Politecnico di Torino

Course in Mechatronic Engineering

Master's degree thesis

Design and Control of a 6-DOF mechanical manipulator

Supervisors

Prof. Marcello Chiaberge

Ing. Marco Ambrosio
Ing. Gianluca Dara
Ing. Mauro Martini

Candidate

Alberto Bonansea

S320198

A.y. 2025/2026
October 2025

Abstract

This thesis presents the development of a 6-dof anthropomorphic manipulator attachment for the RoboTO team's fleet of remote-controlled mobile robots, encompassing all its aspects.

After a brief introduction, the project requirements are discussed, including the manipulator's objectives and design constraints, followed by an overview of robotic manipulators and a discussion of alternative design solutions. These introductory chapters are followed by the mechanical design of the manipulator in SolidWorks, entailing the motor choice, selection of stock components to use in the assembly, and design of custom parts, a portion of which were designed with 3-D printing capabilities in mind.

The entire manipulator assembly is divided into two distinct modules: the shoulder, a simple serial robot, and the wrist, which is spherical in nature and rather compact due to the presence of a differential assembly controlling two of the three axes of rotation.

The work proceeds with a discussion of forward, differential, and inverse kinematics for which the Denavit-Hartenberg convention is employed. Following this discussion on kinematics the SolidWorks model, exported to MATLAB via Simulink's multibody link plugin, is simulated and used as the plant in the control loops to evaluate various control schemes which are suitable for remote control.

Having settled on a control scheme, the next step is writing code in C to add to the already preexisting code base of the RoboTO team, allowing for the control of the manipulator via an embedded STM32 microcontroller.

The final step before the conclusion of the work is a stage of physical prototyping to verify the viability of both the mechanical design and control schema.

Table of Contents

Table of figures

Tables

Equations

1: Introduction

Student teams serve multiple purposes within the fabric of a university organization: they function as incubators for fledgeling engineers allowing them to develop real world skill in multiple fields ahead of their peers, they also function as gathering spaces for likeminded individuals to share their passions and interests, and recruitment pools for businesses while at the same time granting individuals the freedom to explore various fields of development before their entrance into the workforce.

The RoboTO team is one of these student teams and is primarily focused on the development of a fleet of robots to participate in competitions around the world, with a focus on the RoboMaster University Series: a series of competitions against teams from other universities from around the world, organized by DJI. This thesis is about the development of a remote-controlled manipulator for the engineer class of robots within the wider fleet of the team to participate in these competitions.

Initially, the thesis will focus on the project requirements, including objectives and constraints, followed by a look into the state of the art for robotic manipulators. The successive steps will be a discussion of the mechanical design and kinematics, followed by a look into the simulation and prototyping of the control systems, followed by the creation of C code for the embedded controller and physical prototyping to refine the control systems' parameters.

SolidWorks, a parametric-based software produced by Dassault Systèmes Company, was employed for the whole design of the manipulator's components and its MODELING add-on was utilized to assess rough torque requirements for the motor selection. MATLAB and Simulink, produced by The MathWorks Inc., were used to simulate the manipulator

and create the control systems for prototyping and subsequently used to generate a part of the code required for the embedded controller. Atom, an open-source code editor, was used to create the rest of the necessary code for the embedded controller.

The final step before the conclusion of the work was the undertaking of a stage of physical prototyping to verify the viability of both the mechanical design and control schema. The fully 3-D printed prototype, which encompasses the entire wrist, was used to ensure the proper fitting of the components.

Keil, a compiler developed by the homonymous Keil company, was used to debug, and compile the code base to ensure the correct functioning of the control schema and, during testing, calibrate the controllers designed in the simulations to properly follow the given references.

2: Project requirements

The objective of the thesis is the development of a robotic manipulator within the constraints of the RoboMaster competition: the competition for the engineer class of robots is point-based, with points assigned upon task completion, based on task difficulty, and, if any, time left.

The competition is divided into 3-minute rounds during which the robot is placed by itself into a 5 m by 5 m arena. The robot always begins the round within a 1 m by 1 m starting zone in the bottom right corner of the arena. Within the arena there are two more notable locations: the resource island, which consists of a low rectangular platform in which three foam cubes "Ores" are flush set, and an exchange station.

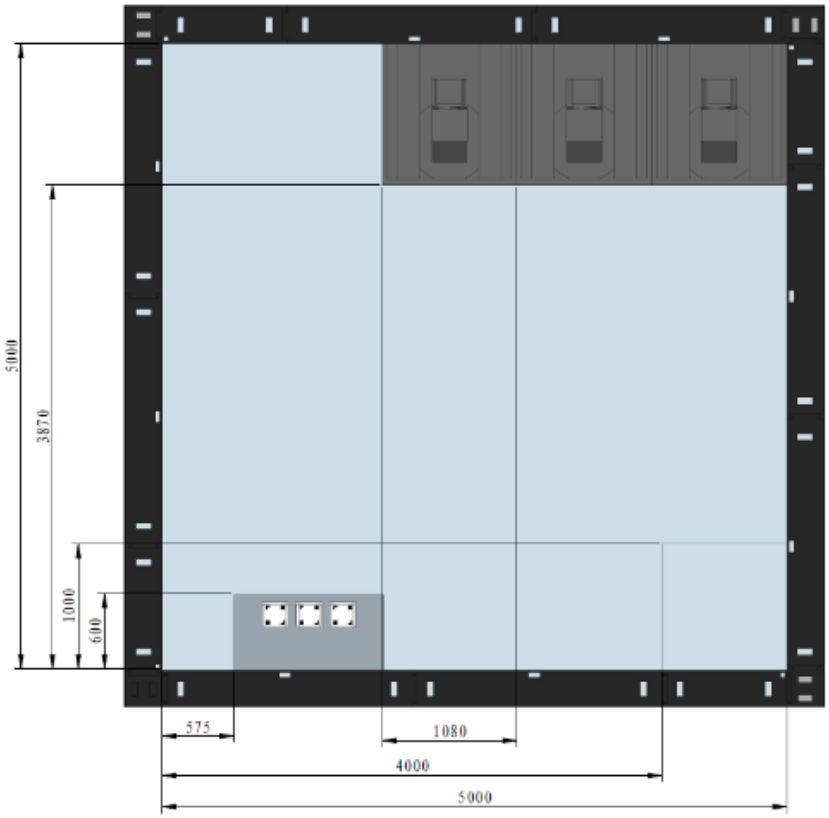

Figure 1 Competition arena

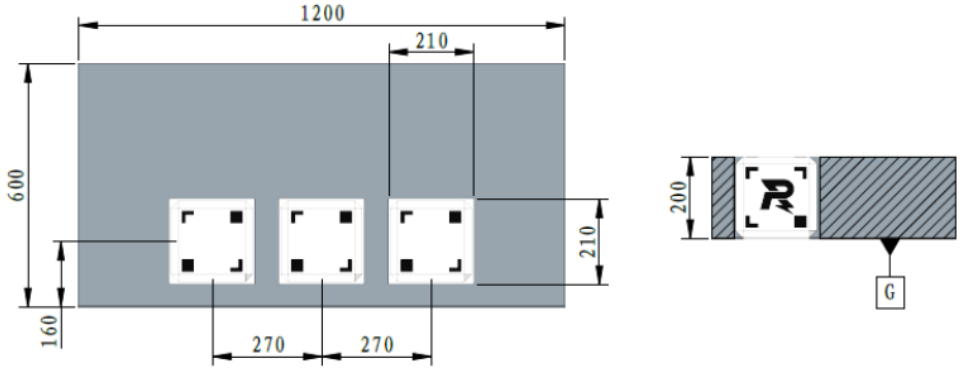

Figure 2 Resource Island

The Ores are EVA (Ethylene-vinyl acetate) cubes with bevelled edges, with nominal dimensions of 200 X 200 X 200 mm (L X W X H), a mass between 550 and 650 g and hardness of 38±5 HC. All Ores share the same screen printing and barcodes, but due to manufacturing variance, the surface roughness may vary. The nominal roughness for a screen print area is 30-40 µm while for a non-screen print area it is 12-16 µm.

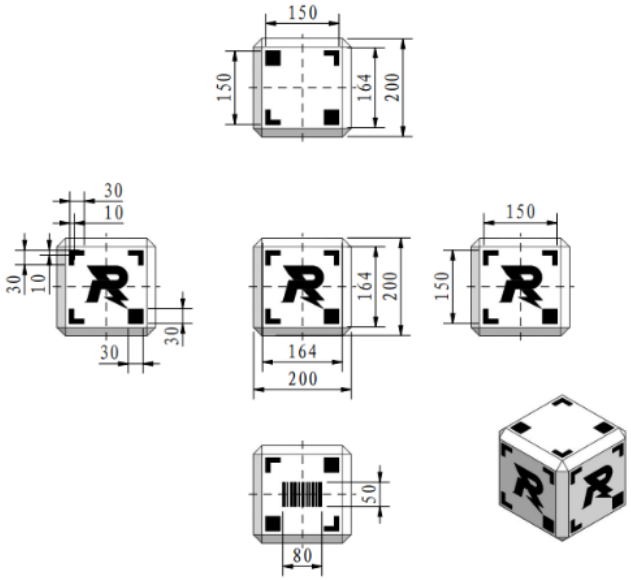

Figure 3 Ore dimensional specifications

The exchange station is comprised of three Ore receptacles, each one equipped with a container where the foam cubes must be placed to score points. These containers are mounted upon arms that, at the start of each round but not during it, will randomize their position and attitude within a given range depending on the task difficulty.

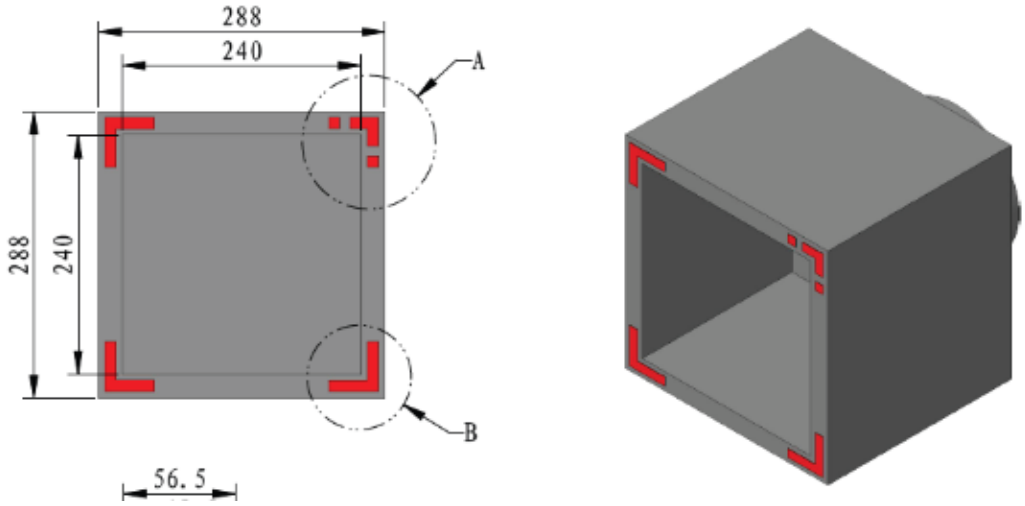

Figure 4 Ore end receptacle

For each Ore Receptacle we can define a right-handed Cartesian reference frame O XYZ with origin O placed on ground level and at the middle point of the Receptacle Base's front. The normal from the base front towards the container can be taken as the negative X-axis for the frame, while the upward direction can be taken as the positive Z-axis. Another frame E X'Y'Z' can be established associated with the container's entry plane: the origin E can be placed at the centre of the entry plane with the X' Y and Z' axes being parallel and consistent with the X, Y, and Z axes.

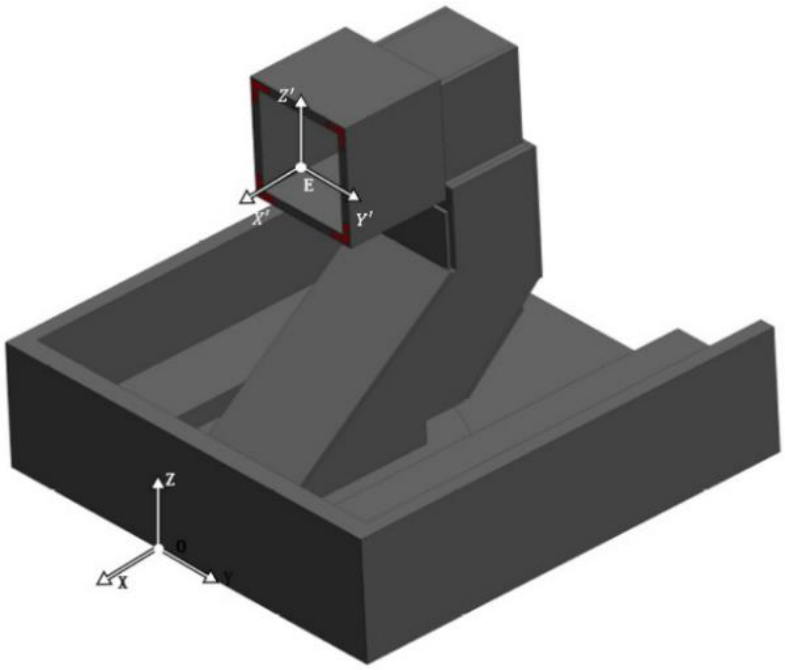

Figure 5 Receptacle position reference frames

The normal of the container's entry plane $\vec{e}$ allows for the creation of a spherical coordinate system ($r\ \theta\ \varphi$) where θ represents the angle between the projection of $\vec{e}$ on the plane X' Y' and X' in the range [-180, 180), φ represents the angle between $\vec{e}$ and the Z'-axis falling in the range (0, 180), and, finally, an additional term α represents the rotation of the container about $\vec{e}$ with counterclockwise rotation being positive.

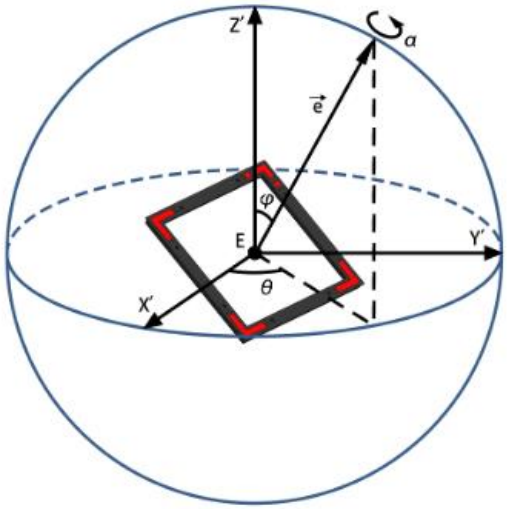

Figure 6 Spherical frame of reference attitude

The pose of the container is defined by the coordinates (x, y, z) of E in the reference frame OXYZ which, combined with the $(\theta, \varphi, \alpha)$ coordinates related to $\vec{e}$, define the pose of the container.

Ore Receptacle	x	y	z	θ	φ	α	Points
A	0	0	830	0	90	0	1
B	0	[-100,100]	[710,910]	0	90	[-45,45]	10
C	0	[-100,100]	[710,910]	[0,90]	90	[-45,45]	100

Table 1 Receptacle pose intervals

As for constraints in the design of the manipulator, there are relatively few:

- The robot must weigh less than 35 kilograms.
- The orthographic projection on the robot's initial configuration must fit into a square of 600 mm while the highest point must not be any higher than 600 mm from the ground.
- During operation, the robot's orthographic projection must fit into a square with side of 1200 mm while its highest point must not be any higher than 1200 mm from the ground.
- The actuators must be either pneumatic or electric.
- The robot is limited to a maximum power supply voltage of 30V and a maximum overall power consumption of 300Wh.

There are more generic constraints that will not be reported here as they concern the design of the wheeled chassis, battery choice, and other components, which were already designed by the team.

3: State of the art

Manipulators and their classification

A robotic manipulator is a structure composed of links which are connected by joints, the joints are actuated by powered components that may be electric, pneumatic, hydraulic, etc. The manipulator ends with an end effector responsible for carrying out tasks.

Links, when considered by themselves, are free to move in 3-D space and, as such, have inherently 6-DOF joints impose restrictions on the relative degree of freedom between the links they connect, typically limiting the relative motion to 1 and, somewhat more rarely, to 2 DOFs. The most common types of joints are the revolute kind, which limits the movement of connected links to a rotation about an axis, and the prismatic kind, which allows for translational motion along a single axis, both thus restrict the relative DOF of connected links down to 1.

Within sufficiently complex manipulators a part of the manipulator can be sectioned off and considered as a discrete entity: the wrist, whose purpose is to increase the dexterity of the manipulator, and often separates the position control from the attitude control for which it is responsible.

Manipulators can be classified by various criteria:

- Structure
    - Serial: The most common type for industrial applications consists of an open-ended chain of links which connect the base to the end-effector
    - Parallel: Rarer, it can consist in part or wholly of closed loop chains of joints and links, increasing complexity for a higher rigidity

- Base mobility: A manipulator is said to be mobile if its base is attached to a mobile robot.
- Link Rigidity: Depending on the application links may either need to be rigid, especially in high load situations, or soft, to obtain a greater degree of compliance. Rigid link manipulators are the older, more well-established type compared to the newer soft link ones, which are a relatively new field of development, often based on the concept of biomimicry and compliance.
- Redundancy: a manipulator is said to be redundant if it has more degrees of freedom than the task requires. This may be intrinsic (for example 7 DOFs robot operating in 3D space) or task-related (a 4 DOFs robot concerned with only the position in 3D space).
- Wrist type: the wrist is typically composed of a series of 3 revolute joints, if the axis of rotation of these 3 joints meet at the same point the wrist is said spherical and greatly reduces the inverse kinematics complexity as it allows for the clean separation of the position and attitude, if the wrist does not meet this requirement, it is said non-spherical.

Alternate solutions

Due to the nature of the competition, which encourages the sharing of methodology and developments between teams, a significant number of past developments were open-sourced.[1] This allowed for a comparison between different approaches before beginning the development of the manipulator. Several alternatives, observed from other teams' projects in the past years, were considered:

- Simple Gantry systems: almost universal in earlier competitions, most teams started from this kind of manipulator due to their ease of construction; they offer some clear advantages, like their considerable stiffness and low requirement for development and

control. This kind of manipulator can be modeled as a chain of prismatic joints with at most a single revolute joint near the end effector to allow for the loading and unloading of the ore. This simplicity also comes with some clear downsides, significantly limiting its mobility, reach, and dexterity and some not-so-apparent ones, such as its carrying capacity. In fact, these systems require some sort of mechanized hopper system to be able to easily carry more than one ore at a time, essentially trading manipulator complexity for ancillary system complexity.

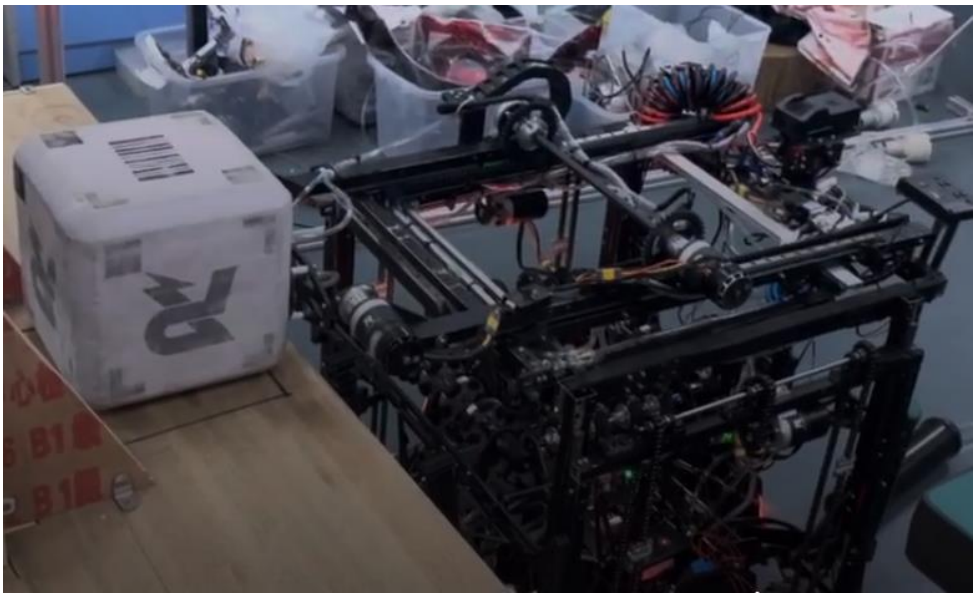

Figure 7 Simple gantry - Liaoning University of Science and Technology

- Advanced Gantry systems: Some of the downsides of simple gantries were partially corrected over the years by various teams through the implementation of a hybrid approach. By essentially stacking on top of the gantry system a revolute jointed robot arm/wrist with low DOF the whole manipulator obtained an overall increase in capability (reach attitude pose adjustment etc.) at the cost of a slightly decreased rigidity and increased complexity, however this approach still doesn't solve some of the intrinsic problems of

the gantry system such a s the need for a mechanized hopper or its inherent bulkiness.

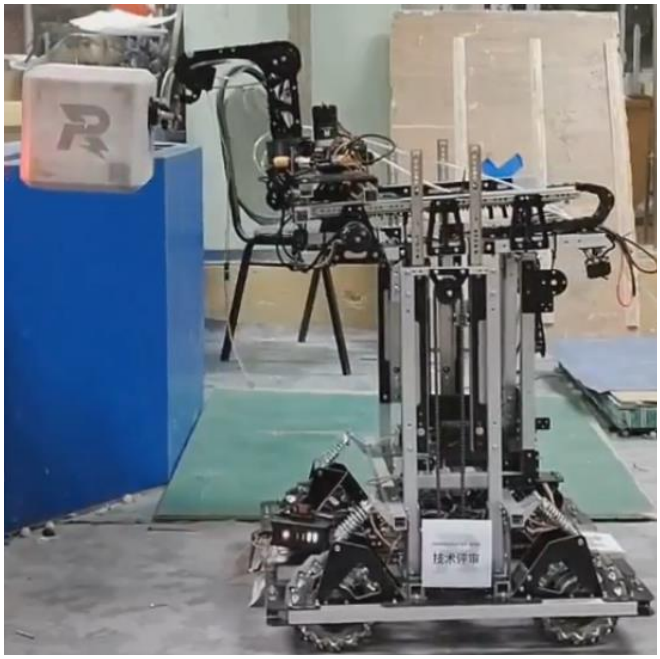

Figure 8 Gantry with a low R robotic arm - Beijing University of Petroleum

- Scara arm: an alternative manipulator scheme that was mostly bypassed in the development of manipulators for the competition. It consists of a singular prismatic joint followed by a planar arm composed of revolute joints, which have their axes of revolution parallel to the axis of translation of the first joint. This kind of robot maintains a high degree of stiffness while being more compact and having a much greater reach and dexterity than the gantry manipulators; however, it is still limited in the height adjustment depending on the singular prismatic joint's limits.

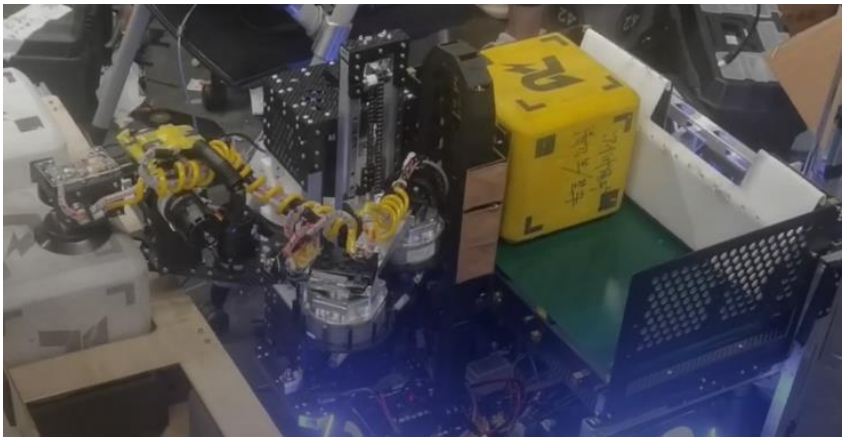

Figure 9 Scara arm - Southern University of Science and Technology

- Anthropomorphic manipulator: Also known as a humanoid arm manipulator, it consists of a series of revolute joints, which result in a remarkably high degree of dexterity and manipulability at the cost of some stiffness. This kind of manipulator has become mainstream for the competition as the challenges become harder and harder to complete while utilizing more primitive designs. It has become favored over the Scara arm due to its greater reach and dexterity and the greater number of resources online about their construction and control, especially at the amateur level.

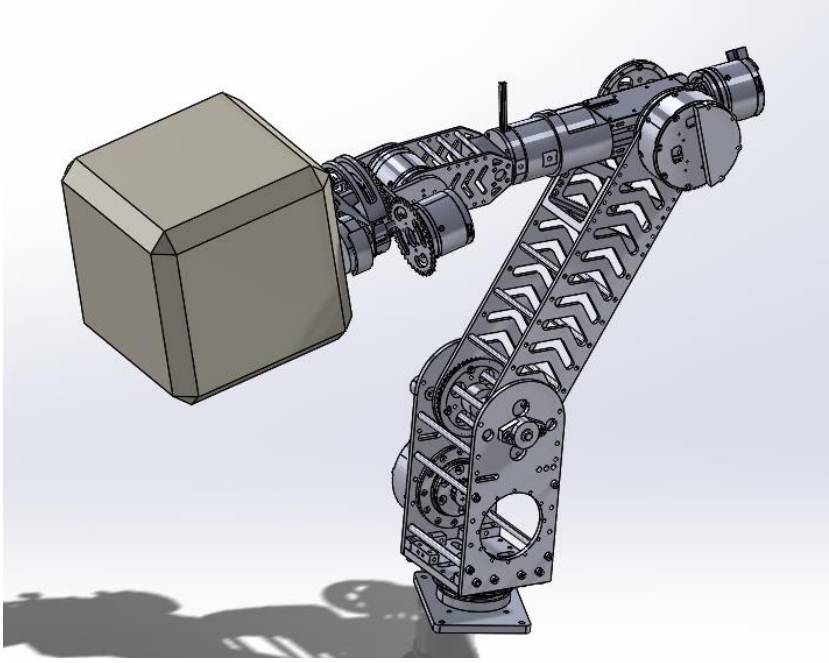

Figure 10 Anthropomorphic arm - Zhejiang University

Another crucial detail is the design of the wrist; two approaches were present:

- Serial chain in which the wrist was composed of a series of links interconnected by revolute joints.
- Differential in which two of the 3 DOF of the wrist are operated by a differential controlled by two separate motors.

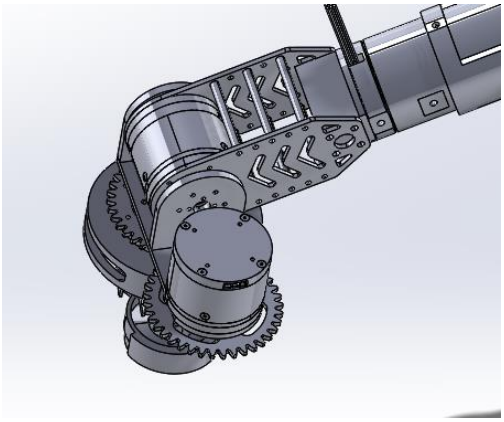

Figure 11 Serial wrist

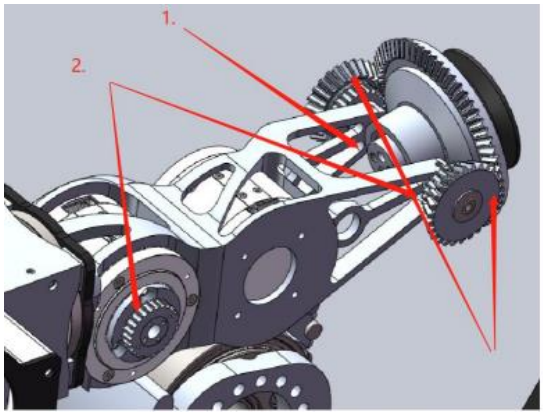

Figure 12 Differential wrist

As for the end effector, while not being truly forced, the only viable option is the use of a suction cup, as other types of end effectors, like clamps, are too unwieldy to use, especially due to the flush set start position of the cubes. Various teams have experimented with various setups over the years; however, the consensus is that a single large suction cup (≈80 mm in diameter) is both sufficiently reliable, rarely losing grip, and low complexity, especially in comparison with setups with multiple smaller cups where adhesion and vacuum loss were more common.

4: Mechanical design

The manipulator, after consideration and an analysis of other teams' past designs, was chosen to be a humanoid arm with a differential wrist and a suction cup end effector was designed in SolidWorks.

The mechanical design process was guided by various needs: the first was to keep costs low by utilizing standard parts, and if custom-machined parts were required, utilizing either plates or extruded profiles modified through milling operations; milling custom parts from blocks had to be avoided to keep costs suitably low. The second principle was to maintain a low complexity both in part and ease of assembly, as the manipulator would most certainly need to be partially or wholly disassembled during transit. Following these two principles means that the design would, in theory, be easily maintainable and repairable as most components can be sourced through online specialist stores and easily modifiable in case the theoretical design of this thesis was required to be updated due to changes in the requirements or constraints.

The design can be split into two pieces: the shoulder, responsible for the larger, rougher pose adjustments, and the wrist, responsible for adjusting the attitude of the end effector.

Shoulder

The shoulder is composed of four links and three revolute joints. It will be discussed in a forward motion starting from the base and moving towards the wrist.

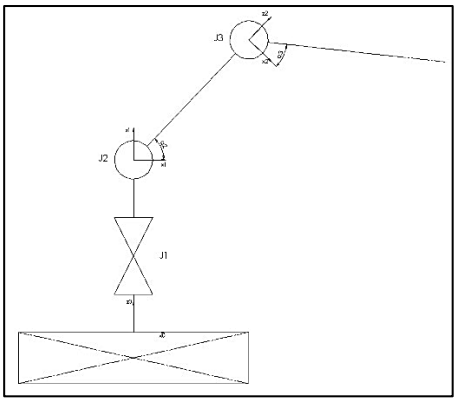

Figure 13 Shoulder simplified model

Base link

The Base link is the part responsible for connecting the manipulator to the wheeled chassis of the robot.

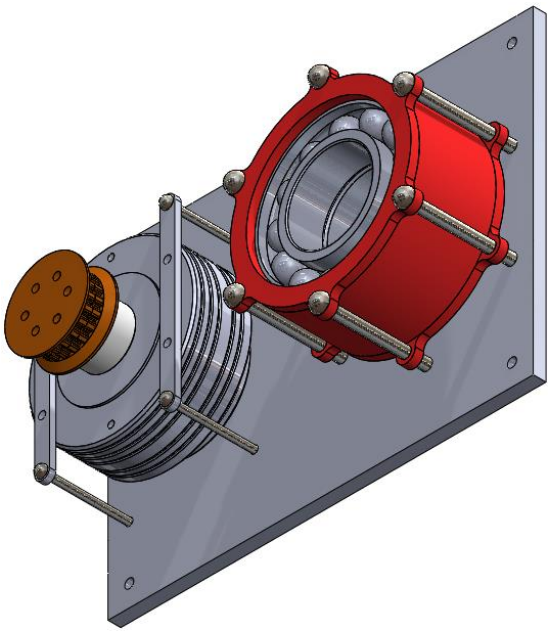

Figure 14 Base link

Its main part consists of a rectangular aluminum plate, which has had several-sized holes bored through it. This plate acts both as the mounting

point of the whole manipulator to the chassis and as the attachment point for Link 1 and its actuator.

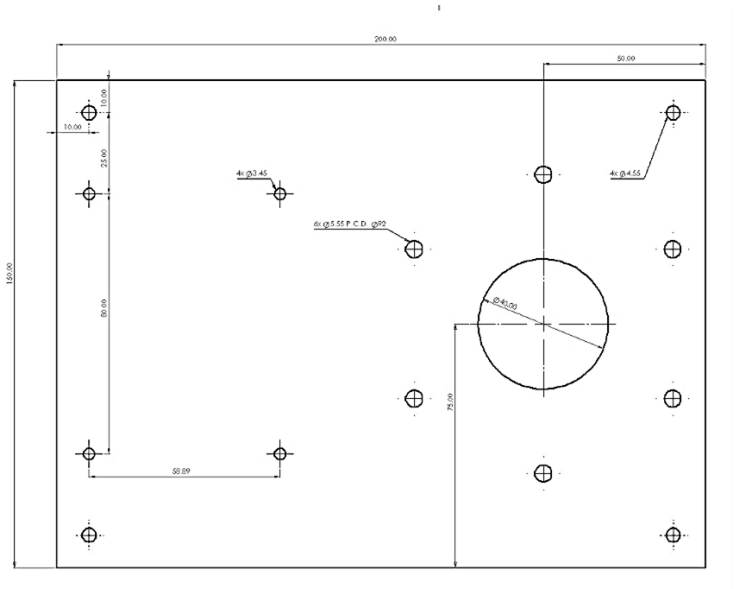

Figure 15 Base link - Plate schematic

As a design decision to minimize accidents, cabling, electrical and pneumatic, must remain contained within the manipulator's structure as much as possible. The loose cabling necessary for a robotic manipulator, if left unprotected, would be at risk of pinching by the manipulator itself and getting caught in protrusions.

The plate has a large circular hole 40 mm in diameter, which is necessary for the passage of the internal cabling, while the surrounding six equally spaced M6 clearance holes are necessary for the mounting of Link 1.

The motor, due to the need for an internal passage for the cabling, cannot be mounted directly to the joint and must be offset and connected to the joint via a pulley system; this constraint however while increasing the complexity also allows to amplify the motor torque somewhat easily via changing the pulley ratio if the base torque provided by the chosen motor is deemed insufficient.

The holes in the corners of the plate serve as mounting holes to connect the plate to the chassis, while the four M3 clearance holes spaced in a

rectangular fashion serve to mount the motor through the proper affixing plates.

Figure 16 Base link - motor affixing plates

Joint 1 is one of the most critical points of the whole manipulator: not only does the whole arm rotate around it, but it also weighs down on it, thus generating not only radial but also axial loads. To withstand these combined loads a singular bearing is not sufficient; thus, two single-crown angular contact bearings were employed. These kinds of bearings are built to withstand not only radial loads but also axial loads in a single direction when placed in a back-to-back "O" configuration, this arrangement not only allows the two bearing to withstand axial loads in both directions but also presents the greatest stiffness within the possible layout of the bearings allowing the whole assembly to withstand greater combined forces.

Again, due to the need to pass cabling through the bearings, they required a significant inner bore and thus the chosen bearing, with an inner bore of 40 mm, are oversized and overbuilt for the application, with even a single bearing having a static load safety rating of well over a hundred.

Angular contact bearings specifications [2]	
Producer and RS product code	RS PRO 291-632
Inner diameter	40 mm
Outer diameter	80 mm
Raceway thickness	18 mm
Static load rating	16.236 kN
Dynamic load	35.363 kN
Mass	0.355 kg

Table 2 Joint 1 Bearings specifications

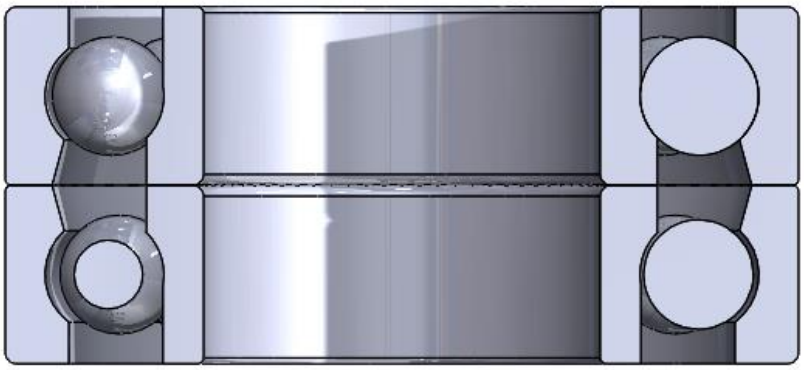

Figure 17 Back-to-back angular contact bearing setup

To align the bearings, some simple outer and inner sleeves are required; these sleeves serve as alignment helpers and not as structural support and therefore can be created through 3-D printing.

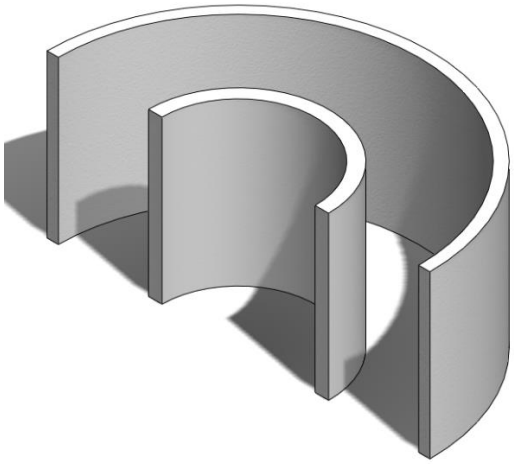

Figure 18 Bearing inner and outer mounting sleeves sections

Angular contact bearings especially when in pairs need to be preloaded to take up any slack between the ball bearings and the raceways, preloading can be achieved by clamping the bearings together and this is exactly what the flanges are designed to do; the outer sets of flanges serve a dual purpose: they clamp the outer raceways together and fix them to the baseplate they do this through the 6 smaller holes through which bolts can be passed through to be tightened via nuts.

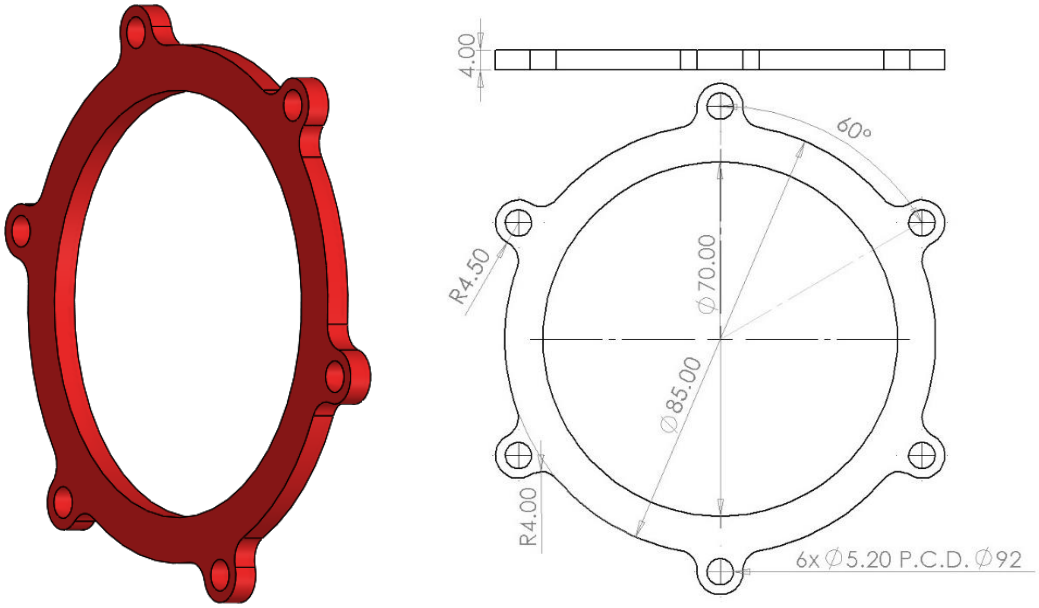

Figure 19 Outer flange Render and schematic

The motor is mounted such that its rotor points upwards and is fixed to the plate through two smaller plates, clamping it on two sides and providing holes for four M3 screws to firmly affix the stator of the motor to the plate. A pulley is connected to the rotor and properly positioned at the right height through a spacer. The pulley itself is left unsupported on one side as the motor's bearings should be sufficient in withstanding the radial forces due to the pulley belt without the need for additional support.

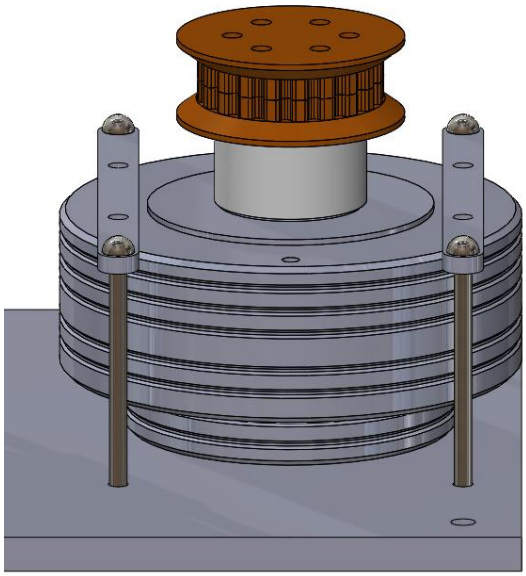

Figure 20 Base-Link motor mounting

Link 1

Link 1, as the name implies, is the first mobile link of the manipulator; its U-shaped form is composed of three plates connected through angular brackets.

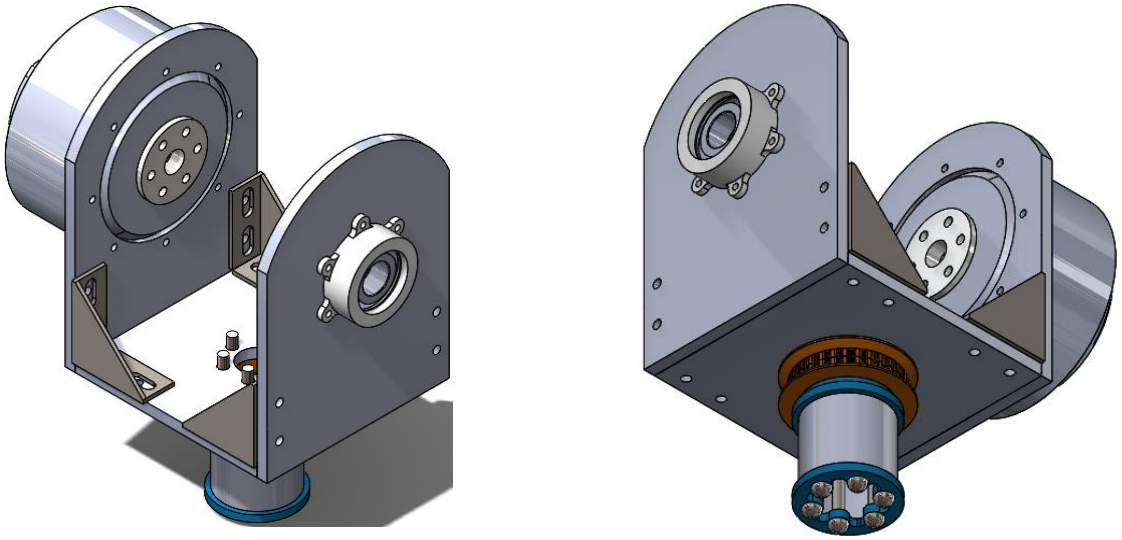

Figure 21 Link 1

The bottom plate similarly to the baseplate of the Base link has a central hole to allow the passage of cabling, surrounded by six equally spaced holes to allow for the passage of the bolts needed to clamp the internal raceways of the bearing through the use of the inner flanges.

An additional eight holes are required for the mounting of the angular brackets [3] to the plate; sized to allow the passage of M5 bolts, they have been placed such that the brackets will offer minimal interference to the movement of Joint 2, allowing for nearly a 150-degree field of motion.

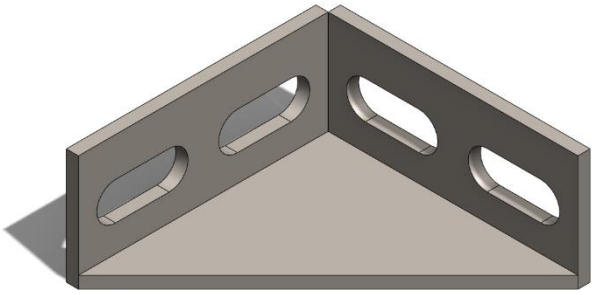

Figure 22 Angular Bracket

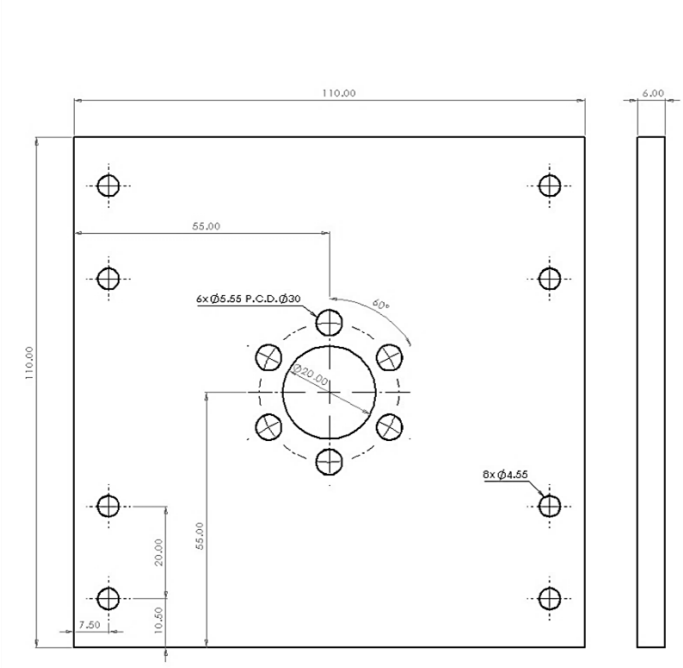

Figure 23 Bottom Plate

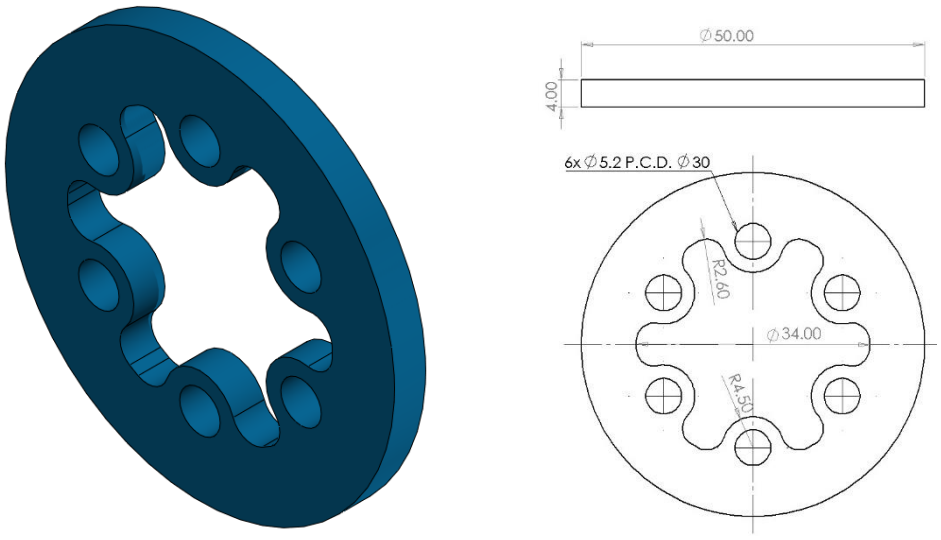

Figure 24 Inner Flange

The two side plates share a basic design being rectangular plates whose top has been rounded, mostly for aesthetic reasons but also to lessen the risk of "scissor-like" effect that a straight edge could have wrought in combination with the movement of Link 2, and drilled to place the mounting holes for the angular brackets, where they differ is on what attaches to them and thus the mounting holes required.

One of the two plates will host the motor actuating Joint 2, therefore requiring the proper openings: a large aperture is necessary to allow the motor's chassis to be countersunk into the plate while the 8 holes surrounding this primary opening are where the M3 bolts, used to fasten the motor to the plate, will be set.

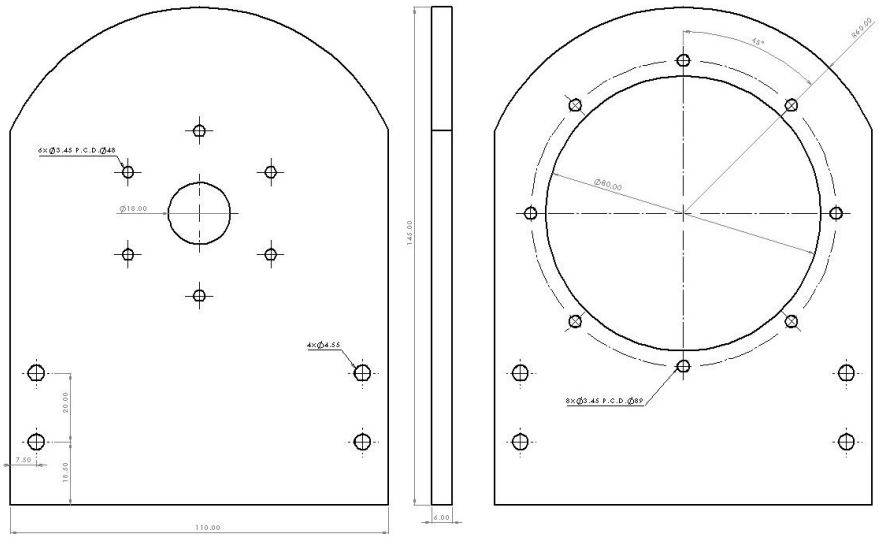

Figure 25 Link 1 - Side plate schematics

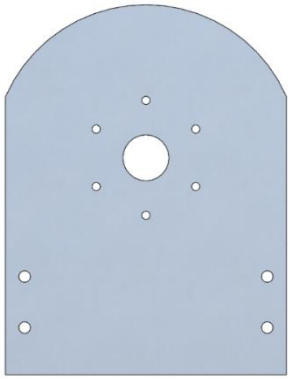

Figure 26 Link 1 - Bearing Side plate

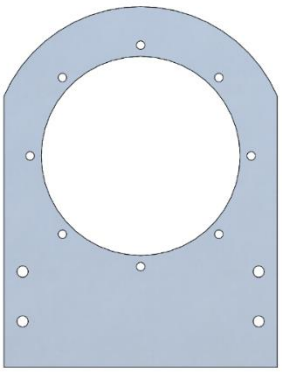

Figure 27 Link 1 - Motor Side plate

The other plate's task is to hold a deep groove ball bearing, it does so through a 3-D printed bearing holder which is attached to the plate through bolts passing through the 6 M3 clearance holes, the central hole is designed to allow the passage of a shaft through the plate and into the bearing allowing it to sustain a part of the radial load so as not to rely entirely on the motor's bearings to sustain the rest of the manipulator.

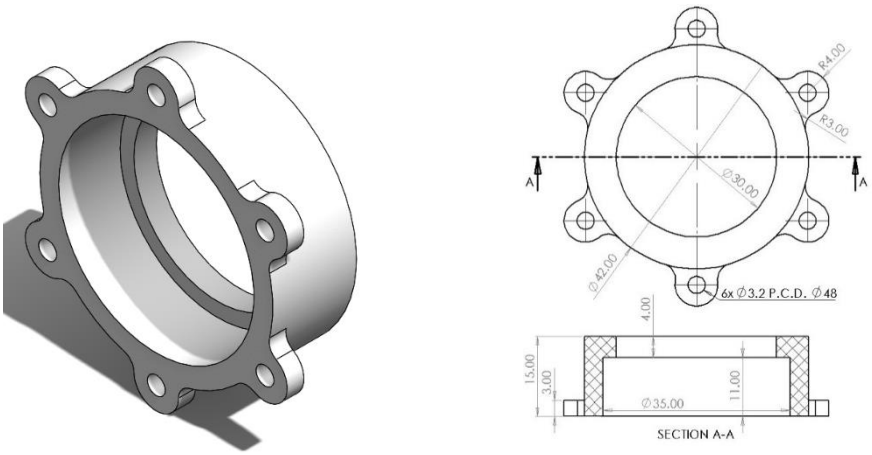

Figure 28 Bearing holder

The bearing chosen is a deep groove bearing, one of the most common types of bearing, and as a result much cheaper than some of the more specialized kind of bearings, these bearings where chosen for their reliability, availability and low cost and to keep assembly complexity and part numbers low this bearing and its associated bearing holder was used for all similar setups within the manipulator (Joint 2 and Joint 3).

Deep groove bearings specifications [4]	
Producer and model number	RS PRO 234-6895
Inner diameter	16 mm
Outer diameter	35 mm
Bearing thickness	11 mm
Static load rating	3.72 kN
Dynamic load rating	7.65 kN
Mass	0.046 kg

Table 3 Joint 2-3 Bearing specifications

Link 2

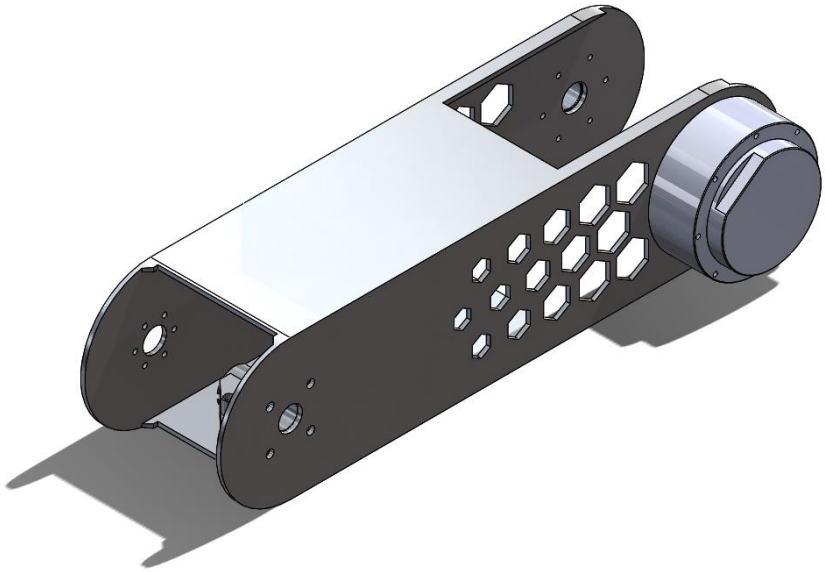

Figure 29 Link 2

Link 2 is composed of only three parts, excluding the mounting hardware such as nuts and bolts: a motor actuating Joint 3, a shaft holder [5] which is a stock component whose purpose is to clamp down on a shaft and prevent it from moving relative to itself and the frame on which the other two are attached, which is a custom component.

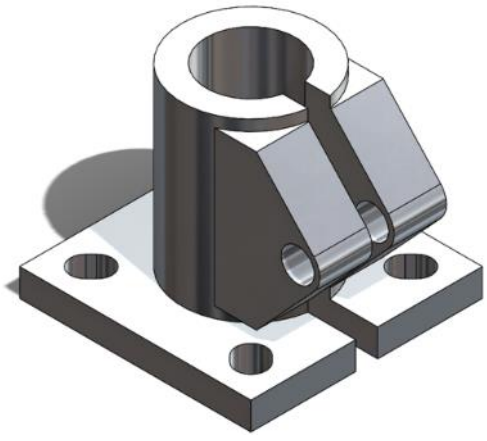

Figure 30 Shaft holder

Link 2's frame is obtained from a piece of square aluminum tubing, specifically, the tubing envisioned for this link is a 100X100 mm tubing with a wall thickness of 4 mm. The choice to use this kind of tubing was

favored over other approaches, such as using multiple plates separated by spacers or the creation of a solid part via milling, by a desire to both keep part numbers low and lower material costs. This kind of tube is widely available in stores at multiple lengths and thus would minimize material wastage and cost. Another advantage this approach presents is the fact that the RoboTO team is already in contact with a manufacturer specialized in milling operations on extruded aluminum parts, which is already their supplier for these kinds of requests, as the team has already implemented similar parts in the robot chassis and other models in their fleet.

Figure 31 Link 2 Frame

The frame could be ideally obtained from a piece of tubing as short as 400 mm and requires some milling operations on all sides; however, a part of these operations, being mostly cosmetic, could be skipped if time or cost were an issue.

The features of the frame will be discussed following their probable order of creation.

At the start the sides are still all the same as we are starting from a plain piece of square tubing; we can start from one arbitrarily chosen side, which from this point on will be called the motor side for reasons that will be clear in short order, the two feature exclusive to this side can be drilled: one of

them is composed of a central hole, sized to allow the passage of a 16 mm shaft surrounded by four M5 clearance holes in a square pattern with a side of 29 mm, this feature is symmetric with respect to the midplane of the side and the central hole has its center placed 50 mm from the tube end, the four holes are the required mounting points for the shaft clamp which will be mounted inside the frame; the other feature on this side is the motor mount placed near the top of the workpiece and similarly to the motor side plate of Link 1is composed of a central hole to inset the motor's rotor casing surrounded by the clearance holes required to firmly attach the motor to the frame.

Figure 32 Link 2 Frame - Motor side

The opposite side to the motor's shall be referred to as the bearing side, and like the motor side, it has two features, one at the top and one at the bottom. The bottom one consists of six holes equally spaced along a circumference of diameter 28 mm these holes are required to mount the frame to the motor actuating Joint 2 the inscribing circumference needs to be concentric to the shaft hole of the bottom feature on the motor side to allow the smooth operation of Joint 2, if these two features are not properly aligned they could lead to crooked or even impossible connections between links and thus are of critical importance. Near the top of the frame on the bearing side the same hole setup that was present on the bearing side plate of Link 1 is present, similarly to the previous feature, this one also needs to be concentric and thus properly aligned with the motor mount on the other side of the part.

Figure 33 Link 2 Frame - Bearing side

There are additional necessary operations not affecting a single side but multiple at a time, such as the rounding of the bottom going from the motor side through to the bearing side, necessary to ensure the correct function of Joint 2 without interference between parts. This feature is concentric to the shaft hole.

The last truly necessary operation is the creation of the notch at the top going from the front through to the back of the frame; this notch, whose purpose is to allow the slotting of Link 3 into Link2, consists of a rectangular cutout centered on the midplane of front side the notch needs to be about 90 mm wide to allow a large amount of clearance between the frames of Link 2 and Link 3 while maintaining the corners of the frame; this fact is crucial as these corners maintain the stiffness of the upper part of the frame despite the presence of the cutout. The height of the cutout limits the travelable angle of Joint 3, with a height of 150 mm with respect to the top edge, allowing around 260 degrees of movement.

Figure 34 Link 2 Frame - Bottom round

Figure 35 Link 2 Frame - Side cut

An additional and debatably cosmetic operation can be done to round out the sharp contours that the bottom round leaves on the front and back; however, this operation may not even require machining and may be carried out by hand with a file to reduce costs.

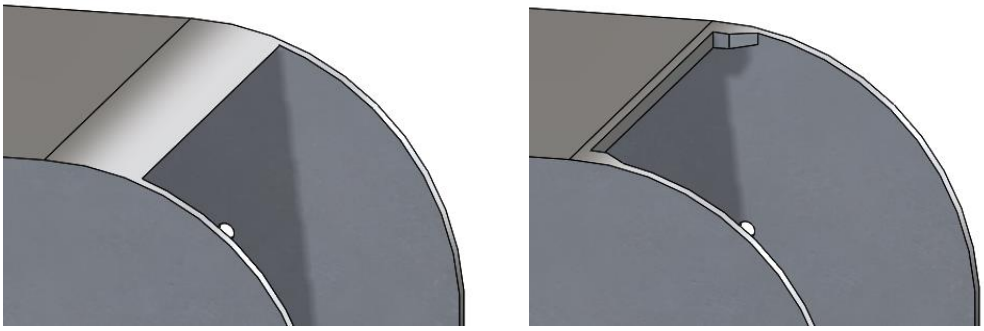

Figure 36 Unfiled and filed Bottom round

As for wholly cosmetic alterations, the fading hexagon pattern on the sides does not serve any purpose other than aesthetics; however, if the RoboTO team chooses to forego its inclusion, a simple hole on the motor side should be drilled to allow for the passage of the wiring for the motor of Joint 3. Another arguably avoidable operation is the top round, concentric with the motor mount, which, while helpful in preventing pinching and slightly reducing the weight, is not strictly necessary.

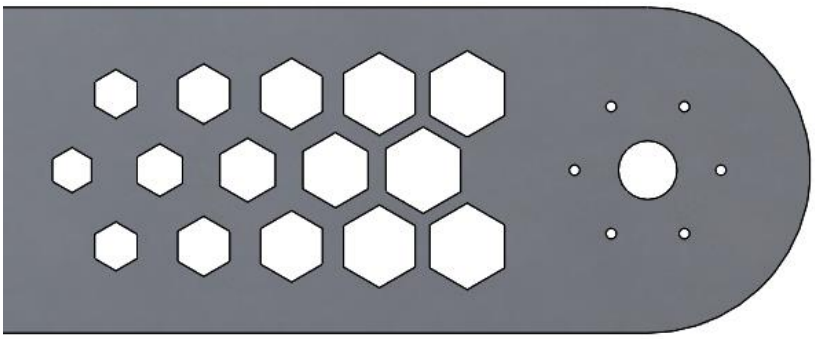

Figure 37 Link 2 Frame - Decorative Hexagonal pattern and Top round

These operations do not require the precision needed for the previous ones as these features are either built with large clearances in mind or serve an aesthetic purpose, meaning high dimensional and positional precision are not truly needed.

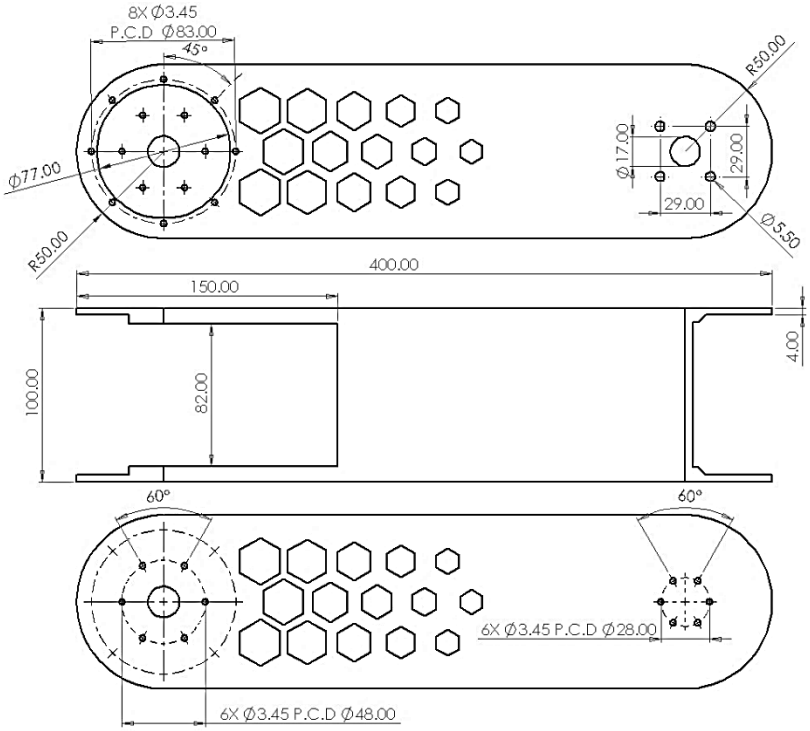

Figure 38 Link 2 - Frame schematic

Link 3

Link 3 is similar in shape to Link 2, being composed of the same three base components: a motor, a shaft holder, and a frame plus an additional shaft holder. The shaft holders are the same model as the one used in Link 2.

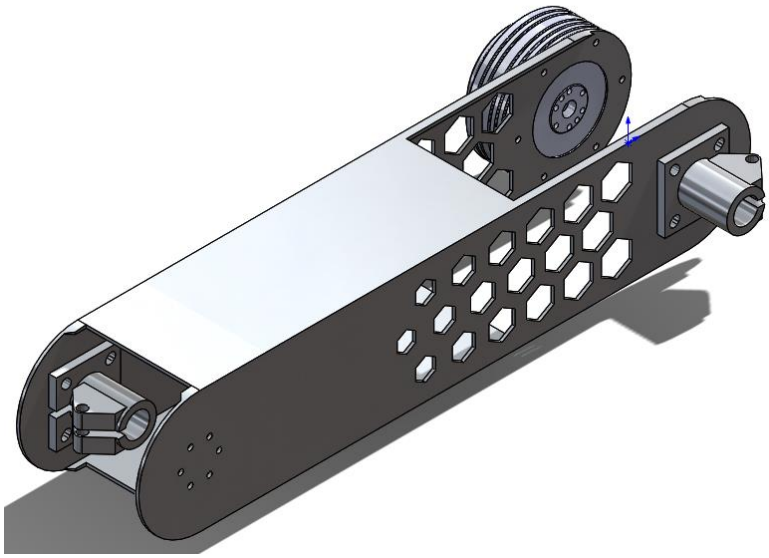

Figure 39 Link 3

The frame, for starters, is obtained through a square piece of tubing of size 80X80 mm and wall thickness of 2 mm, with the whole frame having a total length of 400 mm.

On the bottom the shaft and rotor mounts and the bottom round are functionally identical to the ones of Link 2 where things do differ is at the top: the motor mount although sized for a different motor is spiritually the same where things differ is on the opposite side where, due to space constraints inside Link 4, an additional shaft holder needs to be positioned where the bearing was placed in Link 2 and thus the frame needs to be modified to accommodate this difference.

The notch for Link 4 is also different in width, down to 70 mm, while maintaining the same height, which allows a theoretical 360 degrees of rotation to Joint 4 when not grasping the ore.

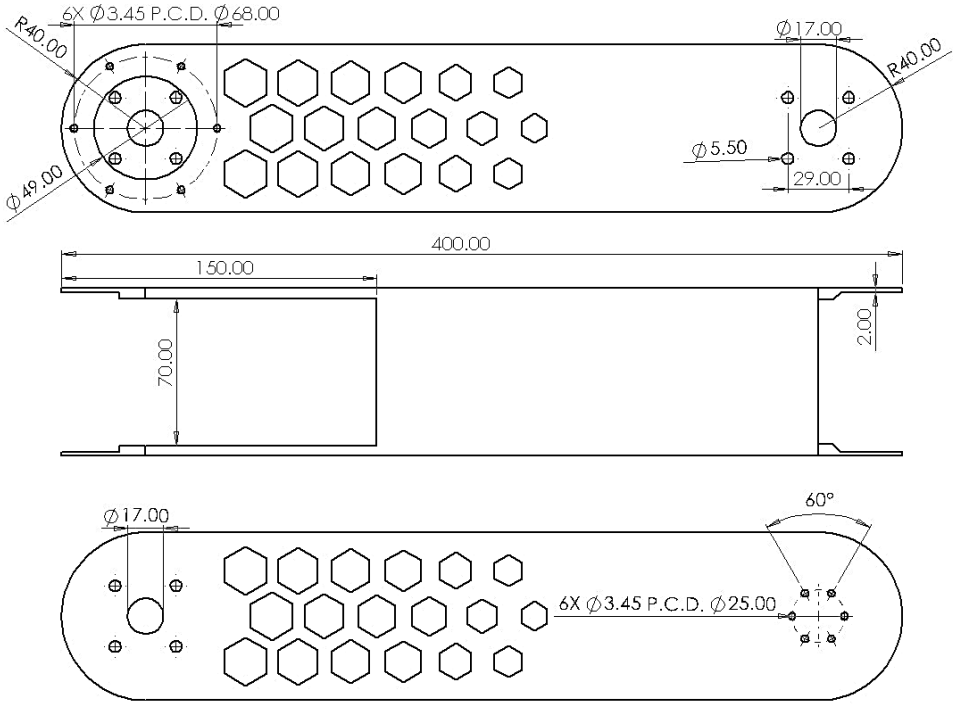

Figure 40 Link 3 - Frame schematic

Joint 1

Joint 1 is not actuated directly through a motor but instead through a 3-D printed pulley (here shown in orange with a split in the middle), this pulley has a central hole for wiring and pass through holes for the bolts, it requires an additional part which can be printed separately or as a whole with the pulley: a spacer to regulate the distance between the first inner flange and the bottom plate of Link 1- Being 3-D printed it allows much greater flexibility for modifications than using standard sized spacers and pulleys.

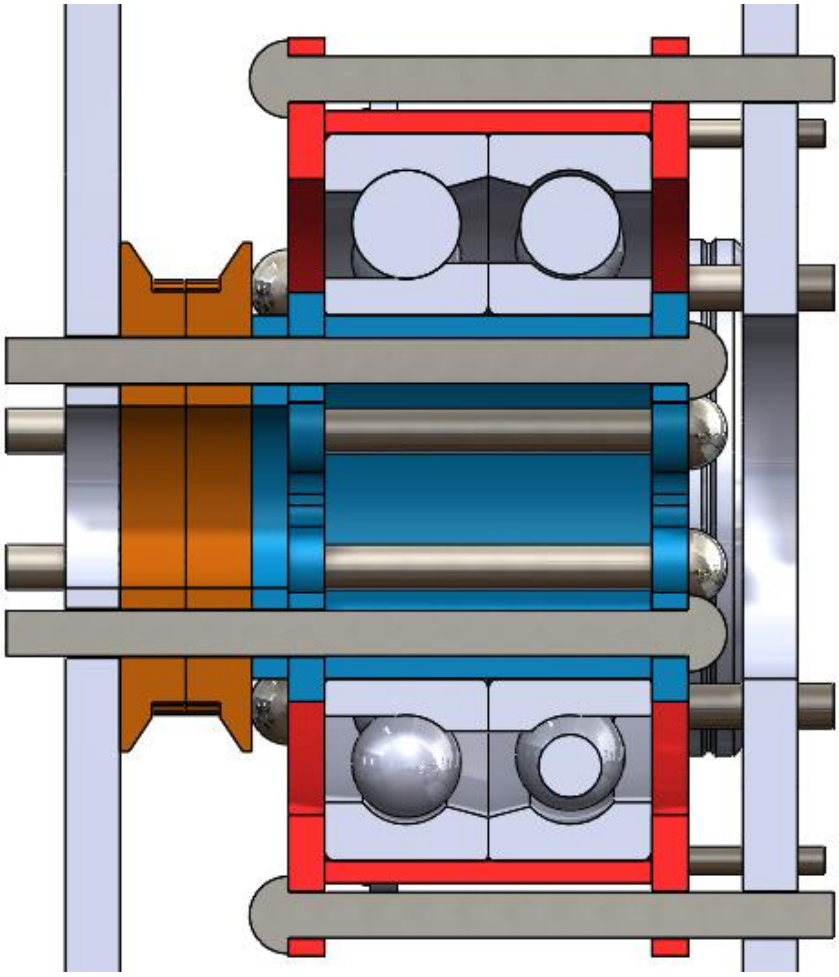

Figure 41 Joint 1 section

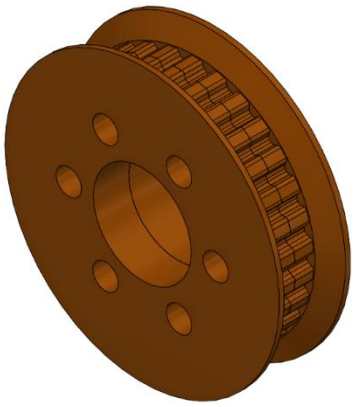

Figure 42 Joint 1 - Link 1 pulley - 30 teeth

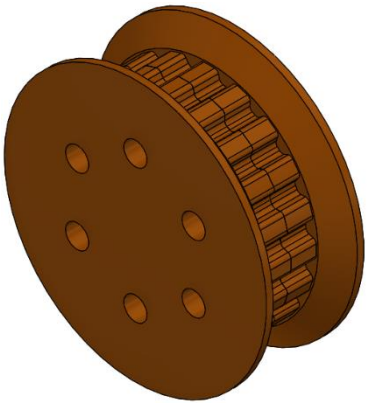

Figure 43 Joint 1 - Base Link pulley - 20 teeth

Cubemars is a company that produces highly integrated motors for robotics in compact sizes and at reasonable prices. For these reasons and more, these motors were already widely employed by the RoboTO team and, as such, were among the first choices when deciding on which motors to use. To actuate Joint 1, an AK60-6 brushless motor produced by Cubemars was chosen. This motor produces a relatively high torque, which, amplified by the pulleys' 1.5:1 ratio, is more than sufficient for rotating the robot arm. This motor was chosen over other models by Cubemars and its competitors since the team has access to a surplus of these motors.

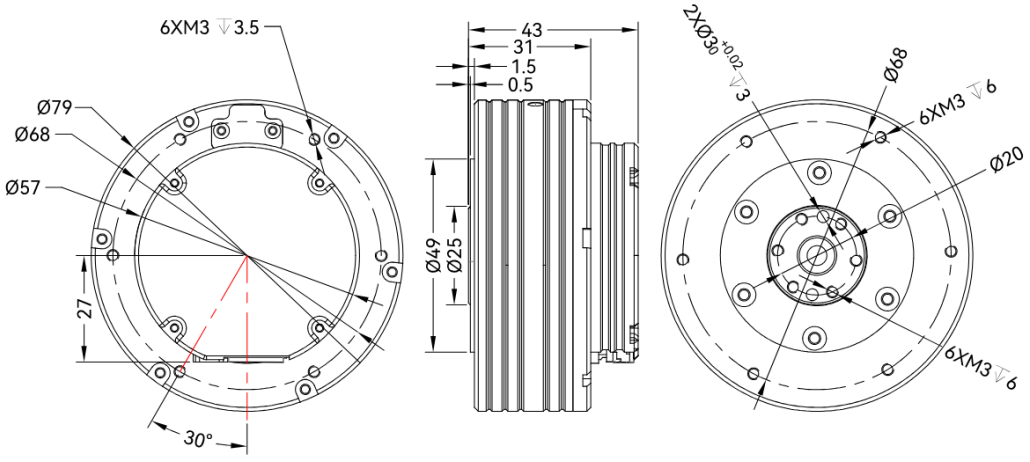

Figure 44 AK60-6 schematic

AK 60-6 specifications [6]	
Internal Reduction ratio	6:1
Rated voltage	24/48 V
Rated torque	3 Nm
Rated speed	233/490 rpm
Rated power	60/125 W
Peak Torque	9 Nm
Weight	380 g
Size (diameter x length)	79 mm x 43 mm

Table 4 Joint 1-4 motor specifications

Joints 2-3

The structure of these joints is much simpler than the previous, consisting of relatively fewer parts, and is almost identical between the two.

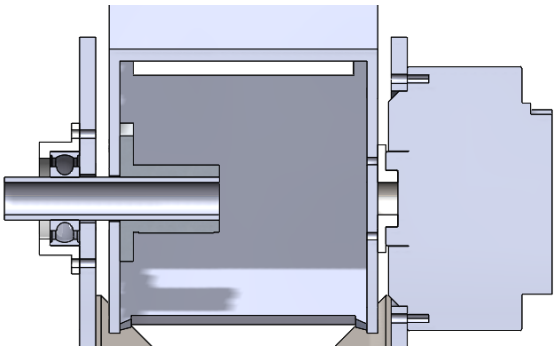

Figure 45 Joint 2 section

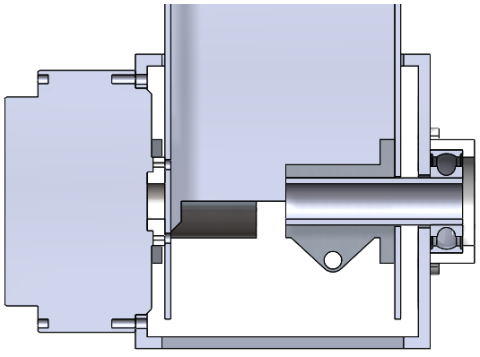

Figure 46 Joint 3 section

To keep costs down, the shaft was envisioned as a cut-down piece of 16 mm aluminum tubing, which during assembly would be inserted from the outside, passing through the bearing and shaft holder, after which the latter would be tightened to clamp down on the shaft. On the other side, the frame

of the succeeding Link connects to the rotor through six M3 screws, to ensure the proper positioning and spacing, a 3-D printed custom spacer is inserted. This spacer differs between the various joints only slightly due to dimensional differences between the rotors while maintaining its rough shape and purpose.

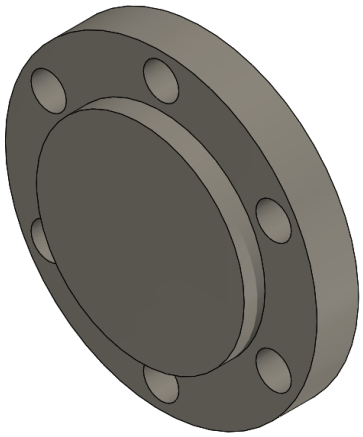

Figure 47 Rotor spacer

For Joint 2, an AK80-64 brushless motor by Cubemars was planned. This motor is oversized for the application, reaching a peak torque of 120 Nm; however, the next smallest category of motors, be it from Cubemars or other companies, reached less than half of its peak torque with only a small price and weight difference. Furthermore, this category of smaller motors, while certainly not undersized, offered a smaller safety margin, which the RoboTO team was not comfortable with when compared to their bigger alternative, and as such, the choice was clear.

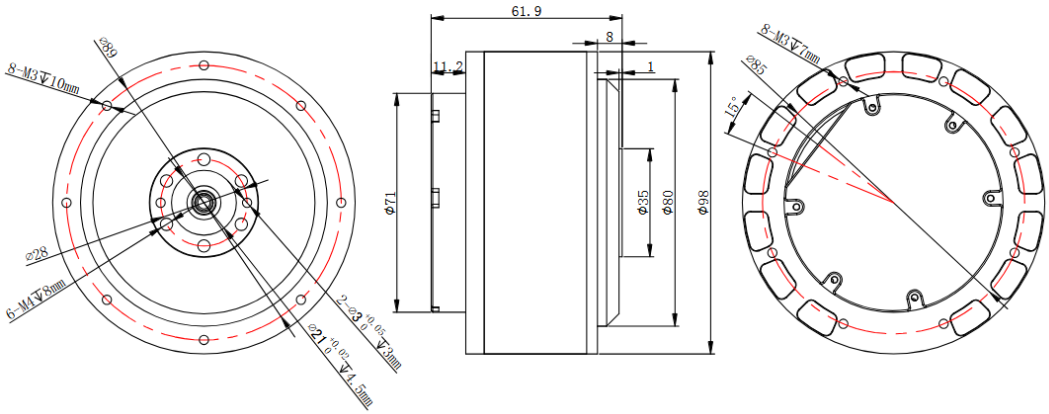

Figure 48 AK80-64 schematic

AK 80-64 specifications [7]	
Internal Reduction ratio	64:1
Rated voltage	24/48 V
Rated torque	48 Nm
Rated speed	23/48 rpm
Rated power	220 W
Peak Torque	120 Nm
Weight	850 g
Size (diameter x length)	98 mm x 61 mm

Table 5 Joint 2 motor specifications

Joint 3 meanwhile employs an AK 70-10 motor, which, besides providing more than adequate performance, had the additional positive of being already available within the team's inventory.

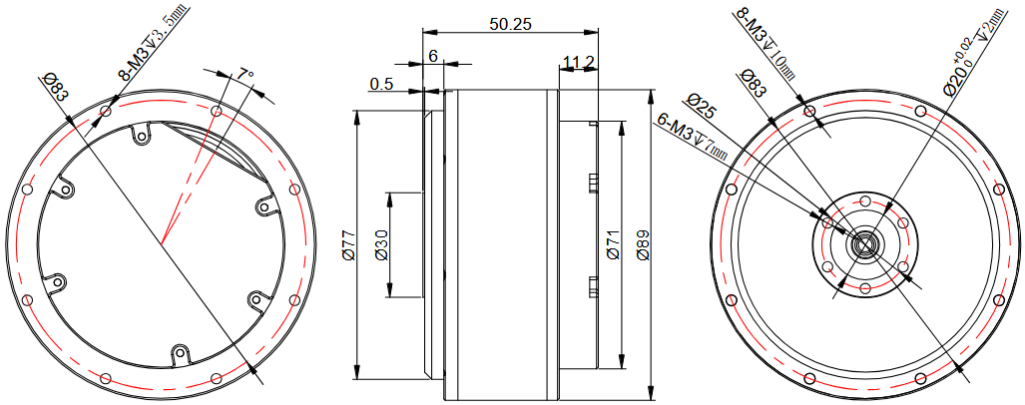

Figure 49 AK70-10 schematic

AK 70-1 specifications [8]	
Internal Reduction ratio	10:1
Rated voltage	24/48 V
Rated torque	8.3 Nm
Rated speed	148/310 rpm
Rated power	230 W
Peak Torque	24.8 Nm
Weight	521 g
Size (diameter x length)	89 mm x 50 mm

Table 6 Joint 3 Motor specifications

Wrist

The wrist can be thought of as three links connected through three revolute joints; however, due to its design, the division between parts is not as clear-cut as for the shoulder part of the manipulator. Links 5 and 6 need to be discussed together as they make up the differential portion of the wrist.

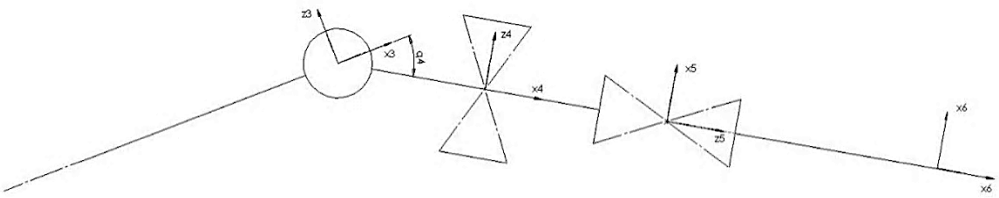

Figure 50 Wrist simplified model

Link 4

Link 4 still maintains some continuity of design with the previous links and is composed of three main parts: the frame and two M2006-P36 motors, including their 16:1 reduction gearbox.

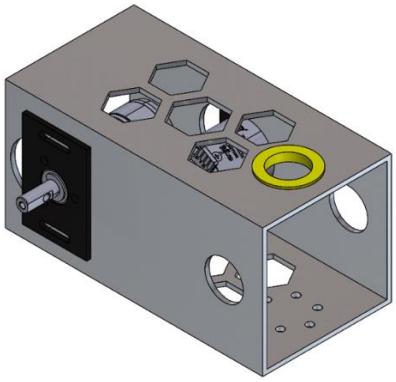

Figure 51 Link 4

These motors connected through pulleys are what actuate the differential system that acts as the last two remaining joints.

The frame is once again obtained from operations on a square piece of tubing, this time with a cross-section of 60X60 mm and a wall thickness of 2 mm. The frame has a length of 120 mm.

On one of the four sides, an 18 mm diameter hole is placed in the middle of the side with its center placed 18 mm from the top edge, and, on the opposite side, six clearance holes for M3 screws regularly spaced along a circumference need to be concentric to it.

On the remaining two sides, which will be known as the motor sides for reasons that will be made clear shortly, similar holes (18 mm diameter, placed on the midplane of their respective side, and 18 mm from the top) are required, with particular care taken to ensure they are aligned with each other.

The two motor sides hold the mounting points for the two motors. The initial design for these mounts was static, consisting of a 13 mm hole necessary to allow passage for the motor shaft, surrounded by 3 clearance M3 holes needed for the actual mounting of the motor. This implantation, however, is flawed, not permitting any positional adjustment for the motor and its connected pulley necessary to reduce the slack of the pulley belt.

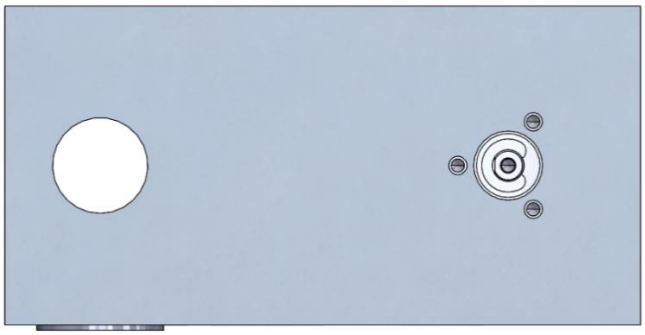

Figure 52 Fixed position design

To resolve this flaw, a different design was used: instead of mounting the motor directly to the frame we mount it to a plate which has slots to allow for a sliding motion. At the same time, the frame has a larger slot for the motor body and two M3 clearance holes reflecting the slots on the plate.

This design gives a full centimeter of adjustability to each motor to reduce the slack in the pulley belts as much as possible.

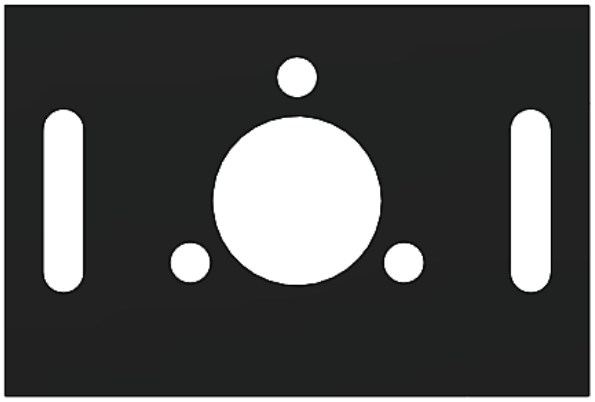

Figure 53 M2006 mounting plate

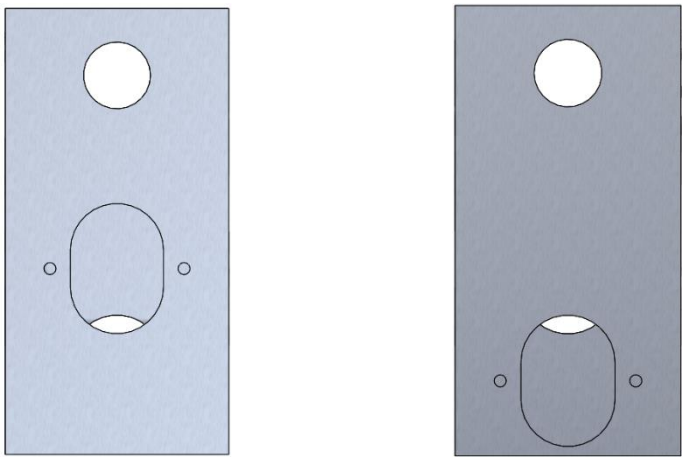

Figure 54 Frame 4 - Motor Slots

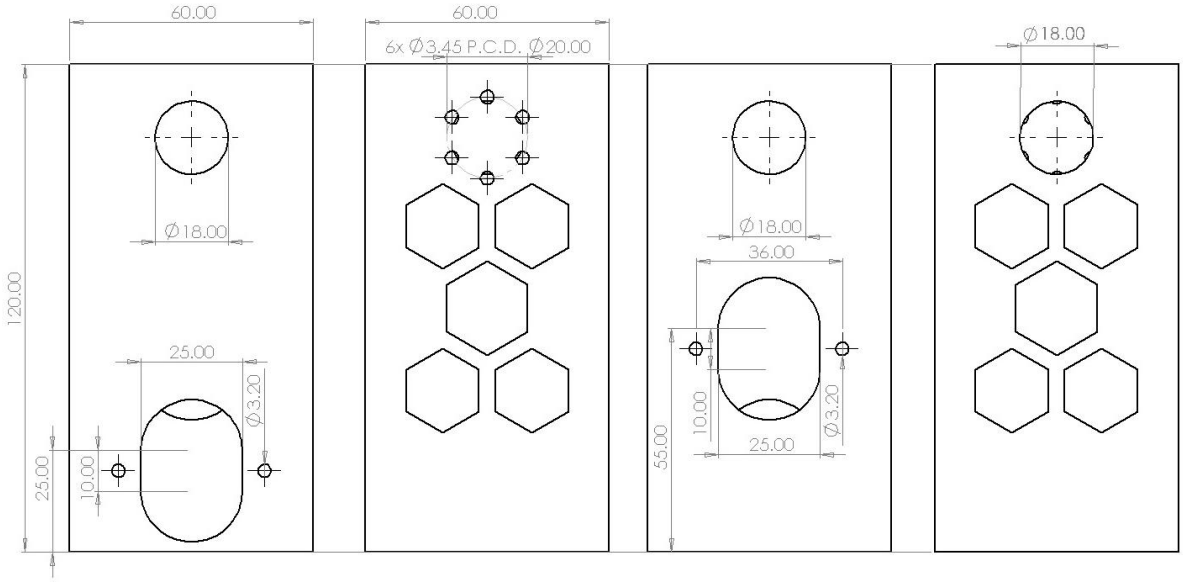

Figure 55 Link 4 Frame schematic

Joint 4

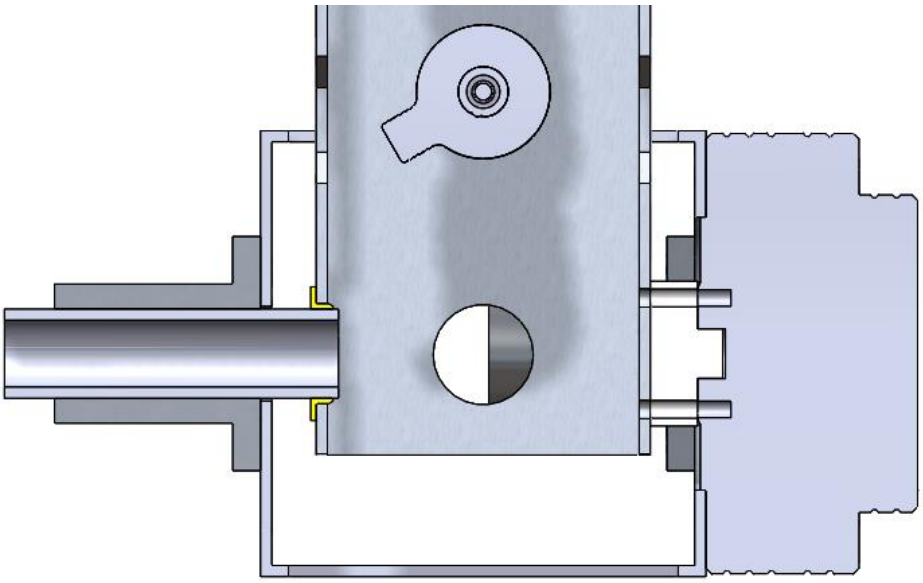

Figure 56 Joint 4 section

This joint has similar if somewhat inverted structure when compared to Joint 2 or 3 on one side the connection to the rotor is the same with a custom 3-D spacer for the proper positioning, where things do differ is on the other side where, as already anticipated when discussing Link 3, the shaft holder is placed externally and mounted to Link 3 instead of internally within Link 4, the placement of the shaft holder means that the shaft will not rotate being instead affixed to Link 3. This linkage however differs from the previous ones in yet another way, as, again due to space constraints, a ball bearing like the ones used for the previous Joints could not be employed, necessitating the use of a baffle as a substitute.

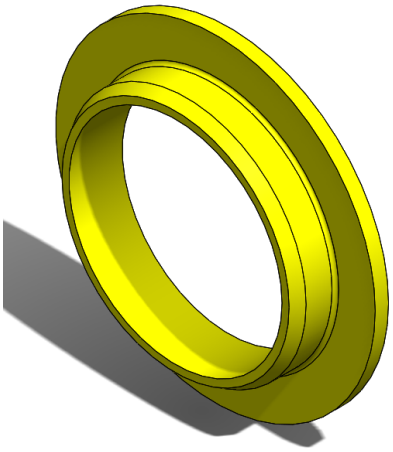

Figure 57 Flanged baffle

Baffle specifications [9]	
Material	Iglidur® G
Inner diameter	16 mm
Outer diameter	18 mm
Total length	4 mm
Flange thickness	1 mm
Flange diameter	24 mm
Dynamic friction coefficient	0.08 – 0.15 μ

Table 7 Baffle specifications

These baffles are produced by igus; they are solid pieces of plastic produced via injection moulding with materials specifically engineered to attain low friction and long longevity without any moving parts. Being pieces of plastic, they are more susceptible to damage and wear when compared to bearings, but under the comparatively light loads that the Wrist assembly is under, they are sufficiently sturdy while as a plus being cheap to replace.

Joint 4 employs another AK60-6 motor as an actuator, this one directly connected to the joint differently from the one actuating Joint 1. It was chosen due to its availability, compactness, and relatively high torque.

Differential

The differential is composed of what could be considered Link 5 and 6; however, due to the complexity of the mechanism, they will be treated as a whole until the discussion on the actual functioning of the mechanism.

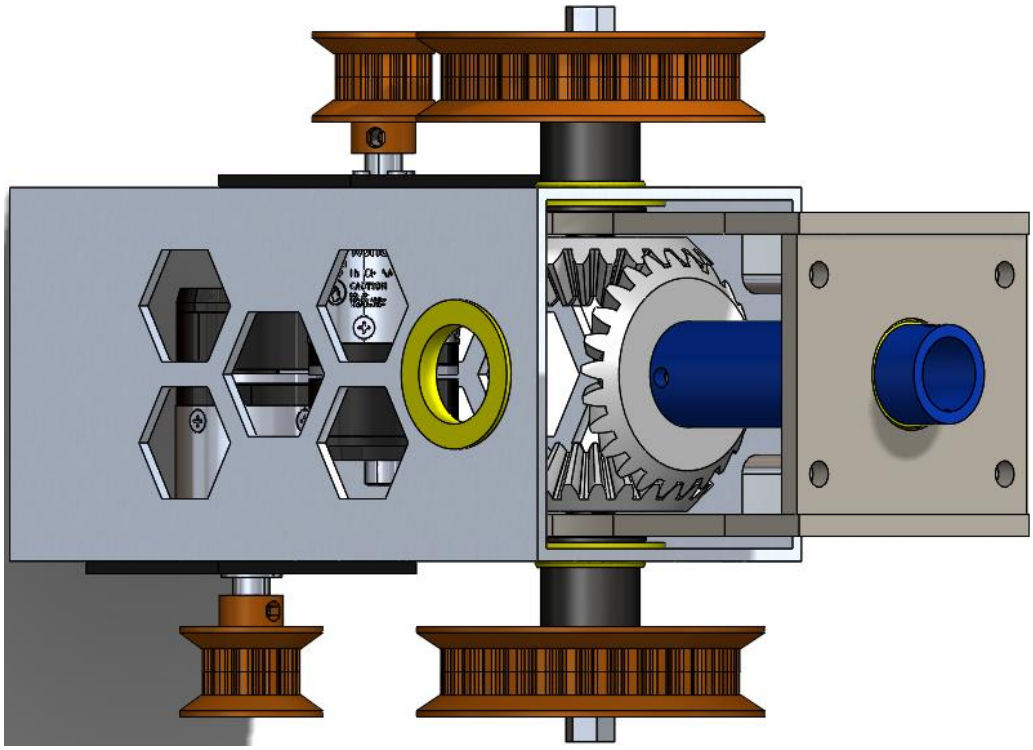

Figure 58 Differential

Before talking about the differential some external components must be addressed first: the two M2006-P36 motors present in Link 4 are connected to the differential via two separate belt drives composed of two pulleys with a reduction ratio of 3:1, these pulleys are during testing where 3-D printed parts to augment the flexibility of the construction and keep costs low however for the final assembly fabric reinforced belts should be used. For the differential XL025 model belts are required: one with 40 teeth and another with 51 teeth.

The first smaller pulley has 10 teeth and has been modeled to interface with the D-shaped shaft of the M2006 motors the other one has 30 teeth and has a hollow hub with six trapezoidal teeth to transmit the power to the pivot.

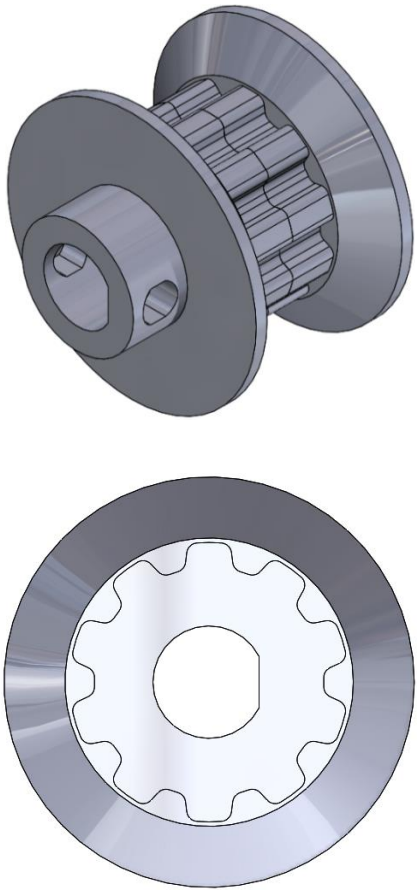

Figure 59 Differential - M2006 pulley -10 teeth

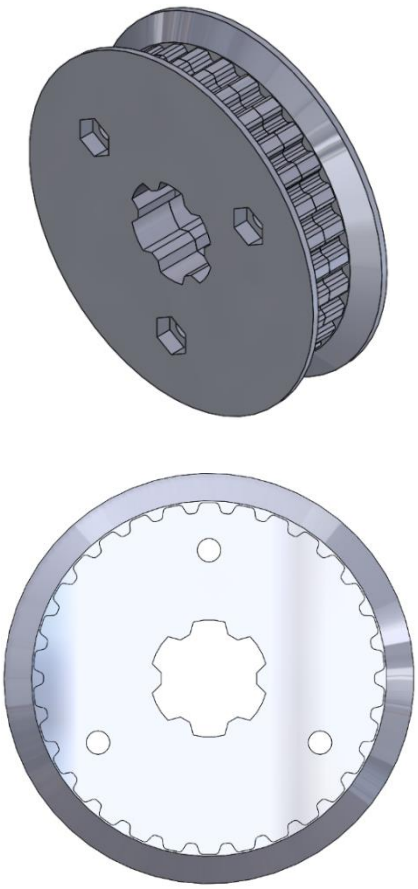

Figure 60 Differential - Pinion Pulley – 30 teeth

The pivot is an additional 3-D printed piece with multiple functions: it not only interfaces with the pulley though the aforementioned trapezoidal teeth on the back but it also acts as the hub to the flanged baffle, as a spacer and clamping point for the bearings of the gear carriage which will be discussed afterwards and finally through the annular sector shaped teeth interfaces with the pinion gears themselves. A central hole is also needed to allow the passage of a bolt through it, with the function of holding the assembly together.

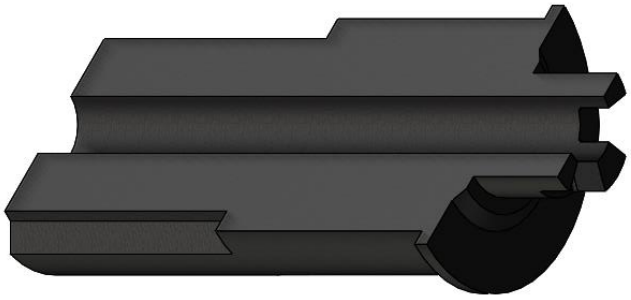

Figure 61 Differential – Pivot

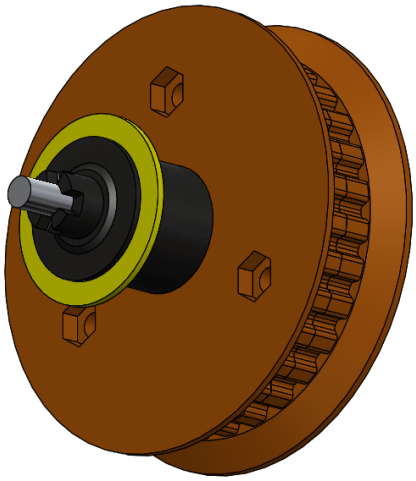

Figure 62 Differential - Pivot assembly

The differential itself is composed of a gear carriage and three bevel gears; all these components were modeled with the intention of being 3-D printed.

The bevel gears were modeled via a base parametric model, which was developed for the project. Bevel gears, unlike spur gears, change their geometry based on both the crown and pinion tooth count and the shaft angle, and as such need to be modeled as pairs. [10], [11]

These bevel gears, modeled with a 90-degree axis angle and an equal number of teeth (25 teeth) between the pinion and the crown, possess involute teeth profiles to reduce vibrations and wear: the pivot gears received a hexagonal indentation on the front to allow the insertion of a nut a clearance hole for the bolt to pass through and four annular section indentations in the back which will interface with the pivot to transmit the

torque received from the motor through the pulleys. The crown gear, meanwhile, was modified with a central bore to allow the fitting of the various pneumatic components required to attach the end effector suction cup and circular standoff on the back, which will interface with a custom spacer and lock to it through two small M3 screws.

The bevel gears should preferably be printed using a resin-based process instead of a filament-based process, to better preserve the various features and achieve a greater smoothness of the contact surfaces.

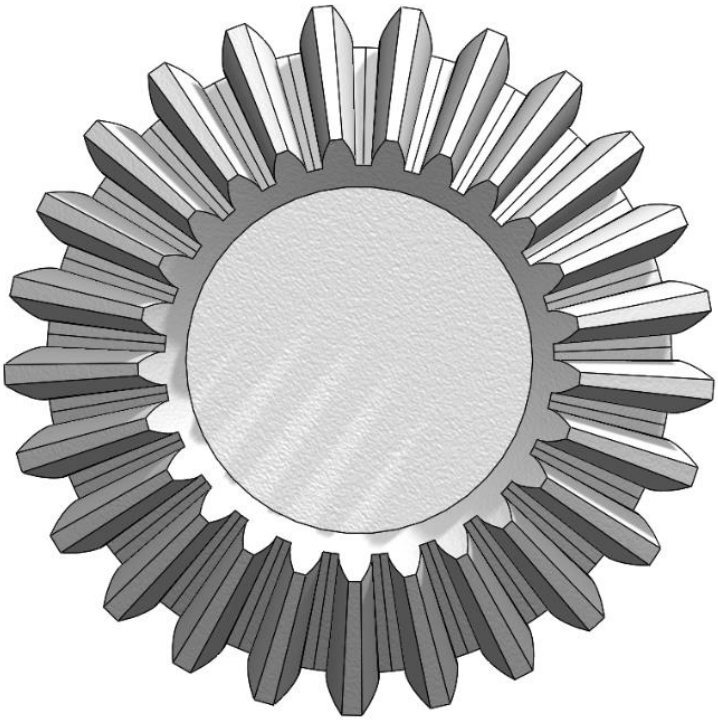

Figure 63 Base parametric gear - 1.5 Modulus 25 teeth

Figure 64 Differential - Crown Gear

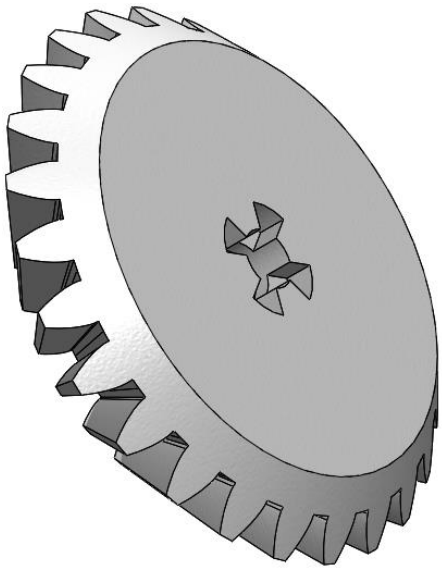

Figure 65 Differential - Pinion Gear

Gear carriage

The gear carriage could be considered Link 5 and is by far the most redesigned part of the whole assembly.

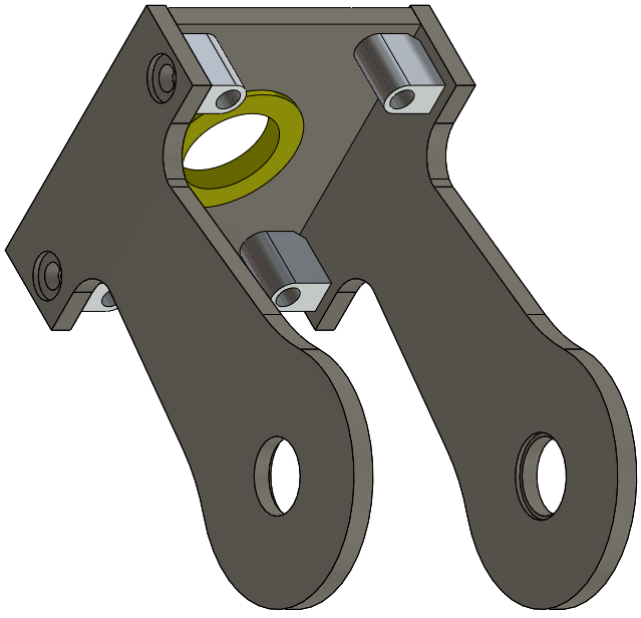

Figure 66 Differential - Gear Carriage

The final version is composed of two side panels and a top panel connected via angular fasteners. [12]

The top panel is a simple rectangle; in the center, a bore allows the insertion of a baffle, which permits the passage of the pneumatic components while reducing friction. The only other notable feature for this part are the clearance holes necessary to affix the fasteners.

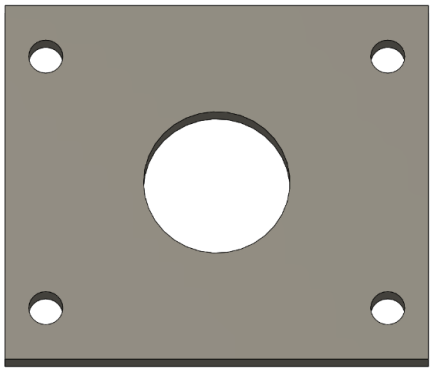

Figure 67 Differential - Gear Carriage - Top panel

The two side panels have a bearing seat for a flanged deep groove bearing near the bottom, where the pivots, in combination with the pinion gears, will clamp down on the inner raceway. The outer raceway, however, is not clamped and must rely on friction and the flange to maintain contact with the side panel. The most notable feature of the panel is its overall shape: the symmetrical cutouts on the sides are designed to allow for a greater range of movement for the carriage within the Joint 4 frame. The design allows for a roughly 70-degree swing without excessively reducing the panel's width.

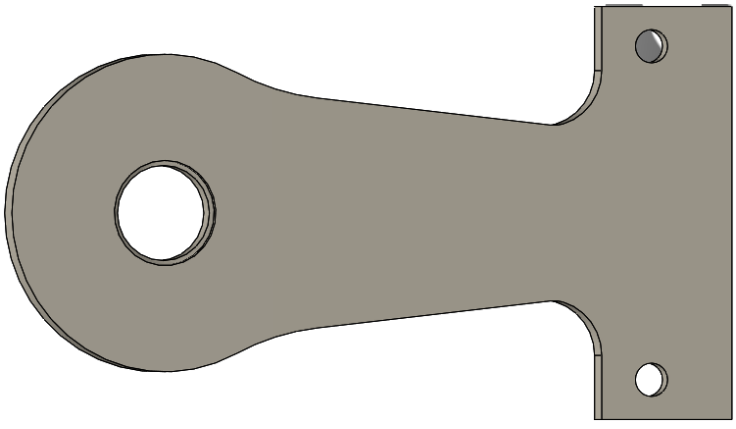

Figure 68 Differential - Gear Carriage - Side panel

Deep groove bearings specifications [13]	
Producer and RS product code	NMB 540-299
Inner diameter	8 mm
Outer diameter	12 mm
Bearing thickness	3,5 mm
Static load rating	246 N
Dynamic load rating	506 N

Table 8 Gear carriage bearing specifications

To keep the crown gear in alignment with the gear carriage' top panel and in contact with the two pinion gears a spacer was designed, this spacer sockets in with the crown gear's standoff while at the same time pressing on the baffle present within the gear carriage's top panel through the flange, like the crown gear the spacer has a central bore to allow for the passage of tubing the area at the top after the flange is equipped with an additional two M3 holes to allow the connection to the end-effector.

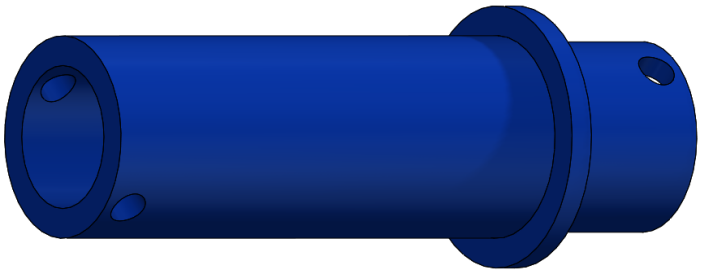

Figure 69 Crown gear spacer

This design for the gear carriage was reached due to the desire to design a spherical wrist, which offers some great benefits in terms of controls, as it allows the separation of pose and attitude control. For the specific application in this thesis, this factoid is not particularly useful as the manipulator will be controlled in real time by an operator; nevertheless, this design allows an easier development path for future integrations, which an easier-to-build non-spherical wrist would not have guaranteed.

Joint 5-6

These two joints are related to the movement of the gear carriage and crown gear respectively, which in turn are linked to the movement of the two pinion gears.

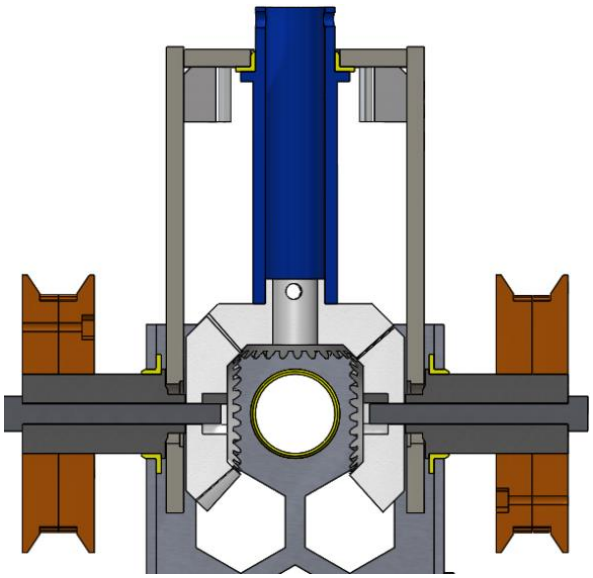

Figure 70 Differential section

Considering counterclockwise movements with respect to the pinions as positive, the formulas for calculating the position of both these components are as follows:

$$\theta_{carriage} = \theta_5 = \frac{\theta_{p1} - \theta_{p2}}{2} + c$$

$$\theta_{crown} = \theta_6 = \frac{\theta_{p1} + \theta_{p2}}{2}$$

Equation 1 Relationship between pinon positions and Link 5-6

For the position of the carriage, the term c accounts for its initial position while a similar term is not required for the crown, at least for the specific end-effector used in this thesis, as, being radially symmetric, the "true" initial angle does not matter, only its variation.

The pinion gears, as previously mentioned, are controlled through two separate pulley systems by two M2006-P36 motors, which were chosen

partially due to their availability within the team's stocks but also due to their compactness, allowing their insertion within the Link 4's frame.

M2006-P36 specifications [14]	
Internal Reduction ratio	36:1
Rated voltage	24 V
Rated torque	1 Nm
Rated speed	416 rpm
Weight	107 g
Size (diameter x length)	64.8 mm x 24.4 mm

Table 9 Differential motor specifications

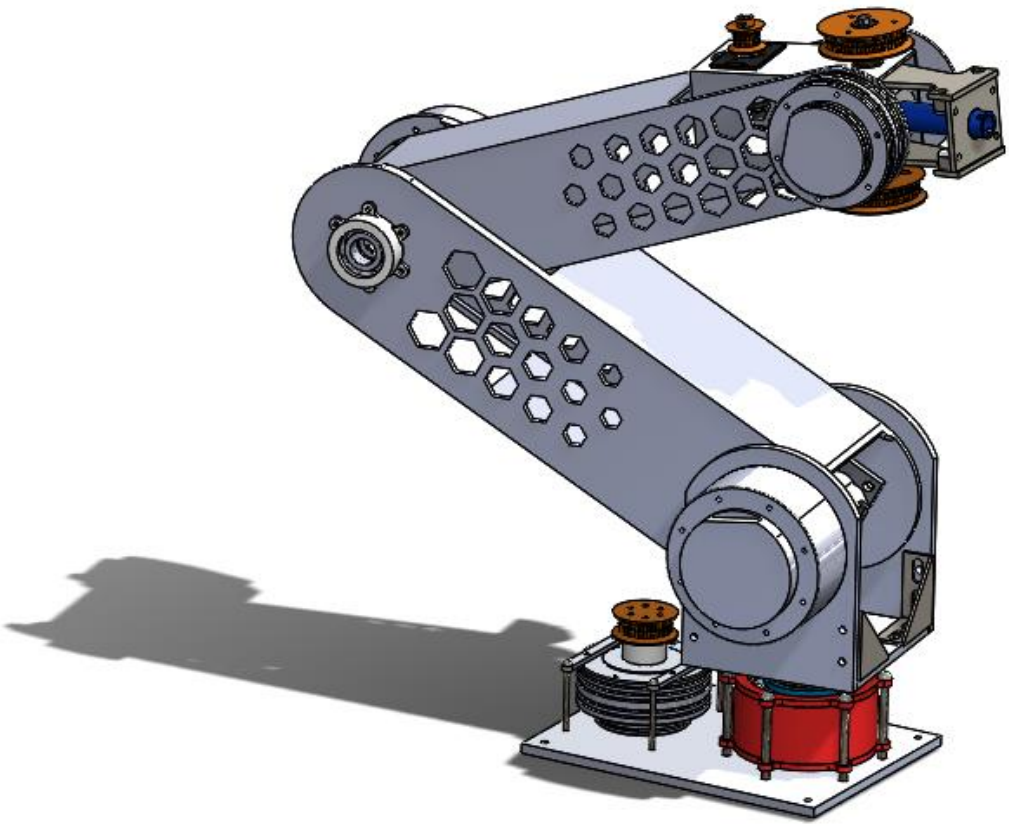

Figure 71 Whole arm at rest position

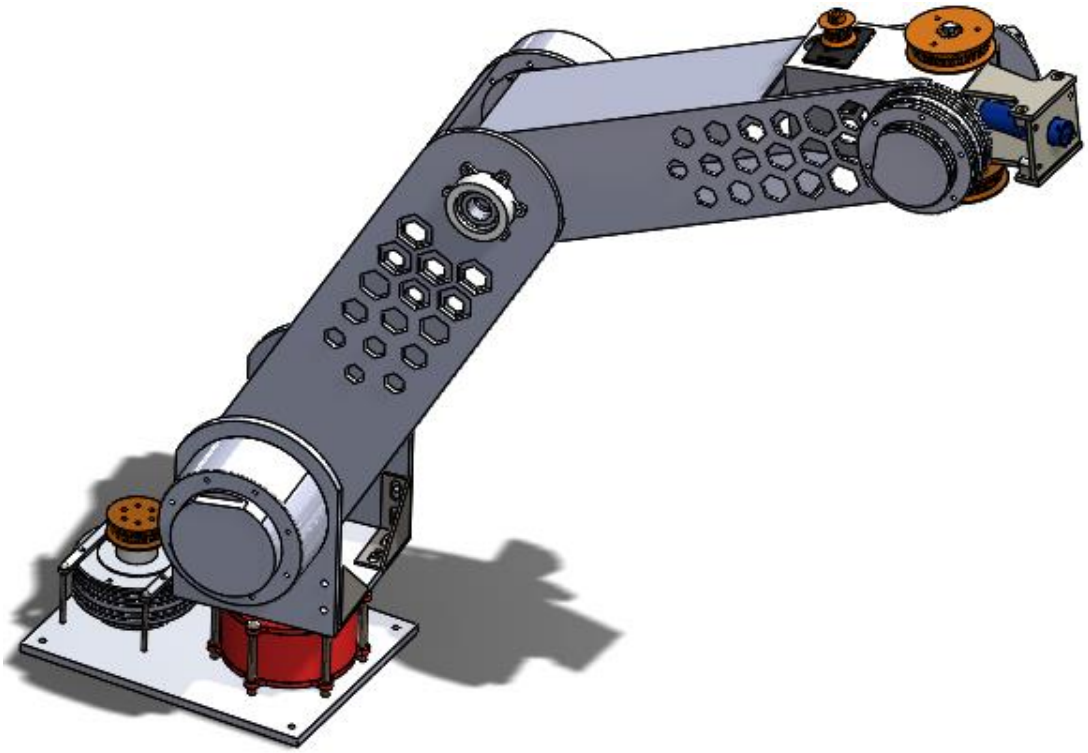

Figure 72 Arm in extension

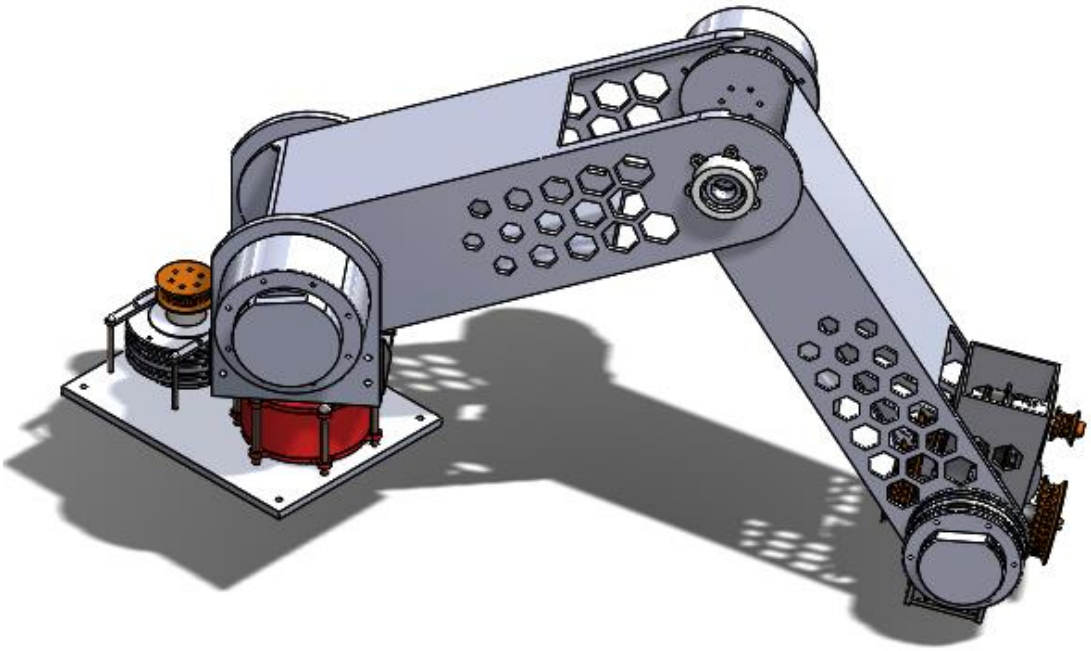

Figure 73 Ore pick up pose

5: Kinematics

Kinematics is a fundamental field of study in robotics whose purpose is linking the joint space of robots, meaning joint movements and positions, to the outer world, also known as the task space. Kinematics do not care about the cause of the movement (forces and torques), only about the effect they have on the manipulator, meaning the acceleration and thus changes in velocities and positions they cause.

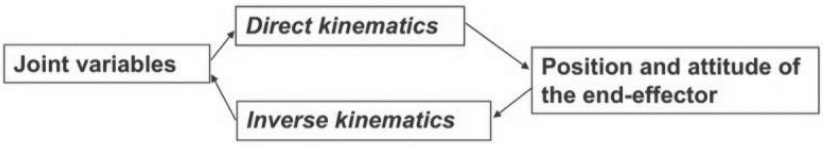

Figure 74 Relationship between joint and task space

DH convention and Forward Kinematics

Forward kinematics, also called Direct kinematics, is the process of mapping the joint variables (angle for a revolute joint or extension for a prismatic one, for example) to the end-effector's or any other link's position and attitude with respect to a given reference frame. Direct kinematics, as the name implies, are a form of direct problem and thus will only ever return a singular pose for the end-effector for a given set of joint parameters, no matter the robot type, be it a simple gantry system or the most advanced 7-axis industrial robot.

However, to calculate the direct kinematics of a robot, joint positions are not the only thing needed; knowledge of its link geometries is also required. This knowledge is used in the creation of a mathematical representation of the robot's geometry, most often via transformation matrices, the form that these matrices take however depend on the robot type and convention used;

for serial robots the most commonly used convention is the Denavit-Hartenberg convention (DH convention) which reduces the geometry of each link down to 4 relevant parameters 3 of which are fixed depending on the preceding joint type.

These 4 parameters (a link length, α link twist, d link offset, and θ joint angle) are obtained by first placing right-handed reference frames (RF) along the joint axes following the rules of the DH convention, consecutively the parameters can be obtained by taking a pair of consecutive RF at a time (ex. RF 1 and RF 2 yield the fixed parameters for the geometry of Link 2 and, since Joint 1 is revolute, the variable parameter θ_1). These parameters can then be plugged into a standardized transformation matrix, which represents the rotation and translation from one RF to the next one.

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

Equation 2 Homogeneous transformation of a single link

Concatenating these transformations by multiplying them allows us to calculate the position of the end effector or any preceding joint in any of the prior RF.

$$T_n^0 = A_1^0 A_2^1 \cdots A_n^{n-1}$$

Equation 3 Homogeneous Transformation of an n-jointed serial manipulator

Sometimes additional transformations are required if the base or end-effector's frames do not coincide with frame 0 and n, respectively:

$$T_e^b(q) = T_0^b T_n^0(q) T_e^n$$

Equation 4 Homogenous Transformation from Base to End-Effector

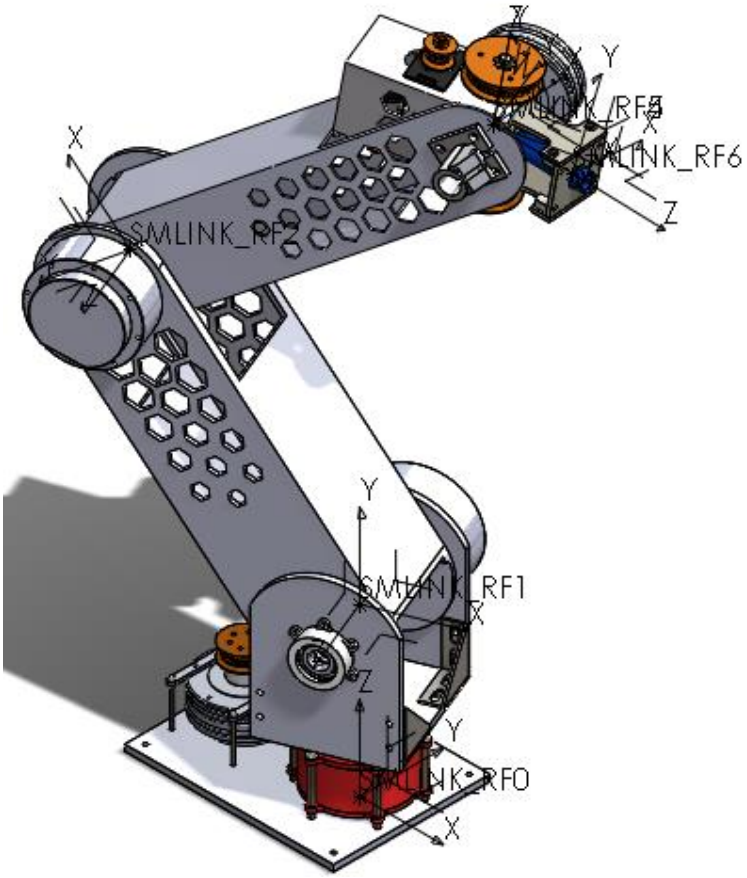

Figure 75 Manipulator DH Reference Frames

From the manipulator's geometry, the fixed DH parameters can thus be obtained:

Parameters	a link length	α link twist	d link offset
Link 1	0 mm	90°	159 mm
Link 2	300 mm	0°	0 mm
Link 3	320mm	0°	0 mm
Link 4	0 mm	-90°	0 mm
Link 5	0 mm	90°	0 mm
Link 6	0 mm	0°	79 mm

Table 10 DH Convention Parameters

Differential Kinematics

Kinematics connect the inner world of a robot, characterized by joint positions, to the outer world, characterized by the pose of the end-effector. Kinematics, however, map only positions and are difficult to invert directly; for this reason, we introduce a new field: differential kinematics, whose objective is the mapping of the joint velocities to the velocities of the end effector.

This mapping can be described through a matrix known as the Jacobian matrix, which changes depending on the robot's configuration. The matrix can also be differentiated based on its form: if the end effector's pose is expressed via a homogenous transformation matrix, we refer to it as a geometric Jacobian; if instead it is expressed through a minimal representation, we refer to it as an analytical Jacobian.

For an n-DOF manipulator:

$$\begin{cases} \dot{p} = J_P(q)\dot{q} & \text{linear velocity of the end effector} \\ \omega = J_O(q)\dot{q} & \text{angular velocity of the ened effector} \end{cases}$$

$$v = \begin{bmatrix} \dot{p} \\ \omega \end{bmatrix} = J(q)\dot{q} \quad J(q) = \begin{bmatrix} J_P(q) \\ J_O(q) \end{bmatrix}$$

Equation 5 Geometric Jacobian definition

$J(q)$ is in the form of a Geometric Jacobian with $J_p(q)$ and $J_o(q)$ being 3xn matrices representing the contribution of the various joint velocities to the end-effector. The analytical Jacobian $J_A(q)$ meanwhile:

$$\begin{cases} \dot{p} = J_P(q)\dot{q} \\ \dot{\varphi} = J_\varphi(q)\dot{q} \end{cases}$$

$$\dot{x} = \begin{bmatrix} \dot{p} \\ \dot{\varphi} \end{bmatrix} = J_A(q)\dot{q} \quad J_A(q) = \begin{bmatrix} J_P(q) \\ J_\varphi(q) \end{bmatrix}$$

Equation 6 Analytical Jacobian definition

As we can see, the difference between these two forms depends not on the position but on the attitude representation. Both approaches hold some advantages and disadvantages: the angular velocity ω is more intuitive, but its integral does not hold any physical interpretation, while $\dot{\varphi}$ in general does not coincide with ω and is less intuitive however its integral corresponds to the attitude and therefore $\dot{\varphi}$ directly expresses the variation in the Euler angles required to pass from one attitude to another.

Computing an analytical Jacobian directly is complicated; fortunately, the geometric Jacobian is far easier to obtain, especially if we have access to the DH parameters, and the analytical form can be obtained from it through a transformation.

To obtain the geometric Jacobian, we add an additional six-element column for each subsequent joint, with joint n occupying column n. How we populate the column depends on the joint type:

$$\begin{bmatrix} J_{Pi} \\ J_{Oi} \end{bmatrix} = \begin{cases} \begin{bmatrix} z_{i-1} \\ \mathbf{0} \end{bmatrix} & \text{for a } prismatic \text{ joint} \\ \begin{bmatrix} z_{i-1} \times (p_e - p_{i-1}) \\ z_{i-1} \end{bmatrix} & \text{for a } revolute \text{ joint} \end{cases}$$

Equation 7 Geometric Jacobian column characterization

Where z_{i-1} is a vector composed of the first three elements of the third column of T_{i-1}^0 and similarly p_{i-1} is the first three components of the fourth column of T_{i-1}^0 and finally $p_e$is the first three elements of the fourth column of T_n^0.

To convert the Jacobian from geometric to analytical, we use a particular relationship between the angular velocity and Euler angle derivative:

$$\omega = \begin{bmatrix} 0 & -\sin(\phi) & \cos(\phi)\sin(\theta) \\ 0 & \cos(\phi) & \sin(\phi)\sin(\theta) \\ 1 & 0 & \cos(\theta) \end{bmatrix} \dot{\varphi} = T(\varphi)\dot{\varphi}$$

Equation 8 Euler angle differential to Angular velocity conversion

And thus

$$v = \begin{bmatrix} I & \mathbf{0} \\ \mathbf{0} & T(\varphi) \end{bmatrix} \dot{x} = T_A(\varphi)\dot{x} \xrightarrow{yields} J = T_A(\varphi)J_A$$

Equation 9 Analytical to Geometric Jacobian conversion

All the configurations for which J decreases in rank are called kinematic singularities, which are of particular interest to us as they imply a loss of mobility of the manipulator and can lead to infinite solutions to the inverse kinematic problem; for these reasons and more, singularities should be avoided.

Singularities occur at the border of the reachable workspace, when the manipulator is totally bent or extended; these can be easily avoided, but there exists a more insidious kind: those appearing within the workspace due to particular configurations.

For manipulators with spherical wrists, the singularity problem can be decoupled, simplifying it: wrist singularities occur when two of the three rotation axes align, but in our case, the design itself prevents this from occurring, meanwhile, for anthropomorphic arms, a singularity occurs when the wrist center intersects z_0.

Inverse Kinematics

Inverse kinematics is the opposite of direct kinematics: its purpose is the mapping of the end-effector's pose to the joint position; however, as said previously, this is hard to do, especially directly through analytical or numerical means. Fortunately, another method exists: through the calculation of the differential kinematics and their inversion, we can obtain the desired joint velocities from a reference trajectory in the task space; the joint velocities afterwards can be integrated to obtain a reference in joint space.

This kinematic inversion through the differential kinematics is mostly done in two principal ways:

- Pseudoinverse of the Jacobian: this method allows for a certain amount of robustness around singularities using a damped-least-square pseudo inverse and, in the case of redundant manipulators, the achievement of secondary goals, such as obstacle or singularity avoidance. This method requires knowledge of the trajectory and the computation of the inverse kinematics, and as such is more computationally heavy.
- Transpose of the Jacobian: this method does not require knowledge of the trajectory and thus is less computationally heavy, as it requires only the computation of direct kinematic functions. This method does not asymptotically track moving references; the tracking error is, however, error-bounded and inversely proportional to the value of K. K in turn is bounded by hardware limits in digital applications.

6: Simulation and control

Having finished the mechanical design for the manipulator, the next step was the design of a control scheme; to do this, MATLAB by The MathWorks Inc. and its extension Simulink were employed to simulate the arm and evaluate the various possible control schemes.

The Simulink multibody link plugin was used to import the SolidWorks model as a Simulink multibody model. The imported model was modified to account for some mates that the plugin could not translate to Simulink, such as gear mates and pulley mates, yielding the following model.

Another modification was yet required: the generated model revolute joints and related motors are generated without internal mechanics and thus friction to compensate for this a Damping coefficient of $0.005\,\frac{Nm}{\mathrm{deg/s}}$ was applied wholesale to simulate the presence of friction. This assumed value assures that the simulation will not accurately and perfectly reflect the real world, and thus any controller based upon the simulation will require further calibration within the real world to compensate for these inaccuracies.

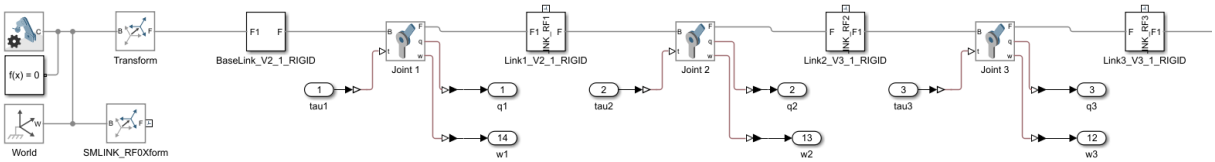

Figure 76 Manipulator Shoulder Simulink Multibody Model

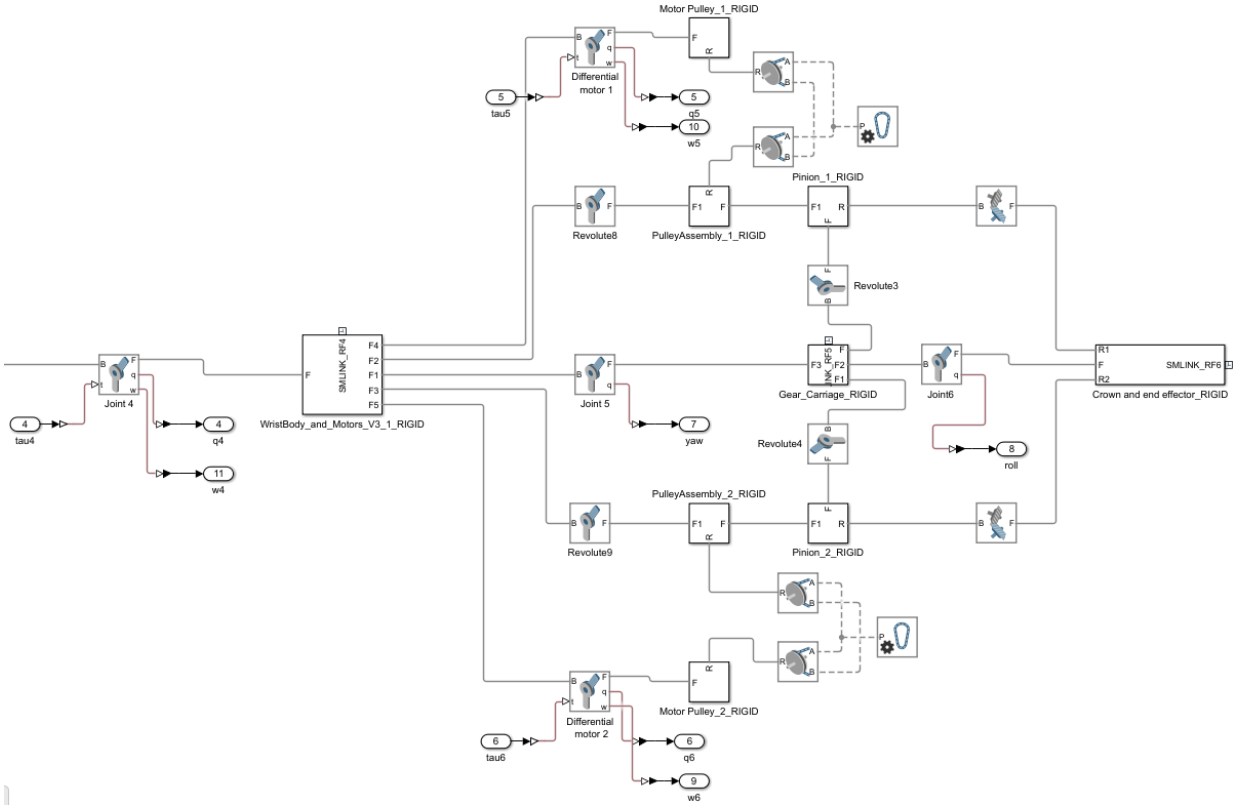

Figure 77 Manipulator Wrist Simulink multibody model

The last modification to the plant was imposing joint rotation limits; these limits were estimated through the SolidWorks model and are reported in accordance with the DH convention:

Joint limits estimations (measured through the DH convention)	
Joint 1	[-85, 275]
Joint 2	[15, 165]
Joint 3	[-130, 130]
Joint 4	[-180, 180]
Joint 5	[125, 55]
Joint 6	[0, 360]

Table 11 Joint limits for simulation

However, before designing the control algorithm to control this simulation, the kind of control to implement needed to be decided; the control scheme for manipulators can be classified in several separate ways:

- Control space: the type of reference signal we wish to use informs the decision on which type of control to use. References are almost always given in terms of the task space as they are the most intuitive, but this reference can be used as is or converted to joint space:
    - Controls in the joint space are more compact, less resource intensive and easier to implement than task space controls however they are heavily dependent on the accuracy of the initial inversion of the reference from task to joint space, the controllers will converge q joint parameters to q_d which does not guarantee that x converges to x_d even for reliable inversion method meaning that, from a certain point of view these controllers act in an open loop configuration with all the downsides that follow such a control strategy. This approach, while having its issues, is still viable and widely employed, especially when used along with reliable inversions.
    - Task/Operational space controls do converge x directly to $x_{d,}$ though this takes more complicated controllers and either additional sensors to directly measure the position of the end effector in task space or indirectly through direct kinematics; furthermore, kinematic inversion is still required to drive the actuators. The inversion in this case, however, does not need to be as precise, as inaccuracies can be treated simply as additional disturbances and compensated for through the controllers.

- Control type: the requirements of the task and hardware limits inform this choice.
    - Centralized control is typically used for high-speed and high-precision tasks; these types of controllers are MIMO systems which act on the whole robot simultaneously, controlling all the actuators at the same time while taking into account the effect that the other actuators may have on any single actuator. This kind of controller is very computationally intense but provides, in exchange, extremely high performances when properly applied.
    - Decentralized control approach applies an independent controller to each joint, treating the effect of other joints as additional disturbances, this may put two or more controllers in direct competition to achieve their own separate goals, thus leading to odd and inefficient behaviors. These kinds of controllers are best used in low-speed applications and, due to their relative simplicity, are cheaper and easier to implement compared to Centralized controllers. The use of motors with high reductions, such as the ones used for our arm, favors this kind of controller as the disturbances due to other actuators scale in a quadratically inverse fashion to the gear ratio of the actuator itself.

Due to the spherical wrist, the controls for the shoulder and the wrist can be kept separate to lower both the complexity and resource use; a decentralized control approach for both parts was first attempted.

Shoulder control

Even though we choose to separate the shoulder controls from the wrist controls standard controls schemas are presented with some difficulties: the typical use for these robots is doing autonomous repetitive tasks which are well known and provide a clear trajectory for the end effector, being remote controlled in real time however results in no clear preset trajectory and the calculation and updating of a trajectory in real time is problematic at best both in terms of implementation and computing cost. This lack of a trajectory proved to be a deciding factor when choosing which kinematic inversion scheme, necessary to translate the task space references to joint space, to use.

A trajectory gives out a lot of information including desired position and speed at any given moment, information which is required for the inverse or the pseudo inverse Jacobian Kinematic inversion methods leaving as the singular available inversion method the Jacobian transpose; this method is most suitable for steady state references where it guarantees convergence, if no singularities are encountered, and norm-bounded error for non-constant references dependent on the value of K.

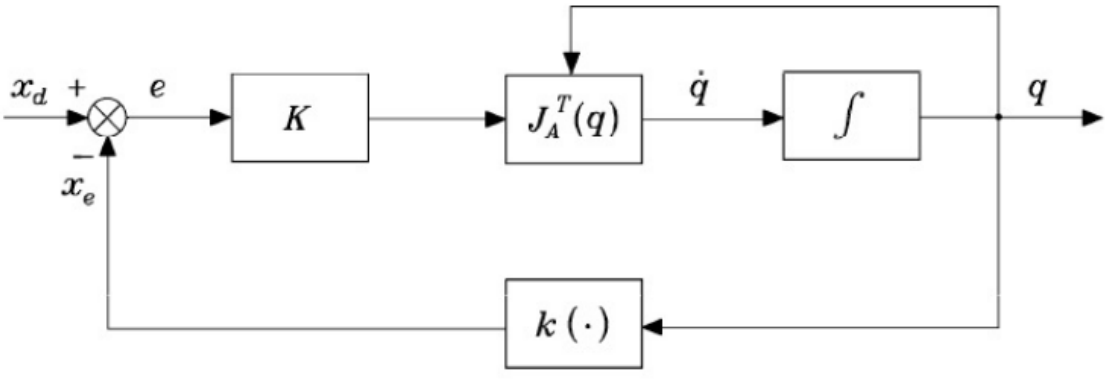

Figure 78 Jacobian transpose kinematic inversion schema

This method while providing less benefits than the other approach (The damped-least-squares Pseudoinverse of the Jacobian matrix gives a degree of robustness around singularities for example) it is still a fitting method for our application: the remote control can be interpreted as shifting the value of the reference parameters by a given rate but, as soon as the user input ceases, the parameter value stops changing and effectively becomes a steady value thus guaranteeing asymptotic convergence even with the use of the Jacobian transpose inversion method.

Another problem posed by the lack of trajectory lies in how to effectively track the references: in preset tasks the reference and thus trajectories are given in terms of the end effector, manipulators with spherical wrist allow us to generate a reference for the arm by calculating the position of the wrist center and using that as the reference for the shoulder, this however requires knowing the desired attitude $[\vec{n}, \vec{s}, \vec{a}]$ and offset d_6 of the wrist at each point in time, or at the very least the final attitude.

$$p_{wrist} = p_{end} - d_6 * \vec{a}$$

Equation 10 Standard Spherical wristed manipulator shoulder reference

The problem with this approach is that to obtain the shoulder reference we must assume that both the position and final attitude of the end effector have already been decided; however, an operator in real time does not know a priori the final end-effector's position or attitude, instead adjusting them on the fly.

Furthermore, to make use of this method we would need to provide references for both position and attitude at the same time, which would make the manipulator difficult to operate and not particularly intuitive.

For these reasons, this method, while technically feasible, was discarded. An alternate approach was implemented: the operator, instead of giving a

reference for the end effector, directly provides a reference for the wrist center. This solution allows an operator to execute rougher positional shifts by moving the wrist center, combined with more precise adjustment through the wrist itself, this solution, although rougher, is more intuitive and more easily controlled with a remote.

The last issue to resolve was the choice of which coordinate format would be used to provide the positional reference values for the wrist center. Two options were explored: providing the reference as Cartesian coordinates and as cylindrical coordinates.

The first option that was explored was the use of Cartesian coordinates, the basis for the coordinates was chosen as the base reference frame of the DH convention RF0, with the z-axis pointing upwards and the x-axis pointing forward, this was done to preclude the need for an additional static transformation A_0^b that a different basis frame would have required.

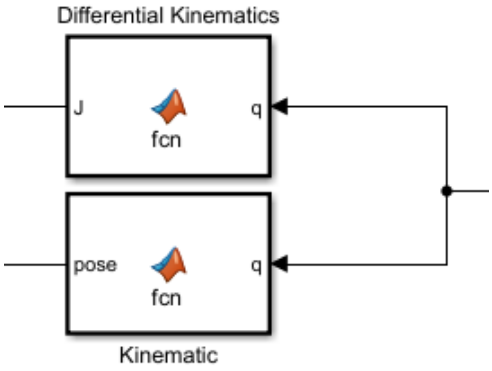

Figure 79 SolidWorks Kinematics and Differential Kinematics blocks

The forward kinematics, computed up to the elbow wrist, are divided into two for loops, the first recalculating each homogeneous transformation matrix A_i^{i-1} at each time instant, while the second calculates and stores in separate values the various concatenations of T up to T_3^0.

```
function pose  = fcn(q)
a=    [0,     0.3, 0.32, 0,  0,  0]'; %link lenghts
alpha=[90,    0,   0,    -90,  90, 0]';  %link twist
d=    [0.146, 0,   0,    0,  0,  0.069]'; %link offset
n=6; %number of joints
n=6; %number of joints
A=cell(1,n);
T=cell(1,n);
for i=1:n
    A{i}=zeros(4,4);
    A{i}=[cosd(q(i)), -sind(q(i))*cosd(alpha(i)),  sind(q(i))*sind(alpha(i)), a(i)*cosd(q(i));
          sind(q(i)),  cosd(q(i))*cosd(alpha(i)), -cosd(q(i))*cosd(alpha(i)), a(i)*sind(q(i));
           0,               sind(alpha(i)),            cosd(alpha(i)),          d(i);
           0,                     0,                      0,                  1];
end
T{1}=A{1};
for i=2:6
    T{i}=eye(4);
    T{i}=T{i-1}*A{i};
end
%Tc=Tb0*T



Tc=T{3};

pose=Tc(1:3,4);
```

Figure 80 Cartesian coordinate - Kinematics block code

As anticipated, the Jacobian transpose inversion method was used to convert the task space reference signal from the operator to reference joint positions, yet the creation of the Jacobian bears repeating: the Jacobian was computed for only the arm up to the wrist, being composed of 3 joints, it yields a 3X6 geometric Jacobian matrix.

The geometric Jacobian depends on the forward kinematics, and as such, the following code can be appended to the code of the kinematics block to obtain it: since the three joints are all revolute, a simple loop allows the calculation of the singular columns with p_e defined as the wrist center and obtained from T_3^0 while the remaining terms are obtained from the previously computed T_i^0.

```
T{1}=A{1};
for i=2:6
    T{i}=zeros(4,4);
    T{i}=T{i-1}*A{i};
end
%End effector position in RF0
pe=T{3}(1:3,4);
z{1}=[0 0 1]';
p{1}=[0 0 0]';

Jp{1}=[cross(z{1},(pe-p{1}));z{1}];
for i=2:6
    z{i}=T{i-1}(1:3,3);
    p{i}=T{i-1}(1:3,4);
    Jp{i}=[cross(z{i},(pe-p{i}));z{i}];
end

Jelbow=[Jp{1},Jp{2},Jp{3}];
```

Figure 81 Cartesian coordinates - Differential Kinematics block additional code

The position x from the direct kinematics is subtracted from the reference x_d giving a task space error vector e_x which is multiplied by a value K_j and then fed into the inverse kinematics block. K_j is a constant value used to tune the inverse kinematics; a higher value leads to a faster response time and a smaller tracking error for non-constant references, but also greater oscillations. During the first tests, a singular value K_j was utilized for all the error components, but different values could be used in a weighted matrix or as element-wise multiplication to heighten the importance of a particular parameter over another, and thus incentivize a faster convergence for that particular reference value

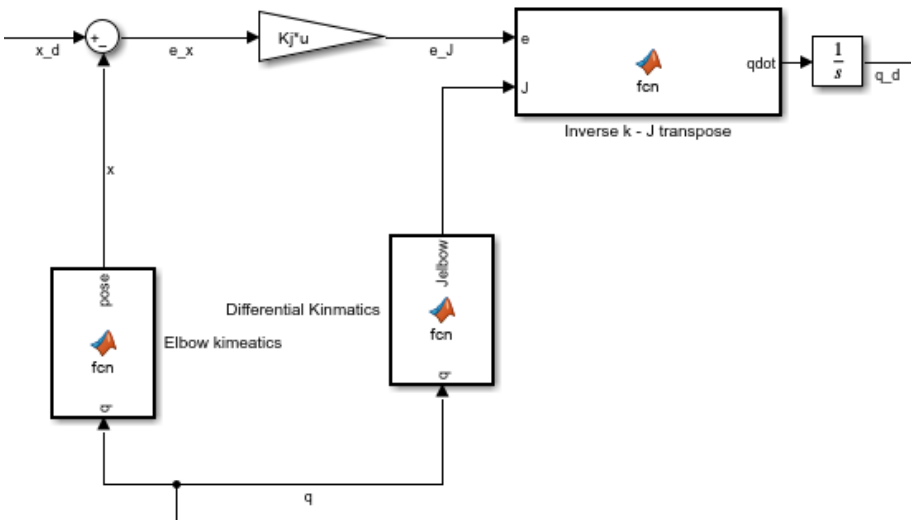

Figure 82 Cartesian coordinates – Inversion schema

The transpose block is quite simple in its function; it extracts from the full Jacobian the 3 relevant rows (the first three corresponding to the linear velocities of the end-effector), transposes the resulting 3X3 matrix, and multiplies it by the input error e_J to obtain $\dot{q_d}$ which is then integrated to obtain q_d.

```
function qdot = fcn(e,J)

J_pos=J(1:3,1:3);
qdot=J_pos'*e;
```

Figure 83 Cartesian coordinates - Transpose block code

The resulting q_d vector then acts as a reference signal for three separate controllers, before separating the vector into its separate components, we subtract the current joint positions q vector from it to obtain the positional errors e_q necessary for the control loops.

Due to an already existing implementation in the code base of the Roboto team, the controllers were at first designed as positional and velocity PIDs with the intent of replacing them with a more advanced form of control if they proved inadequate. An initial tuning was carried out, but it was not a

particular focus as the simulation was not accurate enough to accurately reflect reality and thus not suitable for real tuning, which would need to be carried out on a physical prototype.

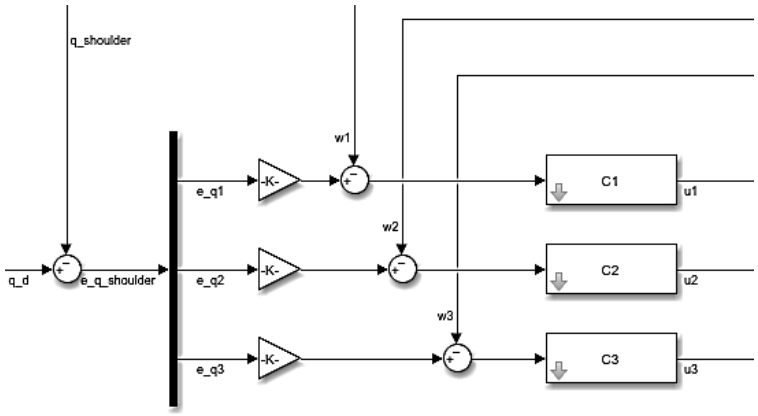

Figure 84 Position and Velocity PID control loops

Initially, the use of Cartesian coordinates showed promise, controlling the arm with precision and reasonable speed. Still, after a brief period of testing within the simulations, a fatal flaw showed itself: this method is not conscious of the joint limits, meaning that the inversion would path q_d for certain joints to impossible configurations and not compensate for them, leading to a stuck manipulator that would very slowly try to compensate to ineffectual results. This phenomena took place when a reference signal and position value switched from being concordant (positive desired x and positive actual x position for example) to being discordant (desired negative x position), in our example instead of turning the whole arm around using Joint 1 the inversion would attempt to only slightly angle the arm with Joint 1 such that the subsequent Joints were aligned on the vertical plane coincident with line described by our desired x and y and the origin and, once this was achieved, attempt to reach the objective using only Joints 2 and 3 often generating joint trajectories which forced one or both of the joints to their limits. Some attempts were made to correct this by tuning the

inversion parameters and the controllers, all unsuccessful; thus, a novel approach was attempted.

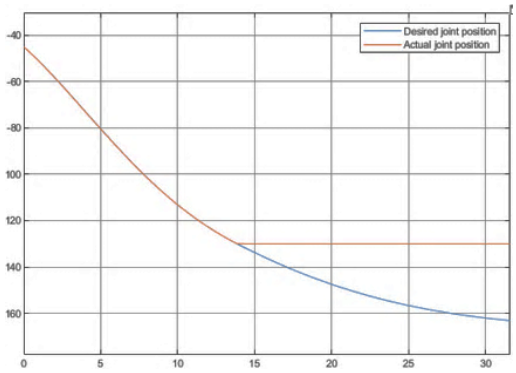

Figure 85 Joint crash example

Since the problem seemed to stem from a too conservative handling of Joint 1, resulting in its capabilities being underused, an alternate approach was attempted to remedy this flaw: by converting from a Cartesian coordinate reference system to a cylindrical coordinate system, we were able to separate the control of Joint 1, responsible for the azimuth ϕ, from the control of Joints 2 and 3, responsible for the radial distance and height.

The cylindrical coordinates result in the creation of a 'hybridized' reference since a part of it can be interpreted directly as a joint reference, while other parts are given in task space and therefore necessitate a conversion to joint space.

The azimuth from the cylindrical coordinates is reflected directly and solely on the contribution of Joint 1, and so it can be interpreted directly as a joint space reference, meanwhile, the radius and height components depend on both the remaining Joints and as such, need to be converted trough kinematic inversion to joint space before they can be used as references for the control loops.

Due to the separation of the azimuth and thus Joint 1 from the rest of the reference, the remaining Joints with their respective links can be interpreted as a 2-link planar arm, which, as the name implies, moves the wrist center on a plane with x and y coordinates corresponding to the radial distance and height.

To effectively invert the planar arm structure the transpose Jacobian method was once again employed, though the Jacobian and direct kinematics Simulink blocks needed to be modified to suit the new control architecture: since Joint 1 receives what effectively is a joint reference, it can be disregarded in the Jacobian. We choose the base frame as RF1 (with a similar reasoning to the Cartesian coordinate control, that is, to reduce the number of computations) and compute the direct kinematics and Jacobian for the remaining two joints up to the wrist center. To do this, we effectively treat Joint 2 as if it were the first joint and Joint 3 as the second joint in a planar arm.

```
function pose  = fcn(q)
a=    [0,     0.3, 0.32, 0,  0,  0]'; %link lenghts
alpha=[90,    0,   0,    -90,  90, 0]';  %link twist
d=    [0.146, 0,   0,    0,  0,  0.069]'; %link offset
n=6; %number of joints
n=6; %number of joints
A=cell(1,n);
T=cell(1,n);
for i=2:3
    A{i}=zeros(4,4);
    A{i}=[cosd(q(i)), -sind(q(i))*cosd(alpha(i)),  sind(q(i))*sind(alpha(i)), a(i)*cosd(q(i));
          sind(q(i)),  cosd(q(i))*cosd(alpha(i)), -cosd(q(i))*cosd(alpha(i)), a(i)*sind(q(i));
           0,               sind(alpha(i)),            cosd(alpha(i)),          d(i);
           0,                     0,                      0,                   1];
end
T{1}=A{2};
T{2}=T{1}*A{3};
Tc=T{2};

pose=Tc(1:2,4);
```

Figure 86 Cylindrical coordinates – Kinematics block code

The direct kinematics composed of T_3^1 supplies the current planar position of the wrist center, which will be subtracted from the reference position to obtain the task error e_x which, in turn, after being multiplied by K_j will be provided to the Transpose Jacobian block.

The Jacobian, due to considering only two joints starting from Joint 2, will be a 2X6 matrix with p_e being the first three elements of the fourth row of T_3^1 and similarly p_{i-1} and z_{i-1} being obtained in a similar fashion.

```
% Wrist center position in RF1
Tpl=A{2}*A{3};
pe=Tpl(1:3,4);
z{1}=[0 0 1]';
p{1}=[0 0 0]';
Jp{1}=[cross(z{1},(pe-p{1}));z{1}];
z{2}=A{2}(1:3,3);
p{2}=A{2}(1:3,4);
Jp{2}=[cross(z{2},(pe-p{2}));z{2}];
Jpl=[Jp{1},Jp{2}];
```

Figure 87 Cylindrical coordinates – Differential Kinematics block additional code

The azimuth position, as previously stated, is treated directly as a joint space coordinate and used directly as the reference for the Joint 1 control loop while the transpose block extracts the first two rows from the Jacobian of the planar arm, corresponding to the x y linear velocities with right to RF1, and uses the resulting 2X2 matrix to obtain the $\dot{q}_a$ for Joint 2 and 3.

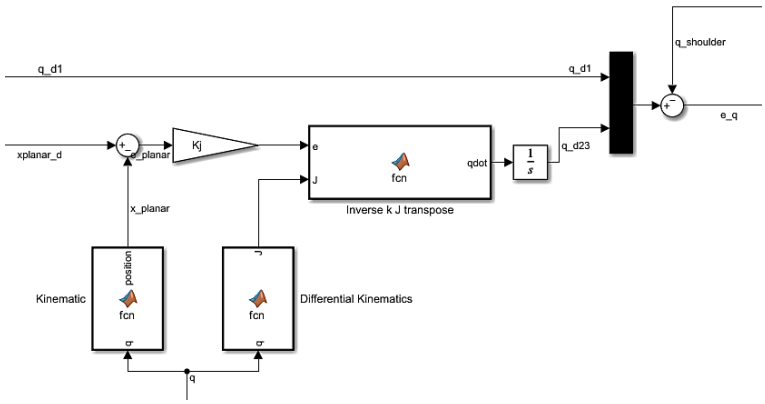

Figure 88 Cylindrical coordinates – Inversion schema

```
function qdot = fcn(J,e)

J_pos=J(1:2,1:2);
qdot=J_pos'*e;
```

Figure 89 Cylindrical coordinates – Transpose block code

This implementation has some distinct advantages compared to the Cartesian coordinate one: the Jacobian inversion upon testing appears to have fewer issues with joint limits: in the simulations, it still crashes in similar circumstances, nonetheless, thanks to the decoupling of Joint 1, the operator can simply turn around the whole arm eliminating most of the instances where the previous implementation would have failed.

Of note is also the fact that the operation of the arm itself is rather streamlined and clearer when compared to the previous attempt, as the operator can decide height and extension of the wrist center separately from the angle which renders maneuvering more instinctual than thinking in 3-D Cartesian space.

After ensuring the effectiveness and correct functioning of this method, the next action undertaken was a calibration of the controllers: the first step consisted of significantly incrementing and separating the values of K_J into two separate values to allow for more customization and increased response times, values in the range of 200-300 resulted in swift movements without destabilization. The next step consisted of adjusting the control loops, which, similarly to the previous implementation, consist of PIDs with positional and velocity feedback.

Wrist control

The wrist control employs a Roll-Pitch-Yaw representation for the attitude; this representation is easier to comprehend for remote operations compared to Euler angle representations, and thanks to the spherical nature of the wrist, we can associate each element directly to a joint parameter in the wrist, removing the need for kinematic inversions.

The yaw component of the attitude is related to and controlled directly by Joint 4; thus, similarly to the azimuth for the shoulder part of the arm, the reference can be interpreted as being given in joint space from the start and used as is in the control loop.

For the Pitch and Roll components however the story is slightly more complicated: the references do map respectively to the movement of Joint 5 and Link 5, which can be interpreted as the Gear carriage, and Joint 6 and Link 6 composed of the crown gear and end-effector, however, due to the differential design of the wrist, this mapping is not usable to control directly the actuators. To map these values to the actuator's position, we must filter them through a matrix that relates the position of Links 5 and 6 to the position of the actuators:

$$A = \begin{bmatrix} 1/2 & -1/2 \\ 1/2 & 1/2 \end{bmatrix} B = \begin{bmatrix} 1/3 & 0 \\ 0 & 1/3 \end{bmatrix} C = A * B = \begin{bmatrix} 1/6 & 1/6 \\ 1/6 & -1/6 \end{bmatrix}$$

Equation 11 Matrix relationship between motor positions and yaw and roll

Matrix A represents the relation between the position of the two Pinion gears and the position of the two Joints, while Matrix B represents the positional relationship between each Pinion gear and its respective actuator, which is the inverse of the pulleys' reduction ratio. If we multiply the two matrices, we obtain Matrix C which represents the full relationship between the actuators and the Joints.

We invert C to obtain the relationship between the Joint positions and the actuators' position:

$$D = (C)^{-1} = \begin{bmatrix} 3 & 3 \\ -3 & 3 \end{bmatrix}$$

Equation 12 Inverted matrix relationship between yaw and roll and motor positions

Having calculated the relationships between Joint positions and actuator positions, and these relationships being fixed and well defined, presented the opportunity to implement a different control schema: task space control loops.

By multiplying the current position of the actuators by C, we can obtain the current yaw and roll, which, when subtracted from the desired Pitch and Roll, yield the error in the task space e_x which, when multiplied by D, returns the joint space error e_q which in turn can be used for the decentralized control loops, which, similarly to the shoulder Controls, consist of PIDs with positional and velocity feedback.

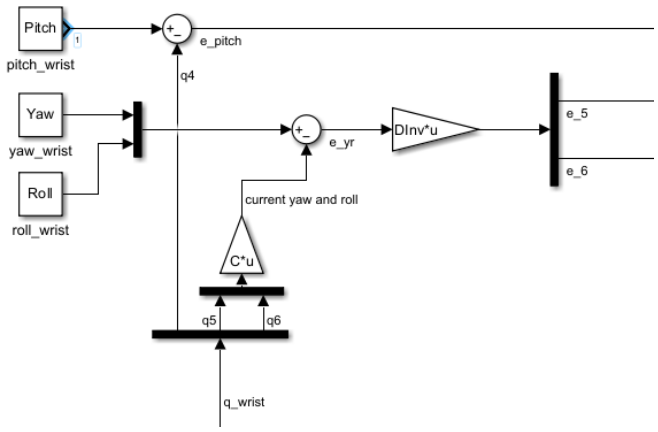

Figure 90 Wrist reference conversion to joint space

```
Kp= [1 5  3 2 5 5];
Kvp=[2 10 10 5 2 2];
Kvi=[0 1  1 1 1 1];
Kvd=[0 1  1 0 0 0];
```

Figure 91 Simulink PID control loop values for the shoulder and wrist

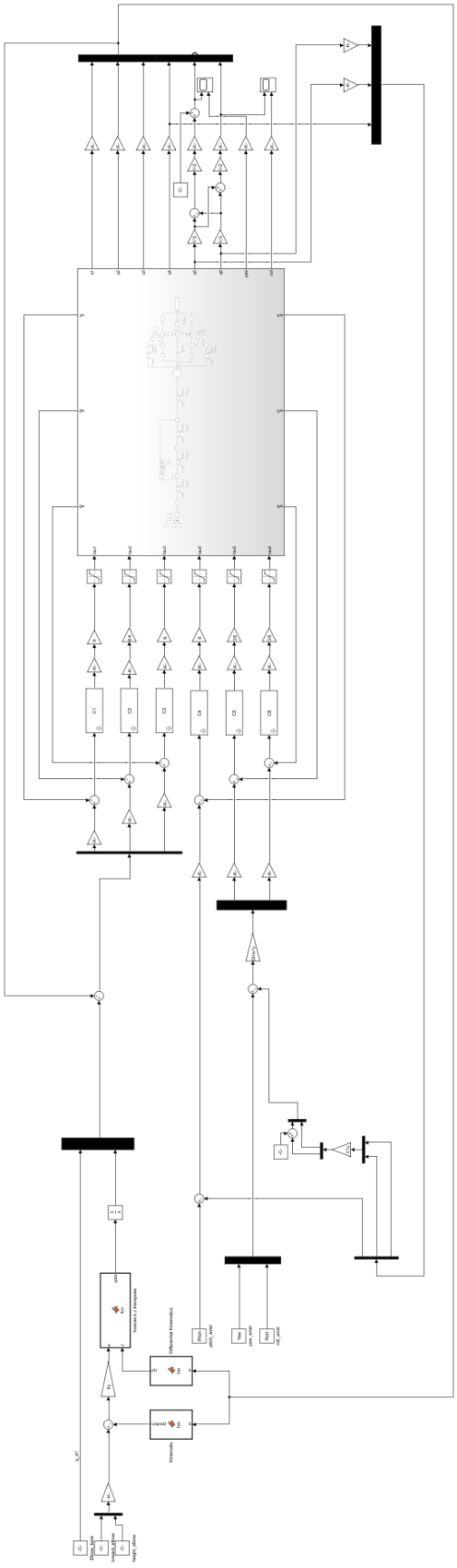

Figure 92 Final control scheme

7: Code

Having designed a control scheme, the last step of the design process was porting the control algorithm from MATLAB to a C code base deployable on an embedded microcontroller. The RoboTO team uses an STM32 407H microcontroller on all robots in their fleet, and as such, code generation efforts were focused on this model.

The code base for the whole fleet of robots is unified in a single repository; some basic components are universal; however, model model-specific code needs to be kept separate and loaded only when required. The RoboTO team's implementation undergoes this selection when compiling: a preprogrammed and preset selector establishes the model of the manipulator, and in so doing, disables all the non-model-specific code. This selector already existed and was only modified to add the engineer model to the fleet.

```
#define IS_STD_CIRC     0
#define IS_STD_RECT     1
#define IS_SENTRY       0
#define IS_HERO         0
#define IS_ENGINEER     0

  /*****************************/
 /*   CHECK ROBOT SELECTION   */
/*****************************/

#define IS_ROBOT_CONFIG_WRONG \
    IS_STD_CIRC     < 0 || \
    IS_STD_RECT     < 0 || \
    IS_STD_SENTRY   < 0 || \
    IS_STD_HERO     < 0 || \
    IS_ENGINEER     < 0 || \
    IS_STD_CIRC     > 1 || \
    IS_STD_RECT     > 1 || \
    IS_STD_SENTRY   > 1 || \
    IS_STD_HERO     > 1 || \
    IS_ENGINEER     > 1 || \
    IS_STD_CIRC + IS_STD_RECT + IS_SENTRY + IS_HERO + IS_ENGINEER!= 1
```

Figure 93 Robot selector code

All header files in the repository have include guards to prevent them from being included multiple times, causing errors during compilation.

```
#ifndef CONTROL_ENG_WRIST_H
#define CONTROL_ENG_WRIST_H
```

Figure 94 Code guard

Generic code

Some parts of the code were not generated but instead repurposed from existing components:

- Motor structs: each motor has a motor struct created for it in which we define its reduction ratio (set to 1 if the motor has a double encoder), additional reductions, the number of partitions in its digital encoder, and the conversion from digital encoder to rad. Furthermore, we use these structs to hold the data we receive from the CAN connections of the motors, including angular position, angular velocity in rpm and rads, turn count, cumulative angular position, and more. An additional component in the struct is set beforehand, decreeing whether to set the initial cumulative angular position of a motor to zero; for our manipulator, this is set to false for all motors except for the two M2006 differential motors.

```
typedef struct motor {

    uint16_t ang_pos_digital;  // Angular position [in range 0-8191]
    uint16_t prev_ang_pos_digital;  // Previous angular position [in range 0-8191]
    int16_t ang_vel_rpm;  // Angular velocity [rpm]
    fp32 ang_vel_rads;  // Angular velocity [rad/s]
    uint32_t turn_count;  // Motor's turn counter
    fp32 cumulative_ang_pos_rad;  // Cumulative angular position [rad]
    int16_t current_ampere;  // Current [A]
    uint8_t temperature_C;  // Temperature [°C]
    fp32 gearbox_ratio;  // Gearbox ratio
    fp32 additional_ratio;  // Additional motor ratio
    uint16_t num_encoder_digital_angle_partitions;  // Number of partitions the encoder's digital angular position is divided into
    fp32 digital_encoder_angle_to_rad;  // Coefficient to convert digital encoder angle into radiants
    uint8_t is_first_measurement_arrived;   // Boolean flag: 1 if first measurement have been already arrived, 0 otherwise
    uint8_t set_initial_cumulative_ang_pos_to_zero;  // Boolean flag: 1 if the initial cumulative angular position must be set to zero, 0 otherwise
    uint8_t is_cubemars; // boolean flag: 1 if motor model is cubemars
} motor_t;
```

Figure 95 General Motor struct

```
motor_t M3508_chassis_wheel[NUM_CHASSIS_WHEEL_MOTORS] = {};

motor_t AK60_arm_yaw = {};

motor_t planar_arm_motor1 = {};

motor_t planar_arm_motor2 = {};

motor_t AK60_wrist_pitch = {};

motor_t M2006_diff[NUM_DIFF_MOTORS] = {};
```

Figure 96 Manipulator's motor structs

- CAN transmissions: the motors are controlled by and communicate with the microcontroller through a CAN interface. This necessitates not only defining which CAN channel each motor belongs and their ordering in it, but also a definition of the protocols for receiving and transmitting data.

```
#if IS_ENGINEER

// Chassis wheels
#define CHASSIS_WHEELS_CAN hcan1
#define CHASSIS_WHEELS_CAN_MESSAGE_HEADER_STD_ID 0x200
#define CHASSIS_WHEEL_1_ID_CAN 0x201
#define CHASSIS_WHEEL_2_ID_CAN 0x202
#define CHASSIS_WHEEL_3_ID_CAN 0x203
#define CHASSIS_WHEEL_4_ID_CAN 0x204

// Shoulder
#define ARM_CAN hcan2
#define ARM_CAN_MESSAGE_HEADER_STD_ID 0x200
#define ARM_YAW_ID_CAN 0x201
#define PLANAR_JOINT_1_ID_CAN 0x202
#define PLANAR_JOINT_2_ID_CAN 0x203
// Wrist
#define WRIST_CAN hcan2
#define WRIST_CAN_MESSAGE_HEADER_STD_ID 0x1FF
#define WRIST_PITCH_ID_CAN 0x204
#define DIFF_LEFT_ID_CAN 0x205
#define DIFF_RIGHT_ID_CAN 0x206

#endif
```

Figure 97 CAN setup

```
void CAN_Tx_chassis_wheels(
    int16_t current_ampere_motor_1,
    int16_t current_ampere_motor_2,
    int16_t current_ampere_motor_3,
    int16_t current_ampere_motor_4) {}

void CAN_Tx_arm(
    int16_t current_ampere_arm_yaw,
    int16_t current_ampere_planar_1,
    int16_t current_ampere_planar_2) {}

void CAN_Tx_wrist(
    int16_t current_ampere_wrist_pitch,
    int16_t current_ampere_differential_1,
    int16_t current_ampere_differential_2) {}
```

Figure 98 CAN Transmit

```
motor_t* get_motor_from_CAN_message(CAN_HandleTypeDef *hcan, CAN_Rx_message_t *Rx_message) {

    // Get ID of received CAN message
    uint32_t motor_ID_CAN = Rx_message->Rx_header.StdId;

    // Return motor struct corresponding to received CAN message
    if (hcan == &CHASSIS_WHEELS_CAN) {}

    if (hcan == &ARM_CAN) {

        switch (motor_ID_CAN) {

            case ARM_YAW_ID_CAN:
                return &AK60_arm_yaw;

            case PLANAR_JOINT_1_ID_CAN:
                return &planar_arm_motor1;

            case PLANAR_JOINT_2_ID_CAN:
                return &planar_arm_motor2;
            default:
                break;
        }
    }

    if (hcan == &WRIST_CAN) {

        switch (motor_ID_CAN) {

            case WRIST_PITCH_ID_CAN:
                return &AK60_wrist_pitch;

            case DIFF_LEFT_ID_CAN:
                return &M2006_diff[0];

            case DIFF_RIGHT_ID_CAN:
                return &M2006_diff[1];

            default:
                break;
        }
    }
```

Figure 99 CAN Receive

- Chassis control: the chassis of the robot was left untouched during the design process, and as such, the control algorithm was taken wholesale from a compatible model and only slightly and superficially modified.

State machine and remote-control integration

The robot including the manipulator is controlled by an operator utilizing either a remote controller or a keyboard; due to the complexity of the task a single operator cannot operate the whole robot at the same time as such a segmentation of the robot is required, this segmentation is handled by a state machine which depending on various factors enable or disables the control of certain parts of the robot.

The remote controller has two 3-position switches on the back which function to manipulate the state machine: the right switch declares whether the motors receive commands and if they do from what source the keyboard or the remote controller itself, the left switch instead decides in case the operator is using the remote controller, which part of the robot he or she will be controlling (chassis, arm or wrist).

```
void robot_states_update_state_machine() {

    // Update previous states
    state_remote_commands_prev = state_remote_commands;
    state_controlled_subsystem_prev = state_controlled_subsystem
    state_chassis_prev = state_chassis;
    state_arm_prev = state_arm
    state_wrist_prev = state_wrist;

    // Remote commands
    state_remote_commands = _state_machine_remote_commands();

    //Controlled subsystem
    state_controlled_subsystem = _state_machine_chassis_controlled_subsystems();

    // Chassis
    state_chassis = _state_machine_chassis();

    // Arm
    state_arm = _state_machine_arm();

    // Wrist
    state_wrist = _state_machine_wrist();
}
```

Figure 100 State Machine

```
uint8_t _state_machine_remote_commands() {

    switch(remote_controller_right_switch) {

        case RC_SW_DOWN:
            // Stop commands
            return COMMANDS_STOP;

        case RC_SW_MID:
            // Get commands from keyboard and mouse
            return COMMANDS_KEYBOARD_MOUSE;

        case RC_SW_UP:
            // Get commands from remote controller
            return COMMANDS_REMOTE_CONTROLLER;

        default:
            return state_remote_commands;
    }
}
```

Figure 101 State Machine - Right switch

```
uint8_t _state_machine_chassis_controlled_subsystems() {

    switch(remote_controller_left_switch) {

        case RC_SW_DOWN:
            return CHASSIS_CONTROL;

        case RC_SW_MID:
            return ARM_CONTROL;

        case RC_SW_UP:
            return WRIST_CONTROL;

        default:
            return state_controlled_subsystem;
    }
}
```

Figure 102 State Machine - Left switch

Each separate component of the robot has its own smaller state machine, which defaults to a state of inactivity, switching to an active state only when the component is being controlled, in turn enabling some part of the controls. What that state looks like depends on the component.

```
uint8_t _state_machine_arm() {

    switch (state_remote_commands) {

        case COMMANDS_REMOTE_CONTROLLER:
            // Remote controller
            return _state_machine_arm_remote_controller();

        case COMMANDS_KEYBOARD_MOUSE:
            // Keyboard and mouse
            return _state_machine_arm_keyboard_mouse();

        default:
            return state_arm;
    }
}

uint8_t _state_machine_arm_remote_controller() {
  if (state_controlled_subsystem == ARM_CONTROL) {
      // Arm is controllable
      return ARM_MANUAL_CONTROL;
  }

  // Arm is locket into position
  return ARM_FIXED_POSE;
}

uint8_t _state_machine_arm_keyboard_mouse() {
  if (is_mouse_key_pressed(MOUSE_RIGHT_KEY) {
      // Arm is controllable
      return ARM_MANUAL_CONTROL;
  }

  // Arm is locket into position
  return ARM_FIXED_POSE;
}
```

Figure 103 State Machine - Wrist state

Control algorithms

The code of the control algorithms for the two parts of the manipulator can be roughly divided into two parts: a standardized part of the code, which was copied from the code base, and a custom-built part related to the idiosyncrasies of the module itself.

Standardized components

For standardized components, we mean the parts of the code that have already been established in the code base and whose use has not been changed. This includes:

- Controlled system structs: these structs, defined by the number of system states, inputs, and outputs they contain, serve as the caches of the various control loops.

```
controlled_system_t eng_wrist_pitch = {

    .n          = 2,  // Number of system states
    .m          = 1,  // Number of system inputs
    .p          = 2,  // Number of system outputs
    .x          = {0},
    .x_prev     = {0},
    .u          = {0},
    .u_prev     = {0},
    .y          = {0},
    .y_prev     = {0},
    .r_x        = {0},
    .r_x_prev   = {0},
    .e_x        = {0},
    .e_x_prev   = {0},
    .ei_x       = {0},
    .ed_x       = {0}
};
```

Figure 104 Controlled system struct

- PID structs: these structs hold the settings for the various PIDs, including the various coefficients of the PID itself and settings to use low-pass filters and saturate separately the proportional, integral, and derivative parts of the control. For each one of the motors, there are two of these structs present, one for its position and one for its velocity.

```
static pid__t pid_pitch_pos = {

  .Kp = 27,
  .Ki = 0,
  .Kd = 0,
  .u = 0,
  .up = 0,
  .ui = 0,
  .ud = 0,
    .lpf_up = NULL,
    .lpf_ui = NULL,
    .lpf_ud = NULL,
    .saturation_up = NULL,
    .saturation_ui = NULL,
    .saturation_ud = NULL
};
```

Figure 105 PID struct

- Parts of the control loops: some parts of the control loops themselves are standard, changing the name of the variables they act upon, but not their function; some of these parts are:
    - A command stop check: before doing anything else, the loop checks whether the robot is supposed to be turned on; if it is not, then the control loop keeps the motors unpowered by setting their provided current to zero before closing the loop.

```
if (state_remote_commands == COMMANDS_STOP) {

    memset(eng_wrist_differential.u, 0, sizeof(eng_wrist_differential.u));
    return;
}
```

Figure 106 Motor shutdown check

- State update: at the beginning of each loop, the motors' position and velocity are checked and stored in the appropriate struct.

```
// Update outputs from sensor data
for (uint8_t i = 0; i < eng_wrist_pitch.p; i++) {
    eng_wrist_pitch.y_prev[i] = eng_wrist_pitch.y[i];
}
eng_wrist_pitch.y[0] = (fp32) AK60_wrist_pitch.cumulative_ang_pos_rad;  // Angular position of REV [rad/s]
eng_wrist_pitch.y[1] = (fp32) AK60_wrist_pitch.ang_vel_rads;  // Angular velocity of REV [rad/s]

// Update states
for (uint8_t i = 0; i < eng_wrist_pitch.n; i++) {
    eng_wrist_pitch.x_prev[i] = eng_wrist_pitch.x[i];
}
eng_wrist_pitch.x[0] = eng_wrist_pitch.y[0];
eng_wrist_pitch.x[1] = eng_wrist_pitch.y[1];

// Update references
for (uint8_t i = 0; i < eng_wrist_pitch.n; i++) {
    eng_wrist_pitch.r_x_prev[i] = eng_wrist_pitch.r_x[i];
}
```

Figure 107 State update

- Module control: within each loop a switch case monitors whether the operator has switched control to the module of the control loop itself: if the state machine declares the intention of the operator to control its module the switch statement enables control operations via either the remote controller or the keyboard, if however, control switches away the loop either disables the motors or locks their position depending on the module.

```
switch (state_wrist) {
  case WRIST_MANUAL_CONTROL:
      switch (state_remote_commands) {

        case COMMANDS_REMOTE_CONTROLLER:
            // Update commands from remote controller
            eng_wrist_pitch.r_x[0] = eng_wrist_pitch.x[0] + (remote_controller_right_joystick_vertical / MAX_RC_TILT) * 20 * DEG_TO_RAD;
            break;

        case COMMANDS_KEYBOARD_MOUSE:
            // Update commands from mouse
            eng_wrist_pitch.r_x[0] = eng_wrist_pitch.x[0] - (pitch_command_mouse_to_remote_controller(dt_manipulator) / MAX_RC_TILT) * 30 * DEG_TO_RAD;
            break;

        default:
            eng_wrist_differential.r_x[0] = eng_wrist_differential.x[0]
            break;
      }

      break;
  case WRIST_FIXED_POSE:
      eng_wrist_pitch.r_x[0] = eng_wrist_pitch.x[0];
      break;
  default:
      break;
}
```

Figure 108 Controller logic

- PID implementation: the 2 PID implementation requires the output of the positional PID to be used as the reference for the velocity PID. This implementation was already present in the code base, requiring only slight modifications to be used within the control loops.

```
// Update errors w.r.t. position states
eng_wrist_pitch.e_x_prev[0]  = eng_wrist_pitch.e_x[0];
eng_wrist_pitch.e_x[0]       = eng_wrist_pitch.r_x[0] - eng_wrist_pitch.x[0];
eng_wrist_pitch.ei_x[0]     += eng_wrist_pitch.e_x[0] * dt_manipulator;
eng_wrist_pitch.ed_x[0]      = (eng_wrist_pitch.e_x[0] - eng_wrist_pitch.e_x_prev[0]) / dt_manipulator;

// Outer control loop: position
pid_control(&pid_pitch_pos, eng_wrist_pitch.e_x[0], eng_wrist_pitch.ei_x[0], eng_wrist_pitch.ed_x[0]);
eng_wrist_pitch.r_x[1] = pid_pitch_pos.u;

// Update errors w.r.t. velocity states
eng_wrist_pitch.e_x_prev[1]  = eng_wrist_pitch.e_x[1];
eng_wrist_pitch.e_x[1]       = eng_wrist_pitch.r_x[1] - eng_wrist_pitch.x[1];
eng_wrist_pitch.ei_x[1]     += eng_wrist_pitch.e_x[1] * dt_shooting;
eng_wrist_pitch.ed_x[1]      = (eng_wrist_pitch.e_x[1] - eng_wrist_pitch.e_x_prev[1]) / dt_shooting;

// Inner control loop: velocity
pid_control(&pid_pitch_vel, eng_wrist_pitch.e_x[1], eng_wrist_pitch.ei_x[1], eng_wrist_pitch.ed_x[1]);
eng_wrist_pitch.u[0] = pid_pitch_vel.u;
```

Figure 109 PID loops

Shoulder control loops

The control loop for the elbow, similarly to the structure in the Simulink model, is divided into two smaller control loops: the first controls the singular motor responsible for the azimuth orientation, while the other controls the two motors actuating the planar arm.

```
void control_loop_eng_arm() {


    _control_loop_eng_arm_yaw();
    _control_loop_eng_arm_planar();

    // Send control signals
    CAN_Tx_arm(
        (int16_t) eng_arm_yaw.u[0],
        (int16_t) eng_arm_planar.u[0],
        (int16_t) eng_arm_planar.u[1]
    );


    is_first_iter = FALSE;
}
```

Figure 110 Shoulder main control loops

The first partial control loop is responsible for the first motor actuating Joint 1, and, as within the Simulink model, the reference can be used directly; thus, the control loop can be copied from the code base with only minor modifications.

The second loop, however, is a different story as it controls the “planar arm” part of the shoulder; to allow for its functioning, custom code not already present in the code base was required. MATLAB, as a software, has the capability of generating code for embedded micro controllers, however, while attempts were made at using this ability to convert the blocks present in the model, the generated code while very efficient and specific was deemed too inflexible for use as it was not adaptable for other possible configurations and thus not open to future changes in the design. The generated code was replaced with more generic and flexible functions, which are usable for any DH convention-based robot.

To make use of these functions, the DH parameters and joint type are stored in float arrays as constants; this method also allows for the quick adjustment and expansion of the controlled manipulator.

```
static const int N_JOINTS = 2;
// Link lengths
// 0-based indexing: a[0] is a(1), a[1] is a(2)
static const fp32 a[N_JOINTS] = {0.3f, 0.32f};
// Link twist, in radians
// Since values are 0,0, they do not affect rotations in this case.
static const fp32 alpha[N_JOINTS] = {0.0f, 0.0f};
//Fixed Link offset always zero for prismatic joint
static const fp32 d[N_JOINTS] = {0.0f, 0.0f};
//Fixed Link angle in radians always zero for prismatic joint
static const fp32 theta_offset_dh[N_JOINTS] = {0.0f, 0.0f};
// Tipi di giunto (Revolute o Prismatic)
// Check che la dimensione di questo array corrisponda a N_JOINTS.
static const int joint_types[N_JOINTS] = {REVOLUTE_JOINT, REVOLUTE_JOINT};

static const fp32 Kj[N_JOINTS] = {100f, 40f};

static uint8_t is_first_iter = TRUE;
```

Figure 111 DH parameters and joint type implementation

Since C unlike MATLAB is not a vector oriented language basic operation functions, necessary for the recreation of the control schema, were recreated these include the summing, subtracting, and cross vectoring for vectors with three elements a function to allow the multiplication between matrices which can be also used to multiply a matrix by a vector or even a compatible row vector to a column vector.

```
void multiplyMatrices(fp32 *A, int rowsA, int colsA,
                      fp32 *B, int rowsB, int colsB,
                      fp32 *C) {…}

void vector_subtract(fp32 A[3],fp32 B[3], fp32 C[3]) {…}

void vector_add(fp32 A[3],fp32 B[3], fp32 C[3]) {…}

void cross_product(fp32 A[3],fp32 B[3], fp32 C[3]) {…}

void calculate_A_matrix(int index, fp32 q_val, fp32 A_matrix[4][4]) {…}

void calculate_pose(fp32 q[N_JOINTS], fp32 *pose_x, fp32 *pose_y) {…}

void calculate_jacobian(fp32 q[N_JOINTS], fp32 Jpl[6][N_JOINTS]) {…}
```

Figure 112 Custom Functions

For the kinematics, the calculation of $A_i^{i-1}(q_i)$ is segregated to its own function for compactness, as it is required for both the direct kinematics and differential kinematics. An if-else statement checks if the interested joint is revolute or prismatic and subsequently handles the variable parameter. To improve run-time, the trigonometric operations are computed once and stored within variables, which are then used to define the various matrix elements.

```
void calculate_A_matrix(int index, fp32 q_val, fp32 A_matrix[4][4]) {
    // Uses the DH parameters: a_dh, alpha_dh, d_offset_dh, theta_offset_dh + joint_types

    fp32 current_theta; // Effective theta angle
    fp32 current_d;     // Effective d offset

    if (joint_types[index] == REVOLUTE_JOINT) {
        current_theta = theta_offset_dh[index] + q_val;
        current_d = d[index]; //constant d
    } else { // PRISMATIC_JOINT
        current_theta = theta_offset_dh[index]; // constant theta
        current_d = d_offset_dh[index] + q_val;
    }

    fp32 cos_current_theta = cosf(current_theta);
    fp32 sin_current_theta = sinf(current_theta);
    fp32 cos_alpha = cosf(alpha_dh[index]);
    fp32 sin_alpha = sinf(alpha_dh[index]);

    A_matrix[0][0] = cos_current_theta;
    A_matrix[0][1] = -sin_current_theta * cos_alpha;
    A_matrix[0][2] = sin_current_theta * sin_alpha;
    A_matrix[0][3] = a_dh[index] * cos_current_theta;

    A_matrix[1][0] = sin_current_theta;
    A_matrix[1][1] = cos_current_theta * cos_alpha;
    A_matrix[1][2] = -cos_current_theta * sin_alpha;
    A_matrix[1][3] = a_dh[index] * sin_current_theta;

    A_matrix[2][0] = 0.0f;
    A_matrix[2][1] = sin_alpha;
    A_matrix[2][2] = cos_alpha;
    A_matrix[2][3] = current_d;

    A_matrix[3][0] = 0.0f;
    A_matrix[3][1] = 0.0f;
    A_matrix[3][2] = 0.0f;
    A_matrix[3][3] = 1.0f;
}
```

Figure 113 A matrix calculation

As anticipated, the A matrix function is used in calculating the pose of the planar arm, which is done through a separate function where, through a loop, all the $A_i^{i-1}(q_i)$ are calculated and, through a separate loop, they are concatenated to obtain T_2^0 for the planar arm, which corresponds to T_3^1 for the wider manipulator. The function then extracts the x and y coordinates of the position.

This function is slightly redundant as the same operation loop is also carried out for the differential kinematics; however, for flexibility's sake, it was kept separate.

```
void calculate_pose(fp32 q[N_JOINTS], fp32 *pose_x, fp32 *pose_y) {
    // Array to hold the individual A matrices (A[0] for A1, A[1] for A2, etc.)
    fp32 A_matrices[N_JOINTS][4][4];
    // Calculate each A matrix using a loop
    for (int i = 0; i < N_JOINTS; i++) {
        calculate_A_matrix(i, q[i], A_matrices[i]);
    }
    // --- Calculate the T matrix (T = A_0 * A_1 * ... * A_n-1)
    // Initialize 'pose' as an identity matrix
    float pose[4][4] = {
        {1.0f, 0.0f, 0.0f, 0.0f},
        {0.0f, 1.0f, 0.0f, 0.0f},
        {0.0f, 0.0f, 1.0f, 0.0f},
        {0.0f, 0.0f, 0.0f, 1.0f}
    };
    // Temporary matrix to store intermediate multiplication results.
    float temp_pose[4][4];

    // Multiply all A matrices in sequence: pose = pose * A_i
    for (int i = 0; i < N_JOINTS; i++) {
        // The result is stored in temp_pose
        multiplyMatrices(&pose[0][0], 4, 4,
                                &A_matrices[i][0][0], 4, 4,
                                &temp_pose[0][0]);
        // Copy the content of temp_pose back to pose for the next iteration.
        // This makes 'pose' accumulate the transformations.
        for(int r = 0; r < 4; r++) {
            for(int c = 0; c < 4; c++) {
                pose[r][c] = temp_pose[r][c];
            }
        }
    }
    // --- Extract position (x, y) ---
    *pose_x = pose[0][3]; // Assign x value to pose_x pointer
    *pose_y = pose[1][3]; // Assign y value to pose_y pointer
}
```

Figure 114 Planar position calculation

The last of the custom functions calculates the Jacobian, in a similar way to the direct kinematics, both the A_i^{i-1} and T_i^0 are computed, stored, and used to obtain the necessary components to compute the geometrical Jacobian column by column.

```
fp32 pe[3];
pe[0] = T_base_to_joint[N_JOINTS - 1][0][3];
pe[1] = T_base_to_joint[N_JOINTS - 1][1][3];
pe[2] = T_base_to_joint[N_JOINTS - 1][2][3];


for (int j = 0; j < N_JOINTS; j++) {
    fp32 z_j[3];
    fp32 p_j[3];

    if (j == 0) {
        z_j[0] = 0.0f; z_j[1] = 0.0f; z_j[2] = 1.0f;
        p_j[0] = 0.0f; p_j[1] = 0.0f; p_j[2] = 0.0f;
    } else {
        z_j[0] = T_base_to_joint[j-1][0][2];
        z_j[1] = T_base_to_joint[j-1][1][2];
        z_j[2] = T_base_to_joint[j-1][2][2];

        p_j[0] = T_base_to_joint[j-1][0][3];
        p_j[1] = T_base_to_joint[j-1][1][3];
        p_j[2] = T_base_to_joint[j-1][2][3];
    }
    fp32 pe_minus_pj[3];
    vector_subtract(pe, p_j, pe_minus_pj);

    if (joint_types[j] == REVOLUTE_JOINT) {
        fp32 v_j[3];
        cross_product(z_j, pe_minus_pj, v_j);
        Jpl[0][j] = v_j[0]; // Vx
        Jpl[1][j] = v_j[1]; // Vy
        Jpl[2][j] = v_j[2]; // Vz
        Jpl[3][j] = z_j[0]; // Wx
        Jpl[4][j] = z_j[1]; // Wy
        Jpl[5][j] = z_j[2]; // Wz
    } else { // PRISMATIC_JOINT
        Jpl[0][j] = z_j[0]; // Vx
        Jpl[1][j] = z_j[1]; // Vy
        Jpl[2][j] = z_j[2]; // Vz
        Jpl[3][j] = 0.0f;   // Wx
        Jpl[4][j] = 0.0f;   // Wy
        Jpl[5][j] = 0.0f;   // Wz
    }
}
```

Figure 115 Jacobian calculation

The control loop itself utilizes these custom functions to translate the functioning of the MATLAB design: after the motor positions and variables are stored, the calculate_pose function is used to obtain the current x and y position of the wrist center; these positions are used as the basis for the reference and to subsequently calculate the error in the task space.

```
fp32 qpos[N_JOINTS]={eng_arm_planar.x[0], eng_arm_planar.x[1]}

fp32 Jpl[N_JOINTS][6];

calculate_pose(qpos, &eng_arm_planar.x[4], &eng_arm_planar.x[5]); //calculate current radial distance and height
```

Figure 116 Control loop position calculation

The Jacobian is calculated through the custom function, then the necessary partial matrix is extracted, transposed, and multiplied by eJ, which in turn is obtained by multiplying the pose error by Kj, to obtain the $\dot{q}$. A simple integration starting from an initial state gives out the positional references for the motors, at which point the standard PID control takes over.

```
// Update pose errors
eng_arm_planar.e_x_prev[4]  = eng_arm_planar.e_x[4];
eng_arm_planar.e_x[4]       = eng_arm_planar.r_x[4] - eng_arm_planar.x[4];
eng_arm_planar.e_x_prev[5]  = eng_arm_planar.e_x[5];
eng_arm_planar.e_x[5]       = eng_arm_planar.r_x[5] - eng_arm_planar.x[5];

fp32 eJ[N_JOINTS];
eJ[0]=Kj[0]*eng_arm_planar.e_x[4];
eJ[1]=Kj[1]*eng_arm_planar.e_x[5];

fp32 Jpl[N_JOINTS][6];
calculate_jacobian(qpos,Jpl);

fp32 J_partial_transpose[2][2];
for (int i = 0; i < 2; i++) {
  for (int j = 0; j < 2; j++) {
      J_partial_transpose[i][j] = Jpl[j][i];  // Extract partial matrix from the full Jacobian and transpose it
  }
}
fp32 qdot[N_JOINTS];
multiplyMatrices(&J_partial_transpose[0][0],2,2,&eJ[0],2,1,&qdot[0])

if (is_first_iter == TRUE){
  eng_arm_planar.r_x[0]=eng_arm_planar.x[0];
  eng_arm_planar.r_x[1]=eng_arm_planar.x[1];
}
eng_arm_planar.r_x[0]+=qdot[0]*dt_manipulator;
eng_arm_planar.r_x[1]+=qdot[1]*dt_manipulator;
```

Figure 117 Transpose J Kinematic Inversion

Wrist control

The wrist control loop, similarly to the shoulder's, can be divided into two partial loops, one responsible for the pitch through a single motor, the other controlling yaw and roll through the differential and its two motors.

```
void control_loop_wrist(void) {

    //Control loops
    _control_loop_eng_wrist_pitch();
    _control_loop_eng_differential();

    is_first_iter = FALSE;

    // Send control signals
    CAN_Tx_wrist(
        (int16_t) eng_wrist_pitch.u[0],
        (int16_t) eng_wrist_differential.u[0],
        (int16_t) eng_wrist_differential.u[1]
    );
}
```

Figure 118 Wrist main control loop

The pitch control loop, similarly to the azimuth/yaw control loop of the shoulder, has basically zero custom code except for renamed variables and structs, as the received reference can be used directly. The differential control loop, meanwhile, requires a minimum of custom code; however, since the inversion itself uses a precalculated fixed matrix, no additional functions were required.

Due to the presence in the motor structs of the 'additional reduction' parameter, the presence of matrix B is made redundant; thus, instead of using the inverted matrix D, we can just invert A, obtaining.

$$E = (A)^{-1} = \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix}$$

Matrix A is used directly to obtain the current yaw and roll, which, after being subtracted from the reference, yields e_x. Using matrix E and e_x we

obtain e_q which is used finally for the PID controllers. These matrix operations, due to the fixed nature of the matrices, are carried out in plain form without the use of functions.

```
// Convert motor values to yaw/roll
eng_wrist_differential.x[4] = (eng_wrist_differential.x[0] - eng_wrist_differential.x[1])/2 // current yaw and roll values position
eng_wrist_differential.x[5] = (eng_wrist_differential.x[0] + eng_wrist_differential.x[1])/2
```

Figure 119 Current yaw and roll calculation

```
// Calculating errors in terms of yaw/roll
eng_wrist_differential.e_x_prev[4]  = eng_wrist_differential.e_x[4];
eng_wrist_differential.e_x[4] = eng_wrist_differential.r_x[4] - eng_wrist_differential.x[4]; //yaw error

eng_wrist_differential.e_x_prev[5]  = eng_wrist_differential.e_x[5];
eng_wrist_differential.e_x[5] = eng_wrist_differential.r_x[5] - eng_wrist_differential.x[5]; //roll error
```

Figure 120 Error in terms of yaw and roll positions

```
eng_wrist_differential.e_x[0] = eng_wrist_differential.e_x[4] +  eng_wrist_differential.e_x[5];
eng_wrist_differential.e_x[1] = -eng_wrist_differential.e_x[4] +  eng_wrist_differential.e_x[5];
```

Figure 121 Error in terms of motor positions

The differential motors, as anticipated, are the only ones that set their cumulative angle to 0 at startup. This is done so that the initial roll will be considered zero; this arrangement, however, presents a problem for the yaw estimation, which depends on the accuracy of the initial position to impose digital limits to prevent damage to the gear carriage. To solve this problem and ensure that the angle measurement can be made compliant with the DH convention for potential future developments, a fork-shaped removable fixture was developed, which ensures the initial position at robot startup is central.

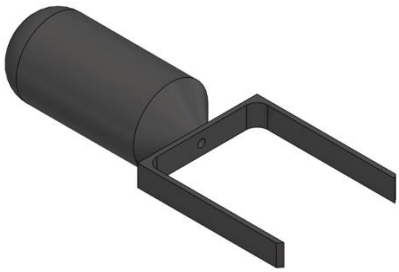

Figure 122 Differential positioner

8: Physical prototyping

The work carried out so far is based entirely on a digital environment, laying a foundation for future modifications, refinements, and improvements; however, there is a need for verification and validation of the project, and thus a prototyping phase was initiated.

The fabrication of a full-scale prototype of the whole model was hindered by long lead times on some of the components, especially so for the motors. Fortunately, the RoboTO team had access to the components needed for a partial prototype, namely the wrist, allowing for a partial build to take place along with the testing and tuning of the control software.

The wrist assembly is mostly composed of 3-D printed parts, but to keep costs low and speed up production of a prototype, the Link 4 frame, a cut-down version of Link 3's frame, and the pulley belts were also 3-D printed. This approach is feasible for testing only, as the relative fragility and dimensional accuracy of these parts are not suitable for real-world deployment; it is, however, perfectly acceptable to use in the controlled conditions a prototype like ours might encounter.

To increase the strength of the frames, the decorative hexagons were not included and, like most of the other components, were printed using a filament printer.

The pulleys, to preserve their form and correctly print, were split into two sides. The differential gears, however, were printed in a single part using a resin-based process to ensure correct dimensionality, part strength, and smoothness of the teeth.

Figure 123 3-D printed pulleys

Figure 124 Old Gear carriage side panel with two 1203 angular connectors installed

Figure 125 Gear carriage top panel with 3-D printed baffle installed (in purple)

At this point, Keil, an embedded software development program, was used to compile the repository, and after debugging was also used to flash the microcontroller memory so that testing could be carried out. The inputs were sent from a remote controller through an antenna connected to the microcontroller.

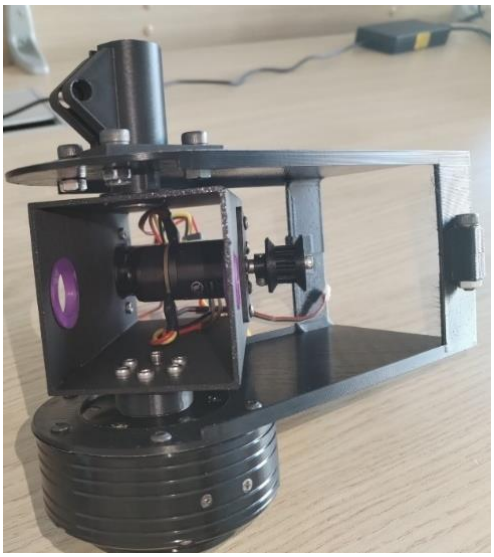

Figure 126 Partially assembled prototype with Joint 4 connections clearly visible

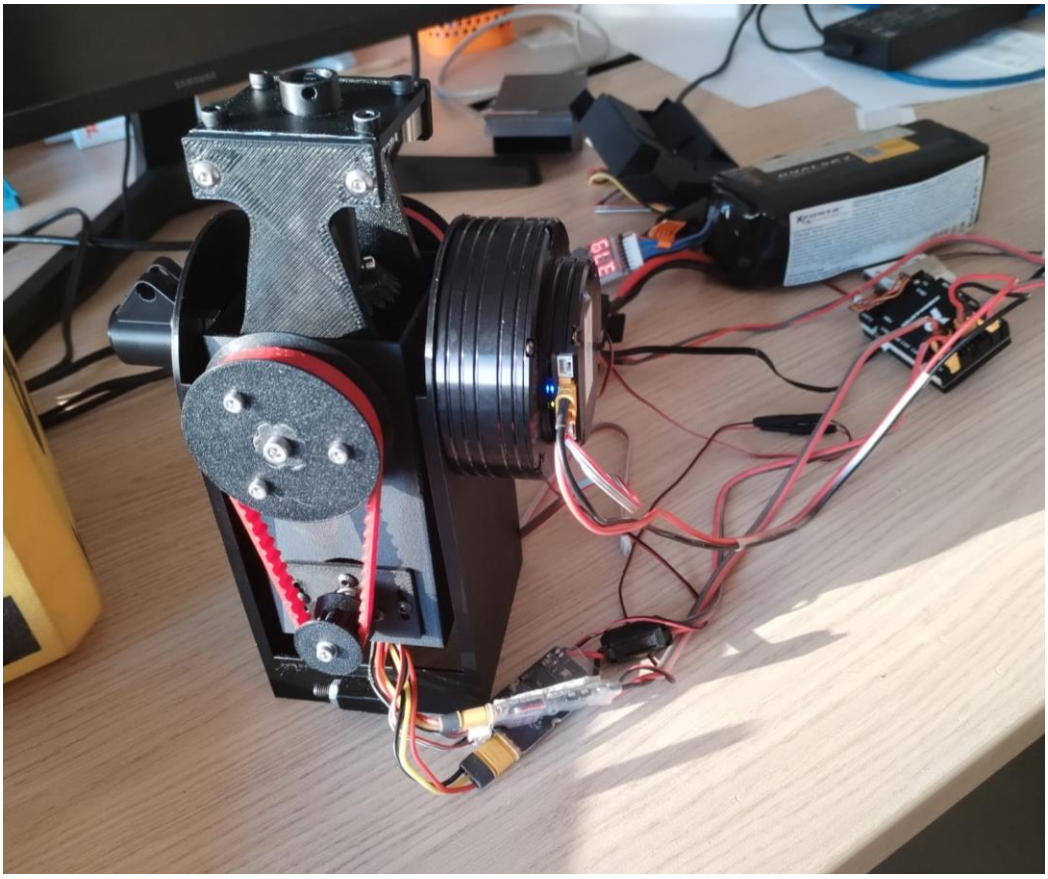

Figure 127 Prototype fully assembled and connected to the microcontroller

Testing revealed snappy, jerky movements, which were initially attributed to overly aggressive initial PID tunings. However, even after adjusting the PID values, the behaviour persisted. The issue was eventually traced back to the input reference system, as the rate changes provided by the operator were scaled too aggressively.

Having corrected this issue and reverted the changes to the PID's parameters, the behavior was much improved, reacting quickly yet smoothly, and properly tracking the operator's commands. Over time, small tracking errors within the differential started to accumulate, the source was tracked down to the 3-D printed belts slipping. After further tensioning, the slippage was reduced but still present, highlighting the need for proper rubber belts and possibly a redesign to include a tensioner for the belts.

During testing, however, another issue presented itself: the radial teeth used to connect the pivot to the pinion were too frail. The pinion being printed vertically meant that the teeth were printed as small, almost dot-like layers stacked upon each other instead of as a part of wider loops that printing the part horizontally would have generated, thus weakening them due to layer separation issues common in 3-D printers.

However, printing them horizontally was not viable; the teeth were replaced with a square insert, which should prove much more resistant owing to a larger surface area and thus greater layer adhesion strength. Accordingly, the pinion gears were redesigned to accommodate this square insert.

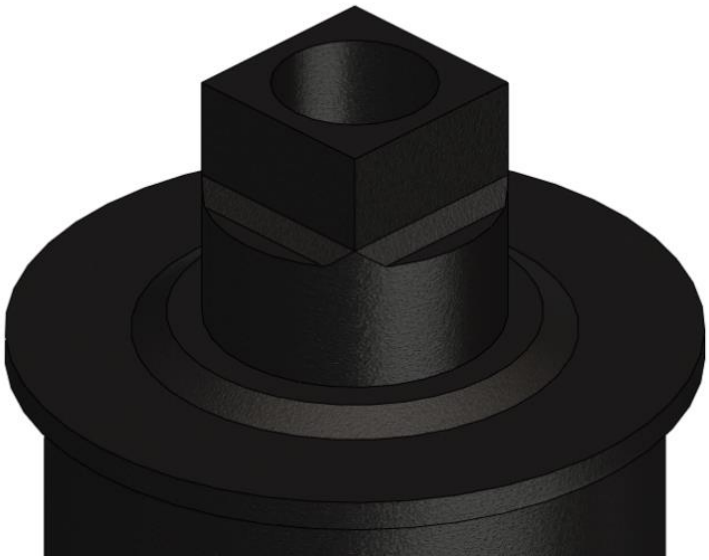

Figure 128 Pivot-Pinion gear connecting insert redesign

9: Conclusions

The objective of the thesis consists of the design of a 6-DOF robotic arm manipulator attachment for the RoboTO student team to participate in the engineer challenge within the RoboMaster competition. After a brief introduction to the requirements and constraints on the manipulator and into the state of the art within the competition, the work proceeds with a discussion on the mechanical design, which was carried out within the SolidWorks environment.

The manipulator is a six-jointed anthropomorphic arm with a spherical wrist and, as such, can be divided into two parts: the shoulder and the wrist. An effort was made to keep both costs and complexity low to fit within the team's budget and requirements, since this arm will most likely need to be disassembled and reassembled for travel multiple times during its lifecycle.

Link 1 is composed of three plates bolted together to form a U shape with a width of 110 mm, while the frames of Link 2 and 3 can be obtained through simple milling operations from different sizes of square aluminum tubing, respectively 100 mm and 80 mm, allowing each subsequent Link to slot in the previous one.

The shoulder structure is thus strictly serial an somewhat repetitive, where things do differ is within the wrist: Link 4 still maintains the philosophy of the previous Links being obtained from a short piece of 60 mm square tubing however to be more compact and use less capable, and thus less expensive, motors the design of the wrist eschews a more standard serial design to instead use a 3-D printed differential to control the final two Links.

The differential consists of a gear carriage and a trio of 90° axis bevel gears: two pinions controlled by two motors and a crown gear which is connected to the end-effector. The whole differential is made up of custom

components; however, the bevel gears themselves are notable as they were created through a custom parametric model to ensure proper meshing of the tooth surfaces.

Joints 2, 3, and 4 are controlled directly by their respective motors since the links are attached directly to the rotors. This design allows the use of the motors themselves not only as actuators but also as parts of the joints themselves, the joints do not rely solely on these motors to function as the sole physical connection between links, introducing additional supports to reduce the strain on the motor bearings in the form of a shaft, clamped to the successive link and passing through a ball bearing or baffle, to act as an additional mechanical connection and support.

The remaining Joints, namely 1, 5, and 6, due to space issues, are not actuated directly; toothed pulleys and belts are used instead, the pulleys themselves are also 3-D printed and modeled using a parametric model, which, differently from the bevel gears' one, utilized linear regression from the standard sizes to obtain the parameters needed for the model.

Once the whole arm was modeled and assembled within the SolidWorks environment, it was used to estimate the various joint limits and, after a cursory discussion on kinematics, to calculate the DH parameters.

The work then continues in MATLAB and its extension Simulink: through the use of the Simscape Multibody Link extension, the SolidWorks model can be used to automatically generate a plant to use within Simulink. The generated plant was slightly modified to include the relationships that the extension could not handle, namely the pulley belts and the bevel gears, to allow the input of torques to the Joints and the output of their respective position and velocities, and a small amount of friction in the joints.

A general discussion on manipulator control approaches is followed by a discussion on how to control our manipulator: since it possesses a spherical

wrist, control can be split between control of the shoulder and control of the wrist. Additional issues to be discussed are due to the particular method of control of the arm: since the manipulator will be controlled in real time by an operator using a remote controller with limited inputs the reference signals need to be adjusted accordingly to be intuitive and usable in real time, furthermore, as the robot is controlled in real time there do not exist pre-planned trajectories and thus the use of some of the kinematic inversion schemas is restricted leaving only the Jacobian transpose method as usable among the more common schemas.

For the references, while the wrist receives as reference the desired attitude of the end effector, the shoulder is another story, receiving as reference the position of the wrist center instead of the end effector. This was done to simplify the control approach and render the operator controls more intuitive.

Two main ways to provide a reference signal to the shoulder were attempted: the first was by providing the Cartesian coordinates of the wrist center. This method was still discarded due to the kinematic inversion method used proving incapable of handling joint limits; thus, a cylindrical reference system was employed. Cylindrical coordinates can be characterized as a hybrid reference where a portion (namely, the azimuth) can be interpreted as already being part of joint space, while the rest (radial distance and height) are given in the task space. This second implementation allows us to treat Links 2 and 3 as a two-jointed planar arm, significantly reducing the instances of joint lock.

For both reference methods, the shoulder was controlled via decentralized joint space controllers, which in turn employed a position-velocity feedback PID control schema.

The wrist reference, meanwhile, shares some similarities to the cylindrical approach for the shoulder, as the reference pitch can be directly connected to Joint 4, while yaw and roll, which belong to the task space, need to be

translated to joint space to be used as signals for the actuators controlling the differential.

Similarly to the shoulder, the actuators were controlled via decentralized controllers; however, while for the pitch and thus Joint 4, this was done in joint space, the pitch and roll controllers were constructed as task space controllers.

Having developed a working control scheme, the next step was translating it from MATLAB Script to a C code base to enable it's use on an embedded controller, the code was written using Atom modifying and reusing much of the code already present within the code base of the RoboTO team however some parts of the code, namely the matrix operations necessary to do kinematics, were written wholesale after the code generation of MATLAB proved insufficient and inflexible.

The thesis concludes with a short prototyping phase, where the wrist was almost wholly 3-D printed and tested to check whether the control scheme functions properly and to properly adjust the PID parameters.

Possible future developments of the mechanical design include, but are not limited to, expanding the range of options for the end effector, improving mobility of the arm by modifying the various link geometries, including mechanical advantages such as gear reductions where not already present to improve capabilities and lower motor requirements. For the control schema instead, future developments may include the creation of a trajectory generator to allow the use of alternate inversion methods, testing whether control of the shoulder via the position of the end effector instead of the wrist center's is a viable control method, the creation of preset poses to allow a quick transition of the manipulator with minimal operator input, the creation of a calibration routine to position the differential more precisely instead of relying on a positioner and more.

10: Bibliography

[1] ‘RoboMaster Open Source database’. Accessed: Oct. 03, 2025. [Online]. Available: https://search.scutbot.cn/search/?query=engineer

[2] ‘RS PRO, Angular contact ball bearing’. Accessed: Oct. 03, 2025. [Online]. Available: https://it.rs-online.com/web/p/cuscinetti-a-sfera/0291632

[3] ‘RS Pro Angular bracket’. Accessed: Oct. 03, 2025. [Online]. Available: https://it.rs-online.com/web/p/accessori-per-morsettiere-su-guida-din/0606866?gb=a

[4] ‘RS PRO Deep groove ball bearing’. Accessed: Oct. 03, 2025. [Online]. Available: https://it.rs-online.com/web/p/cuscinetti-a-sfera/2346895

[5] ‘Rose+Krieger Shaft holder’. Accessed: Oct. 03, 2025. [Online]. Available: https://it.rs-online.com/web/p/piedini-e-accessori-per-ruote/5097661

[6] ‘AK60-6 V3.0’, CubeMars. Accessed: Oct. 03, 2025. [Online]. Available: https://www.cubemars.com/product/ak60-6-v3-0-kv80-robotic-actuator.html

[7] ‘AK80-64 KV80’. Accessed: Oct. 03, 2025. [Online]. Available: https://www.cubemars.com/product/ak80-64-kv80-robotic-actuator.html

[8] ‘AK70-10’. Accessed: Oct. 03, 2025. [Online]. Available: https://www.cubemars.com/product/ak70-10-kv100-robotic-actuator.html

[9] ‘iglidur® Flanged baffle’. Accessed: Oct. 03, 2025. [Online]. Available: https://www.igus.it/iglidur-ibh/boccole-flangiate/product-details/iglidur-g-m

[10] Richard Kokstein, *Modeling Parametric Timing Pulley in SolidWorks | ISO 5294 | Part 2: Modeling w/ Download*, (Apr. 11, 2021). Accessed: Oct. 03, 2025. [Online Video]. Available: https://www.youtube.com/watch?v=4EKMdqsQmik

[11] B. Piombo and G. Jacazio, *Meccanica applicata alle macchine*. Torino: Levrotto e Bella, 1977.

[12] ‘1203 Series Block Mount (1-1)’, goBILDA®. Accessed: Oct. 03, 2025. [Online]. Available: https://www.gobilda.com/1203-series-block-mount-1-1-4-pack/

[13] ‘NMB Flanged deep groove ball bearing’. Accessed: Oct. 03, 2025. [Online]. Available: https://it.rs-online.com/web/p/cuscinetti-a-sfera/0540299

[14] ‘M2006 P36’. Accessed: Oct. 03, 2025. [Online]. Available: https://www.robomaster.com/en-US/products/components/detail/1277

[15] C. Basso, ‘Design and control of a four degrees of freedom manipulator for a mobile robot for automatic wine sampling in a winery.’, Master’s Thesis, Politecnico di Torino, 2025. Accessed: Oct. 04, 2025. [Online]. Available: https://webthesis.biblio.polito.it/35511/

[16] M. W. Spong, S. Hutchinson, and M. Vidyasagar, *Robot Modeling and Control*. Wiley, 2005.

[17] J. J. Craig, *Introduction to Robotics: Mechanics and Control*. Pearson, 2017.