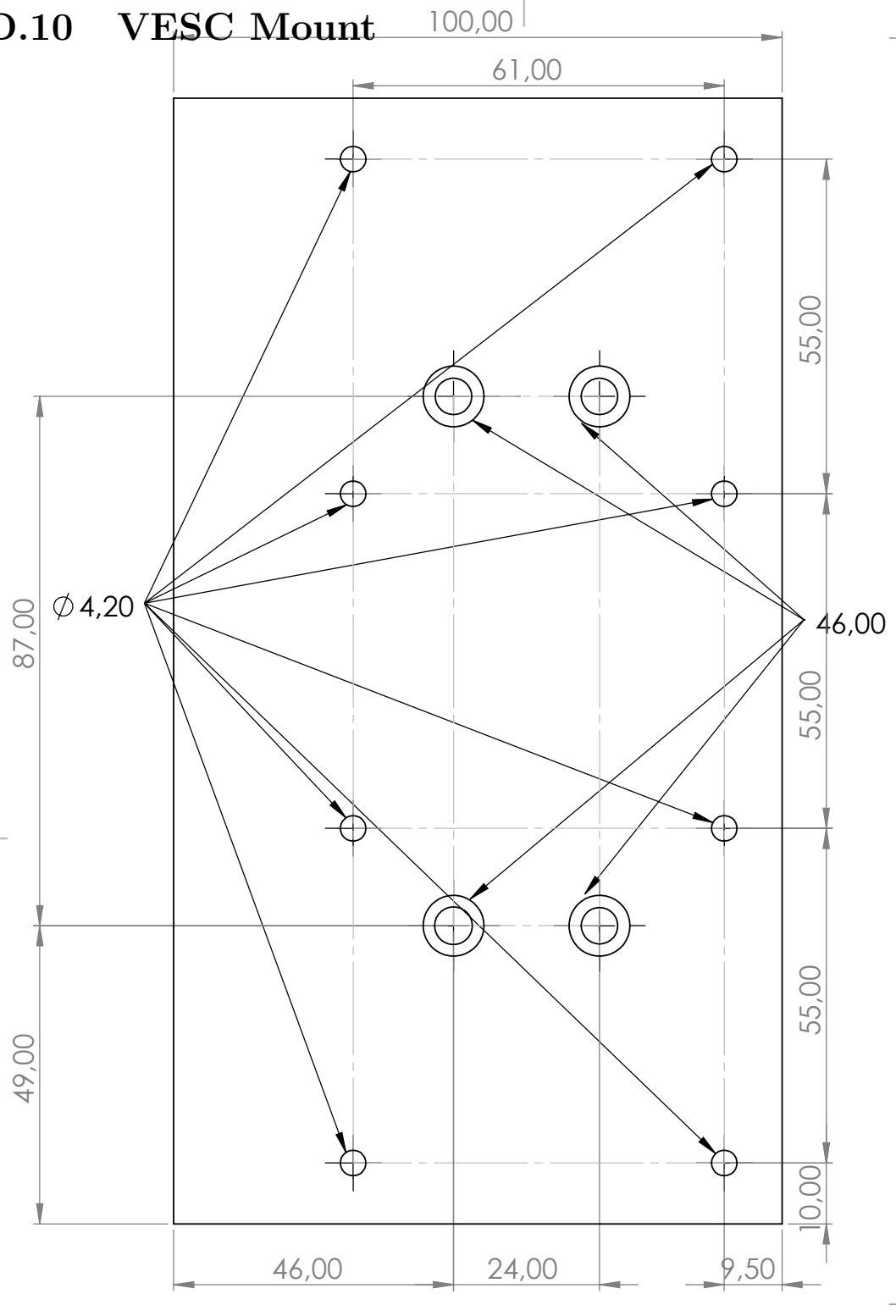
# D.9 Battery Mount




| | | | | | | | |
|---|---|---|---|---|---|---|---|
| UNLESS OTHERWISE SPECIFIED:<br>DIMENSIONS ARE IN MILLIMETERS<br>SURFACE FINISH:<br>TOLERANCES:<br>LINEAR:<br>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. Battery Mount | A4 |
| | | | | | | | |
| | | | | WEIGHT: | | SCALE:1:5 | SHEET 1 OF 1 |

# D.10 VESC Mount




| | | | |
|---|---|---|---|
| UNLESS OTHERWISE SPECIFIED:<br>DIMENSIONS ARE IN MILLIMETERS<br>SURFACE FINISH:<br>TOLERANCES:<br>LINEAR:<br>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: PLA

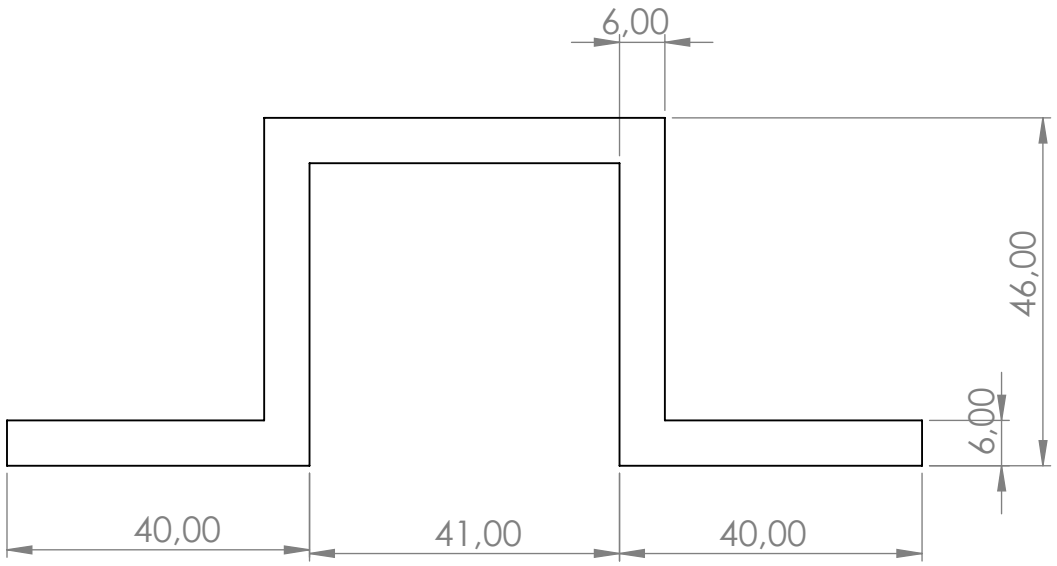
DWG NO. Vesc Mount

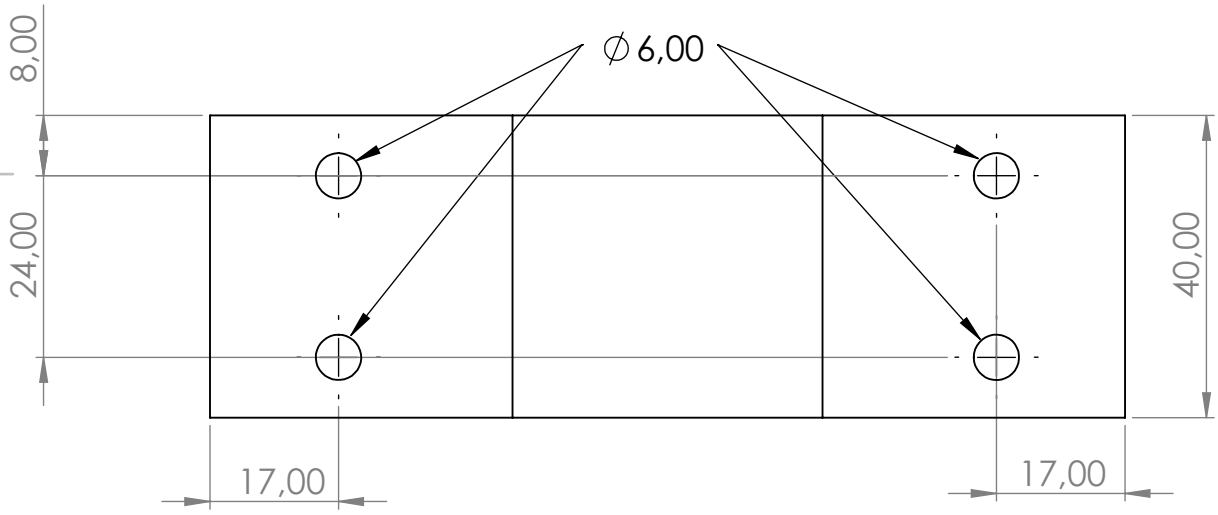
A4

WEIGHT:

SCALE:1:2

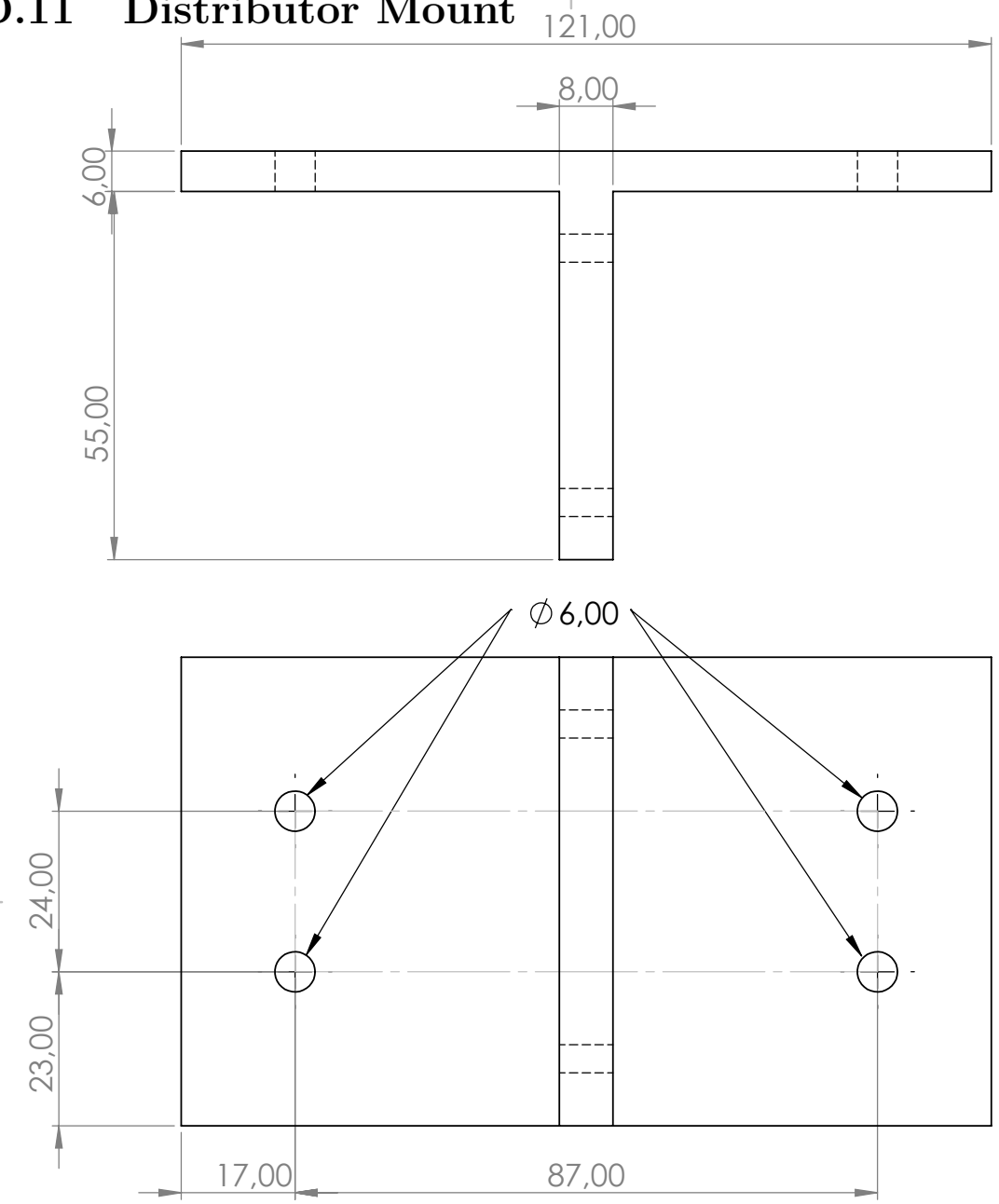
SHEET 1 OF 2



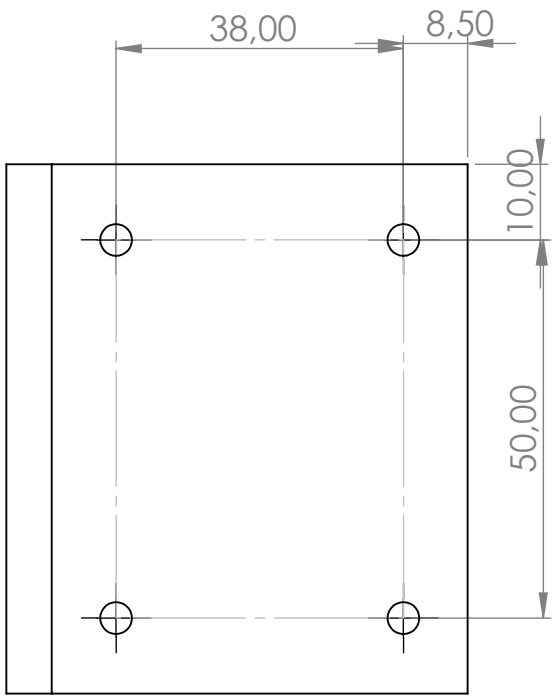



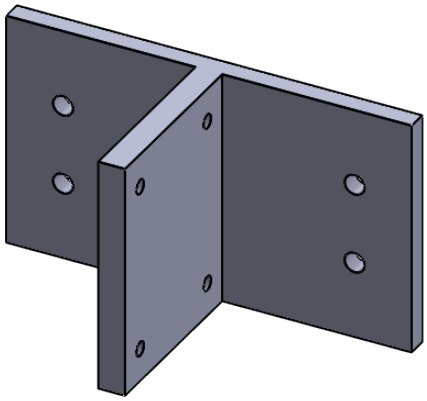

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| UNLESS OTHERWISE SPECIFIED:<br>DIMENSIONS ARE IN MILLIMETERS<br>SURFACE FINISH:<br>TOLERANCES:<br>LINEAR:<br>ANGULAR: | FINISH: | | DEBURR AND BREAK SHARP EDGES | | DO NOT SCALE DRAWING | REVISION | |
| | NAME | SIGNATURE | DATE | | | TITLE: | |
| DRAWN | | | | | | | |
| CHK'D | | | | | | | |
| APPV'D | | | | | | | |
| MFG | | | | | | | |
| Q.A | | | | MATERIAL: PLA | | DWG NO. Vesc Mount | A4 |
| | | | | | | | |
| | | | | WEIGHT: | | SCALE 1:1 | SHEET 2 OF 2 |

# D.11 Distributor Mount




| | | | | | | | |
|---|---|---|---|---|---|---|---|
| UNLESS OTHERWISE SPECIFIED:<br>DIMENSIONS ARE IN MILLIMETERS<br>SURFACE FINISH:<br>TOLERANCES:<br>LINEAR:<br>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: PLA | | DWG NO. Distributor Mount | A4 |
| | | | | | | | |
| | | | | WEIGHT: | | SCALE: 1:1 | SHEET 1 OF 3 |



| | | | |
|---|---|---|---|
| UNLESS OTHERWISE SPECIFIED:<br>DIMENSIONS ARE IN MILLIMETERS<br>SURFACE FINISH:<br>TOLERANCES:<br>LINEAR:<br>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: PLA

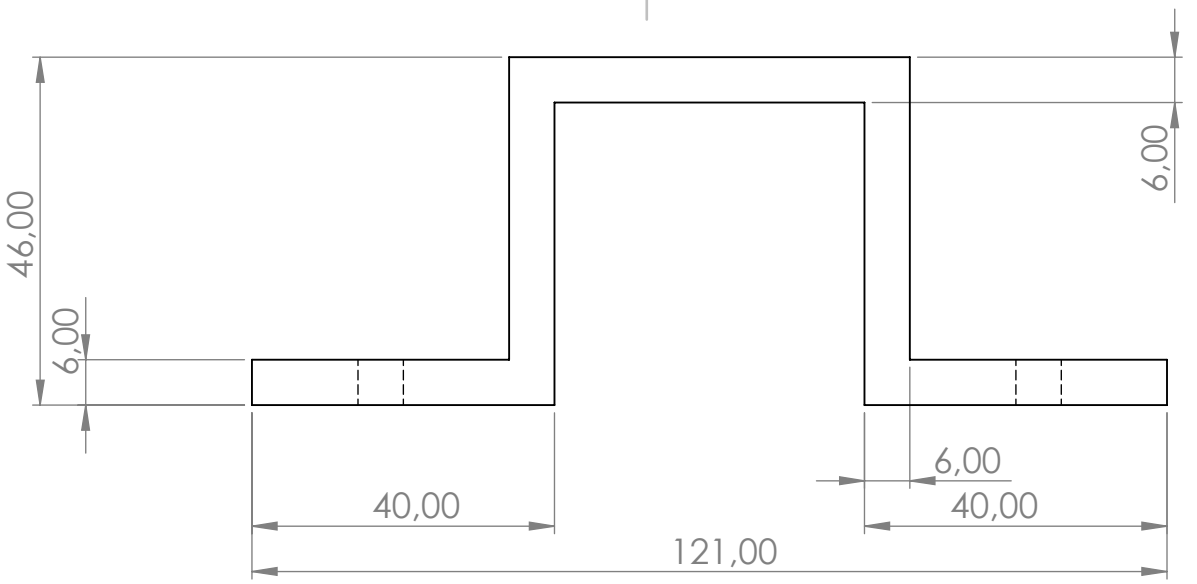
DWG NO. Distributor Mount

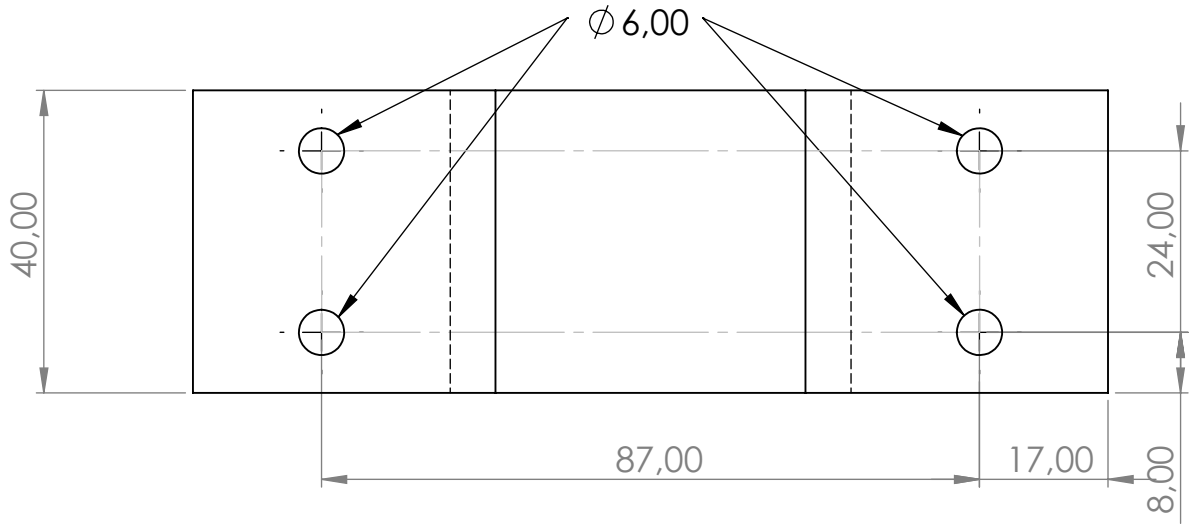
A4

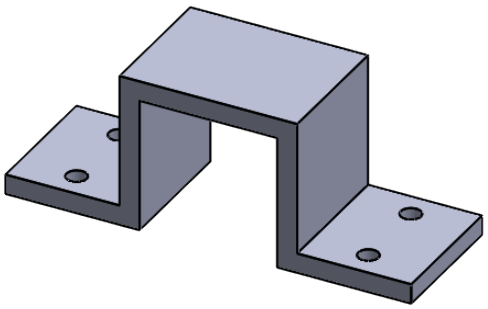
WEIGHT:

SCALE: 1:1

SHEET 2 OF 3



| | | | |
|---|---|---|---|
| UNLESS OTHERWISE SPECIFIED:<br>DIMENSIONS ARE IN MILLIMETERS<br>SURFACE FINISH:<br>TOLERANCES:<br>LINEAR:<br>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: PLA

DWG NO. Distributor Mount

A4

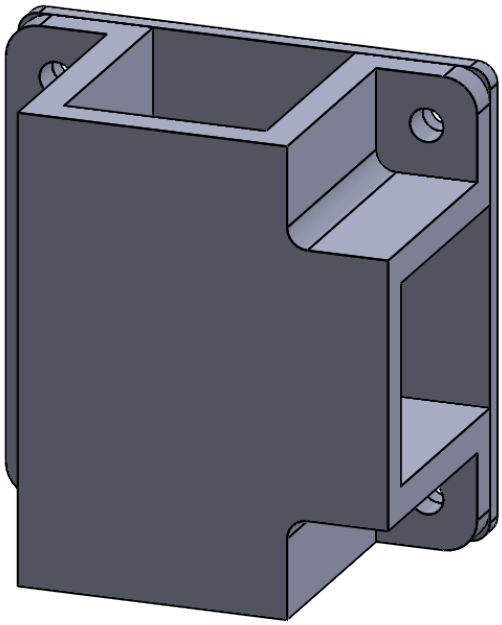
WEIGHT:

SCALE: 1:1

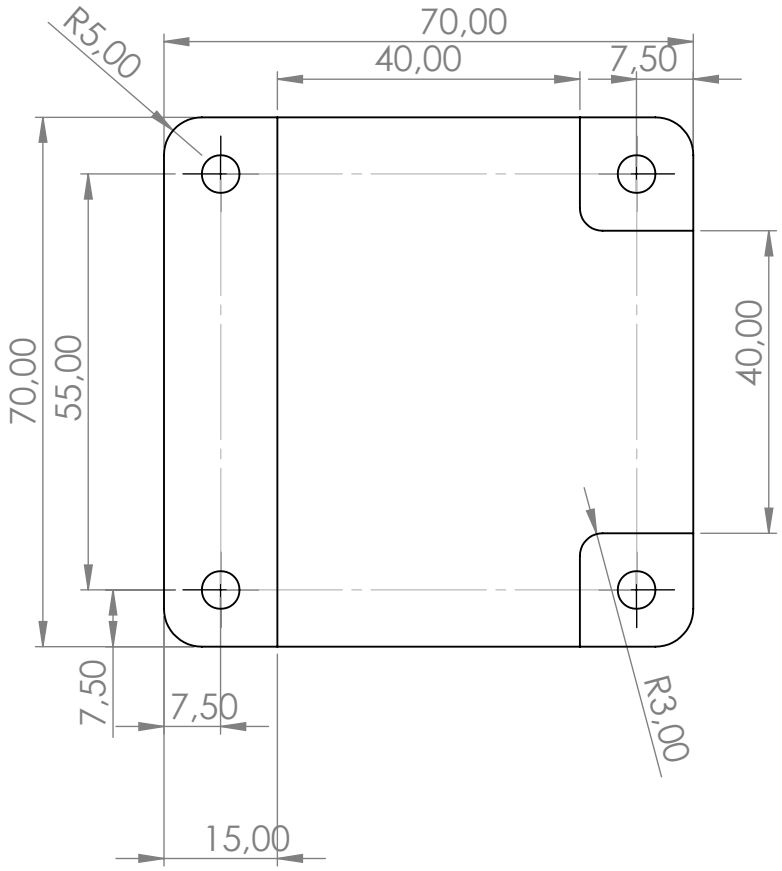
SHEET 3 OF 3

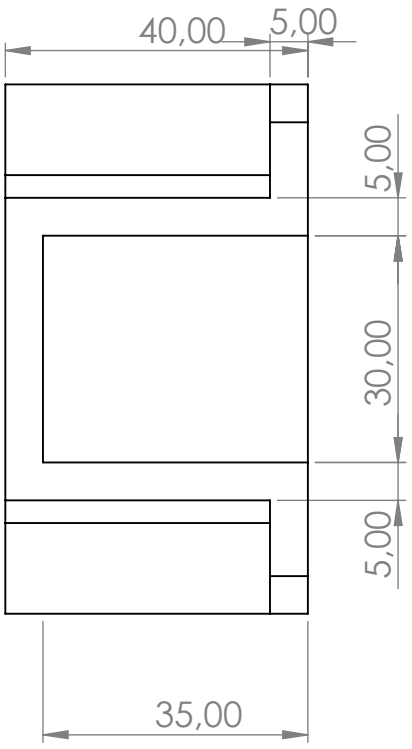
# D.12 Wireconnection Cover

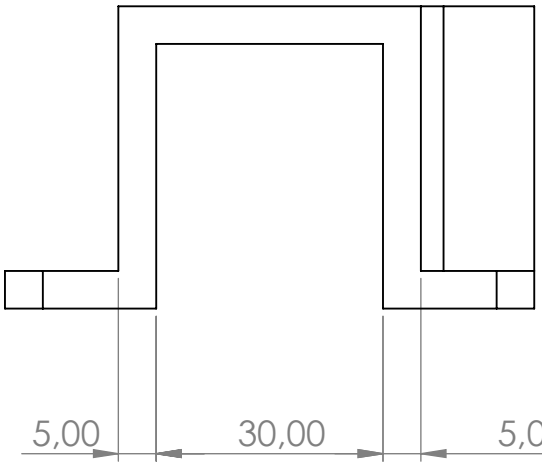
| ITEM NO. | PART NUMBER | DESCRIPTION | QTY. |
|---|---|---|---|
| 1 | WireBox_bot | | 1 |
| 2 | WireBox_top | | 1 |

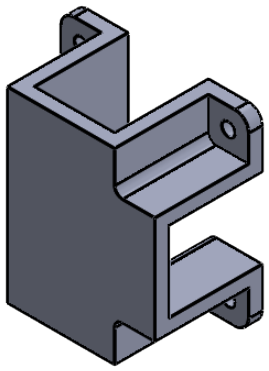


| | | | | | | | |
|---|---|---|---|---|---|---|---|
| UNLESS OTHERWISE SPECIFIED:<br>DIMENSIONS ARE IN MILLIMETERS<br>SURFACE FINISH:<br>TOLERANCES:<br>LINEAR:<br>ANGULAR: | FINISH: | | DEBURR AND BREAK SHARP EDGES | DO NOT SCALE DRAWING | REVISION | | |
| | NAME | SIGNATURE | DATE | | | TITLE: | |
| DRAWN | | | | | | | |
| CHK'D | | | | | | | |
| APPV'D | | | | | | | |
| MFG | | | | | | | |
| Q.A | | | | MATERIAL: PLA | | DWG NO. Wireconnection Cover | A4 |
| | | | | WEIGHT: | | SCALE:1:1 | SHEET 1 OF 3 |



| | | | |
|---|---|---|---|
| UNLESS OTHERWISE SPECIFIED:<br>DIMENSIONS ARE IN MILLIMETERS<br>SURFACE FINISH:<br>TOLERANCES:<br>LINEAR:<br>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: PLA

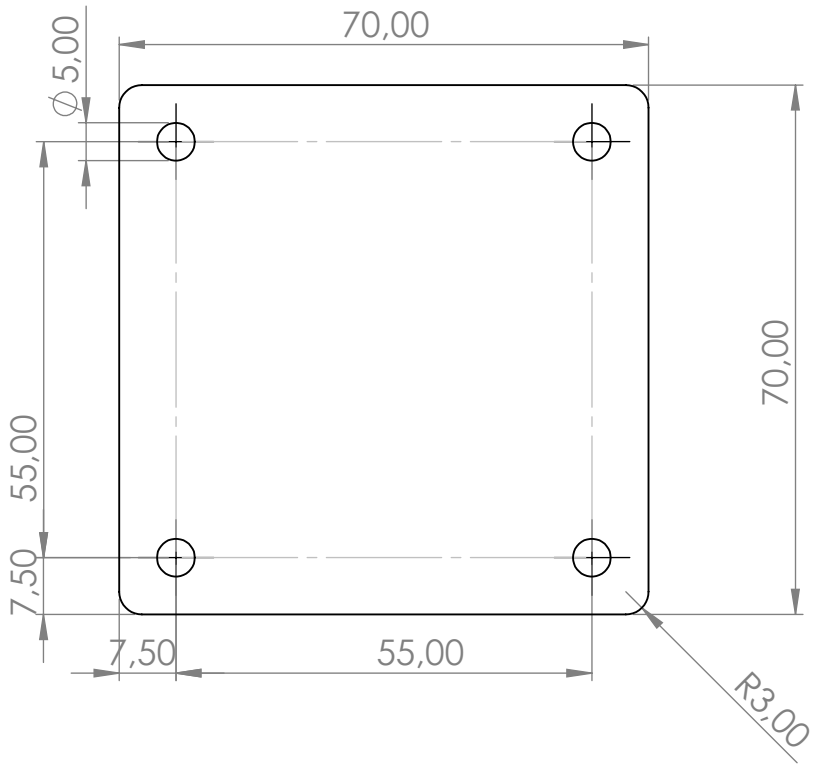
DWG NO. Wireconnection Cover

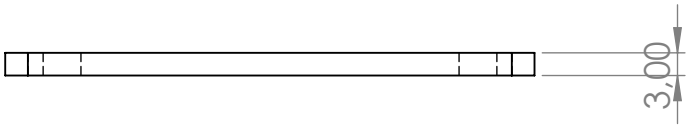
A4

WEIGHT:

SCALE:1:1

SHEET 2 OF 3




| | | | |
|---|---|---|---|
| UNLESS OTHERWISE SPECIFIED:<br>DIMENSIONS ARE IN MILLIMETERS<br>SURFACE FINISH:<br>TOLERANCES:<br>LINEAR:<br>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: PLA

DWG NO. Wireconnection Cover

A4

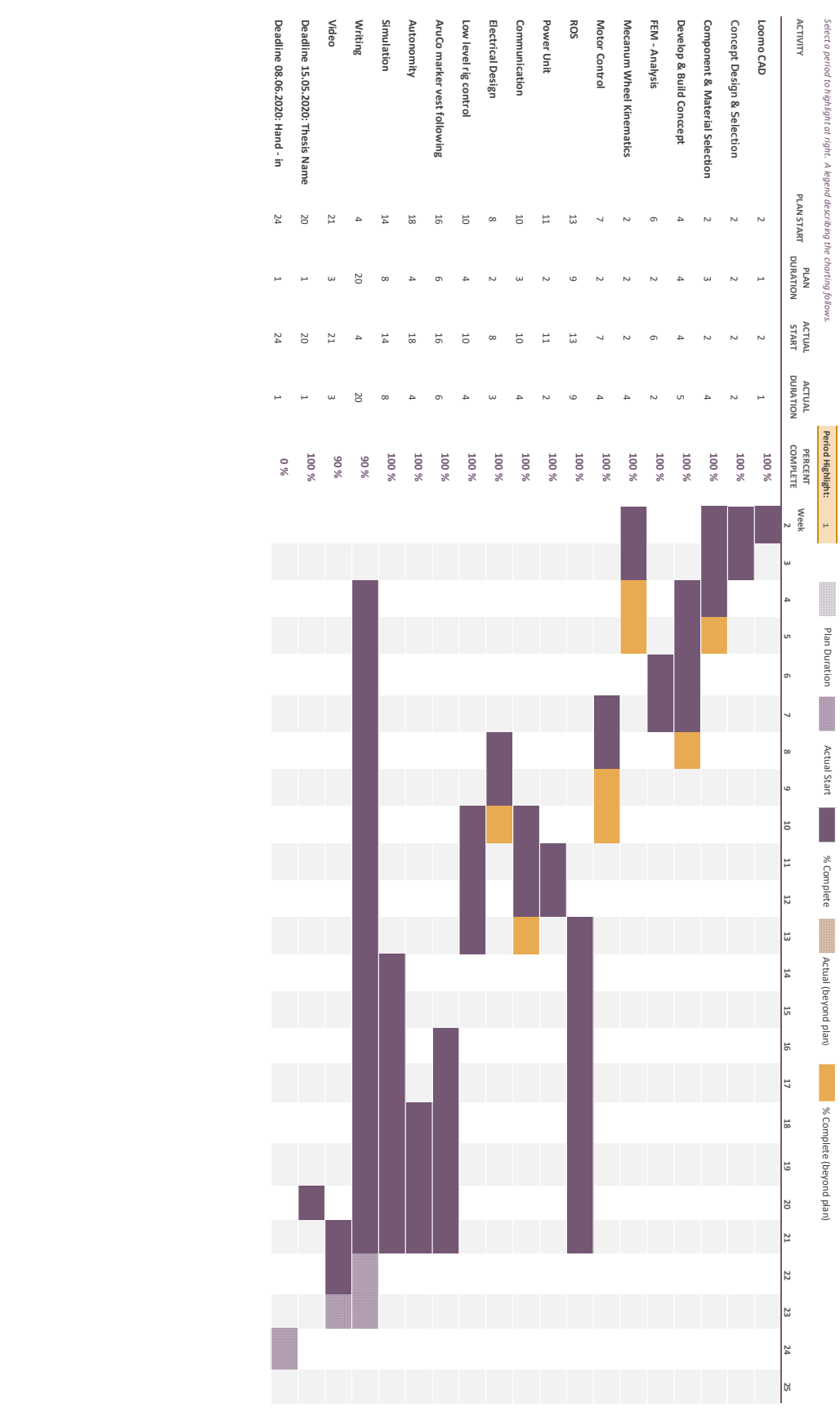
WEIGHT:

SCALE:1:1

SHEET 3 OF 3

# E. Project Management - Gantt chart

## E.1 Gantt chart - Main



## E.2 Gantt chart - Mechanical

## E.3 Gantt chart - Electrical

## E.4 Gantt chart - Programming

# F. ROS files

## F.1 Leap Motion assistive actuation node

```
#include "ros/ros.h"
#include <geometry_msgs/Twist.h>
#include <tf/transform_datatypes.h>

#include "leap_motion/Human.h"
#include "leap_motion/Hand.h"
#include "leap_motion/Finger.h"
#include "leap_motion/Bone.h"
#include "leap_motion/Gesture.h"
#include "leap_motion/leapros.h"

#include <iostream>

#include "../inc/lmc_listener.h"
#include "Leap.h"


// INIt Variables
float speed = 0;
float rot = 0;
float in_min = 0.1;
float out_min = -0.1;
float in_max = 0.3;
float out_max = 0.1;

//This tutorial demonstrates simple receipt of messages over the ROS system.
void LeapPoseCallback(const leap_motion::Human::ConstPtr& LeapPosemsg)
{
  //speed = LeapPosemsg->direction.y;
  //rot = LeapPosemsg->direction.x;
  //ROS_INFO("Number of hands [%i], Grab Strength [%f]", LeapPosemsg->
   nr_of_hands, LeapPosemsg->right_hand.grab_strength);
  //ROS_INFO("X: [%f], Y: [%f], Z: [%f],", LeapPosemsg->right_hand.direction.x
   , LeapPosemsg->right_hand.direction.y,  LeapPosemsg->right_hand.direction.
   z);


  //ROS_INFO("X: [%f], Y: [%f], Z: [%f]", LeapPosemsg->right_hand.palm_center.
   x,LeapPosemsg->right_hand.palm_center.y,LeapPosemsg->right_hand.
   palm_center.z);



  speed = LeapPosemsg->right_hand.palm_center.y;
  rot = -(LeapPosemsg->right_hand.direction.x);


  // mapping speed from 0.1-0.3 to -0.1-0.1
  speed = (speed-in_min)*(out_max-out_min)/(in_max-in_min) + out_min;
  if (speed < -0.1){
    speed = -0.1;
  }
```

```
  else if (speed > 0.1) {
    speed = 0.1;
  }
  else {
    speed = speed;
  }

// Check if hand is interpreted in stop or drive state

  if(LeapPosemsg->nr_of_hands >=1 && LeapPosemsg->right_hand.grab_strength
   <=0.15){
  speed = speed;
  rot = rot;
  }
  else{
    speed = 0;
    rot = 0;
  }

  //ROS_INFO("speed: [%f]", speed);


}

int main(int argc, char **argv){
  // Name of node
  ros::init(argc, argv, "leap_controller");

  ros::NodeHandle n;
  //subscriber
  ros::Subscriber sub = n.subscribe("/leap_motion/leap_filtered", 1000,
   LeapPoseCallback);
  //publisher
  ros::Publisher pub = n.advertise<geometry_msgs::Twist>("cmd_vel", 1);
  while(ros::ok()){
  // Create Twist message
    geometry_msgs::Twist twist;


    twist.linear.x = (speed)*15/2;
    twist.linear.y =0;
    twist.linear.z = 0;

    twist.angular.x = 0;
    twist.angular.y = 0;
    twist.angular.z = rot;
    //ros::spin();
    pub.publish(twist);
    ros::spinOnce();
  }


  return 0;
}
```

## F.2 Leap motion launch file

```
<launch>

   <!-- Load the listener parameters and start the driver node -->
   <rosparam file="$(find leap_motion)/config/listener_params.yaml" command="
      load" />
   <node pkg="leap_motion" type="leap_motion_driver_node" name="leap_driver"
      output="screen" />

   <!-- Load the filter parameters and start the filter node -->
   <rosparam file="$(find leap_motion)/config/filter_params.yaml" command="
      load" />
   <node pkg="leap_motion" type="leap_motion_filter_node" name="leap_filter"
      output="screen" />

   <group ns="leap_motion">
      <!-- Start the node that gets raw images from the Leap Motion controller
          -->
      <node pkg="leap_motion" type="leap_motion_camera_node" name="leap_camera
         " output="screen"/>
    </group>

</launch>
```

Listing F.1: Leap Motion Launch file

## F.3 ArUco Tracking and Navigation goals

```
#! /usr/bin/python

import numpy as np
import rospy
import roslib
from std_msgs.msg import String, Int32, Float32, Float64
from fiducial_msgs.msg import FiducialTransform, FiducialTransformArray,
   FiducialArray
from geometry_msgs.msg import Transform, Quaternion, Vector3
from nav_msgs.msg import Odometry
import math as m
import actionlib
from move_base_msgs.msg import MoveBaseAction, MoveBaseGoal

transform = FiducialTransform()
name = String()

tx = 0, ty = 0, tz = 0, rz = 0

x0 = 0, y0 = 0, x1 = 0, y1 = 0, x2 = 0, y2 = 0, x3 = 0, y3 = 0

pose_x = 0, pose_y = 0

goal_x = 0, goal_y = 0, yaw = 0

markerLength = 0.2

def fiducial_callback(msg):

    global img_seq, transform, tx, ty, tz, rz, name
    transform = msg.transforms
    name = msg.header.frame_id
```

```
    for f in transform:
        tx = f.transform.translation.x
        ty = f.transform.translation.y
        tz = f.transform.translation.z
        rz = f.transform.rotation.z

def vertecies_callback(msg):

    global x0, y0, x1, y1, x2, y2, x3, y3

    fiducials = msg.fiducials
    for n in fiducials:

        x0 = n.x0
        y0 = n.y0
        x1 = n.x1
        y1 = n.y1
        x2 = n.x2
        y2 = n.y2
        x3 = n.x3
        y3 = n.y3

def odom_callback(msg):
    global pose_x, pose_y, yaw

    pose_x = msg.pose.pose.position.x
    pose_y = msg.pose.pose.position.y

    yaw = msg.pose.pose.orientation.z

def movebase_client(target_x, target_y, target_r_z, target_r_w):

    client = actionlib.SimpleActionClient('move_base',MoveBaseAction)

    client.wait_for_server()

    goal = MoveBaseGoal()
    goal.target_pose.header.frame_id = "map"
    goal.target_pose.header.stamp = rospy.Time.now()

    goal.target_pose.pose.position.x = target_x
    goal.target_pose.pose.position.y = target_y

    goal.target_pose.pose.orientation.z = target_r_z
    goal.target_pose.pose.orientation.w = target_r_w

    client.send_goal(goal)
    wait = client.wait_for_result()
    if not wait:
        rospy.logerr("Action server not available!")
        rospy.signal_shutdown("Action server not available!")
    else:
        rospy.loginfo("Goal finished, send new one")
        return
    rospy.loginfo("Goal Sent with (1,0)")

def main():
    rospy.sleep(.25)

    # Initialize node
```

```
    rospy.init_node("aruco_2_navgoal", anonymous=True)
    rospy.loginfo("node Initialize")
    # Subsribers
    aruco_t_sub = rospy.Subscriber("fiducial_transforms",
FiducialTransformArray, fiducial_callback)
    aruco_vetecies_sub = rospy.Subscriber("fiducial_vertices", FiducialArray,
vertecies_callback)
    odom_pose_sub = rospy.Subscriber("odom", Odometry, odom_callback)

    rate = rospy.Rate(10)

    rospy.loginfo(">> calculating navigation goal")

    if x0 != 0.0 and tx != 0.0:
        #print(tx, ty, tz)
        t = np.array([tx, ty, tz])
        aruco_x = np.linalg.norm(t)
        a_cx = ((x0 + x1 + x2 + x3)/4)

        goal_r_z = m.sin(yaw/2)
        goal_r_w = m.cos(yaw/2)
        ratio = markerLength/(((x1 - x0)+(x2 - x3))/2)

        aruco_y = (256 - a_cx) * ratio

        angle = m.atan(aruco_y/aruco_x)

        alpha = yaw + angle
        if name == "frnt_cam_opt" and transform != []:
            if alpha >= 0 and alpha < m.pi/2:
                goal_x = pose_x - aruco_x*m.cos((alpha))
                goal_y = pose_y - aruco_y*m.sin((alpha))
            elif alpha > m.pi/2:
                goal_x = pose_x + aruco_y*m.sin((alpha))
                goal_y = pose_y - aruco_x*m.cos((alpha))
            elif alpha < 0 and alpha > -m.pi/2:
                goal_x = pose_x - aruco_x*m.cos((alpha))
                goal_y = pose_y + aruco_y*m.sin((alpha))
            elif alpha < -m.pi/2 :
                goal_x = pose_x + aruco_x*m.cos((alpha))
                goal_y = pose_y + aruco_y*m.sin((alpha))
            else:
                goal_x = 0
                goal_y = 0
        elif name == "rear_cam_opt" and transform != []:
            if alpha >= 0 and alpha < m.pi/2:
                goal_x = pose_x + aruco_x*m.cos((alpha))
                goal_y = pose_y + aruco_y*m.sin((alpha))
            elif alpha > m.pi/2:
                goal_x = pose_x - aruco_y*m.sin((alpha))
                goal_y = pose_y + aruco_x*m.cos((alpha))
            elif alpha < 0 and alpha > -m.pi/2:
                goal_x = pose_x + aruco_x*m.cos((alpha))
                goal_y = pose_y - aruco_y*m.sin((alpha))
            elif alpha < -m.pi/2 :
                goal_x = pose_x - aruco_x*m.cos((alpha))
                goal_y = pose_y - aruco_y*m.sin((alpha))
            else:
                goal_x = 0
                goal_y = 0
        elif name == "left_cam_opt" and transform != []:
```

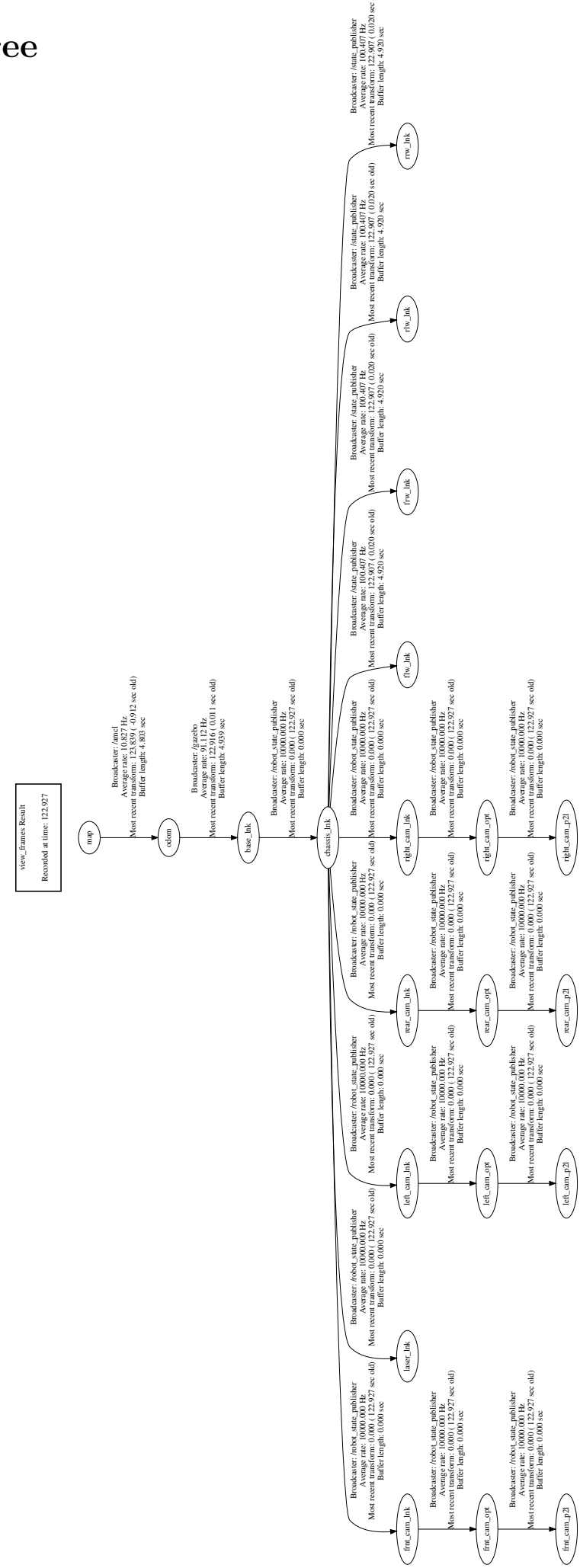
```
            if alpha >= 0 and alpha < m.pi/2:
                goal_y = pose_y - aruco_x*m.cos((alpha))
                goal_x = pose_x + aruco_y*m.sin((alpha))
            elif alpha > m.pi/2:
                goal_y = pose_y + aruco_y*m.sin((alpha))
                goal_x = pose_x + aruco_x*m.cos((alpha))
            elif alpha < 0 and alpha > -m.pi/2:
                goal_y = pose_y - aruco_x*m.cos((alpha))
                goal_x = pose_x - aruco_y*m.sin((alpha))
            elif alpha < -m.pi/2 :
                goal_y = pose_y + aruco_x*m.cos((alpha))
                goal_x = pose_x - aruco_y*m.sin((alpha))
            else:
                goal_x = 0
                goal_y = 0
        elif name == "right_cam_opt" and transform != []:
            if alpha >= 0 and alpha < m.pi/2:
                goal_y = pose_y - aruco_x*m.cos((alpha))
                goal_x = pose_x - aruco_y*m.sin((alpha))
            elif alpha > m.pi/2:
                goal_y = pose_y - aruco_y*m.sin((alpha))
                goal_x = pose_x + aruco_x*m.cos((alpha))
            elif alpha < 0 and alpha > -m.pi/2:
                goal_y = pose_y + aruco_x*m.cos((alpha))
                goal_x = pose_x - aruco_y*m.sin((alpha))
            elif alpha < -m.pi/2 :
                goal_y = pose_y + aruco_x*m.cos((alpha))
                goal_x = pose_x + aruco_y*m.sin((alpha))
            else:
                goal_x = 0
                goal_y = 0
        else:
            goal_x = 0
            goal_y = 0

        rospy.loginfo(">> Sending Navigation goal")
        movebase_client(goal_x, goal_y, goal_r_z, goal_r_w)


    else:

        movebase_client(pose_x, pose_y, 0, 1)
        print("No aruco detected")


if __name__ == "__main__":

    while True:
        main()
```

## F.4 Aruco launch file

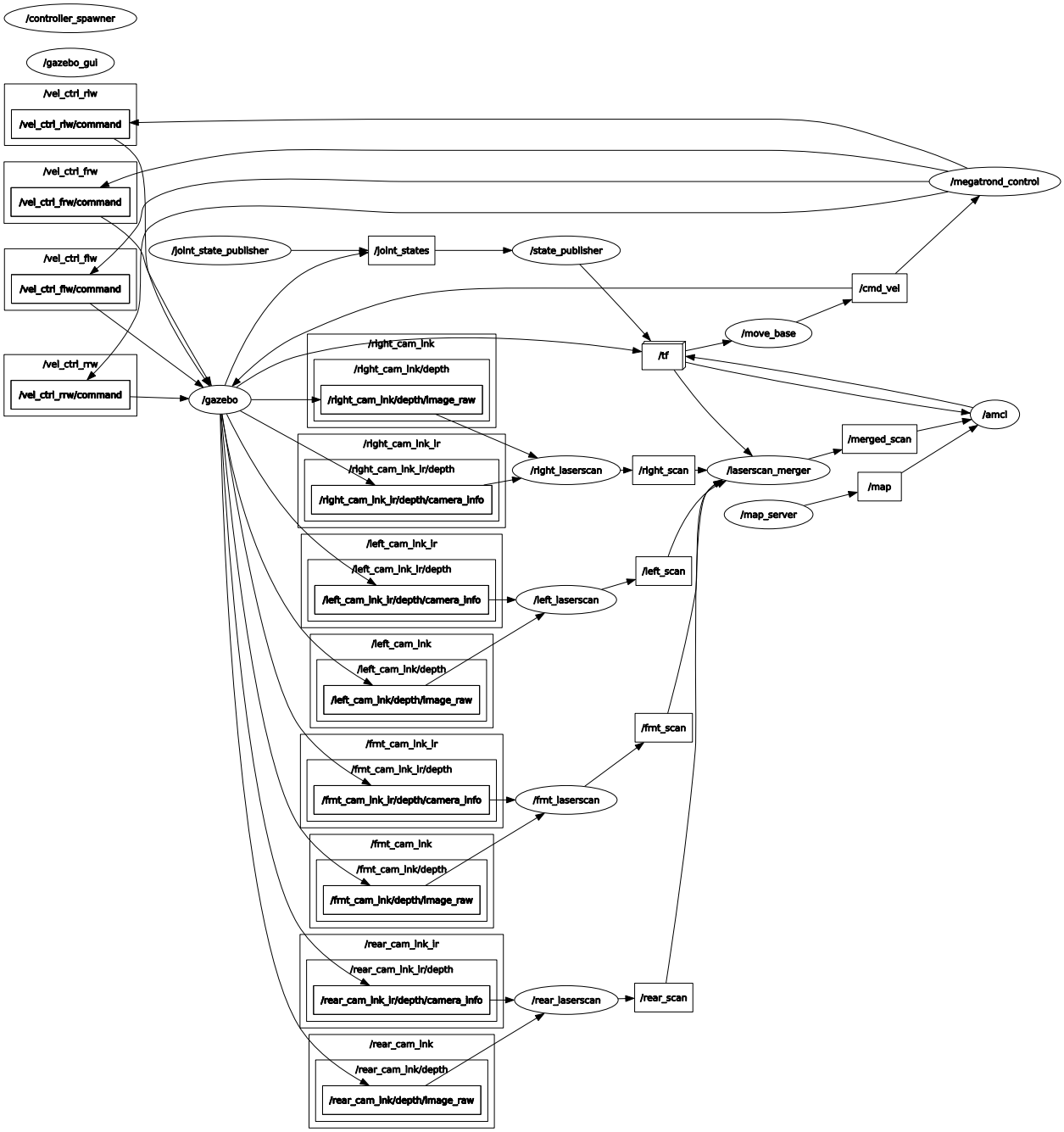
```
<!-- Run the aruco_detect node -->
<launch>
  <!-- namespace for camera input -->
  <arg name="camera" default="/frnt_cam_lnk/color"/>
  <arg name="image" default="image_raw"/>
  <arg name="transport" default="compressed"/>
  <arg name="fiducial_len" default="0.20"/>
  <arg name="dictionary" default="3"/>
  <arg name="do_pose_estimation" default="true"/>
  <arg name="ignore_fiducials" default="" />
  <arg name="fiducial_len_override" default="" />

  <node pkg="aruco_detect" name="aruco_detect_front"
    type="aruco_detect" output="screen" respawn="false">
    <param name="image_transport" value="$(arg transport)"/>
    <param name="publish_images" value="true" />
    <param name="fiducial_len" value="$(arg fiducial_len)"/>
    <param name="dictionary" value="$(arg dictionary)"/>
    <param name="do_pose_estimation" value="$(arg do_pose_estimation)"/>
    <param name="ignore_fiducials" value="$(arg ignore_fiducials)"/>
    <param name="fiducial_len_override" value="$(arg fiducial_len_override)"/>
    <remap from="/camera/compressed"
       to="$(arg camera)/$(arg image)/$(arg transport)"/>
    <remap from="/camera_info" to="$(arg camera)/camera_info"/>
  </node>
</launch>
```

## F.5 Transform Configuration Tree

# F.6 tf-tree


view_frames Result
Recorded at time: 122.927
map
Broadcaster: /amcl
Average rate: 10.827 Hz
Most recent transform: 123.839 ( -0.912 sec old)
Buffer length: 4.803 sec
odom
Broadcaster: /gazebo
Average rate: 91.112 Hz
Most recent transform: 122.916 ( 0.011 sec old)
Buffer length: 4.959 sec
base_link
Broadcaster: /robot_state_publisher
Average rate: 10000.000 Hz
Most recent transform: 0.000 ( 122.927 sec old)
Buffer length: 0.000 sec
chassis_link
Broadcaster: /robot_state_publisher
Average rate: 10000.000 Hz
Most recent transform: 0.000 ( 122.927 sec old)
Buffer length: 0.000 sec
laser_link
Broadcaster: /state_publisher
Average rate: 100.407 Hz
Most recent transform: 122.907 ( 0.020 sec old)
Buffer length: 4.920 sec
rfw_link
rrw_link
lfw_link
lrw_link
front_cam_link
front_cam_opt
front_cam_p2l
left_cam_link
left_cam_opt
left_cam_p2l
rear_cam_link
rear_cam_opt
rear_cam_p2l
right_cam_link
right_cam_opt
right_cam_p2l

# F.7 ROS graph - node structure



# G. Matlab Calculations

## G.1 Shaft Analysis (Force diagrams)

```
clear all;
close all;
%Lengths:
L = (204.5-31)*10^(-3); %m
a = (31/2+15+5)*10^(-3); %m
r = (203/2)*10^(-3); %m
%Loads:
F_w = 1000; %N
F_t = 250; %N
F_a = 250;
g = 9.81; %m/s^2
%Reaction Forces:
R_bx = F_t/2;
R_ax = F_t-R_bx;
R_ay = F_a;
R_bz = (a*F_w/2+(L-a)*F_w/2+r*F_a)/L;
R_az = F_w/2+F_w/2-R_bz;
R_ay = F_a;
%Counters:
counter = 1;
x = 0;
dx = 1e-4;
x_max = 116.76e-3;
while x<=L
    if x==0
        M_xy = 0;
        V_xy = R_ax;
        N_xy = R_ay;
        M_zy = 0;
        V_zy = R_az;
        N_zy = R_ay;
    elseif x>0 && x<a
        M_xy = x*R_ax;
        V_xy = -R_ax;
        N_xy = R_ay;
        M_zy = x*R_az;
        V_zy = -R_az;
        N_zy = R_ay;
    elseif x>=a && x<(L-a)
        M_xy = x*R_ax-(x-a)*F_t/2;
        V_xy = -R_ax+F_t/2;
        N_xy = R_ay-F_a/2;
        M_zy = x*R_az-(x-a)*F_w/2+r*F_a/2;
        V_zy = -R_az+F_w/2;
        N_zy = R_ay-F_a/2;
    elseif x>=(L-a) && x<L
        M_xy = x*R_ax-(x-a)*F_t/2-(x-a-(L-2*a))*F_t/2;
        V_xy = -R_ax+F_t/2+F_t/2;
        N_xy = R_ay-F_a/2-F_a/2;
        M_zy = x*R_az - (x-a)*F_w/2 - (x-a-(L-2*a))*F_w/2 + r*F_a;
        V_zy = -R_az+F_w/2+F_w/2;
        N_zy = -F_a+R_ay;
```

```
    end

    M_xy_plot(counter)= M_xy;
    V_xy_plot(counter)= V_xy;
    N_xy_plot(counter)= N_xy;

    M_zy_plot(counter)= M_zy;
    V_zy_plot(counter)= V_zy;
    N_zy_plot(counter)= N_zy;

    x_plot(counter) = x;
    counter = counter + 1;
    x = x+dx;
end

figure('Renderer', 'painters', 'Position', [500 200 623.62204724 400])
subplot(3,1,1)
plot(x_plot*1000,M_xy_plot,'linewidth',1.5)
xlim([0 L*1000])
title('Bending Moment')
xlabel('Shaft Length [mm]')
ylabel('Moment [Nm]')
subplot(3,1,2)
plot(x_plot*1000,V_xy_plot,'linewidth',1.5)
xlim([0 L*1000])
title('Shear Force')
xlabel('Shaft Length [mm]')
ylabel('Force [N]')
subplot(3,1,3)
plot(x_plot*1000,N_xy_plot,'linewidth',1.5)
xlim([0 L*1000])
title('Normal Force')
xlabel('Shaft Length [mm]')
ylabel('Force [N]')

figure('Renderer', 'painters', 'Position', [500 200 623.62204724 400])
subplot(3,1,1)
plot(x_plot*1000,M_zy_plot,'linewidth',1.5)
xlim([0 L*1000])
title('Bending Moment')
xlabel('Shaft Length [mm]')
ylabel('Moment [Nm]')
subplot(3,1,2)
plot(x_plot*1000,-V_zy_plot,'linewidth',1.5)
xlim([0 L*1000])
title('Shear Force')
xlabel('Shaft Length [mm]')
ylabel('Force [N]')
subplot(3,1,3)
plot(x_plot*1000,N_zy_plot,'linewidth',1.5)
xlim([0 L*1000])
title('Normal Force')
xlabel('Shaft Length [mm]')
ylabel('Force [N]')
```

## G.2 Deflection

```
clear all;
close all;

%Forces:
F_w = 1000; %N
F_t = 250; %N
F_a = 250; %N
M = 25.3750; %Nm
g = 9.81; %m/s^2
%Lengths:
L = (204.5-31)*10^(-3); %m
a = (31/2+15+5)*10^(-3); %m
r = (203/2)*10^(-3); %m
d = 20e-3; %m
%Properties:
E = 200*10^9; %Pa
I = (pi/64)*(d^4);
%Counters:
counter = 1;
x = 0;
dx = 1e-3;
x_max = 116.76e-3;

while x<=L
    a1 = a;
    b1 = L-a;
    a2 = b1;
    b2 = a;
    if x>=0 && x<a1
        v_load_xy = -(F_t/2*x)/(6*E*I)*(3*a*L-3*a^2-x^2);
        v_load_zy = -(F_w/2*x)/(6*E*I)*(3*a*L-3*a^2-x^2);
        v_M_a1 = (M*x)/(6*L*E*I)*(6*a1*L-3*a1^2-2*L^2-x^2);
    else
        v_load_xy = -(F_t/2*a)/(6*E*I)*(3*L*x-3*x^2-a^2);
        v_load_zy = -(F_w/2*a)/(6*E*I)*(3*L*x-3*x^2-a^2);
        v_M_a1 = -(M*(L-x))/(6*L*E*I)*(6*b1*L-3*b1^2-2*L^2-(L-x)^2);
    end
    if x>=0 && x<a2
        v_M_a2 = (M*x)/(6*L*E*I)*(6*a2*L-3*a2^2-2*L^2-x^2);
    else
        v_M_a2 = -(M*(L-x))/(6*L*E*I)*(6*b2*L-3*b2^2-2*L^2-(L-x)^2);
    end
    v_xy = v_load_xy;
    v_zy = v_load_zy + v_M_a1 + v_M_a2;
    v = sqrt(v_xy^2+v_zy^2);
    v_plot(counter)= v;
    x_plot(counter) = x;
    counter = counter + 1;
    x = x+dx;
end

figure('Renderer', 'painters', 'Position', [500 300 623.62204724 150])
plot(x_plot*1000,-v_plot*1000,'linewidth',1.5)
title('Deflection')
xlabel('Shaft Length [mm]')
ylabel('Deflection [mm]')
xlim([0 L*1000])
```

## G.3 Fatigue - Smith Diagram

```
close all; clear; clc;

d = 20; % mm
N = 250; % N
M_xy = 4.44*1000; % Nmm
M_zy = 25.23*1000; % Nmm
Mb = sqrt(M_xy^2+M_zy^2); % Nmm

y = d/2; % mm
I = pi*d^4/64; % mm^4
A = pi*(d/2)^2;
K_fb = 2.6; % -
K = 0.88; % -
K_red = K*K_fb; % -

sigma_b = Mb*y/I; % N/mm^2
sigma_a = sigma_b;% N/mm^2
sigma_ea = K_red*sigma_a; % N/mm^2
sigma_N = N/A;
sigma_m = sigma_N;
sigma_em = sigma_m;

sigma_f = 470; %N/mm^2
sigma_Ab0 = 280;
sigma_Ab = 225;
b1 = 0.9;
b2 = 0.92;
sigma_Ab0_red = sigma_Ab0*b1*b2;
sigma_Ab_red = sigma_Ab*b1*b2;

%Creating reduced valus corresponding to shaft
% slope_1 = (2*sigma_Ab_red - sigma_Ab0_red)/(sigma_Ab-0);
% slope_2 = (sigma_f-2*sigma_Ab_red)/(sigma_f-sigma_Ab);

slope_1 = (2*sigma_Ab_red - sigma_Ab0_red)/(sigma_Ab_red-0);
slope_2 = (sigma_f-2*sigma_Ab_red)/(sigma_f-sigma_Ab);

x = linspace(0,sigma_f,1000)';
sigma_shaft = zeros(length(x),1);

for i = 1:length(x)
    if x(i) < sigma_Ab
        sigma_shaft(i) = slope_1*x(i)+sigma_Ab0_red;
    else
%         sigma_shaft(i) = slope_2*(x(i)-sigma_Ab0_red)+2*sigma_Ab_red;
        sigma_shaft(i) = slope_2*x(i)+2*sigma_Ab0_red - (462.2 - 401.8);
    end
end
%Creating Smith-diagram for data of 10mm specimen
plot([0,sigma_f],[0,sigma_f],"k")
hold on
grid on
plot([0,sigma_Ab,sigma_f],[sigma_Ab0,2*sigma_Ab,sigma_f],"k")
%ploting reduced diagram ontop
%  plot(x,sigma_shaft);
%plot([sigma_em,sigma_em],[0,sigma_f])
plot([0,sigma_Ab,sigma_f],[sigma_Ab0_red,401.8,sigma_f],"r")

```

```
59 sigma_AN_red = sigma_shaft(find(x<sigma_em, 1, 'last' ));
60 SF = sigma_AN_red/sigma_ea;
```

## G.4 Center of Mass Calculations

```
close all;
clear;
clc;

%% Center of Mass calculation

M_loomo = 19.5; %kg
M_Rig = 80; % kg
M_Components = 50; % kg - Wheels, battery, bearings, motors, gearboxes, etc
M_Person = 80; % Weight of average person

X_1 = 120 + 40 + 40 + 125; % mm - distance to COM of lowest row stored wtih
    loomo
X_2 = 470 + 40 + 40 + 125; % mm - distance to COM of middel row with loomo
X_3 = 870 + 40 + 40 + 125; % mm - distance to COM of heigest store row with
    loomo;

W_loomo = 170+20+20; % mm - distance from COM loomo to center of rig
    horizontally
W_loomo_full = 0;
W_Rig = 0;
W_Comp = 0;
W_Person = 740/2;
d_Wheel = 203;
x_bearing = 33.3;
X_G = d_Wheel/2 + x_bearing;
X_rig = 365 + X_G;
X_Comp = X_G + 100;
X_Person = 1800/2 + X_G; % Person average height 180 cm

% Distance to COM of oneside with looms above ground
X_loomo_COM = (2*M_loomo*(X_1 + X_G) + M_loomo*(X_2+ X_G) + 2*M_loomo*(X_3+
    X_G))/(M_loomo*5);

% Distance to COM of full rig with looms above ground
X_loomo_COM_full = (4*M_loomo*(X_1 + X_G) + 2*M_loomo*(X_2+ X_G) + 4*M_loomo*(
    X_3+ X_G))/(M_loomo*10);


%% Only Two on top at one side

X_COM_twotop_H = (X_3*2*M_loomo + X_rig*M_Rig + X_Comp*M_Components)/(M_loomo
    *2 + M_Rig + M_Components);

X_COM_twotop_W = (W_loomo*2*M_loomo + W_Rig*M_Rig + W_Comp*M_Components)/(
    M_loomo*2 + M_Rig + M_Components);

B_twotop = 560 - 2 * X_COM_twotop_W;

theta_twotop = atan(B_twotop/(2*X_COM_twotop_H))*(180/pi);

%% One side
X_COM_oneside_H = (X_loomo_COM*5*M_loomo + X_rig*M_Rig + X_Comp*M_Components)
    /(M_loomo*5 + M_Rig + M_Components);

X_COM_oneside_W = (W_loomo*5*M_loomo + W_Rig*M_Rig + W_Comp*M_Components)/(
    M_loomo*5 + M_Rig + M_Components);

```

```
B_oneside = 560 - 2 * X_COM_oneside_W;

theta_oneside = atan(B_oneside/(2*X_COM_oneside_H))*(180/pi);


%% Full top 4
X_COM_full4_H = (X_3*4*M_loomo + X_rig*M_Rig + X_Comp*M_Components)/(M_loomo*4
    + M_Rig + M_Components);

X_COM_full4_W = (W_loomo_full*4*M_loomo + W_Rig*M_Rig + W_Comp*M_Components)/(
    M_loomo*4 + M_Rig + M_Components);

B_full4 = 560 - 2 * X_COM_full4_W;

theta_full4 = atan(B_full4/(2*X_COM_full4_H))*(180/pi);


%% Full
X_COM_full_H = (X_loomo_COM*10*M_loomo + X_rig*M_Rig + X_Comp*M_Components)/(
    M_loomo*10 + M_Rig + M_Components);

X_COM_full_W = (W_loomo_full*10*M_loomo + W_Rig*M_Rig + W_Comp*M_Components)/(
    M_loomo*10 + M_Rig + M_Components);

B_full = 560 - 2 * X_COM_full_W;

theta_full = atan(B_full/(2*X_COM_full_H))*(180/pi);

%% One side with person
X_COM_oneside_H_wp = (X_loomo_COM*5*M_loomo + X_rig*M_Rig + X_Comp*
    M_Components + X_Person*M_Person)/(M_loomo*5 + M_Rig + M_Components +
    M_Person);

X_COM_oneside_W_wp = (W_loomo*5*M_loomo + W_Rig*M_Rig + W_Comp*M_Components +
    W_Person*M_Person)/(M_loomo*5 + M_Rig + M_Components + M_Person);

B_oneside_wp = 560 - 2 * X_C;
OM_oneside_W_wp;

theta_oneside_wp = atan(B_oneside_wp/(2*X_COM_oneside_H_wp))*(180/pi);

%% Full with person stepping on one side
X_COM_full_H_wp = (X_loomo_COM*10*M_loomo + X_rig*M_Rig + X_Comp*M_Components
    + X_Person*M_Person)/(M_loomo*10 + M_Rig + M_Components + M_Person);

X_COM_full_W_wp = (W_loomo_full*10*M_loomo + W_Rig*M_Rig + W_Comp*M_Components
    + W_Person*M_Person)/(M_loomo*10 + M_Rig + M_Components + M_Person);

B_full_wp = 560 - 2 * X_COM_full_W_wp;

theta_full_wp = atan(B_full_wp/(2*X_COM_full_H_wp))*(180/pi);
```

## G.5 Kinematics verification

```
close all; clear; clc;

l = 0.7;
h = 0.635;
w = 0.285;
r = 0.203/2;

A = [1, 1, w+h;
    -1, 1, -w-h;
     1, 1, -w-h;
    -1, 1, w+h];

A_inv = 1/4*[1, -1, 1, -1;
             1, 1, 1, 1;
            1/(w+h),-1/(w+h),-1/(w+h),1/(w+h)];


dt = 0.01;
t = 0;
t_end = 10;
x0 = 0;
y0 = 0;
theta0 = 0;
x=x0; x_f =x0;
y=y0; y_f= y0;
theta=theta0;
theta_f=theta0;

% Initializing arrays ---------------------------------- %
B = zeros(length(t:dt:t_end),3);
C = B;
x_dot = zeros(1,length(t:dt:t_end)); t = x_dot;
y_dot = x_dot; theta_dot = x_dot; states=x_dot;
x_dot_f = x_dot; y_dot_f = x_dot; theta_dot_f = x_dot;
x_f = x_dot; y_f = x_dot; theta_f = x_dot;
% ------------------------------------------------------ %


for i = 1:t_end/dt+1
if i <=25
x_dot(i) = 1;
y_dot(i) = 0;
theta_dot(i) = 0;
elseif(i<= 50)
x_dot(i) = 0;
y_dot(i) = 0.5;
theta_dot(i) = 0.5;
else
x_dot(i) = 0;
y_dot(i) = 1;
theta_dot(i) = 0;
end
t(i+1) = t(i)+dt;
B(i,1) = x_dot(i);
B(i,2) = y_dot(i);
B(i,3) = theta_dot(i);

C(i,1) = (-1/r*(A(1,1:3)*[x_dot(i); y_dot(i); theta_dot(i)]))';
```

```
C(i,2) = (-1/r*(A(2,1:3)*[x_dot(i); y_dot(i); theta_dot(i)]))';
C(i,3) = (-1/r*(A(3,1:3)*[x_dot(i); y_dot(i); theta_dot(i)]))';
C(i,4) = (-1/r*(A(4,1:3)*[x_dot(i); y_dot(i); theta_dot(i)]))';

x_dot_f(i) = -r*A_inv(1,1:4)*C(i,1:4)';
y_dot_f(i) = -r*A_inv(2,1:4)*C(i,1:4)';
theta_dot_f(i) = -r*A_inv(3,1:4)*C(i,1:4)';
states(i,1) =x_dot_f(i);
states(i,2) =y_dot_f(i);
states(i,3) =theta_dot_f(i);

x(i+1) = x(i) + x_dot(i)*dt;
y(i+1) = y(i) + y_dot(i)*dt;
theta(i+1) = theta(i) + theta_dot(i)*dt;

x_f(i+1) = x_f(i) + x_dot_f(i)*dt;
y_f(i+1) = y_f(i) + y_dot_f(i)*dt;
theta_f(i+1) = theta_f(i) + theta_dot_f(i)*dt;

end
% Deleting last array number to make all dimensions correspond. 1x1002 -> 1
    x1001
t(end)= [];
x(end) = []; x_f(end) = [];
y(end) = []; y_f(end) = [];
theta(end)=[]; theta_f(end)=[];

error_x = round(mean(x-x_f),5);
error_y = round(mean(y-y_f),5);
error_theta = round(mean(theta-theta_f),5);


figure(1)
plot(t,C)

figure(2)
plot(t,states)
```

# H. Megatrond workspace setup

When the repository has been downloaded, it has to be builded as a katkin_workspace with the catkin_make command.

Then, two files has to be given executable properties:
Make executable: chmod +x

- /home/usr/megatrond_ws/src/megatrond_control/gazebo_control/gazebo_control.py
- /home/usr/megatrond_ws/src/ros_control/controller_manager/scripts/spawner

Changing the file directory:

- replace usr with your username
- replace megatrond_ws

Most likely the map file path has to be changed.

Go to:

- /home/usr/megatrond_ws/src/megatrond_navigation

Change image path to:

- /home/usr/megatrond_ws/src/megatrond_description/world/uia_section_v2

A list of the main launch files are listed:

Start Gazebo model

- roslaunch megatrond_description main.launch

Start Rviz (requires that main.launch is started to get information)

- roslaunch megatrond_navigation rviz_amcl.launch

Starging Leap motion assistive acuation

- roslaunch megatrond_description main.launch
- roslaunch leap_motion command_leap.launch
- rosrun leap_motion leap_read

# I. Installing Arduino and Teensyduino on ARM architecture

To enable the UART communication between the Teensy 3.6 and Jetson Xavier, multiple software are required. These are teensydunio,the Arduino IDE for Linux with ARM architecture and the rosserial package.

First, the Arduino IDE is installed onto the Jetson Xavier. It is going to be installed using a tarball. This is done by downloading the Arduino IDE version of choice, here version 1.8.12 of the 64bit ARM is used. Then change to the directory where the tarball has been downloaded and run the following line in listing (I.1) when in the directory. Where, for this case, FILENAME is arduino-1.8.12-linuxaarch64.tar.xz

```
tar xvf FILENAME
```

Listing I.1: Tarball execution

This will begin to extract the compressed folder. When it is done extracting, change directory to the created Arduino version folder. Whilst inside the folder run the following line and the application will install.

```
sudo ./install.sh
```

Next, teensyduino for linux has to be installed in order to compile the teensy 3.6 with the Arduino IDE. Download the Linux ARM64 file from: (Here, and select Linux Installer (AARCH64/Jetson TX2)). If the installer does not work, try the one from the following link: (installer). Then, run the following lines to execute the exetutable-file. The first line is to set the executable bit if the installer cannot be launched with the second line (Source on changing to executable files)

```
sudo chmod +x ./TeensyduinoInstall.linuxaarch64
sudo ./TeensyduinoInstall.linuxaarch64
```

Then, an installer wizard should pop up. Simply follow its instructions. As the final step the teensy 3.6 has to be allowed to use the usb serial. To achieve this create the file with *.rule*-extension containing the following [92].

```
# UDEV Rules for Teensy boards, http://www.pjrc.com/teensy/
ATTRS{idVendor}=="16c0", ATTRS{idProduct}=="04[789B]?", ENV{
    ID_MM_DEVICE_IGNORE}="1", ENV{ID_MM_PORT_IGNORE}="1"
ATTRS{idVendor}=="16c0", ATTRS{idProduct}=="04[789A]?", ENV{MTP_NO_PROBE}="1"
SUBSYSTEMS=="usb", ATTRS{idVendor}=="16c0", ATTRS{idProduct}=="04[789ABCD]?",
    MODE:="0666"
KERNEL=="ttyACM*", ATTRS{idVendor}=="16c0", ATTRS{idProduct}=="04[789B]?",
    MODE:="0666"
```

Change the terminal directory to where the *.rule*-file was created and execute the following line in order to install the rules into the apprioprirate folder. Finally, as the *.rule*-file mentions, physically disconnect and reconnect the teensy, which now should be fully funcational with the Arduino IDE on the Jetson Xavier.

```
sudo cp 49-teensy.rules /etc/udev/rules.d/49-teensy.rules
```