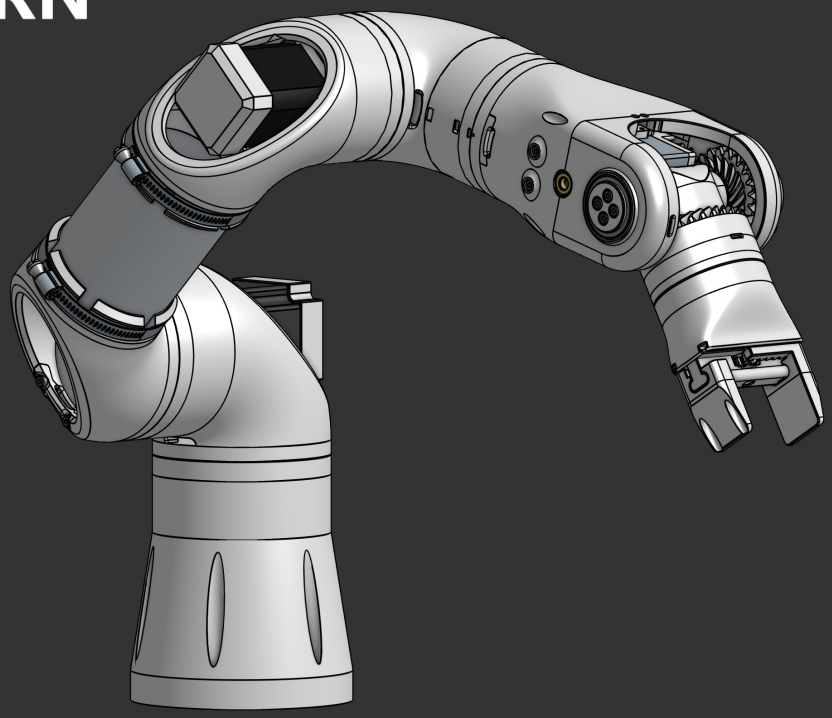
KANTON
LUZERN



Developing a 6 DoF Robotic Arm

Matura Thesis

Andrin Winzap and Remi Gerber, G22l

26. September 2025

Advisor: Pascal Basler, Department of Physics

Kantonsschule
Alpenquai Luzern | ksalpenquai.lu.ch

Abstract This thesis documents the development of a 6 Degree of Freedom robotic arm built from scratch, including the mechanical and electronic design, kinematic analysis and software control. Mechanically, the arm integrates custom eccentric drives, a differential wrist and a toolchanger enabling rapid end-effector swaps. Individual joints are controlled by dedicated microcontrollers running a closed-loop feedback system for the motors and connected to a PC through a custom printed circuit board. After completing the physical structure, we carried out a kinematic analysis to model and control the arm's movements. Using Denavit–Hartenberg parameters, we derived the forward kinematics and implemented an analytical inverse kinematics solution for reliable trajectory planning. On the software side, we developed a control stack with real-time visualization and a Python programming interface for user interaction. During integration, we encountered electromagnetic interference on sensor connections, which we addressed by redesigning the cable harness, separating signal and power lines, and adopting shielded wiring. With these refinements, the final platform successfully demonstrated coordinated motion and proved to be a functional and educational platform. The product prompts further exploration in the form of countless options for expansion, adaptation and refinement such as creating a broad range of tools, integration with computer vision and machine learning systems and adopting a more robust sensor communication protocol to mitigate interference issues.

Contents

1. Preface

The field of robotics long fascinated both of us, not only because of the technical complexity but also due to the broad spectrum of possible applications. We chose to take on the challenge of designing and building a functional 6 Degree of Freedom (DoF) robotic arm as our Matura Thesis. This project brought together many fields of knowledge, including mechanics, electronics, software engineering and mathematics, and required us to apply theoretical concepts and practical skills in all of those.

We aimed to gain a deeper understanding of the fundamental principles behind robotic motion by building a system that would demonstrate these ideas. We were enticed by the technical challenges this project presented and deemed it suitable to allow us to make use of and improve on our preexisting knowledge in the involved fields.

Throughout this work, we faced challenges that required us to problem-solve, adapt, and learn far beyond what we initially anticipated. These moments proved to be the most rewarding once completed and helped us grow as students by teaching us about our own way to deal with stress and frustration.

This paper documents our journey from idea, the theoretical background of kinematics, through the design and fabrication of hardware, to the implementation of control software. We hope it provides insight into the process of developing a robotic system from the ground up, and that it encourages others to explore robotics as an exciting combination of physics, engineering and computer science.

2. Acknowledgements

We would like to express our sincere gratitude to all those who contributed to the successful completion of this thesis:

- Pascal Basler, our advisor, for his supervision during the project.
- LABOR Luzern for graciously allowing us to make use of their facilities during the project.
- Emrus Sezairi for his assistance with the fabrication of the metal components.
- Lucas Renfer for his expert advice at every stage of the project.
- Benjamin Kirn for dedicating extensive time to reviewing and refining our printed circuit board design.
- Jonas Oehen for his insight into the design of the eccentric drive.
- Sandro Covo for his thorough review and valuable input.
- Christian Schütz for carefully reviewing the manuscript.

3. Introduction

Robotics has become a central field of modern technology, with countless applications. At the heart of many robotic systems is the manipulator: a machine designed to reproduce the dexterity of the human arm. Among the different architectures, robotic arms with 6 DoFs are especially interesting, as they provide complete control over both the position and orientation of a tool in three-dimensional space.

For this thesis we set ourselves the goal of designing and building a 6 DoF robotic arm from the ground up. We limited the scope of the project: While industrial manipulators are often equipped with advanced features such as computer vision or artificial intelligence, our arm focuses purely on motion in joint and Cartesian space and the integration of a mechanical toolchanger. This allowed us to concentrate on the fundamental aspects of robotic motion while keeping the system achievable within the framework of a Matura Thesis.

The theoretical foundation of our work lies in kinematics, which describes the relationship between joint movements and end-effector motion. By using mathematical tools such as Denavit–Hartenberg parameters, forward kinematics, and inverse kinematics, we were able to model the motion of the arm and control its position and orientation precisely.

First, we researched existing arm architectures and compared different actuator and gearbox concepts. From this, we developed and fabricated our own mechanical design, followed by the design and implementation of the electronics and kinematic analysis. In parallel, we prepared a software stack using Robot Operating System 2 (ROS 2), which allowed us to connect the hardware with higher-level motion control. Finally, we combined all components into a complete system.

4. Theory

This section provides an overview of the theoretical foundation for the robotic arm. First, the Denavit–Hartenberg (DH) convention, which systematically describes the geometry of each link in the manipulator, is explained. Next, the computation required for coordinated motion of the robotic arm is derived, including forward kinematics to calculate the end-effector pose from joint angles, and inverse kinematics to determine the joint angles required to reach a given end-effector position and orientation.

4.1. Denavit–Hartenberg Parameters

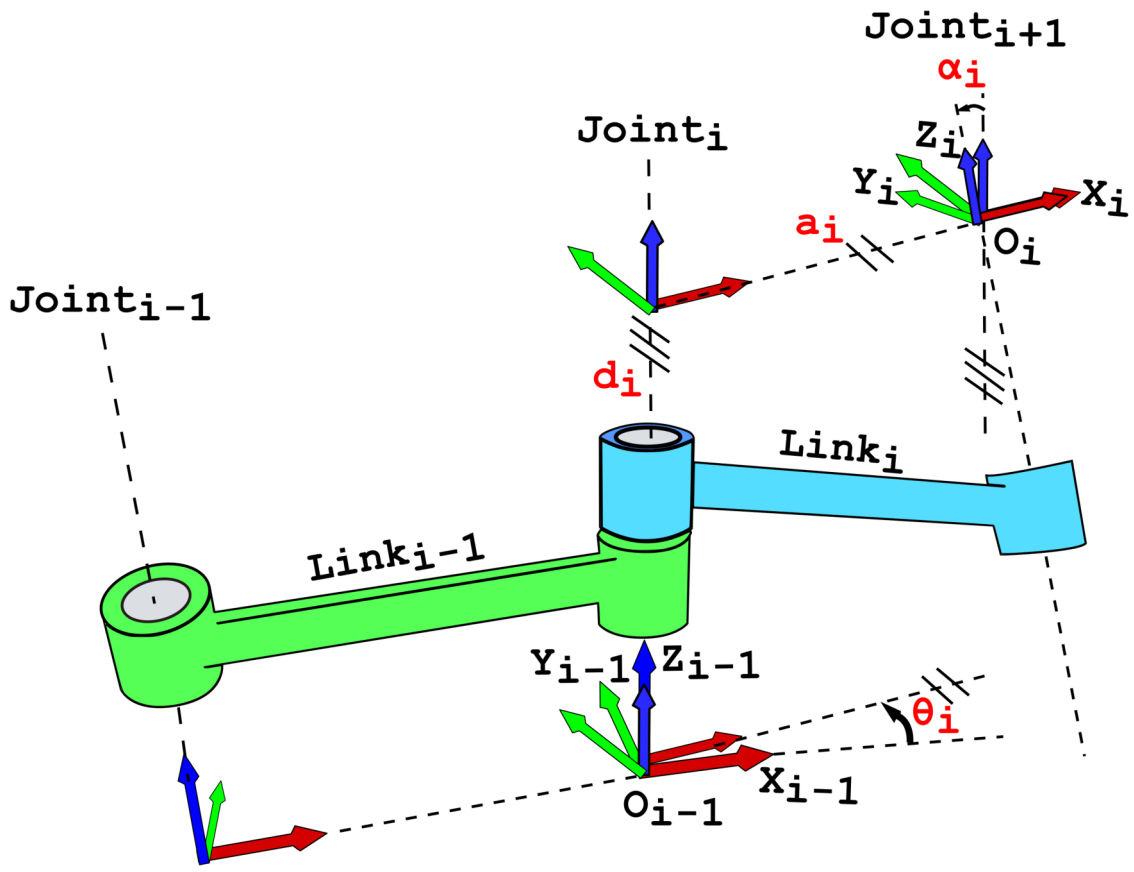


Figure 1: Classic Denavit–Hartenberg convention [1]

A robotic manipulator is composed of links and joints. Links are the rigid segments that make up the structure of the robot, while joints are the movable connections between the links that allow relative motion. Each joint's configuration affects the position and orientation of all subsequent links, forming the robot's kinematic chain.

The DH convention is a widely used method to describe the kinematics of robotic systems. As illustrated in Figure 1, each link in the kinematic chain is described by four parameters [2]:

a: link length

α: link twist

d: distance between links

θ: angle between links

The DH parameters of our robotic arm can be seen in Table 1. For an overview of the joint structure refer to the technical drawing (Appendix E).

Table 1: Denavit–Hartenberg parameters of the robotic arm (extracted from the 3D Model)

Joint	$\boldsymbol{\theta_i}$ (rad)	$\boldsymbol{d_i}$ (m)	$\boldsymbol{\alpha_i}$ (rad)	$\boldsymbol{a_i}$ (m)
1	θ_1	0.18200	$-\frac{\pi}{2}$	0
2	$\theta_2 - \frac{\pi}{2}$	0.01350	0	0.2
3	$\theta_3 + \frac{\pi}{2}$	0	$\frac{\pi}{2}$	0
4	θ_4	0.18850	$-\frac{\pi}{2}$	0
5	θ_5	0	$\frac{\pi}{2}$	0
6	θ_6	0.05813	0	0

4.2. Forward Kinematics

Forward kinematics is the process of computing the position and orientation of a robot's end-effector using the joint angles. The links of the robot can be described using a Homogeneous Transformation Matrix (HTM), which is a 4×4 matrix representing both rotation and translation. Using the DH parameters it can easily be computed.

The HTM matrix for a given link i is:

$$T_{i+1}^{i} = \begin{bmatrix} \cos\theta_i & -\sin\theta_i \cos\alpha_i & \sin\theta_i \sin\alpha_i & a_i \cos\theta_i \\ \sin\theta_i & \cos\theta_i \cos\alpha_i & -\cos\theta_i \sin\alpha_i & a_i \sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1}$$

A general kinematic chain can be described by:

$$T_b^a = \begin{bmatrix} R_b^a & P_b^a \\ 0 & 1 \end{bmatrix} = \prod_{i=a+1}^{b} T_i^{i-1}, \qquad b > a \tag{2}$$

The specific transformations are:

$$T_6^0 = \begin{bmatrix} R_6^0 & P_6^0 \\ 0 & 1 \end{bmatrix} \qquad \text{end-effector transform} \tag{3}$$

$$T_3^0 = \begin{bmatrix} R_3^0 & P_3^0 \\ 0 & 1 \end{bmatrix} \qquad \text{wrist center} \tag{4}$$

$$T_6^3 = \begin{bmatrix} R_6^3 & P_6^3 \\ 0 & 1 \end{bmatrix} \qquad \text{wrist rotation} \tag{5}$$

4.3. Inverse Kinematics

Inverse Kinematics, as the name suggests, is the inverse of forward kinematics. It is the process of computing the joint angles to reach a given point in the Cartesian space. Unlike with forward kinematics, there is not just one single solution. In fact, with our robot, there are 8 joint configurations for any given end-effector pose, except at singularities.

There are numerous different approaches to inverse kinematics. Analytical approaches use the geometry of the robot to derive formulae for the exact solutions. Numerical approaches, such as the Jacobian Transpose, Pseudoinverse and Damped Least Squares methods [3], use the current joint configuration as a starting point and attempt to iteratively reduce the error in the end-effector pose. This is computationally expensive and only results in an approximation. Furthermore, many of the approaches suffer from poor numerical stability around singularities.

Since the last three axes of our manipulator intersect, Pieper's criterion [4] guarantees the existence of an analytical inverse kinematics solution. We therefore adopted an analytical approach derived from Asif and Webb [2].

In order to obtain the joint configurations for a given end-effector pose we first compute the wrist center position. Since we know the orientation of the end-effector we can compute the wrist center position W by subtracting the joint offset d_6.

$$W = P_6^0 - d_6 \cdot R_6^0 \hat{z} \tag{6}$$

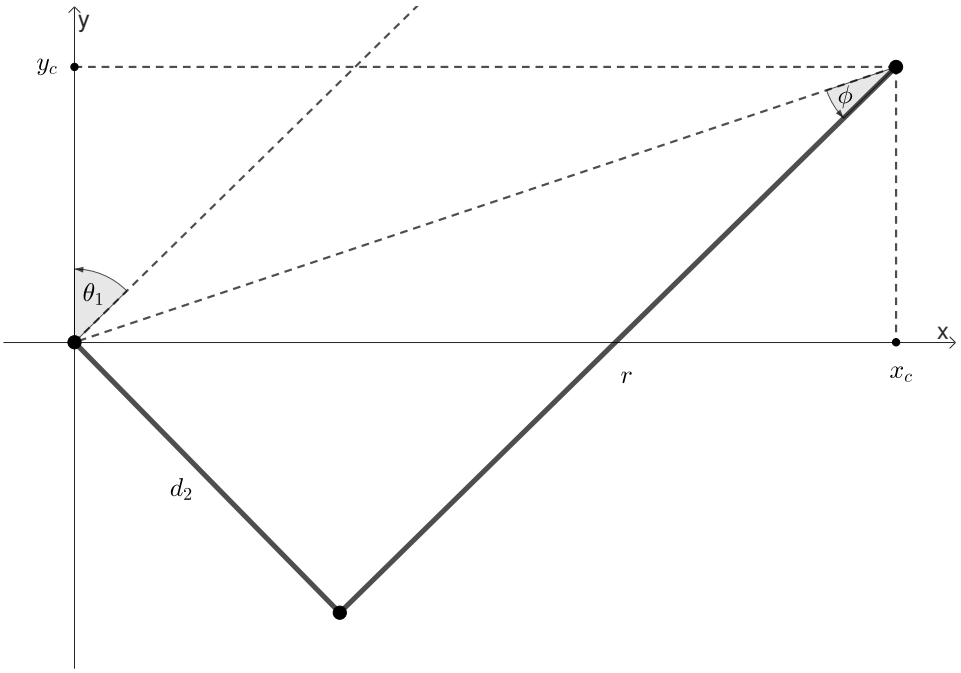


Figure 2: Projection of wrist center onto the XY plane

The projection of the wrist center onto the XY plane, shown in Figure 2, provides the basis for computing the first joint angle θ_1. We first define the auxiliary angle ϕ, which accounts for the shoulder offset d_2:

$$\phi = \text{atan2}\left(d_2, \sqrt{x_c^2 + y_c^2 - d_2^2}\right). \tag{7}$$

From the coordinates x_c and y_c, we then obtain two possible values for θ_1:

$$\theta_1 = \text{atan2}(y_c, x_c) - \phi \tag{8}$$
$$\theta_1 = \text{atan2}(y_c, x_c) + \pi + \phi \tag{9}$$

The second solution corresponds to the configuration with the shoulder flipped.

Figure 3 illustrates the projection of the wrist center onto the plane spanned by links 2 and 3. Using this geometry, θ_3 can be derived via the law of cosines:

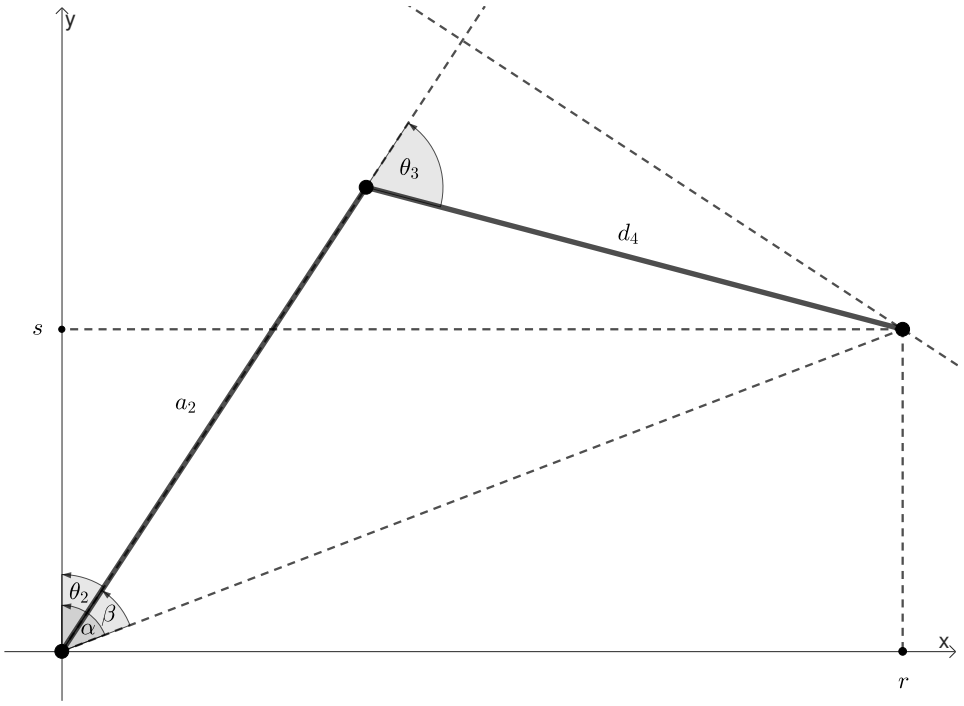


Figure 3: Projection of wrist center onto the plane formed by link 2 and 3

$$r = \sqrt{x_c^2 + y_c^2 - d_2^2} \tag{10}$$

$$s = z_c - d_1 \tag{11}$$

$$\cos\theta_3 = \frac{r^2 + s^2 - a_2^2 - d_4^2}{2a_2 d_4} \tag{12}$$

$$\sin\theta_3 = \sqrt{1 - \cos^2\theta_3} \tag{13}$$

Let $\gamma = \cos\theta_3$, then:

$$\theta_3 = atan2(\gamma, \pm\sqrt{1-\gamma^2}) \tag{14}$$

θ_2 can then be computed using θ_3:

$$\alpha = atan2(r, s) \tag{15}$$

$$\beta = atan2(d_4 \sin\theta_3, a_2 + d_4 \cos\theta_3) \tag{16}$$

$$\theta_2 = \alpha - \beta \tag{17}$$

The wrist can be represented by a ZYZ Euler transformation[1], therefore the remaining three angles can be extracted from the previously derived matrix R_6^3 as follows:

$$c_i = \cos\theta_i \tag{18}$$
$$s_i = \sin\theta_i \tag{19}$$

$$R_6^3 = \begin{bmatrix} c_4c_5c_6 - s_4s_6 & -c_4c_5s_6 - c_6s_4 & c_4s_5 \\ c_4s_6 + c_5c_6s_4 & c_4c_6 - c_5s_4s_6 & s_4s_5 \\ -c_6s_5 & s_5s_6 & c_5 \end{bmatrix} \tag{20}$$

θ_5 can be computed using r_{33}:

$$\theta_5 = \text{atan2}(r_{33}, \pm\sqrt{1 - {r_{33}}^2}) \tag{21}$$

θ_4 can be computed using r_{13} and r_{23}:

$$\tan\theta_4 = \frac{r_{23}}{r_{13}} = \frac{\sin\theta_4 \sin\theta_5}{\sin\theta_5 \cos\theta_4} \tag{22}$$
$$\theta_4 = \text{atan2}(r_{13}, r_{23}) \tag{23}$$

θ_6 can be computed using $-r_{31}$ and r_{32}:

$$\tan\theta_6 = \frac{r_{32}}{-r_{31}} = \frac{\sin\theta_5 \sin\theta_6}{\sin\theta_5 \cos\theta_6} \tag{24}$$
$$\theta_6 = \arctan 2(-r_{31}, r_{32}) \tag{25}$$

A summary of the derived formulae can be seen in Table 2.

[1] A ZYZ Euler transformation describes the orientation of a rigid body as three sequential rotations about the z, y, and z axes of a coordinate system [5].

Table 2: Summary of the inverse kinematics formulae

Angle	**Computation**
θ_1	$\text{atan2}(y_c, x_c) - \phi, \quad \text{atan2}(y_c, x_c) + \pi + \phi$
θ_2	$\text{atan2}(r, s) - \text{atan2}(d_4 \sin\theta_3, a_2 + d_4 \cos\theta_3)$
θ_3	$\text{atan2}\left(\gamma, \sqrt{1-\gamma^2}\right), \quad \text{atan2}\left(\gamma, -\sqrt{1-\gamma^2}\right)$
θ_4	$\text{atan2}(r_{13}, r_{23})$
θ_5	$\text{atan2}\left(r_{33}, \sqrt{1-{r_{33}}^2}\right), \quad \text{atan2}\left(r_{33}, -\sqrt{1-{r_{33}}^2}\right)$
θ_6	$\text{atan2}(-r_{31}, r_{32})$

The final solution is determined by first eliminating all solutions which are not within the physical limits of the robot, computing the distance in Euclidean joint space for each of the remaining solutions and finally choosing the solution which requires the smallest overall joint movement.

5. Hardware

This chapter provides a detailed overview of the hardware components and design considerations of our robotic arm. It covers the mechanical structure, actuation mechanisms, sensors, and electronics that collectively enable precise, reliable operation.

5.1. Mechanics

In this section, we describe the mechanical design, including the inspiration for the architecture, computer-aided design workflow, fabrication methods, and the selection of actuators and reduction mechanisms.

5.1.1. Inspiration

Our robotic arm architecture is inspired by the works of Universal Robots [6], Robotastic [7], Arctos Robotics [8], Crnjak [9, 10], and ATI Industrial Automation [11]. Additionally, the concept of the eccentric drive is derived from the designs presented by Mishin [12] and Fritsch et al. [13].

5.1.2. Computer-Aided Design

Computer-Aided Design (CAD) was a critical component of this project. It enabled us to quickly translate our conceptual ideas into a detailed 3D model (Figure 4). Furthermore it allowed us to identify and resolve many potential design issues early, well before fabrication. Using CAD, we not only created the custom components we later manufactured ourselves, but also incorporated many commercial parts into the model. This integration provided a comprehensive understanding of the robot's mechanical structure and facilitated efficient planning of the assembly process. The ability to visualize the design in 3D streamlined development and allowed us to align our individual ideas into a cohesive final design.

Initially, we used Autodesk Fusion 360, but later transitioned to Onshape for several reasons. Being browser-based, Onshape simplified our workflow, particularly since both of us use Linux-based systems, which Fusion 360 does not natively support. In addition, its online collaboration features and built-in version control helped us coordinate more effectively and avoid design conflicts.

The technical drawing of the robotic arm can be seen in Appendix E.

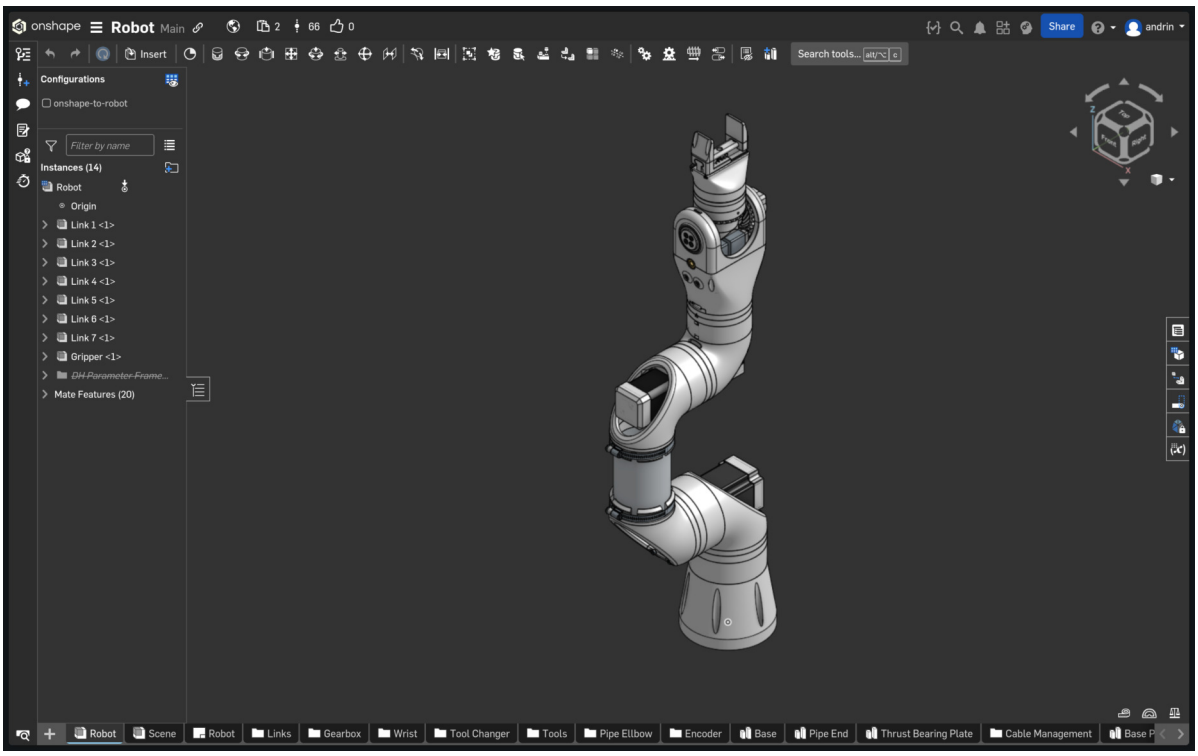


Figure 4: Computer-Aided Design model (Onshape)

5.1.3. Fabrication

Fused Deposition Modeling (FDM) 3D printing has become a highly versatile fabrication technique over the last decade. Its main advantages include rapid design iteration, low material and equipment costs, and the ability to produce complex shapes and geometries [14]. Given that we already owned 3D printers and were familiar with the process, FDM quickly emerged as the most practical method for producing the majority of our custom parts (Figure 5).

We selected Polylactic Acid (PLA) as the printing material because it is affordable and well suited for rapid prototyping. Its stiffness also makes it a strong candidate for components such as 3D-printed gears. The primary limitation of PLA for our application is its relatively low glass transition temperature of around 60 °C [15]. To ensure reliable and durable screw connections, we employed heat-set brass inserts for fastening.

Other than the 3D printed components, we also incorporated a few additional custom parts, such as the motor couplings or the aluminum pipe between the second and third joint. These parts were purchased commercially and then modified by the physics assistant at our school to meet the specific dimensional and functional requirements of our design.

Figure 5: 3D printed prototype parts

Furthermore, we used a laser cutter to manufacture the board on which the robot and its electronics are mounted.

5.1.4. Precise Actuation

We evaluated several motor technologies for the first four joints, focusing primarily on brushless Direct Current (DC) motors and stepper motors. Each type offered distinct advantages and trade-offs in terms of torque, precision, weight, and driver requirements.

Brushless DC motors provide high power-to-weight ratios and are generally lighter than stepper motors of comparable size [16]. We tested a set of these motors with basic drivers to assess their suitability. Our testing showed that achieving accurate operation would have required more expensive drivers. Considering the additional cost and complexity, the benefits of brushless DC motors did not justify their use in our design.

Stepper motors operate in discrete angular steps, allowing precise control across their operating range with relatively inexpensive drivers [16]. This combination of accuracy

and simplicity made them the preferred choice for the first four joints. The load requirements vary along the arm: the third and fourth joints experience lower forces than the first and second.

Early testing showed that NEMA 17 motors might not provide sufficient torque in the heavier joints. To address this, the final configuration uses two NEMA 23 motors with holding torques of 1.8 N m [17] and 3 N m [18] for the first and second joints, and two upgraded NEMA 17 motors rated at 0.8 N m [19, p. 4] for the lighter third and fourth joints. This increase in torque over the 0.4 N m NEMA 17 motors we first considered [19, p. 4] ensures all joints can be actuated reliably. A NEMA 17 motor can be seen in Figure 6.

Since stepper motors generate a considerable amount of heat when continuously energized to maintain holding torque, keeping them powered while the robot is idle not only increases energy consumption but also risks thermal stress on the motor windings and nearby components. We thus avoided continuous operation of the motors.

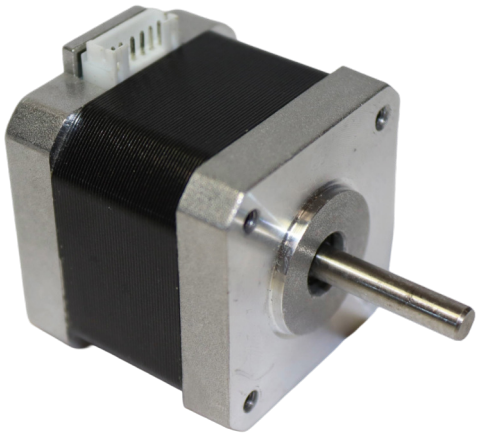


Figure 6: NEMA 17 stepper motor

5.1.5. Compact Actuation

For the last two joints, we evaluated stepper motors again, comparing them with geared, brushed DC motors we had previously used in other projects. Considering weight and size constraints, the lighter and stronger DC motors emerged as the better option.

The first prototypes used motors with a plastic gearbox and a reduction ratio of 1:45, achieving a maximum speed of 120 rpm. These were later replaced by a drop-in upgrade featuring a metal gearbox with a higher reduction ratio of 1:90. This change reduced speed, increased torque, and improved reliability thanks to the metal gearing. Eventually, we adopted an all-metal version with a stronger motor, worm drive gearing, and higher-quality construction (Figure 7).

We tested various reduction ratios and determined that 40 rpm offered the best balance between precision and speed. A faster 90 rpm variant, which we initially hoped would improve wrist responsiveness, introduced significantly more backlash from the motor's internal gearbox. This negatively affected overall precision, so we opted to retain the 40 rpm configuration.

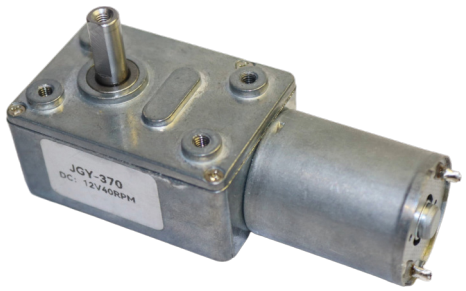


Figure 7: JGY-370 geared direct current motor

5.1.6. Position Feedback

To obtain accurate data of the joint position, we employed magnetic rotary position sensors. These detect the rotation of the magnetic field of a permanent magnet positioned on the rotating axis, eliminating the need for any physical contact between the sensor and moving parts [20, p. 1]. We chose a widely available module based on the popular AS5600 (Figure 8), boasting a 12-bit output enabling high precision measuring and convenient to integrate as its supports the Inter-Integrated Circuit (I^2C) communication protocol [20, p. 1].

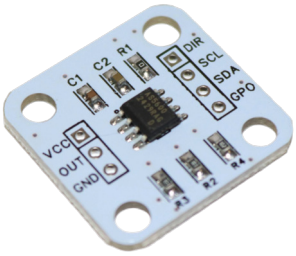


Figure 8: AS5600 magnetic angle encoder module

5.1.7. Homing Reference

Since the encoders monitoring the joint position need a point of reference at startup to determine a known home position, we looked for suitable homing sensors. It was clear to us that it would be beneficial to avoid having to establish physical contact between part and switch. Consequently, a sensor relying on the hall effect would best be used here. When a certain threshold of magnetic flux density is reached, these switches turn on, so by integrating a small permanent magnet into the moving part, it can be detected without direct contact [21, p. 2]. We chose the A3144 (Figure 9) as we had already used it in previous projects and could integrate it without any delay due to order times.

Figure 9: A3144 hall effect switch

5.1.8. Reduction Mechanism

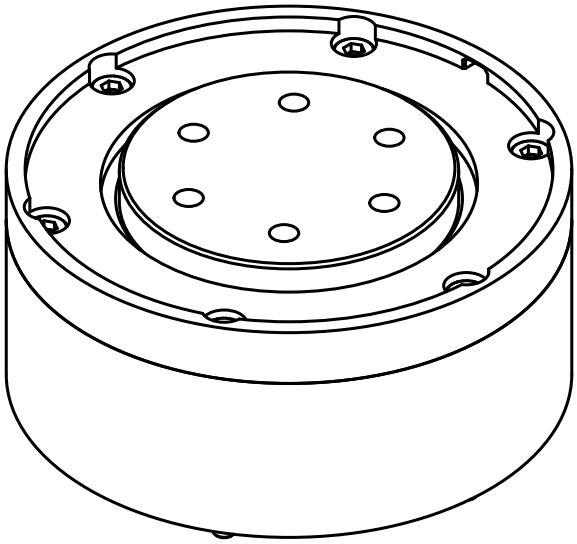


Figure 10: 75 mm variant of the gearbox

Working Principle There are several reduction mechanisms commonly used in robotics, such as strain wave gearing [22], epicyclic gearing [23], and cycloidal drives [24]. For our design, we chose a variant of eccentric drives, as described by Fritsch et al. [13].

This drive operates by pressing rolling elements into the profile of an outer gear through the motion of a crank. The gear profile is designed so that the rolling elements are always radially constrained between the crank and the gear, ensuring continuous engagement along their path. As the crank drives the rolling elements, they are forced onto a ramp-like contour that converts the radial input force into a tangential output force, enabling power transmission without relying on friction. An illustration of the mechanism can be seen in Figure 11.

The rolling elements are guided by a cage that maintains even spacing around the axis of rotation. The cage also moves the rolling elements that are not directly driven by the crank along their trajectory.

For speed reduction, the crank serves as the input, while either the outer gear or the cage can be used as the output, with the remaining component held fixed. By offsetting the position of the rolling elements in the cage, multiple eccentricities can be used with the same gear. Employing a crank with more than one eccentricity provides the additional benefit of balancing static forces and reducing dynamic loads [13].

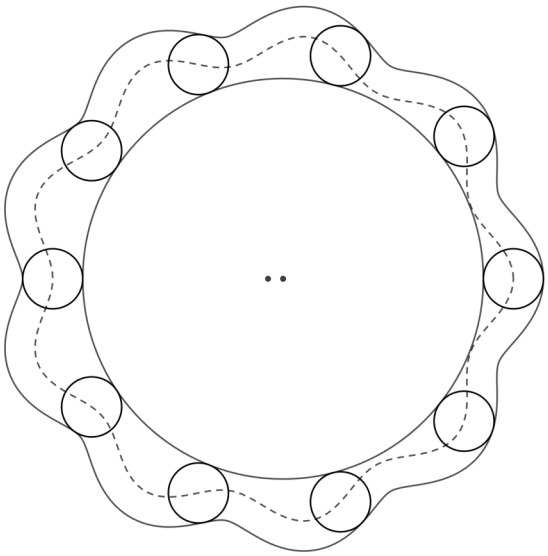


Figure 11: Illustration of the eccentric drive mechanism

Mathematical Model and CAD Implementation To implement this mechanism in our CAD design, we developed a custom Onshape FeatureScript that generates a 2D sketch of the gear profile using the geometric model in Figure 12.

The distance to the center is computed as:

$$c = e\cos\beta + \sqrt{(a+b)^2 - e^2\sin^2\beta} \tag{26}$$

The outward-pointing normal vector is obtained by differentiating the profile to get the tangent vector and taking the perpendicular vector:

$$\vec{n} = \frac{c}{z} \cdot \begin{pmatrix} \cos\frac{1}{z}\beta \\ \sin\frac{1}{z}\beta \end{pmatrix} + e\sin\beta \left(1 + \frac{e\cos\beta}{\sqrt{(a+b)^2 - e^2\sin^2\beta}}\right) \cdot \begin{pmatrix} -\sin\frac{1}{z}\beta \\ \cos\frac{1}{z}\beta \end{pmatrix} \tag{27}$$

The vector is normalized:

$$\vec{N} = \frac{\vec{n}}{|\vec{n}|} \tag{28}$$

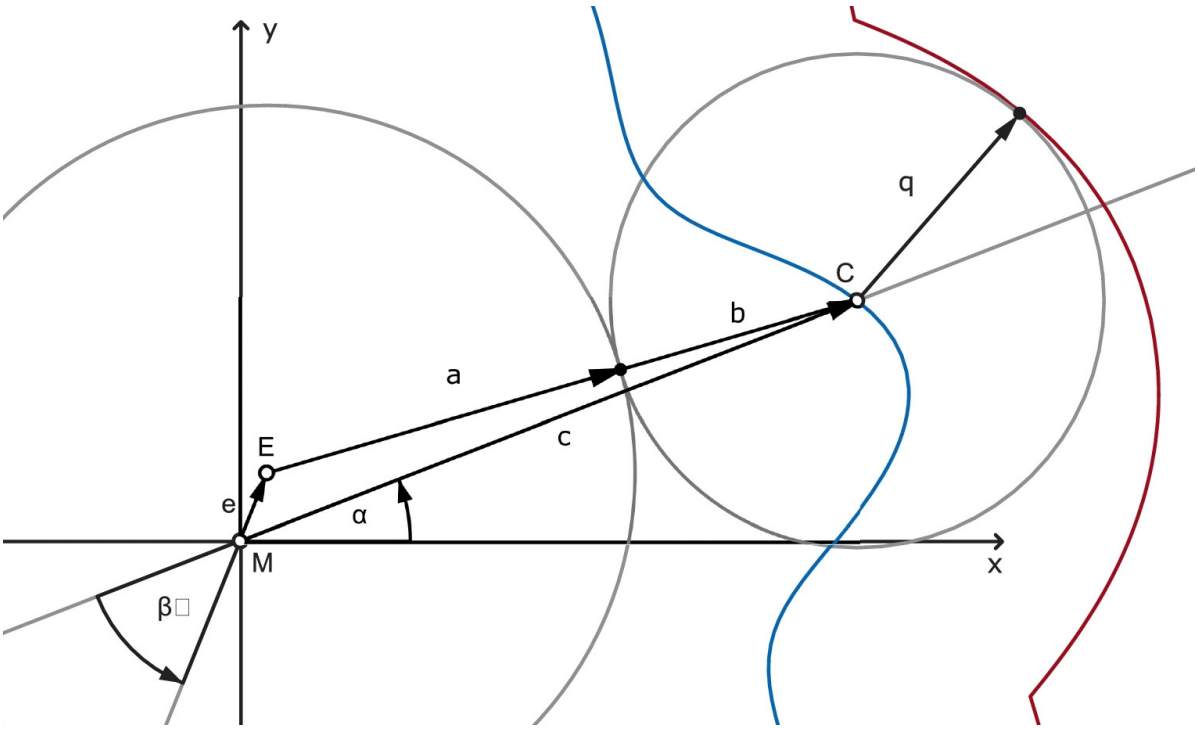


Figure 12: Mathematical model of the gear geometry [13]

The corresponding point on the outer profile is then be computed as follows:

$$\vec{r} = \vec{c} + b \cdot \vec{N} \tag{29}$$

This process is repeated for each point on a single tooth, and the final profile is generated by rotating copies of the tooth and creating a continuous spline. If the profile self-intersects (Figure 13), the spline is segmented and clipped. This results in potential play [13].

Design and Prototyping We developed two gearbox sizes to accommodate different stepper motors: 90 mm for NEMA 23 and 75 mm for NEMA 17. The latter can be seen in Figure 10. For a description of the individual parts refer to the technical drawing (Appendix E). The crank incorporates two eccentricities offset by 180°, each fitted with a bearing to reduce friction with the rolling elements. The cage is constrained by a single bearing, and a thrust bearing (Figure 14) was added to counteract the perpendicular forces within the gearbox.

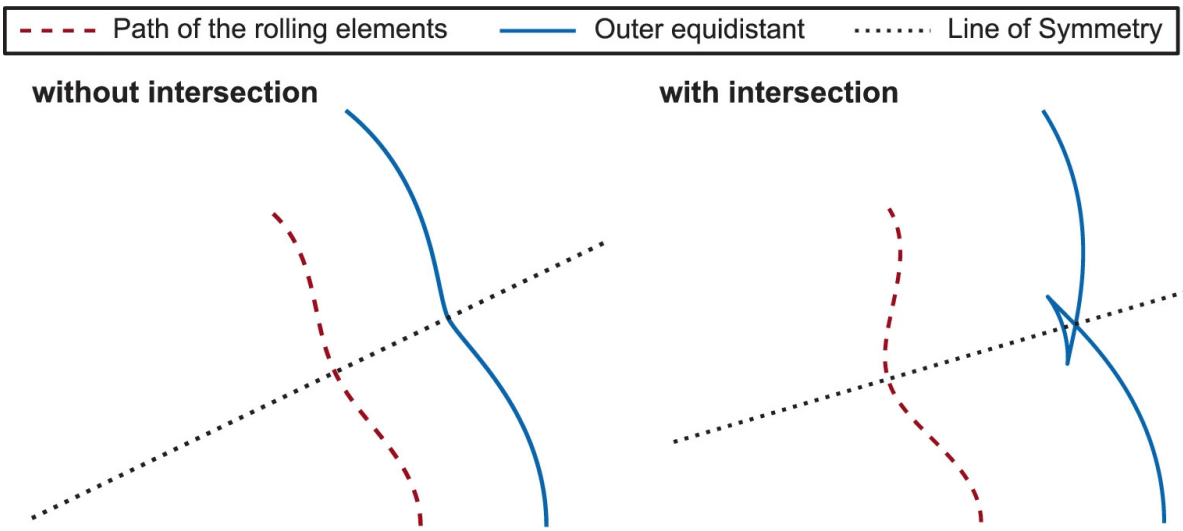


Figure 13: Comparison of curves with and without self-intersection [13]

Figure 14: Thrust bearing

In our first prototypes, we used a grub screw in a D-profile cutout to secure the crank. We quickly noticed that this caused deformation and misalignment, which in turn increased friction and damaged the gear profile. From this, we learned that a stronger connection was necessary. We solved this problem by switching to a flange coupling, with custom threads (Figure 15).

Initially, we thought self-intersecting profiles would help with 3D printing tolerances [13]. Through testing, we discovered that they introduced excessive play, which caused backlash, reduced accuracy, and lowered the number of rolling elements in contact. This increased the forces on each element and made the gearbox less reliable.

Reducing eccentricity avoids self-intersections, but it also lowers the pressure angle and reduces efficiency. We therefore selected the highest eccentricity that still worked reliably. Rolling element size also shaped the design: it influences both the reduction ratio and the gearbox's durability. In the 75 mm gearbox, we used 5 mm elements to achieve a

Figure 15: Modified flange coupling

13 : 1 ratio, while in the 90 mm version we used 6 mm elements for a 15 : 1 ratio, without compromising reliability.

Several design considerations influenced the cage. Initially, we were concerned about the number of rolling elements, as the material between the slots could become very thin. Due to the limitations of FDM 3D printing, the cage had to be printed with layers perpendicular to the axis of rotation. Since the inter-layer strength of 3D-printed parts is significantly lower than the bulk material, the cage needed to withstand the forces transmitted through these layers. In practice, this concern proved less critical, especially because the number of rolling elements was already limited by the self-intersections in the gear profile.

Cage slot tolerances were a major focus during testing. FDM printers struggle with overhangs, leading to inaccuracies in mid-air features. Our initial solution was to increase tolerances, but this introduced several problems. Vertical clearance caused the rolling elements to tilt in their slots, complicating assembly, while horizontal clearances affected gearbox performance. Insufficient tolerances increased friction, while excessive tolerances amplified backlash. Contrary to typical practice, where slightly larger tolerances reduce jamming, the additional clearance in this drive actually worsened jamming. Rolling elements deviated from their theoretical paths opposite to the gearbox rotation, forcing them through the eccentric gaps incorrectly. This created uneven motion, clicking sounds, and accelerated wear on the gear profile. We therefore carefully optimized the tolerances, balancing free movement with reliability and minimal backlash.

5.1.9. Wrist

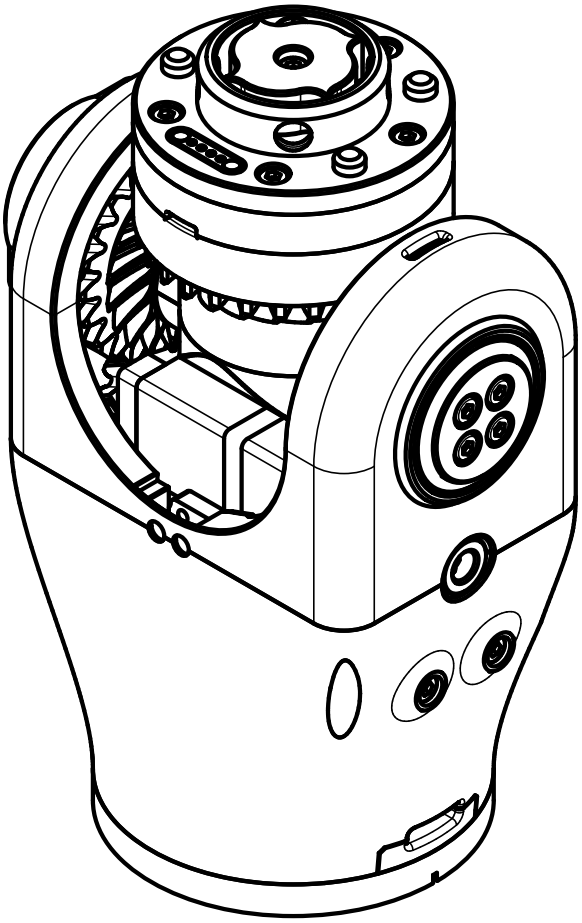


Figure 16: Differential wrist

Working Principle The hardware selection for the last two joints of the robotic arm posed several challenges. These joints needed to be smaller and lighter than the preceding joints, but from our experience with the eccentric drive, we were concerned about the reliability of a smaller gearbox. We decided to implement a differential mechanism for the wrist. This design allowed the heavy motors to be positioned closer to the previous joint, reducing forces on the structure, and introduced a novel mechanical concept to explore. Our wrist design draws inspiration from a prototype by Crnjak [9].

Design and Prototyping Initially, we considered mounting the encoders on the motor shafts, similar to the stepper motors. Ultimately, we mounted them directly on the joint axes. This approach preserves absolute positioning by avoiding the reduction between the encoder and joint axes and eliminates possible inaccuracies in the measured position

caused by backlash in the transmission. A drawback is that the second encoder is now mounted on a moving part, which complicates wiring and requires careful strain relief and routing.

As with the stepper motors, we first underestimated the forces transmitted from the motor shafts to the drive gears. The 3D-printed shaft couplings with heat-set inserts began slipping almost immediately. We replaced them with spare hex couplings originally intended for the stepper motors (Figure 17). Since they did not match the shaft diameter and had partial cutouts, we adjusted them accordingly. This modification eliminated the slipping issue and ensured reliable torque transmission.

Figure 17: Modified hex coupling

The most critical challenge with the wrist was backlash, particularly between the drive gear and the large input gears; the bevel gears were unaffected. Backlash not only introduces mechanical play, but also affects motor control, as the high reduction ratio of the motors limits their responsiveness to sudden position changes. To mitigate this, we implemented anti-backlash techniques in the mechanical design. We selected double-helix gears to combine the benefits of helical gears while avoiding axial forces. Additionally, we increased the pressure angle, root fillet radius, and gear thickness to provide a larger contact surface. These modifications reduced backlash noticeably, improving precision and control.

The final design can be seen in Figure 16.

5.1.10. Toolchanger

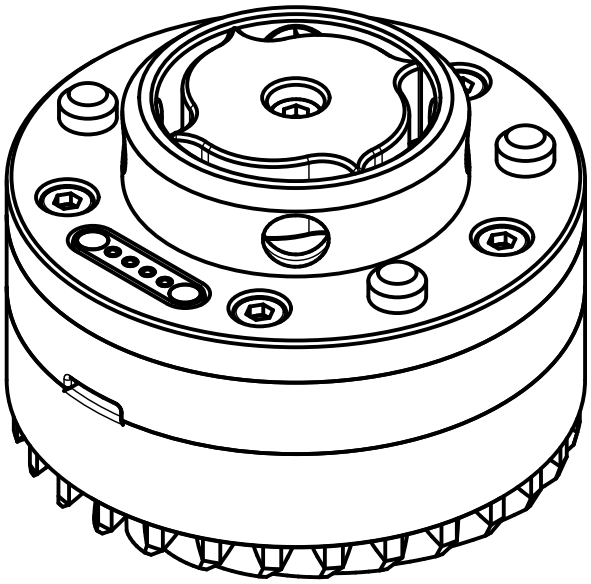


Figure 18: Toolchanger

Working Principle The robotic arm is equipped with a toolchanger to allow automatic switching between different end-effectors. The design had to satisfy the following requirements:

- Securely lock the tool in place under the full payload.
- Require power only during attachment or detachment.
- Provide precise and repeatable alignment for mechanical and electrical connections.

Design and Prototyping Instead of the original pneumatic actuation used by ATI Industrial Automation [11], we used a servo motor. A small crank converts rotational motion into linear displacement, following a smooth circular-arc profile that starts and ends tangentially, ensuring gentle engagement of the ball bearings. To reduce the overall length of the tool changer, the servo is mounted horizontally and transmits force via bevel gears.

Tools require both power and data transmission. Pogo pin connectors, combined with magnets, provide a reliable and quick connection between the tool and the changer. The spring-loaded pins maintain consistent contact and accommodate slight misalignment.

To ensure correct orientation, alignment pins with chamfers guide the tool into its precise position. This guarantees both mechanical stability and proper placement of the electrical connectors.

The final design can be seen in Figure 18.

5.1.11. Gripper

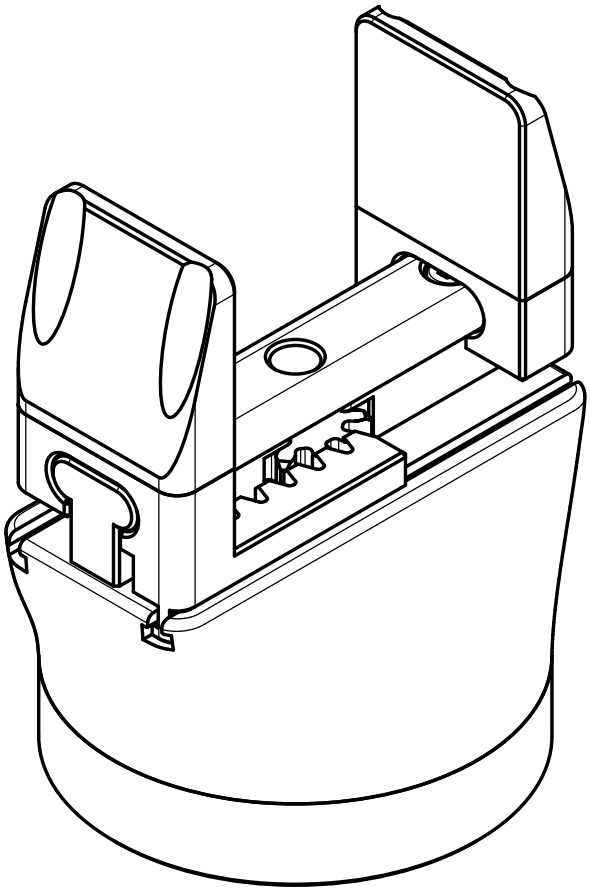


Figure 19: Gripper

Working Principle The gripper is inspired by the design of Crnjak [10]. It employs a servo motor with a pinion gear that drives two sliders equipped with racks, resulting in a symmetrical opening and closing motion.

Design and Prototyping In the first prototype, the gripper exhibited slipping due to short sliders and insufficient alignment of the central guide rail. To address this, guide

rails were added to the bottom of each rack, ensuring that the sliders remain properly aligned with the pinion throughout the motion. This modification resolved the issue.

Initially, the pinion slipped on the servo shaft, making the gripper unusable. Attempts to secure it with a screw failed because the plastic shaft could not withstand the torque. This issue was resolved by replacing the servo with a higher-quality metal shaft servo, which provides sufficient strength and eliminates slipping.

The final design can be seen in Figure 19.

5.2. Electronics

In this section, we describe the electronic system, including the microcontroller architecture, motor drivers, power supply, sensors, and communication interfaces. We also discuss the design, prototyping, and iterative improvements of the printed circuit boards that integrate these components. Throughout, we highlight the design decisions, challenges encountered, and solutions implemented to achieve a functional system.

5.2.1. Microcontrollers

To interface with the hardware directly and handle basic functionality like homing and running Proportional-Integral-Derivative (PID) loops, we need a microcontroller acting as the bridge between sensors and actuators and the higher-level control running on a PC. Since we had plenty of experience with the powerful, well-known, widely available and affordable ESP32, we used them early on in our testing and later selected specifically the ESP32-S3 N16R8 for its dual-core architecture and 8 MB of builtin Pseudo-Static Random-Access Memory (PSRAM) to be used in the robot (Figure 20).

While we initially wanted to use a single ESP32 to control all the joints, this turned out not to be practical, primarily due to the required processing power and the number of available input and output pins. Our communication solution introduces significant overhead and handling all the joints on two cores appeared challenging, with the advantages of that approach not justifying the additional complexity and effort. Instead, we distributed each joint to a dedicated ESP32. Only the wrist joints share one since there is no distinction between the motors for the fifth and the sixth joint. This approach gives us flexibility and simplifies the development and implementation of our firmware, however requires a higher overall component count, increases power consumption and introduces new challenges like the need for dedicated UART-to-USB bridges for each ESP32. In addition to the five joint ESP32s, we introduced a toolchanger ESP32 and a utility ESP32 for any potential other requirements like processing emergency stop signals.

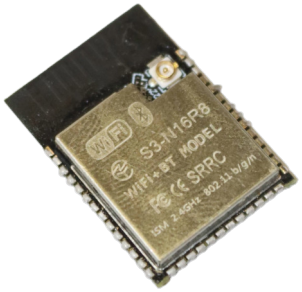


Figure 20: ESP32-S3 N16R8 module

5.2.2. Motor Control

This section details how the control signal from the ESP32s is applied to the motors.

Stepper Motors For the stepper motors, we chose modules based on the Toshiba TB6600 stepper driver Integrated Circuit (IC) due to their low cost, wide availability and convenient package with screw terminals and heat-sink (Figure 21). These drivers also support microstepping[2], which in turn enables smoother motion profiles due to the smaller step size. The driver features a configurable motor current of up to 4.5 A per phase, which is sufficient for the joint actuators used in the arm [26].

During extended testing, we discovered that for the larger stepper motors, practical limitations of the TB6600 modules arise. We observed sudden and unpredictable shutdowns which we attributed to over temperature protection. To mitigate this, we temporarily added a computer fan cooling the affected drivers. While adequate for the smaller motors, we considered switching to larger and more expensive drivers for the more powerful motors; however, we ultimately opted for mounting the drivers to a metal rail in the final design, facilitating cooling.

DC Motors For the two DC motors used in the design, we initially employed a widely available L298N H-bridge driver module. These boards are commonly encountered in hobbyist and prototyping contexts, as they are inexpensive and can handle moderate current levels. We found the module heated up significantly and transitioned to TB6612FNG-based drivers as a consequence (Figure 22). The TB6612FNG is a more modern MOSFET-based dual H-bridge design, offering higher efficiency, smaller physical

[2]Microstepping is a method of controlling stepper motors in fractional steps by modulating the winding currents. This allows smoother motion, reduced vibration, and finer positional resolution, though not necessarily greater absolute accuracy [25].

Figure 21: TB6600 stepper motor driver module

footprint, and lower heat generation [27]. This upgrade reduced conduction losses thus thermal load, enabling a smaller module size.

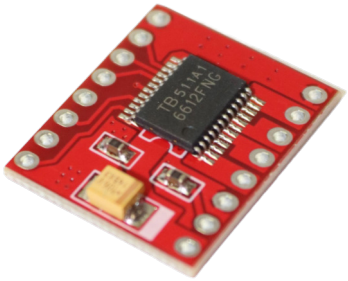


Figure 22: TB6612 Direct Current motor driver module

5.2.3. Prototyping

From the first movement to the early testing of multiple actuators working together we had a prototyping setup with breadboards and ESP32 development kits (Figure 23). This kind of setup allowed us to easily test, evaluate and modify the circuit at the cost of being prone to irritating problems like loose or missing connections. In this state, we ensured our chosen components worked together reliably.

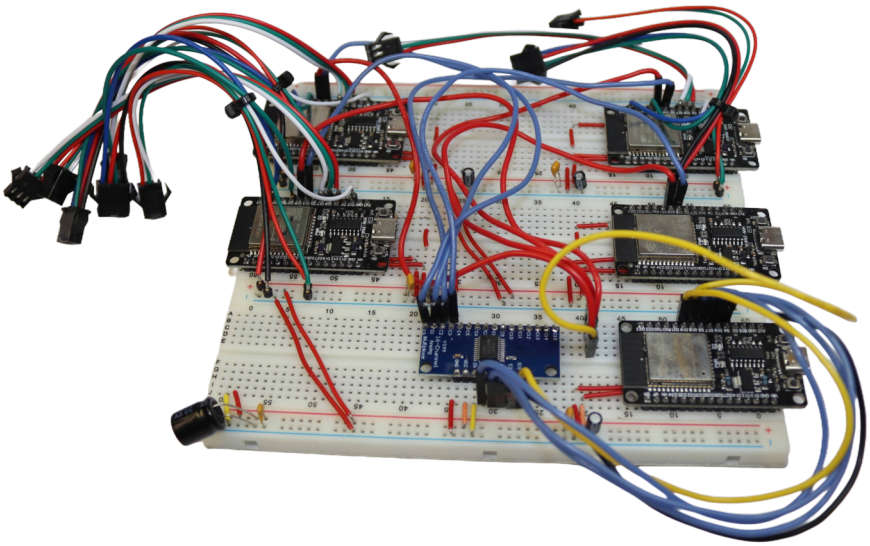


Figure 23: Prototyping setup with development kits on breadboards

5.2.4. Schematic

With the primary electronics architecture defined, we developed a full schematic using the open-source design tool KiCad. This schematic formally captures all circuits required for the custom controller functions and includes the complete set of components: power supply, connectors, USB-to-UART bridges, USB hubs, status and indicator LEDs, pull-up and pull-down resistors, line termination resistors, decoupling capacitors, transient suppression devices, reset buttons and configuration jumpers.

To establish a clear overview, the schematic is organized hierarchically. A top-level file provides the root structure and links to individual sub circuits. These sub files are partitioned by function, comprising separate schematics for power distribution, USB communication and conversion, and ESP32-specific circuitry, with dedicated files for each ESP32 module. The schematic files can be found in Appendix F.

ESP32s For the final design, we chose bare ESP32 modules in place of the previously used development kits. This saves space and drives down cost but requires support components like resistors, a reset switch and regulated, decoupled 3.3 V power.

The schematic files for the ESP32 modules are largely uniform, with only minor variations as dictated by their specific roles. Each ESP32 is supplied with regulated, decoupled power, includes a hardware reset button with a 1 µF capacitor to Ground (GND),

and implements the required pull-up resistors on strapping pins, as specified in the ESP32 datasheet [28]. In addition, every module features three status LEDs, each connected to 3.3 V and a General-Purpose Input/Output (GPIO) pin in an active low configuration through 470 Ω resistors, which we can use to provide a clear visual indication of operational states.

The first four ESP32s share an identical design, differing only in pin assignments. Each interfaces with one hall-effect sensor, one magnetic angle encoder, and one stepper driver.

The wrist ESP32 supports two end-stop sensors, dual magnetic angle encoders, and two DC motor outputs which connect to a motor driver for the wrist motors.

The toolchanger ESP32 connects to a servo, while also offering two configurable general-purpose interfaces with selectable pull-up, pull-down, and line termination resistors for unanticipated peripherals.

The utility ESP32 omits the servo connection, but retains freely assignable general-purpose interfaces, maximizing flexibility for future extensions.

Power The setup is powered from a centralized 5 V supply which is provided either through a USB-C Connector or by a dedicated 5 V input terminal. A number of linear regulators convert this voltage down to 3.3 V as required by the ESP32 and other peripherals. We chose the AMS1117-3.3 LDO for its popularity and resultant availability, low drop out voltage as well as suitable current rating of 1 A per regulator provided proper cooling [29]. The power requirements of the relevant components are listed in Table 3.

Table 3: Current consumption of the developed circuit

Component	**Current** (mA)	**Quantity**	**Combined Current** (mA)
ESP32	150	7	1050
CH340C	20	7	140
FE1.1s	25	2	50
LEDs	3	21	63
AS5600	6.5	6	39
A3144	9	5	45
TB6612FNG	1.5	1	1.5
Total	1311.5 mA		

We followed a conservative and robust power budgeting strategy in the calculation for the ESP32 module. Although the typical current draw of an ESP32 in active mode with wireless disabled is around 100 mA, as specified in the datasheet [28, p. 31], a higher

current value of 150 mA was used in power supply calculations to provide a wide safety margin for current spikes and unexpected additional load.

The indicator LEDs consume only insignificant power due to the series resistor. This also applies to the AS5600 magnetic angle encoders and A3144 hall effect switches which for each joint include connections to be powered by their respective ESP32s regulator. Since we are not using any of the integrated wireless communication on the ESP32s, its reduced power requirements allowed us to combine power supplies of two ESP32s and connect them to one regulator, thus reducing the number of components needed.

Decoupling Decoupling capacitors are employed to stabilize the local power supply by filtering out noise and providing transient current during switching events. This prevents voltage dips and reduces the risk of malfunction.

A standard combination of 100 nF and 1 µF multilayer ceramic capacitors was selected for decoupling the ESP32 modules in accordance with the ESP32 datasheet. For the CH340 converters, 100 nF and for the FE1.1s controllers, 10 µF was chosen, following their respective datasheet and reference schematics.

Debugging Hardware debugging interfaces provide direct access to a microcontroller's internal state, enabling low-level monitoring, code stepping, and troubleshooting during firmware development. A widely used method is JTAG. We implemented access to both the primary console UART and the JTAG debugging functionality to facilitate firmware development and troubleshooting on each ESP32. To support this, the board incorporates two dedicated connectors per microcontroller: one mapped to the JTAG pins and the other to the UART0 interface used as the default console. This arrangement allows us to attach an Espressif ESP-Prog [3] (Figure 24) directly for debugging a single ESP32 at runtime, while the separate UART-to-USB bridges continue to provide uninterrupted connectivity in parallel. Since this represents a debug interface, no additional series resistors or protection components were included on the JTAG or UART lines, prioritizing simplicity of routing and minimizing component overhead at the cost of some robustness against mishandling.

UART-to-USB converters Each ESP32 is connected to the computer for Micro Robot Operating System (Micro-ROS) communication with a dedicated UART-to-USB bridge. We selected the CH340C converter for its ease of use, affordability and because it operates without an external crystal oscillator, reducing component count and saving space. Since the ESP32 is a 3.3 V logic level device, it was necessary to power the CH340 converters

[3] The ESP-Prog is a development and debugging board by Espressif that provides UART and JTAG interfaces for programming and debugging ESP32-based microcontrollers [30].

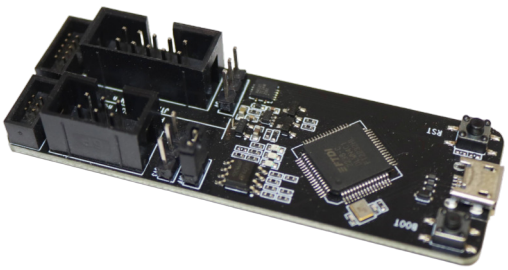


Figure 24: ESP-Prog board

with 3.3 V so as not to exceed the specified maximum ratings of the ESP32. Fortunately, this logic level is compatible with the FE1.1s as per its datasheet [31] and the CH340C tolerates 5 V logic level on its USB input side when being powered from 3.3 V [32].

USB All converters are aggregated through two FE1.1s USB hub controllers, which we cascaded in order to support connecting seven CH340C devices simultaneously. The FE1.1s was chosen over other hub ICs primarily for its easy availability, low cost, and straightforward integration, only requiring a single crystal oscillator and a few passive components to operate [31]. Moreover, sample schematics and open-source reference designs are available online, which simplified the design process and reduced design risk.

For Electrostatic Discharge (ESD) protection, a USBLC6-2SC6 suppression IC was integrated on the USB-C input lines. It provides low-capacitance transient-voltage suppression to protect the board from discharge events without significantly impacting high-speed USB signaling [33]. Additionally, two 5.1 kΩ resistors were added to the Configuration Channel (CC) lines to meet the USB standard for USB-C devices. In addition, we placed two 1 µF capacitors from the data lines to GND and two 27 Ω resistors in series with the data lines to assist ESD protection and provide some dampening to the signal.

The board features one dedicated USB-C connector as the main upstream link to the host PC. Through this, all converters are exposed as independent devices.

Connectors To accommodate peripheral connections, debugging, and future expandability, each ESP32 is wired to multiple connectors. For the driver and sensor connectors, we selected JST-PH 2.0 connectors (Figure 25). This type was chosen not only for its compact footprint and widespread availability but also for practical advantages such as the locking mechanism and its non-reversible design, which ensures that cables cannot be plugged in backwards, protecting against wiring errors during assembly or maintenance. All JST-PH connections provide both GND and 3.3 V as part of their pinout if needed for the respective components operation.

For development and expansion, we also included 2.54 mm standard pin headers, exposing unused GPIO pins and power rails. These expansion headers generally supply GND and 3.3 V, though one header line was intentionally routed with 5 V available as well, allowing for more versatile prototyping with peripherals that cannot be powered directly from 3.3 V. These flexible 2.54 mm headers for unforeseen requirements and additions aimed to give the board robustness and adaptability in practice.

For the debug UART and JTAG interfaces, we used standard 2×3 and 2×5 pin 2.54 mm box headers, the same type found on the ESP-Prog.

Figure 25: JST-PH 2.0 connector

5.2.5. Printed Circuit Board

Printed Circuit Boards (PCBs) are flat boards that use copper traces to connect electronic components, allowing for high-density wiring with an organized design. We again used KiCad to create a PCB layout integrating our control circuitry from the electronic design schematic. Figure 26 shows a rendering of the finished board and a photo of the mounted board can be found in Figure 37. The layouts can be found in Appendix G.

Packages When working with PCBs, Surface-Mount Device (SMD) packages are often preferred over Through-Hole Technology (THT) components because they require considerably less space and do not interfere with routing on other layers. Since the assembly of the PCB was carried out manually, we chose the imperial 1206 package (3216 metric), which measures 3.2 mm × 1.6 mm, while the height depends on the specific component used.

Layers For our application, a simple double-sided PCB with traces on both top and bottom turned out to be sufficient to allow for all necessary components to be integrated. This choice reduced manufacturing cost and lead time as well as complexity while still leaving enough routing options to cleanly separate power, GND, and signal traces. One drawback to this is the absence of any GND and power planes which are usually relied on to provide stable reference planes for signal traces as well as easy access to GND or power, greatly simplifying routing.

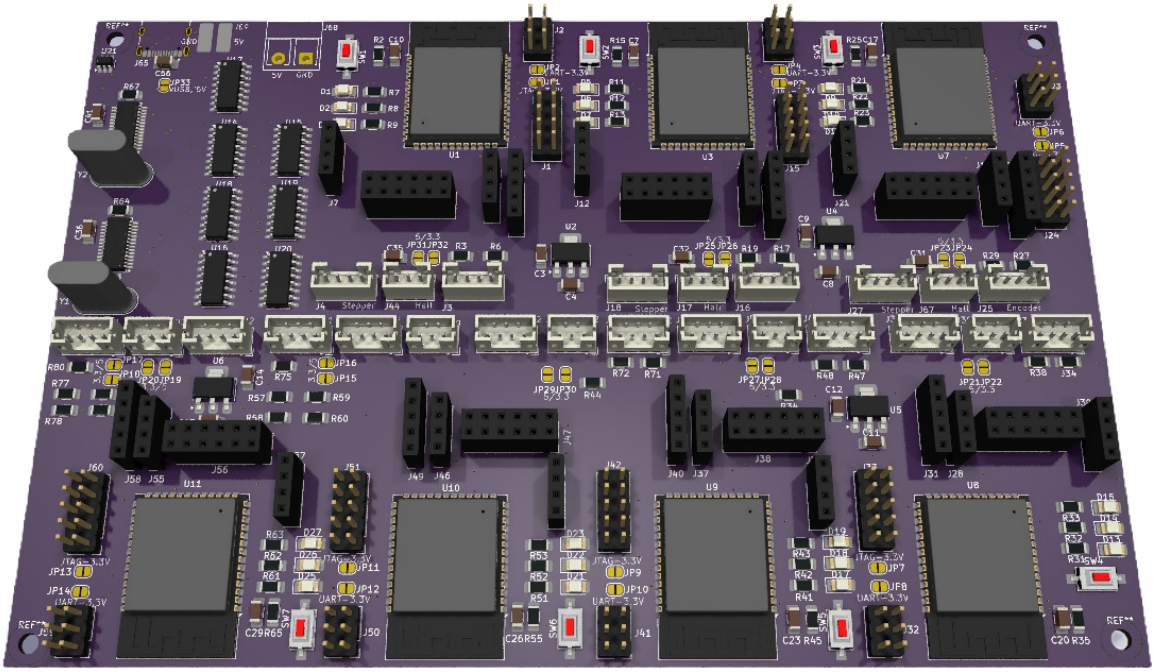


Figure 26: Rendering of the printed circuit board

USB The USB-C connector was placed in a corner alongside with the 5 V power input connector. In terms of signal integrity, we took particular care with the USB data lines and placed the hub crystal as close as possible to the FE1.1s package to minimize trace length and parasitic loading[4] , ensuring stable oscillation and reliable USB operation. Routing of the differential USB pairs was carried out with length matching, while stubs and via counts were minimized to prevent signal degradation according to standard best practices for routing high-speed signal traces.

Power Each microcontroller features its own decoupling network with ceramic capacitors physically close to the Voltage Common Collector (VCC) pins to prevent voltage dips when current spikes occur, along with wider traces for power lines to further reduce potential issues.

[4]Parasitic loading refers to the unintended capacitance, inductance, and resistance introduced by PCB traces, component leads, vias, and connectors. These parasitic elements distort signal waveforms, cause impedance mismatches, and reduce signal integrity, especially in high-speed circuits such as USB. The effects include signal reflections, crosstalk, timing errors, and noise, which can degrade the overall performance and reliability of high-frequency digital designs [34].

Connectors To reduce the risk of confusion, every connector on the board is explicitly labeled on the silkscreen with its intended purpose. We positioned the connectors for the robot centrally on the PCB to simplify cable routing and keep wiring organized within the final system, moving debugging and unassigned connectors outwards next to their respective ESP32s.

Issues of the first version The initial revision of the PCB was largely functional, but we encountered several issues during bring-up and debugging.

The most critical problems concerned the UART interfaces. Specifically, we had not swapped the Receive (RX) and Transmit (TX) lines between the ESP32 modules and the CH340 converters, rendering the UART bridges unusable since the RX line must be connected to the TX line and vice versa. Additionally, we wired both the ESP-Prog and CH340 devices to the same UART 0 interface, which prevented simultaneous use of debugging and Micro-ROS communication.

We also faced challenges with the USB-C interface due to multiple shortcomings. Notably, we mistakenly placed 1 µF capacitors from the D+ and D− USB data lines to GND, which disrupted signal timing and prevented data communication. These capacitors were intended to support ESD protection and should have been orders of magnitude smaller, in the picofarad range. Moreover, the small SMD USB connector footprint caused soldering difficulties. The connector's metal housing compromised the solder mask, creating shorts with underlying traces. To restore USB-C functionality, we resorted to soldering an external breakout board directly to the relevant PCB traces.

The JTAG and UART connector pinouts also did not match the ESP-Prog cable, requiring us to solder custom adapter cables. To resolve these issues, we performed several hardware modifications: we corrected the UART line swap by carefully cutting traces and rewiring with thin jumper wires. Later, to enable a secondary UART interface while maintaining the ESP-Prog connection, we soldered jumper cables from the CH340 chips to free GPIO pins on the ESP32 modules. These fixes allowed us to continue development despite the shortcomings of the first revision.

Before designing and ordering the second revision, we consulted a professional PCB layout engineer for feedback on the original design. He recommended improvements in several key areas:

- The 1 µF capacitor on the reset switch to GND was suboptimal, as it delayed ESP32 startup. The correct value per datasheet is 100 nF.
- The oversized capacitors and series resistors on the USB data lines stood out to him and he recommended their removal.

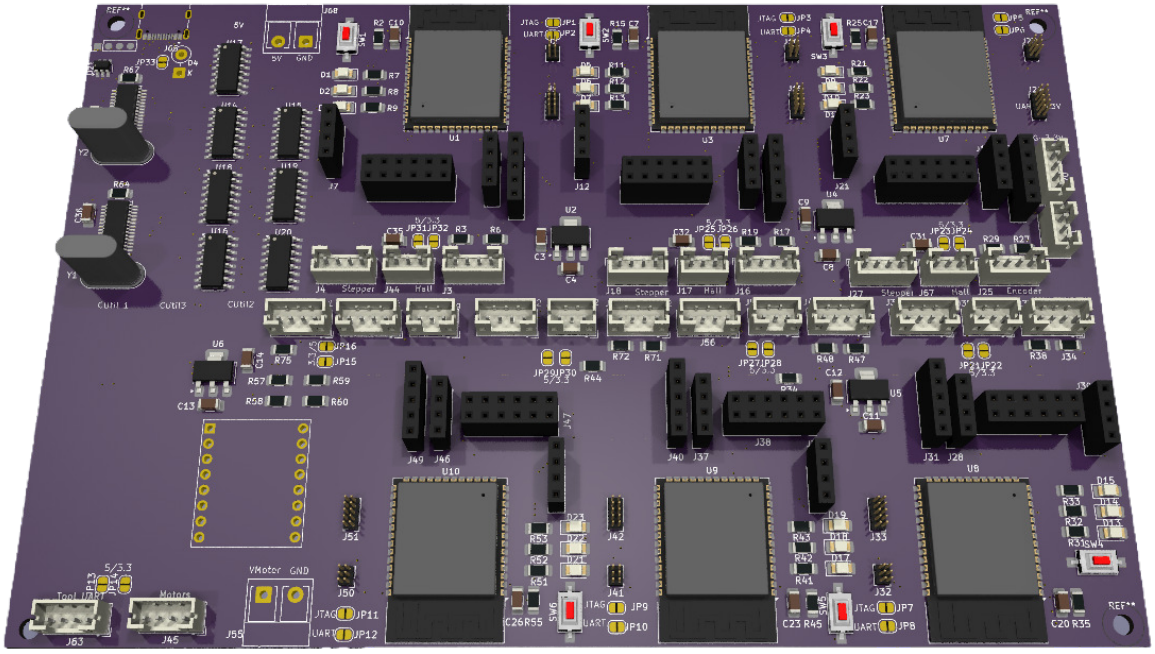


Figure 27: Rendering of the second revision printed circuit board

- In the CH340C power supply, a connection required for powering the chip from 3.3 V instead of 5 V was missing. Despite this, we had managed to run the circuit, but the engineer advised correcting it.
- Robust protection circuitry should be added to safeguard connected USB devices and ESP32 modules:
  - Fuses to provide over-current protection
  - ESD protection ICs to handle transient discharges
  - Protection diodes to prevent over-voltage on the 5 V line from feeding back into an attached USB device
- Stable power and good reference planes for signal traces should be ensured. The engineer recommended a 4-layer PCB design with dedicated GND and power planes to improve overall reliability and signal integrity.

Improvements on the second revision Based on the problems encountered in the first version and changes we found would be beneficial during testing, we designed a second revision of the PCB (Figure 27) that incorporates several important improvements.

The UART lines were corrected to ensure proper communication, and we rerouted the USB-to-UART converters to unused pins connected to the UART 1 interface. This change enables both debugging with the ESP-Prog and communication with Micro-ROS to take place simultaneously without conflicts.

On the USB interface, we removed the incorrectly sized capacitors and unnecessary series resistors on the data lines. To avoid the short circuits caused earlier by the USB connector housing wearing through the solder mask, no signal or power traces were placed underneath the connector; instead, the area was filled with a GND plane. We chose to retain the small connector footprint but intended to rely on improved soldering tools and techniques to assemble it more reliably.

To match the standard ESP-Prog cable, we redesigned the UART and JTAG pinouts and changed the connectors to 1 mm pitch box headers to save space.

Further corrections were made to the power and reset circuitry. The capacitor on the reset switch was changed to the recommended value of 100 nF [28], decreasing ESP32 startup time. The missing configuration connection on the CH340 was fixed so that the device can operate correctly from a 3.3 V supply. In addition, a Schottky protection diode (SB360) was introduced to prevent reverse current flow into the USB host device from the board's 5 V input.

Finally, while we considered external fuses and additional ESD protection ICs, these were left out of the board itself because the ESP32 modules already feature basic integrated protection and we added external fusing to the 5 V input.

We applied GND and power plane fills in suitable spots to both sides to still achieve as much of the desired reduction in electromagnetic interference and to provide reference planes for the microcontrollers and communication circuitry as possible without pivoting to a 4-layer design due to the involved complexity, additional potential for unforeseen difficulties and manufacturing cost.

While previously we had used a dedicated module breaking out the TB6612FNG IC and integrating a few capacitors, on the second revision we decided to place the driver IC directly onto the board for easier wiring and compactness.

Integration of a possible future feature was realized by routing a dedicated GPIO pin from each of the first four ESP32 microcontrollers to an additional connector on the PCB.

The schematics (found in Appendix H) and layout (Appendix I) corresponding to this PCB can be found in the appendix. Due to time constraints, we did not implement its use.

5.2.6. Wiring

The wiring of the robot can be sectioned into four stages, beginning with an improvised prototype and ending in a revised dual cable harness setup.

In the beginning, we connected the sensors and PCB using premade JST SM connector pairs which we crimped to fit our PCB, soldered extension cables from and connected to the sensors. The original motor cables were screwed directly into the drivers. This approach was sufficient for early testing but features apparent limitations: every sensor and motor carried its own separate power lines, redundant conductors accumulated, and numerous cables hung from the joints, obstructing free motion.

We therefore required a structured solution and decided to implement a cable tree that would distribute power and signals through the arm in a tidy and compact manner. For this purpose, we selected six-conductor flat silicone wire with a cross-section of 24 AWG for its high flexibility, mechanical robustness, and ability to carry the expected current. The flat geometry also helped maintain order in the harness. At the base of the arm, the complete tree carried 44 conductors, which was only achievable after consolidating the common supply rails 3.3 V and GND, as otherwise the conductor count would have been considerably higher.

We constructed the cable tree progressively: starting at the wrist, it incorporates connections for the local motors, encoders, sensors, toolchanger, and tool interface via with JST-PH and JST-SM connectors for detachability. At each subsequent joint, additional stepper and sensor lines were added until the full set of conductors reached the base. Power and communication lines are terminated on the PCB through JST-PH connectors, while motors are connected via screw terminals and soldered cable extensions.

For mechanical reliability, we routed the harness inside PET braided sleeving and provided strain relief at every transition between moving joints. Clamps designed into the CAD model secured the harness at relevant points, and we dimensioned cable loops to allow each axis to rotate $\pm 180°$.

During testing of the first version of the cable tree, we encountered significant Electromagnetic Interference (EMI): whenever the stepper motors were active, the I^2C communication with the magnetic encoders became unstable. Using a series of tests with shielded and unshielded wiring moved in proximity of the motor wires, we traced the problem to coupling between the unshielded signal lines and the motor supply wiring. To resolve this, we decided to add shielded twisted pairs from Cat6a S/FTP Ethernet cable into the harness. Each encoder bus was routed through its own twisted pair along with the tool UART lines and the shields were tied to GND. Initially, communication seemed more stable, however we were again faced with repeated failed readings from the encoders when the stepper motors drew a lot of current.

To ultimately address the interference problem, we concluded that the only reliable option was to revert to the physical separation of power and signal lines, as we had initially used during prototyping. To achieve this, we introduced an additional dedicated harness specifically for communication. Again, we employed twisted pair strands extracted from Cat6a S/FTP Ethernet cables, cutting seven pairs to the required lengths and bundling them together. On the outside, the bundle was enclosed in metal mesh braid that also surrounded the Cat6a S/FTP cable.

This auxiliary harness was routed along the side of the robot and fixed in place with zip ties, with cable loops dimensioned to permit joint movements. All I^2C lines for the encoders, as well as the tool UART lines, were migrated into this separate cable tree, thereby ensuring they remained physically distant from the stepper motor wiring.

While the resulting solution was less compact and less visually clean compared to the integrated harness design, we found it to be necessary to ensure communication reliability. Without this physical separation, it would not have been possible to achieve stable encoder communication during motor operation, making the dual-harness system a necessary robust compromise.

5.2.7. Power

The robot's power architecture relies on high-quality, dedicated power supplies to ensure stable operation of both motors and electronics. Three 12 V MeanWell power supplies form the core of the system. Two LPF-40 units rated at 5 A each, together with a higher-current NPF-90 unit rated at 7.5 A, are connected in series to provide a total of 36 V for the stepper motor drivers. This configuration allows the stepper drivers to operate efficiently while supplying sufficient current to the most heavily loaded joints. The 12 V DC motors are powered directly from the higher-current NPF-90 unit, ensuring they receive sufficient voltage and current without affecting the stepper motor supply.

The control electronics, including all ESP32 modules and associated peripherals, are powered separately via a dedicated RS-25-5 5 V power supply rated at 5 A. This separation between motor and logic circuits prevents voltage fluctuations caused by motor activity from affecting the microcontrollers or sensors, ensuring stable operation.

In the final version of the system, fuses were added to all major power lines to provide over-current protection. The 36 V stepper motor supply uses a 5 A fuse, the 12 V DC motor supply a 2 A fuse, and the 5 V PCB supply a 3 A fuse. These fuses protect both the components and wiring while remaining easily replaceable in case of an overload or failure.

Thanks to the use of high-quality, brand-name power supplies and the clear separation of power domains, the system has shown stable and reliable performance throughout

testing. No additional filtering or voltage stabilization was required, and no power-related issues were encountered. A photo of our power supplies can be found in Figure 36.

6. Software

In this section, we describe the development of the software for the robotic arm, including the design choices, frameworks, and tools used to enable control, integration, and programmability. The goal was to provide a simple Python Application Programming Interface (API) that enables users to write programs for the robot using straightforward commands, while also allowing integration with other software.

Initially, we planned to implement the system entirely ourselves using Python and Arduino. However, we quickly realized that much of our time was being spent on basic functionality not directly related to the scope of this thesis, such as managing communication between the different components. To simplify the implementation, we migrated to a widely-used robotics framework, ROS 2 (Section 6.1), which allowed us to focus on the more interesting aspects of the software. We also switched to Espressif IoT Development Framework (ESP-IDF) for the microcontrollers, as it provides better support for Micro-ROS (Section 6.1.1) and offers more powerful and efficient control of the hardware. An overview of the software stack is shown in Figure 29.

6.1. Robot Operating System

ROS 2 is a modular, open-source robotics framework that provides standardized tools and libraries for robot software development. It supports distributed computation, real-time communication, and hardware abstraction, reducing the need to implement low-level communication and control routines from scratch. Key concepts in ROS 2 include *Nodes* (processes that perform computation), *Topics* (publish/subscribe communication channels), *Services* (request/response interactions), and *Actions* (preemptible long-running tasks). ROS 2 separates high-level software components such as planning, control, and perception from the specific hardware implementation, which improves modularity, simplifies integration with simulation, and facilitates system reuse [35].

6.1.1. Microcontroller Implementation

Micro-ROS is a lightweight implementation of the ROS 2 framework for microcontrollers. It simplifies development by allowing microcontrollers to use ROS 2 concepts such as topics, publishers, and subscribers directly, without the need for a custom communication solution. A Micro-ROS agent runs within the main ROS 2 system to integrate the microcontroller nodes. Micro-ROS supports multiple transport options, such as WiFi or custom protocols [36].

In our setup, we used the serial transport provided by the ESP-IDF Micro-ROS examples [37]. Each ESP32 runs a dedicated task on a separate core, which handles all Micro-ROS communication and allows real-time control loops to run on the other core. A key improvement we implemented was increasing the baud rate[5], which substantially enhanced communication performance and enabled faster, more responsive actuator control. The configuration of each ESP32 is managed through a dedicated header file, simplifying parameter tuning and deployment.

6.1.2. Robot Hardware Interface

In the `ros2_control` framework [38], hardware components provide an abstraction layer that decouples robot controllers from the specific physical implementation of the hardware. Three types of hardware components are distinguished: Actuators, sensors, and systems. Actuators represent one DoF devices such as motors driving individual joints, while Sensors provide measurements such as encoder feedback. Systems encapsulate more complex hardware that combines multiple DoFs, for example an entire robot arm.

The structure and capabilities of these components are described in the robot's Unified Robot Description Format (URDF) file, where command and state interfaces are defined. Command interfaces typically specify desired values, e.g., position or velocity, whereas state interfaces provide feedback for the same or related quantities. To generate the URDF file from our Onshape CAD model we used onshape-to-robot [39].

Our hardware system is implemented as a C++ plugin that integrates with the `ros2_control` framework. It forwards position and velocity commands from controllers to the microcontrollers. Since each actuator is equipped with its own ESP32 running Micro-ROS, the communication is distributed and modular, which simplifies the hardware interface. The library simply publishes the desired position and velocity values to the corresponding actuator topics and reads the state feedback.

Additionally, the hardware interface supports a fake hardware mode. In this mode, commands are internally integrated to simulate the actuator motion without forwarding them to the physical hardware. This feature enables the testing and visualization in RViz 2 before executing them on the real robot, thereby improving both safety and development speed.

6.1.3. Joint State Broadcaster

The Joint State Broadcaster is a standard `ros2_control` component that collects all available state interfaces from the hardware, such as encoder readings, and publishes

[5]The baud rate is the speed for a serial communication channel.

them as a unified stream. This topic serves as a central source of truth for the current joint positions and velocities. Many ROS 2 tools rely on this topic, including the Robot State Publisher, visualization in RViz, and higher-level frameworks. By maintaining a consistent and continuous stream of joint states, the broadcaster ensures that other nodes in the system can remain synchronized with the physical or simulated robot [40].

6.1.4. Robot State Publisher

The Robot State Publisher uses the URDF description of the robot together with the joint states reported by the Joint State Broadcaster to compute the corresponding kinematic transforms. It publishes these transforms as a tree of coordinate frames, which describes the spatial relationships between all robot links over time. This information is essential for visualization in RViz (Section 6.1.5). In addition, the Robot State Publisher republishes the robot description, allowing client applications to retrieve the full kinematic and geometric model directly from the ROS 2 system [41].

6.1.5. Visualization

RViz 2 is a 3D visualization tool for the ROS 2 framework (Figure 28). It uses the robot description from the URDF file as well as the joint states provided by the robot state publisher to render the robot model in real time. This makes it possible to verify that the kinematic structure, link transforms, and current configuration of the robot are consistent with the hardware. In our case, RViz 2 was primarily used to monitor the execution of trajectories and confirm that Cartesian and joint space motions behaved as expected before running them on the physical system. Beyond robot visualization, RViz 2 also supports displaying coordinate frames, sensor data, and markers, making it a valuable tool for debugging and validating robotic applications [42].

6.1.6. Joint Trajectory Controller

The `ros2_control` framework provides a set of standardized controllers, among which the Joint Trajectory Controller is the most commonly used for robot arms. It executes time-parameterized trajectories on multiple joints in a coordinated fashion. The controller receives trajectories through a standard action interface. Each trajectory consists of a list of waypoints, where each waypoint specifies a timestamp, joint positions, velocities, and optionally accelerations. The controller interpolates between these waypoints in real time to generate smooth reference commands for the actuators.

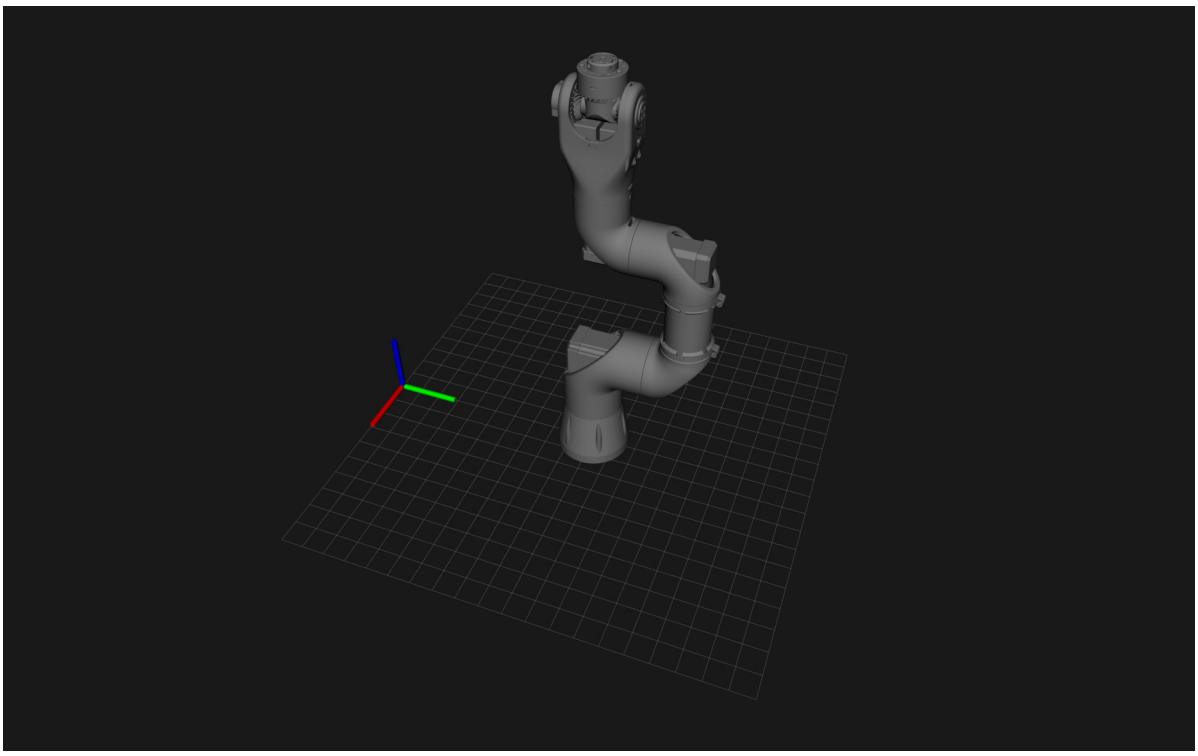


Figure 28: Screenshot of RViz 2

In our setup, the Joint Trajectory Controller acted as the bridge between the high-level Python API, which generated joint trajectories, and the low-level actuator controllers running on the ESP32s. This ensured that all joints moved synchronously according to the commanded trajectory. Although the controller provides an optional PID loop for joint control, we did not enable it since closed-loop control was already implemented directly on the ESP32s (Section 6.3). The same controller could be used both with the physical hardware and with the simulation mode of our hardware interface, enabling trajectory validation before execution on the real system.

6.1.7. Programming Interface

The Robot API is a Python package we developed that serves as the user interface for controlling the robot. It allows the user to write programs for the robot using the Python programming language. The API provides access to the current joint configuration as well as the end-effector pose in Cartesian space, which is computed using the forward kinematics derived in 4.2.

Motion in both joint and Cartesian space is supported:

- **Cartesian space:** A path in cartesian space consists of a sequence of end-effector poses. For a each pose in the sequence, the inverse kinematics (derived in 4.3) are computed (with an average computation time of around 150 µs). Among the feasible joint configurations, the solution closest (in Euclidean joint space) to the previous configuration is selected. A cubic spline is then generated for each joint using these configurations and a proposed duration, which is determined from an approximation of the path length and the specified speed. The spline is constructed with boundary conditions enforcing zero velocity at the start and end, ensuring smooth motion and a complete stop. To ensure the limits of the robot are not exceeded, the trajectory is checked and the proposed time is scaled if necessary. The resulting joint trajectories are then passed to the trajectory controller.

  Instead of an entire path, the robot is also able to move to a single target pose. In which case it generates a path from the current to the target pose through linear interpolation in cartesian space.

- **Joint space:** Linear interpolation in Cartesian space is not required. Instead, a cubic spline is generated directly between the current configuration and the desired target configuration.

Earlier versions of the API computed the inverse kinematics only at the final target pose and then generated the trajectory in joint space. However, since this approach did not guarantee linear motion in Cartesian space, we pivoted towards a method interpolating along the Cartesian path before trajectory generation.

Beyond motion control, the API allows the user to configure and operate the end-effector tool, adjust the execution speed, and interface with the tool changer to attach and operate tools such as a gripper. It also provides additional modes, including:

- **Fake hardware mode:** Simulates ideal hardware without using the physical robot, useful for program visualization and testing.

- **Debug mode:** Provides additional information such as position and velocity errors for the joints.

6.2. Microcontroller Firmware Components

We developed several custom ESP-IDF components for this project. They represent reusable code abstractions that encapsulate low-level control of motors, encoders, and sensors, without being tied to the overall firmware structure. These components provide modular interfaces that can be integrated with higher-level software, ensuring precise and deterministic operation while simplifying maintenance and future extensions.

6.2.1. Stepper Motor

The stepper motor drivers we used are controlled through a step pin and a direction pin. Each rising edge on the step pin (transition from Low to High) advances the motor by one step, with the motor's rotational speed determined by the frequency of these pulses.

To ensure that both the control loop and system communication can run in parallel, pulse generation is executed in the background. The GPTimer peripheral of the ESP32 is leveraged by the library to generate the required square wave signal on the step pin [43]. The library provides a function to set the desired motor speed in radians per second. This value is automatically converted into the corresponding pulse frequency, taking into account the number of steps per revolution, the selected microstepping mode, and the gear ratio.

A key modification we implemented was the use of a speed change threshold. Updating the pulse frequency too frequently leads to jitter in motor motion. By applying updates only when the change in speed exceeds a defined threshold, we achieved smooth and stable motor behavior.

6.2.2. Direct Current Motor

The DC motor driver we used is capable of controlling two separate brushed DC motors. The driver provides two pins per motor for direction and braking, as well as a Pulse Width Modulation (PWM) pin for controlling the applied effort [27]. The MCPWM peripheral of the ESP32 is used by the library to generate the PWM signal [44]. Input values in the range $[-1, 1]$ are mapped to the effective PWM range of the motor.

6.2.3. Encoder

The AS5600 encoders interface via the I^2C communication protocol [20]. The direction and output scale factor can be configured and functions are provided to update readings, retrieve the position in radians as well as detect the presence of a magnet.

On the stepper motor actuators, the encoders are mounted at the back of the motors and measure the motor shaft position directly. Due to the reduction mechanism, the absolute position of the gearbox output cannot be read directly. A cumulative angle variable is maintained to store the current position. For each measurement, the change in position is calculated as the difference between the current and previous readings. This change is added to the cumulative angle. After a full rotation, the angle wraps around to 0, which is handled by using the shortest angular distance: if $\Delta > \pi$, 2π is

subtracted, and if $\Delta < -\pi$, 2π is added. A scale factor parameter is included to report the gearbox output position accurately.

6.2.4. Proportional-Integral-Derivative Controller

To accurately control the motors under varying loads and conditions, PID controllers were implemented (Figure 30), which are widely used in industrial applications. The proportional term depends on the current error, the integral term accumulates error over time, and the derivative term dampens the output if the error changes rapidly [45]. A dedicated library was developed to simplify usage across the project.

The initial implementation relied solely on PID, meaning the control signal was based only on the error. While this works for stable setpoints, it causes lag and reduced accuracy when the setpoint changes rapidly. To improve response, we added a feedforward term, representing the anticipated action required to follow the setpoint more accurately [46].

Integral windup occurs when the control signal saturates and the integral term continues to accumulate, leading to overshoot as the accumulated error unwinds [47]. To address this, the integral term is only updated when the control signal is not saturated without the integral contribution, preventing windup and improving system stability.

6.2.5. Differential Speed Controller

The wrist is actuated through a differential gear set, meaning the virtual output axes are not directly coupled to individual motors. Instead, the first virtual axis is controlled by the speed difference between the two motors, while the second virtual axis is controlled by the speed sum. Furthermore, the gear ratio introduces asymmetry: the second virtual axis is scaled by a factor of 20:29, whereas the first virtual axis remains unaffected. To manage this complexity, we developed a dedicated differential speed controller component to map desired joint velocities to motor commands. If commanded velocities exceed the physical limits of the motors, synchronization between the virtual axes can be lost. In such cases, the actual speeds are governed by the mechanical constraints of the differential gear set rather than the controller. To prevent this, controller outputs are clamped to $[-1; 1]$, and the lower of the two values is proportionally scaled, ensuring synchronous motion within feasible velocity limits.

6.3. Microcontroller Firmware

Each ESP32 runs its own firmware, which combines Micro-ROS with our custom ESP-IDF components (Section 6.2) for hardware management. While the components provide reusable building blocks, device-specific implementations, such as homing sequences and control loop logic, are implemented directly in the firmware.

6.3.1. Stepper Motor Actuators

Control Logic The control logic for the stepper motor actuators (Figure 31) is executed in a periodic loop. In each iteration, encoder values are updated and used to derive velocity feedback based on the loop period. To suppress measurement noise, an exponential moving average is applied to the velocity signals. The control input consists of a desired position and velocity. The position setpoint is updated each cycle by integrating the velocity reference. The position PID controller compares the updated position setpoint with the feedback, incorporates the velocity setpoint as a feedforward term, and outputs a velocity command. Since stepper motors are inherently discrete-position devices, they generally follow commanded velocity unless stalling occurs. Consequently, direct velocity control is sufficient, and no secondary velocity PID controller is required. Acceleration limiting is applied to ensure smooth motion. This is achieved by constraining the difference between successive velocity commands, thereby enforcing an upper bound on actuator acceleration before the velocity is forwarded to the stepper driver.

Homing Homing posed particular challenges due to the absence of absolute position feedback. After power-off, the actuator can be displaced manually, making the zero reference unreliable. A fully automated homing sequence would therefore risk mechanical damage through unexpected collisions or cable strain. To address this, a semi-passive homing procedure was implemented. Initially, the actuator remains disabled until it is manually positioned near the zero reference. Once the encoder detects proximity, the motor is enabled, and the system prevents further displacement. After a short delay, the actuator executes a localized search around the approximate zero to detect both edges of the magnetic sensing range. The zero reference is then defined as the midpoint of this interval, and the actuator repositions itself to this calibrated origin.

6.3.2. Wrist

Control Logic The control logic for a single differential wrist axis (Figure 32) follows a cascaded position–velocity design. As with the stepper motor actuators (Section 6.3.1), filtered encoder signals are compared against the position setpoint in a position PID

controller, producing a velocity command. After acceleration limiting, this command serves as the input to a velocity PID controller, which regulates axis speed based on velocity feedback. This structure is necessary because the wrist actuators are DC motors driven by voltage (effort control), which requires closed-loop velocity regulation to ensure stable behavior. This logic is applied to both axes of the wrist, after which the resulting signals are mapped to the motors by the differential speed controller (Section 6.2.5).

Homing Unlike the stepper actuators, the wrist can determine its position absolutely due to the direct coupling of encoders and the use of worm gears, which also prevent manual backdriving. Consequently, homing is fully automated and serves only to refine the zero reference. For the first axis, an end stop switch is used; for the second, a Hall effect sensor is employed. The sequence begins by driving the first virtual axis against its end stop at high speed, followed by a slower reversal for precise localization. Simultaneously, the second axis moves toward the approximate position of its Hall effect sensor. After the first axis completes homing, the second axis continues until the Hall effect sensor is triggered, completing calibration.

6.3.3. Toolchanger

The tool changer is actuated by a servo motor, which is controlled through the ESP32's LEDC peripheral [48] to generate a PWM signal. The interface is deliberately minimal: a single Boolean topic is exposed, allowing the tool to be either attached or detached.

6.3.4. Gripper

The gripper firmware follows the same design principles as the tool changer (Section 6.3.3). Instead of discrete commands, the interface accepts a continuous target value representing the desired gripper width. The commanded value is mapped to the corresponding servo angle, enabling continuous adjustment of the gripper opening. This approach provides a simple yet flexible interface for object manipulation tasks.

7. Reflection and Outlook

Designing a robotic system from scratch is a complex, multidisciplinary task. This section reflects on the development of our robotic arm, critically assessing mechanical, electrical, and software choices, addressing challenges in time management and division of work, and outlining potential improvements and future directions.

7.1. Mechanical Design

A key consideration in our design was motor placement. We were aware that mounting them directly on or close to the joints would increase the load on the arm by adding mass to the moving links. Nevertheless, we opted for this approach because it simplified the mechanical design and reduced the complexity of the transmission. In retrospect, relocating the motors closer to the base would have reduced inertial forces on the joints, improving efficiency, lowering stress, extending component life, and increasing accuracy.

Another key consideration was the transmission. Our custom gearboxes achieved the required torque, but the limited precision of 3D printing and the material properties introduced noticeable backlash, causing positioning errors and complicating control. In hindsight, belt drives would have been more effective, offering nearly backlash-free operation, smoother motion, lower noise, and easier replacement, which is particularly suited to a 3D-printed system.

7.2. Electrical Design

A central strength of our electrical design was its modularity: dedicating a separate ESP32 to each of the first four joints and the wrist simplified firmware development, allowed parallel debugging, and ensured sufficient processing power, while standard components reduced cost and eased sourcing. However, this approach introduced significant overhead, requiring six microcontrollers, local power regulation, UART-to-USB bridges, and extra circuitry. A more conventional solution would have used a single high-capacity microcontroller with expanded Input/Output (IO) or IO extenders to centrally manage all joints, reducing complexity, component count, board space, power consumption, and simplifying PCB routing and cabling.

Beyond microcontroller selection, our sensor choices introduced additional limitations. The AS5600 magnetic encoders, though widely used and accurate, have fixed I^2C addresses, forcing separate I^2C buses for each encoder and increasing wiring complexity and pin requirements. Similarly, the A3144 Hall effect sensors could have been more efficiently managed via a custom bus-integrated module. Using sensors with configurable

addresses would have enabled a shared digital bus, placing all joint sensors on a single interface and simplifying wiring. We also encountered significant EMI challenges on long unshielded I^2C lines, especially when multiple stepper motors operated simultaneously. This required shielded twisted-pair cabling and careful harness grounding. A more robust and scalable approach would have employed a differential bus system, such as Controller Area Network (CAN) or Recommended Standard 485 (RS-485), commonly used in robotics hardware. These standards are inherently resistant to EMI, particularly in tight, cable-dense environments, and would have reduced both debugging overhead and the risk of communication faults.

In summary, while the system was largely functional and offered valuable learning opportunities, a more centralized architecture using bus-capable microcontrollers and sensor modules would have enhanced maintainability and robustness, while reducing the need for extensive shielding.

7.3. Software Design

Looking back, the most important realization in our software development process is that we underestimated the value of established frameworks. By initially trying to build everything ourselves with Python and Arduino, we placed a heavy burden on ourselves to reinvent solutions to problems that the robotics community has already solved many times. While this did give us a taste of the underlying mechanics, it ultimately slowed down progress and created unnecessary stress. If we had adopted these tools earlier, we could have avoided weeks of frustration and focused on the work that added the most value, like refining control strategies and exploring kinematics.

7.4. Artificial Intelligence

Throughout this project, we experimented with the AI tools ChatGPT and Perplexity for both software development and writing. These tools proved beneficial for productivity but also have limitations that require careful oversight.

In coding, AI helped reduce the need to consult documentation. For larger or more complex tasks, however, output was often inconsistent: hallucinated functions, incorrect library calls, or contradictions to explicit instructions were common. AI sometimes reverted to previous prompts, further reducing reliability. As a result, all AI-generated code was carefully reviewed, adapted, or rewritten by us, making large-scale automated code generation impractical. In practice, AI served more as an assistant than a source of reliable final implementations.

For writing, AI accelerated drafting, grammar, and spelling checks. It also helped with restructuring passages, sketching how paragraphs or sections could be organized, and suggesting alternative formulations that improved readability. Occasionally, it highlighted genuine mistakes, though false positives were frequent and required verification. AI was additionally useful for brainstorming, proposing alternative directions, structural outlines, or example formulations that served as starting points for our edits.

Overall, these experiences show that AI tools, while imperfect, can meaningfully enhance productivity and creativity when used critically and with thorough human oversight.

We included two sample prompts demonstrating our use of AI in Appendix C.

7.5. Time Management

We started working on the project in November of 2024, designing the first prototypes of the eccentric drive and beginning testing. In this stage we heavily experimented with the different parameters of the eccentric drive as well as different motors to determine which configuration to use in the final actuators. By March of 2025 we had mostly settled on our configuration and began designing the first four joints of the arm. Around the same time we began development of the wrist. We had planned to develop the software during the study week in May of 2025.

This early momentum gave us confidence that we were well ahead of schedule. However, as we moved from design and prototyping into the software implementation phase, it became clear that we had severely underestimated the time required to develop a fully functioning and integrated system. What had appeared manageable on paper quickly turned into an enormous effort once we began integrating mechanics, electronics and software.

Furthermore the design of the wrist was not nearly ready for use, becoming an additional obstacle. By June of 2025, it was clear that we needed to rethink our approach in light of the apparent issues and reorganized responsibilities based on our individual strengths and weaknesses we had discovered during the process. Andrin took over the design of the wrist and Remi began developing the PCB. This new approach allowed us to work much more efficiently and regain confidence in the project. Nevertheless the software development was still not progressing at the pace it needed to, prompting us to reconsider our initial approach. We decided to switch to ROS 2 and started migrating and adapting the existing code.

Approaching the summer holidays, the firmware for the microcontrollers was largely functional and we ordered the first version of PCB. We spent every free minute we had working on the project. After four weeks of work during the holidays with days regularly

reaching more than 12 hours, the robot had begun taking shape and we were able to see it working for the first time.

In the following weeks we again spent every free minute we had finalizing the design and writing the paper. With around two weeks to go, the moment came when we put everything together in its supposed final form. Although individual components had performed well in isolation, their combination revealed several integration problems and we subsequently came to experience a phenomenon commonly referred to as "Integration Hell". Most devastatingly, encoder communication was rendered impossible by motor EMI.

The following period proved exceptionally stressful. We had invested a great deal of time and effort and were determined not to fail at the final stage. After initially reworking the cable tree and adding shielding, reducing but not eliminating interference, we routed the encoder wiring separately along the outside of the robot, physically separating it from high-current motor cables. These efforts were time-consuming but finally restored reliable encoder signals and allowed us to resolve the last integration problems in time.

Because we set very high expectations for ourselves, we invested far more time than we had originally planned. Countless hours went into meeting our own ambitious goals. While this dedication allowed us to push the project further even when unexpected challenges arose, it also made the workload extremely demanding.

Looking back, we would focus on a smaller key area of interest and aim to complete that within a smaller timeline before considering to expand our work further if time permits. This way we could avoid the stress and increased workload towards the end of the project.

7.6. Division of Work

The project represented a considerable effort and proved challenging in terms of distributing work effectively. At the outset, one team member had prior experience with robotic arms and related systems, combined with a determined vision for the project's direction. This enabled early testing, evaluation, and development of key components, including the four base joint mechanisms with their eccentric drive, associated mathematical modeling, CAD design, and scripting. While this rapid progress was valuable, it also created a noticeable mismatch in both experience and work pace between team members.

In the subsequent phases, differences in familiarity with CAD tools, mechanical design principles, and the amount of time invested in the project led to contributions that were largely uneven. We identified distinct individual strengths: one team member had a clear

advantage in advanced mechanical concepts and mathematics, while the other was more proficient in electronics and circuit design. The resulting imbalance required adjustments in workflow and constructive discussions to clarify responsibilities and expectations.

Through ongoing collaboration and iterative refinement, many of these difficulties were resolved. Specific responsibilities were gradually reassigned according to expertise: one member focused on electronics, wiring, and PCB development, while the other concentrated on mechanical design and software. We found regular check-ins, periods of working together, and joint problem-solving to be essential in coordinating progress and addressing inter-dependencies between hardware and software components.

This process involved challenges and occasional frustration but ultimately showed us the importance of communication, adaptability, and leveraging complementary skills. By the later stages of the project, contributions had become more balanced, with both team members actively engaged in assembly, wiring, PCB redesign, software integration, and documentation. This presented a more sustainable and equitable approach to collaboration, which we should have adopted earlier on to increase our productivity and prevent frustration. In a future project, assigning tasks based on expertise and constant communication could help us to avoid the difficulties we experienced.

7.7. Outlook

We can implement the use of the revised PCB design to streamline our setup and implement additional functionality:

The addition of a relay circuit could address unnecessary heat buildup in the stepper motors. By using a relay board, each motor coil can be shorted when the relay is unpowered, creating a direct path across the coils. This increases the motor's passive resistance, keeping the joints stable without continuous holding current and preventing overheating. Energizing the relays under firmware control restores normal operation. An additional advantage of this approach is that it prevents joint sagging when power is removed, eliminating the risk of uncontrolled motion due to gravity. In the second revision, a connector with a dedicated pin from the first four ESP32s facilitates connection to the relay board.

A natural next step for the robotic arm is to expand the range of available end-effectors. Thanks to the integrated toolchanger, specialized tools such as screwdrivers, welding heads, or suction cups could be added with minimal modification, increasing system versatility and enabling applications beyond simple pick-and-place tasks.

Another promising direction is to equip the robot with a camera system and learning-based algorithms. This would enable object recognition, environmental awareness, and

adaptive trajectory planning, allowing the arm to identify, track, and manipulate objects autonomously while continuously improving performance in unstructured or changing environments.

References

[1] Wikimedia Foundation, Classic DH Parameters Convention, Accessed: 2025-12-9, **2020**, https://commons.wikimedia.org/wiki/File:Classic_DH_Parameters_Convention.png.

[2] S. Asif, P. Webb, "Kinematics Analysis of 6-DoF Articulated Robot with Spherical Wrist", *Mathematical Problems in Engineering* **2021**, 6647035, DOI 10.1155/2021/6647035, https://onlinelibrary.wiley.com/doi/abs/10.1155/2021/6647035.

[3] S. R. Buss, Introduction to Inverse Kinematics with Jacobian Transpose, Pseudoinverse and Damped Least Squares Methods, tech. rep., Accessed: 2025-09-19, University of California, San Diego, **2009**, https://www.cs.cmu.edu/~15464-s13/lectures/lecture6/iksurvey.pdf.

[4] D. L. Pieper, PhD thesis, Stanford University, **1968**, https://archive.org/stream/DTIC_AD0680036.

[5] Robotic Systems Lab, ETH Zurich, Robot Dynamics Lecture Notes, Accessed: 2025-09-20, **2017**, https://ethz.ch/content/dam/ethz/special-interest/mavt/robotics-n-intelligent-systems/rsl-dam/documents/RobotDynamics2017/RD_HS2017script.pdf.

[6] Universal Robots, UR3e Collaborative Robot, Accessed: 2025-08-16, https://www.universal-robots.com/products/ur3-robot/.

[7] Robotastic, 6R-Roboter, Accessed: 2025-08-16, https://youtube.com/playlist?list=PLR1TMhyq3UhvNwTSHJ-CX06DgdUD9nVTz&si=m9N3mZB22kMod3wM.

[8] Arctos Robotics, Create your own robotic arm from scratch, Accessed: 2025-08-16, https://arctosrobotics.com/.

[9] P. Crnjak, Differential robot wrist, Accessed 2025-09-20, https://www.printables.com/model/918284-differential-robot-wrist.

[10] P. Crnjak, SSG48 adaptive electric gripper, Accessed: 2025-08-16, https://github.com/PCrnjak/SSG-48-adaptive-electric-gripper.

[11] ATI Industrial Automation, Robotic Tool Changer, Accessed: 2025-08-16, https://www.ati-ia.com/Products/toolchanger/robot_tool_changer.aspx.

[12] S. Mishin, Wave Drive with Rolling Elements — Cycloidal Gear Alternative!, Accessed: 2025-08-16, **2024**, https://www.youtube.com/watch?v=zOLQw-TxE7s&t=17s.

[13] S. Fritsch, S. Landler, M. Otto, B. Vogel-Heuser, M. Zimmermann, K. Stahl, "Discussion of a variant of eccentric drives utilizing rolling elements", *Forschung im Ingenieurwesen* **2023**, *87*, 1221–1230, ISSN: 1434-0860, DOI 10.1007/s10010-023-00688-1, https://doi.org/10.1007/s10010-023-00688-1.

[14] I. Gibson, D. Rosen, B. Stucker, M. Khorasani, *Additive Manufacturing Technologies*, Springer Nature Switzerland AG, **2021**, https://link.springer.com/book/10.1007/978-3-030-56127-7.

[15] J. Suder, Z. Bobovsky, M. Safar, J. Mlotek, M. Vocetka, Z. Zeman, "Experimental Analysis of Temperature Resistance of 3D Printed PLA Components", *MM Science Journal* **2021**, 4322–4327, DOI 10.17973/MMSJ.2021_03_2021004, https://www.mmscience.eu/journal/issues/march-2021/articles/experimental-analysis-of-temperature-resistance-of-3d-printed-pla-components/download.

[16] C. Bhamra, Comparative Study: Stepper Motors vs. Brushless DC Motors, Accessed: 2025-08-21, https://www.ezmotion.co/comparative-study-stepper-motors-vs-brushless-dc-motors.

[17] Changzhou Oukeda Electric Appliance, 57H1876-420-8-21B Stepper Motor Datasheet, Accessed: 2025-09-14, https://www.mini-tech.com.ua/image/catalog/cnc/57bygh76-dimensions-11.jpg.

[18] SIMAC Electronics GmbH, NEMA23-03 Bipolar Stepper Motor Datasheet, Accessed: 2025-08-16, **2021**, https://joy-it.net/files/files/Produkte/NEMA23-03/NEMA23-03_Datasheet.pdf.

[19] Shenzhen Leadshine Control Technology, CM Series Stepper Motor Datasheetc, Accessed: 2025-09-14, https://www.leadshine.com/upfiles/downloads/d6a4181edd0c406edd6017264d3b5099_1662082534844.pdf.

[20] AMS AG, AS5600 Magnetic Rotary Position Sensor Datasheet, Accessed: 2025-08-16, **2018**, https://look.ams-osram.com/m/7059eac7531a86fd/original/AS5600-DS000365.pdf.

[21] Allegro MicroSystems, 3141 THRU 3144 Datasheet, Accessed: 2025-08-30, **2005**, https://www.elecrow.com/download/A3141-2-3-4-Datasheet.pdf.

[22] C. W. Musser, *US Patent*, US2906143A, **1959**, https://patents.google.com/patent/US2906143A/en.

[23] R. August, R. Kasuba, J. J. Coy, Dynamics of Planetary Gear Trains, tech. rep., National Aeronautics and Space Administration, **1984**, https://ntrs.nasa.gov/api/citations/19840017959/downloads/19840017959.pdf.

[24] L. K. Braren, *US Patent*, US1817405A, **1931**, https://patents.google.com/patent/US1817405A/en.

[25] Linear Motion Tips, Microstepping for Stepper Motors, Accessed: 2025-09-14, **2025**, https://www.linearmotiontips.com/microstepping-basics/.

[26] TOSHIBA, TB6600HG Datasheet, Accessed: 2025-08-21, **2016**, https://toshiba.semicon-storage.com/info/TB6600HG_datasheet_en_20160610.pdf?did=14683&prodName=TB6600HG.

[27] TOSHIBA, TB6612FNG Datasheet, Accessed: 2025-08-21, **2012**, https://cdn-shop.adafruit.com/datasheets/TB6612FNG_datasheet_en_20121101.pdf.

[28] Espressif Systems, ESP32-S3 Datasheet, Accessed: 2025-08-21, **2025**, https://www.espressif.com/sites/default/files/documentation/esp32-s3-wroom-1_wroom-1u_datasheet_en.pdf.

[29] Advanced Monolithic Systems, AMS1117 Datasheet, Accessed: 2025-08-21, http://www.advanced-monolithic.com/pdf/ds1117.pdf.

[30] Espressif Systems, Introduction to the ESP-Prog Board, Accessed: 2025-09-14, https://docs.espressif.com/projects/esp-iot-solution/en/latest/hw-reference/ESP-Prog_guide.html.

[31] Dongguan Qixin Electronics, FE1.1s Datasheet, Accessed: 2025-08-21, **2008**, https://cdn-shop.adafruit.com/product-files/2991/FE1.1s%20Data%20Sheet%20(Rev.%201.0).pdf.

[32] Nanjing Qinheng Microelectronics, CH340 Datasheet, Accessed: 2025-08-21, https://www.mpja.com/download/35227cpdata.pdf.

[33] STMicroelectronics, USBLC6-2SC6 Datasheet, Accessed: 2025-08-21, **2021**, https://www.st.com/content/ccc/resource/technical/document/datasheet/06/1d/48/9c/6c/20/4a/b2/CD00050750.pdf/files/CD00050750.pdf/jcr:content/translations/en.CD00050750.pdf.

[34] D. Marrakchi, Enhancing Signal Integrity in PCB Design: Key Considerations and Strategies, Accessed: 2025-09-14, **2024**, https://resources.altium.com/p/enhancing-signal-integrity-in-pcb-design-key-considerations-and-strategies.

[35] Open Source Robotics Foundation, Robot Operating System (ROS), Accessed: 2025-08-21, https://www.ros.org/.

[36] micro-ROS Project, micro-ROS: ROS for Microcontrollers, Accessed: 2025-08-21, https://micro.ros.org/.

[37] micro-ROS Project, micro-ROS ESP32 IDF Component and Sample Code, https://github.com/micro-ROS/micro_ros_espidf_component.

[38] Open Source Robotics Foundation, Welcome to the ros2_control documentation - Jazzy!, Accessed: 2025-09-21, https://control.ros.org/jazzy/index.html.

[39] Rhoban, Onshape to Robot (URDF, SDF, MuJoCo), https://github.com/Rhoban/onshape-to-robot.

[40] Open Source Robotics Foundation, joint_state_broadcaster User Documentation, Accessed: 2025-08-30, https://control.ros.org/jazzy/doc/ros2_controllers/joint_state_broadcaster/doc/userdoc.html.

[41] Open Source Robotics Foundation, robot_state_publisher: ROS 2 Jazzy Documentation, Accessed: 2025-08-30, https://docs.ros.org/en/jazzy/p/robot_state_publisher/.

[42] Open Source Robotics Foundation, RViz Tutorial — ROS 2 Jazzy Documentation, Accessed: 2025-08-30, https://docs.ros.org/en/jazzy/Tutorials/Intermediate/RViz/RViz-Main.html.

[43] Espressif Systems, General Purpose Timer (GPTimer), Accessed: 2025-08-21, https://docs.espressif.com/projects/esp-idf/en/v5.2.5/esp32s3/api-reference/peripherals/gptimer.html.

[44] Espressif Systems, Motor Control Pulse Width Modulator (MCPWM), Accessed: 2025-08-21, https://docs.espressif.com/projects/esp-idf/en/v5.2.5/esp32s3/api-reference/peripherals/mcpwm.html.

[45] National Instruments, The PID Controller & Theory Explained, Accessed: 2025-08-30, **2025**, https://www.ni.com/en/shop/labview/pid-theory-explained.html.

[46] P. Nachtwey, Feed forwards augment PID control, Accessed: 2025-08-30, **2015**, https://www.controleng.com/feed-forwards-augment-pid-control/.

[47] IACS Engineering, Integral Windup, Accessed: 2025-08-30, https://iacsengineering.com/integral-windup/.

[48] Espressif Systems, LED Control (LEDC), Accessed: 2025-08-30, https://docs.espressif.com/projects/esp-idf/en/v5.2.5/esp32s3/api-reference/storage/nvs_flash.html.

[49] Bonsystems, Backlash and Backdrive: What Gear Design Really Needs, Accessed: 2025-09-14, **2025**, https://en.bonsystems.com/gear-backlash/.

[50] Farm-ng, Transforms & Poses, Accessed: 2025-09-14, https://amiga.farm-ng.com/docs/concepts/transforms_and_poses/.

[51] Universal Robots, What is a singularity?, Accessed: 2025-09-14, https://www.universal-robots.com/articles/ur/application-installation/what-is-a-singularity/.

List of Figures

Figures without explicit source references were created by the authors.

List of Tables

All tables were created by the authors.

Declaration of Honesty in Academic Work

We hereby declare

- that this Matura Thesis is our own work, and that we did not use any other sources than the cited ones,
- that we explicitly mention any help by third party,
- that we disclose any use of artificial intelligence,
- that we will inform the school management as well as our advisor in case that we
  - publish this entire thesis or parts of it,

  or
  - hand out copies of this thesis to third party for further distribution.
- that we are aware of the regulations for prevention of plagiarism and also of the consequences of plagiarism.

Our Matura Thesis contains 79'187 characters (without white space, front page, table of contents, declaration of honesty and hand-in information, appendix, bibliography, and any other lists).

We hereby declare the use of ChatGPT and Perplexity throughout the project to assist in coding, writing and proofreading.

Together with the Matura Thesis copies, we hand in the following objects or data storage devices:

USB flash drive with our digital files

Lucerne, 26. September 2025

Andrin Winzap and Remi Gerber

Appendix

Glossary

backdriving Backdriving refers to the ability of an external force to move the motor or actuator shaft, allowing manual motion of the mechanism without using the motor's own power [49]. 49

backlash The clearance or lost motion in a mechanical system caused by gaps between mating components, such as gears or screw threads; backlash can lead to positioning errors and reduced precision in robotic mechanisms [49]. 15, 20, 21, 23, 50

ESP32 A family of low-cost, widely used 32-bit microcontrollers developed by Espressif Systems, featuring wireless connectivity, multiple peripherals, and strong community support. [28]. 26, 27, 28, 29, 30, 31, 32, 35, 36, 37, 39, 42, 44, 46, 48, 49, 50, 54

pose The position and orientation of a robot or its end-effector in space, usually represented as a combination of a 3D position vector and a rotation matrix or quaternion [50]. 6, 7, 44, 45

singularity A configuration of a robot manipulator where it loses one or more degrees of freedom [51]. 6

Acronyms

API Application Programming Interface. 41, 44, 45

AWG American Wire Gauge. 38

CAD Computer-Aided Design. 11, 18, 38, 42, 53

CAN Controller Area Network. 51

Cat6a S/FTP Category 6a Shielded Foiled Twisted Pair. 38, 39

CC Configuration Channel. 32

DC Direct Current. 13, 14, 27, 30, 39, 46, 49

DH Denavit–Hartenberg. 4, 5

DoF Degree of Freedom. 1, 3, 42

EMI Electromagnetic Interference. 38, 51, 53

ESD Electrostatic Discharge. 32, 35, 36, 37

ESP-IDF Espressif IoT Development Framework. 41, 42, 45, 48

FDM Fused Deposition Modeling. 12, 21

GND Ground. 29, 32, 33, 35, 36, 37, 38

GPIO General-Purpose Input/Output. 30, 33, 35, 37

HTM Homogeneous Transformation Matrix. 5

IC Integrated Circuit. 27, 32, 36, 37

IO Input/Output. 50

I²C Inter-Integrated Circuit. 15, 38, 39, 46, 50, 51

JTAG Joint Test Action Group. 31, 33, 35, 37

LDO Low Dropout Regulator. 30

Micro-ROS Micro Robot Operating System. 31, 35, 37, 41, 42, 48

NEMA National Electrical Manufacturers Association. 14, 19

PCB Printed Circuit Board. 33, 35, 36, 37, 38, 39, 50, 54

PID Proportional-Integral-Derivative. 26, 44, 47, 48, 49

PLA Polylactic Acid. 12

PSRAM Pseudo-Static Random-Access Memory. 26

PWM Pulse Width Modulation. 46, 49

ROS 2 Robot Operating System 2. 3, 41, 43, 52

RS-485 Recommended Standard 485. 51

RX Receive. 35

SMD Surface-Mount Device. 33, 35

THT Through-Hole Technology. 33

TX Transmit. 35

UART Universal Asynchronous Receiver/Transmitter. 26, 29, 31, 33, 35, 37, 50

URDF Unified Robot Description Format. 42, 43

VCC Voltage Common Collector. 34

A. Project Resources

A.1. Source Code

Main Git Repository: https://github.com/andrinwinzap/robot

Stepper Motor Actuator: https://github.com/andrinwinzap/robot-stepper-actuator

Differential Wrist: https://github.com/andrinwinzap/robot-differential-wrist

Toolchanger: https://github.com/andrinwinzap/robot-toolchanger

Gripper: https://github.com/andrinwinzap/robot-gripper

PID Controller Component: https://github.com/andrinwinzap/pid_espidf_component

Encoder Component: https://github.com/andrinwinzap/as5600_espidf_component

A.2. 3D Model

Onshape Document: https://cad.onshape.com/documents/35244f3032e1dd11e6e40d38/w/ff4e0b78bc305b470c9c54d8/e/67b9fda3e47a6e7d889f8684

A.3. Electronics

Electronics: https://github.com/RemiGerber/robot-electronics

B. Software Diagrams

B.1. Software Architecture Overview

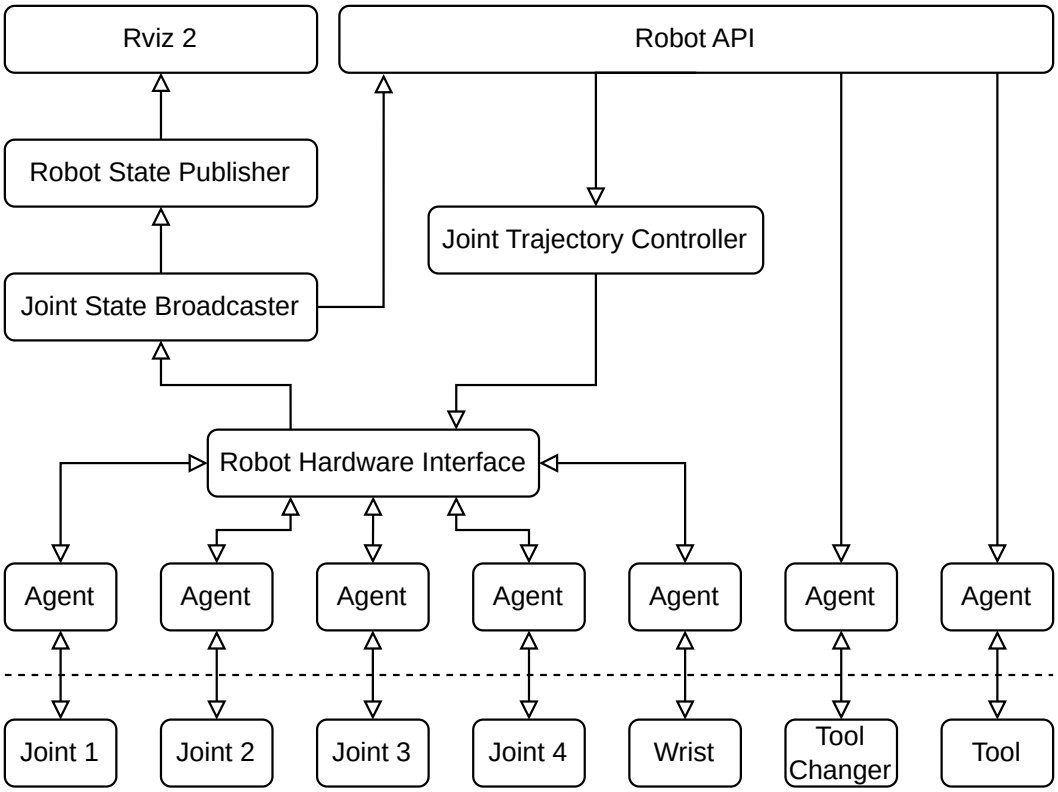


Figure 29: Software architecture overview

B.2. Proportional-Integral-Derivative Controller

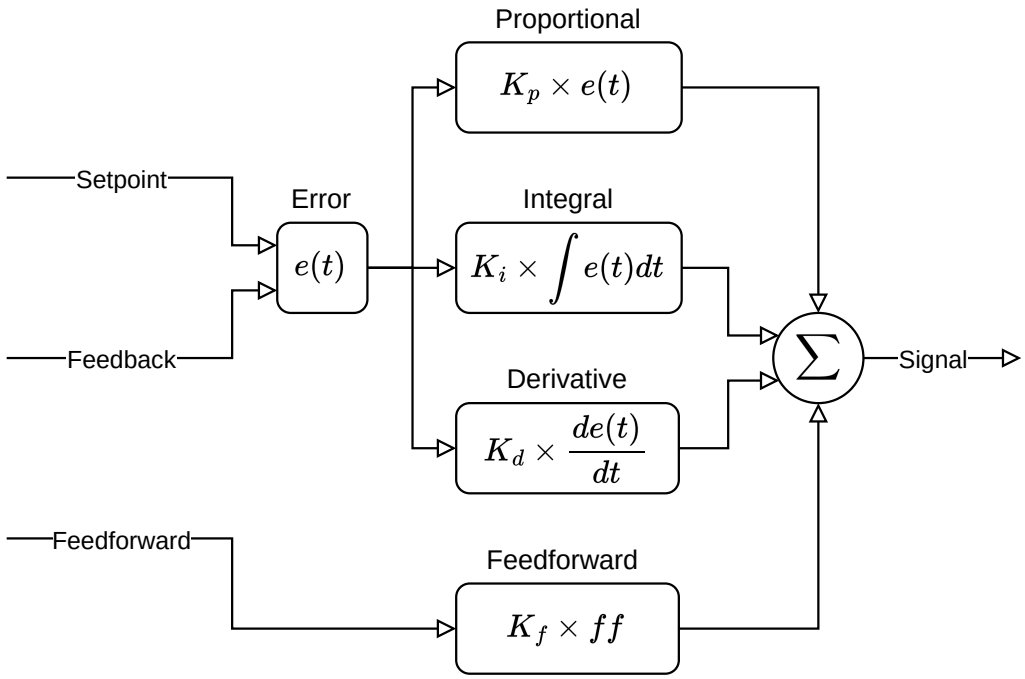


Figure 30: Proportional-Integral-Derivative controller

B.3. Stepper Motor Actuator

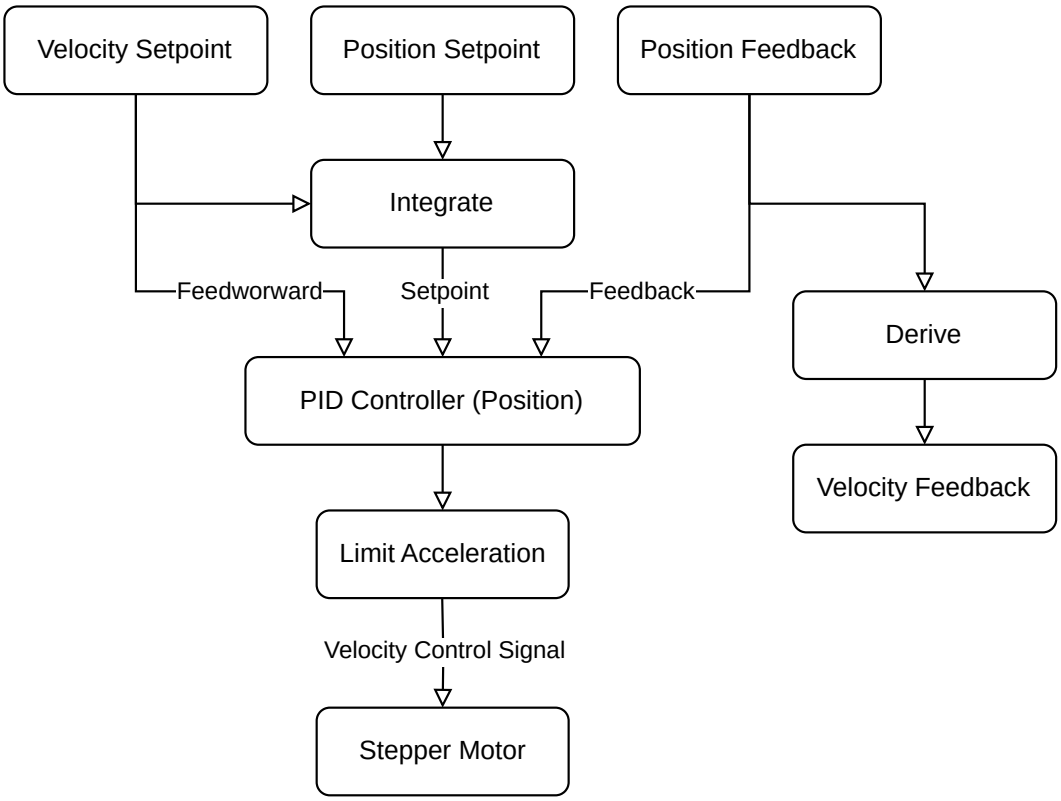


Figure 31: Control logic for a stepper motor

B.4. Wrist

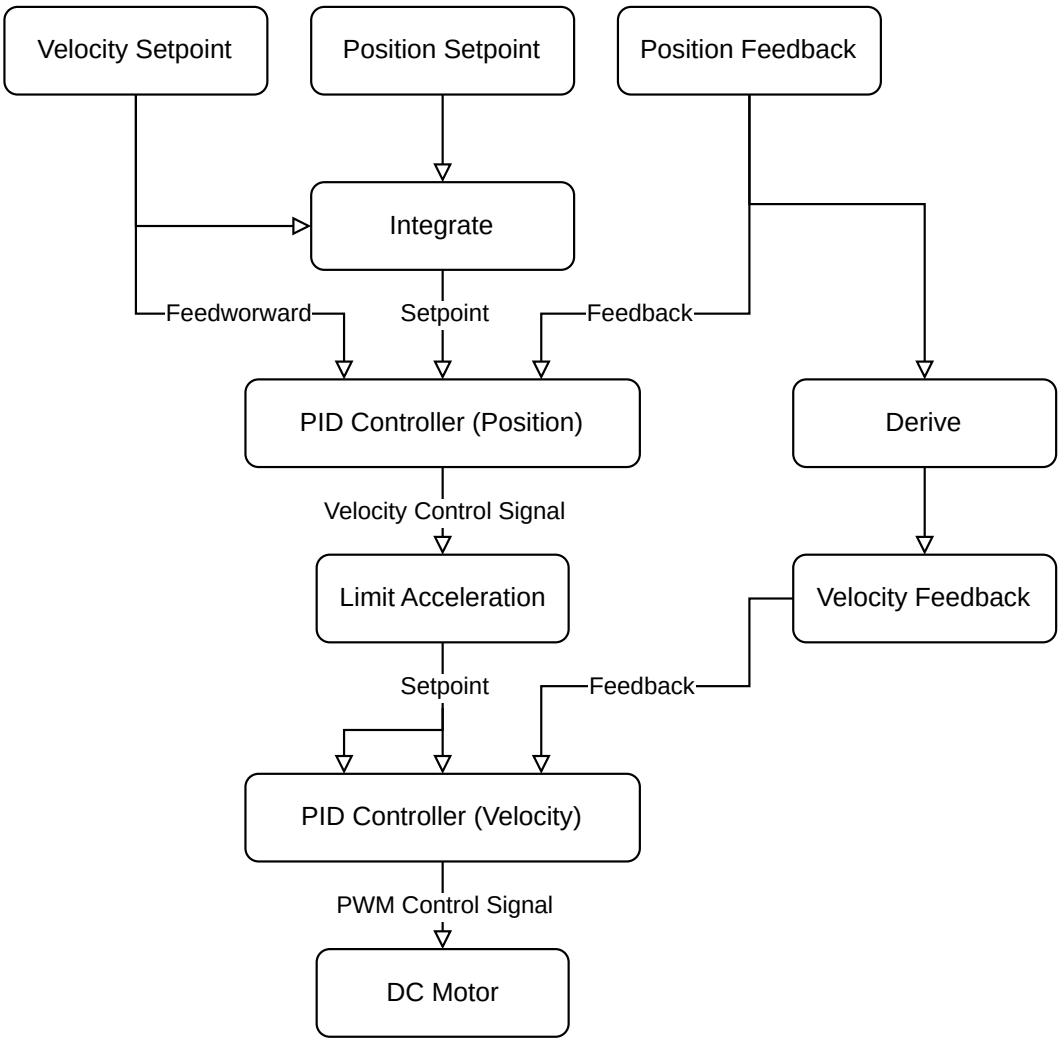


Figure 32: Control logic for a single wrist axis

C. Artificial Intelligence Prompts

C.1. Sample prompt during coding

"Can you outline the basics of writing firmware in C for controlling a single servo on an ESP32 using ESP-IDF? I want the outline to focus on servo control only, including the key ESP-IDF APIs such as LEDC, initializing the servo, setting angles, and any recommended timing or update considerations. Please keep it focused and practical for direct implementation."

C.2. Sample prompt during writing

"Please review the following LaTeX document draft, checking the text for consistency of style and tone, word choice and clarity, spelling and grammar errors, clumsy or overly long sentences that could be rephrased more elegantly, and any obvious content-related mistakes (e.g. inconsistencies, redundancies). It is a high school graduation paper."

D. Photos

Figure 33: Front View of the Robot

Figure 34: Full View of the Robot

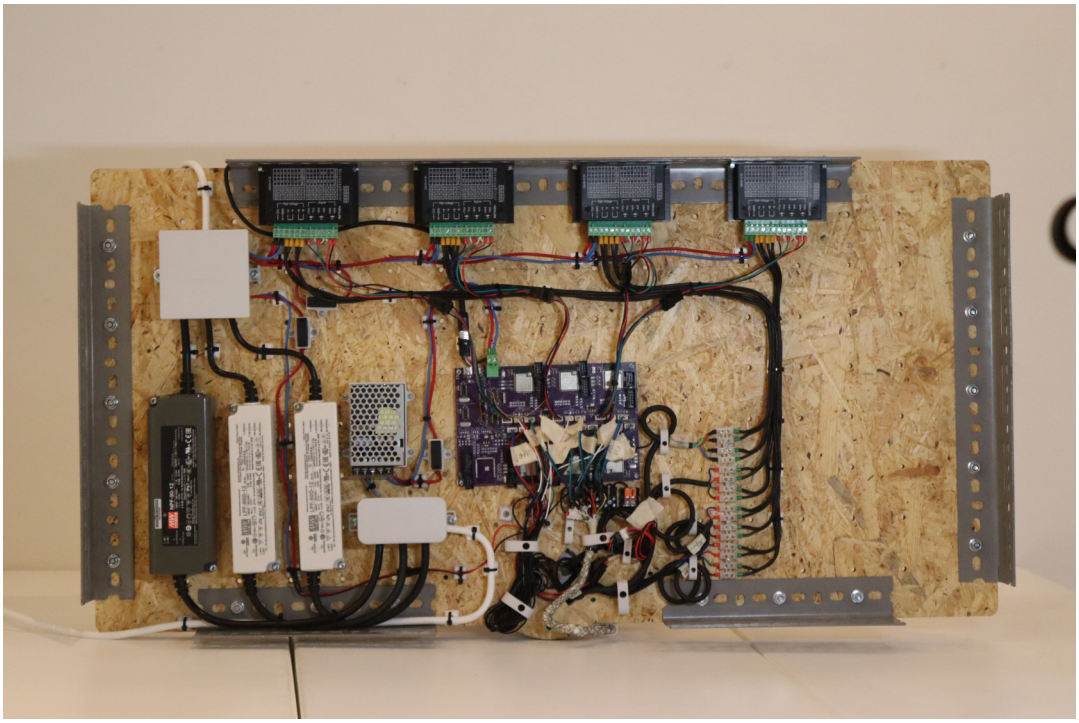


Figure 35: Bottom of the robot

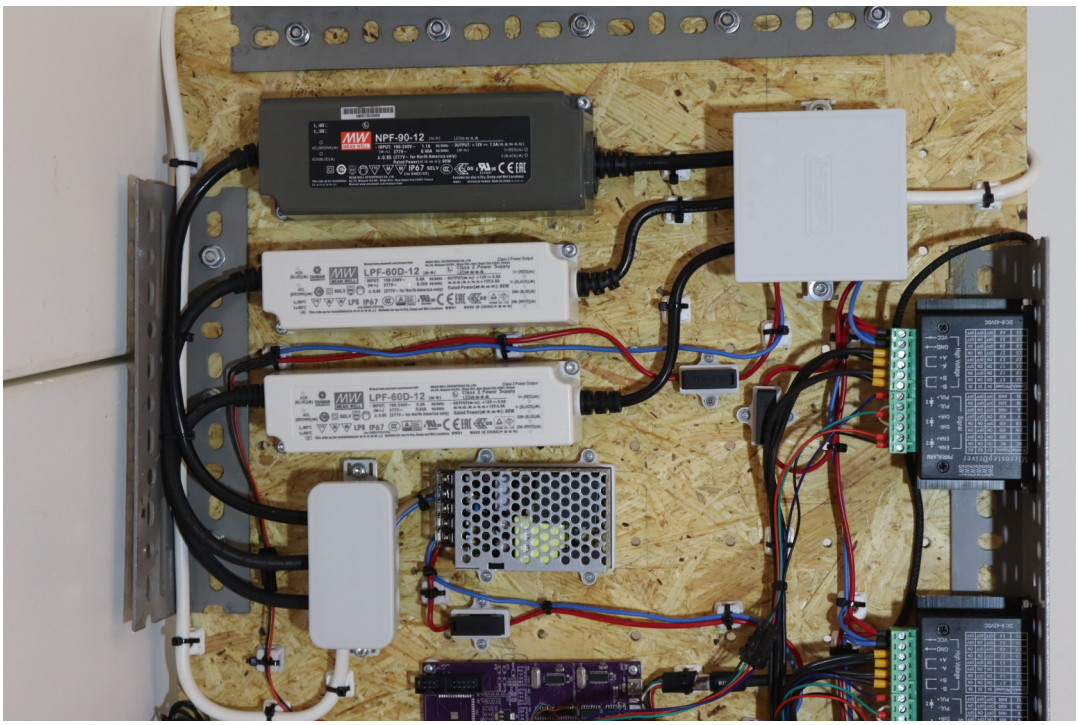


Figure 36: Power supplies of the robot

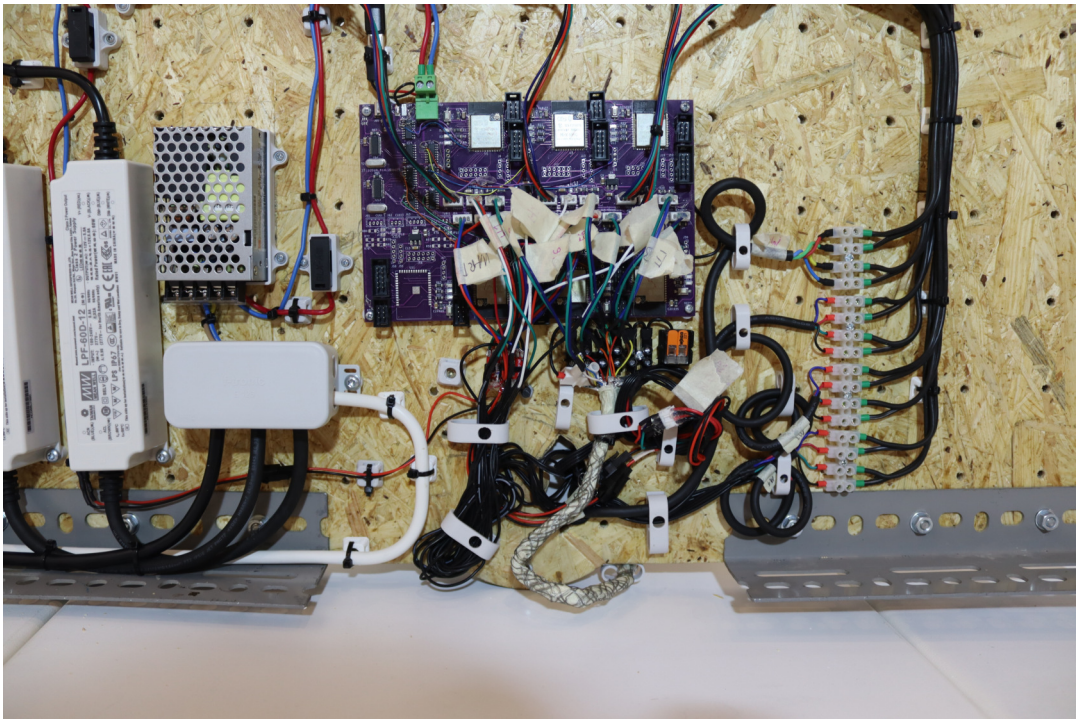
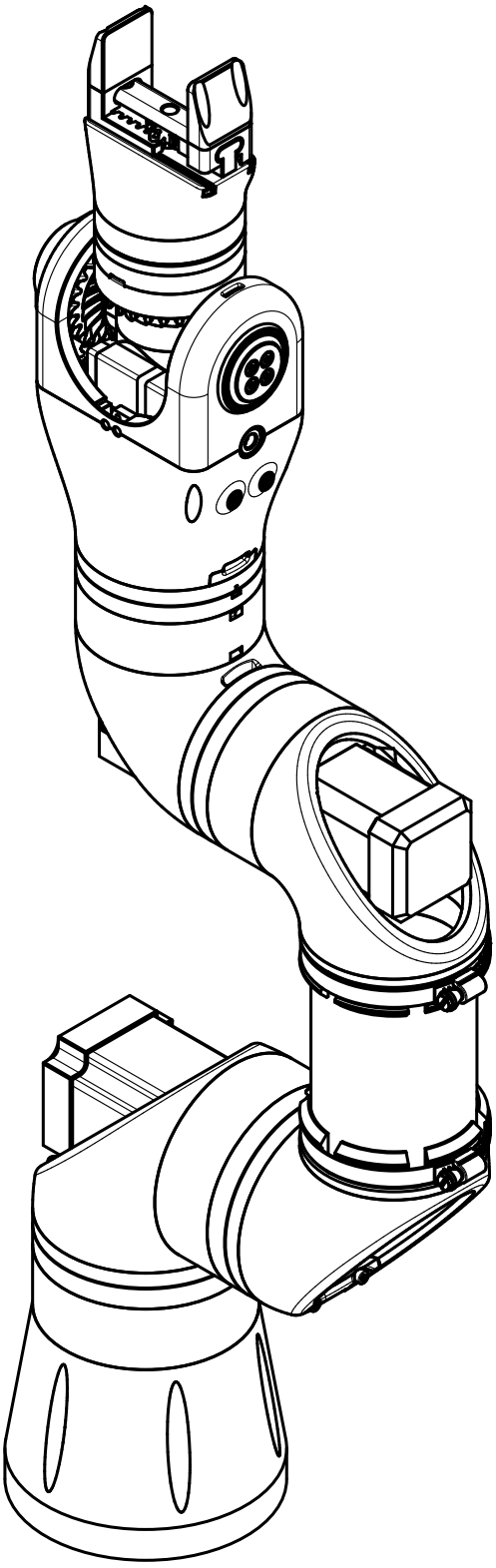


Figure 37: Printed Circuit Board

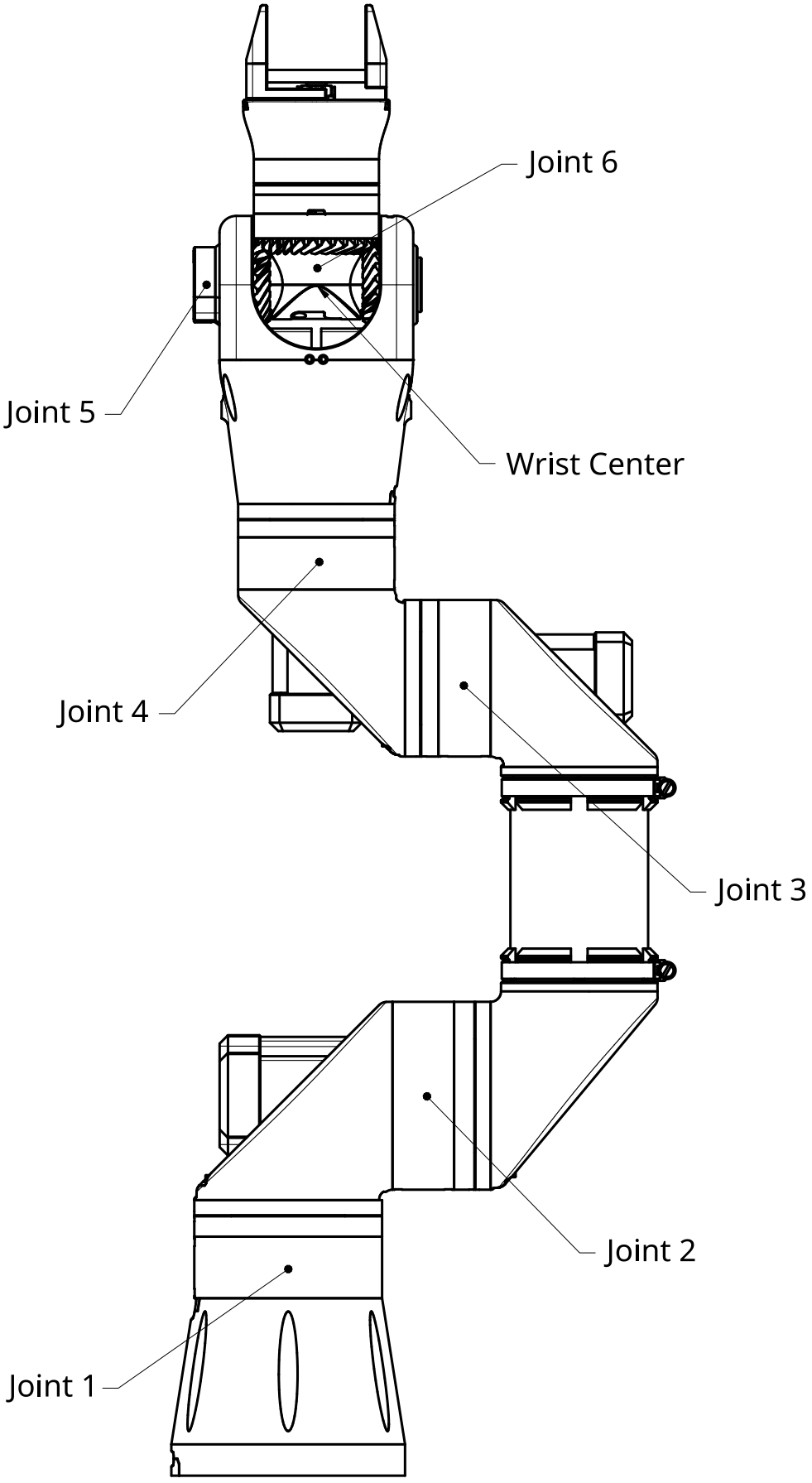
E. Technical Drawing



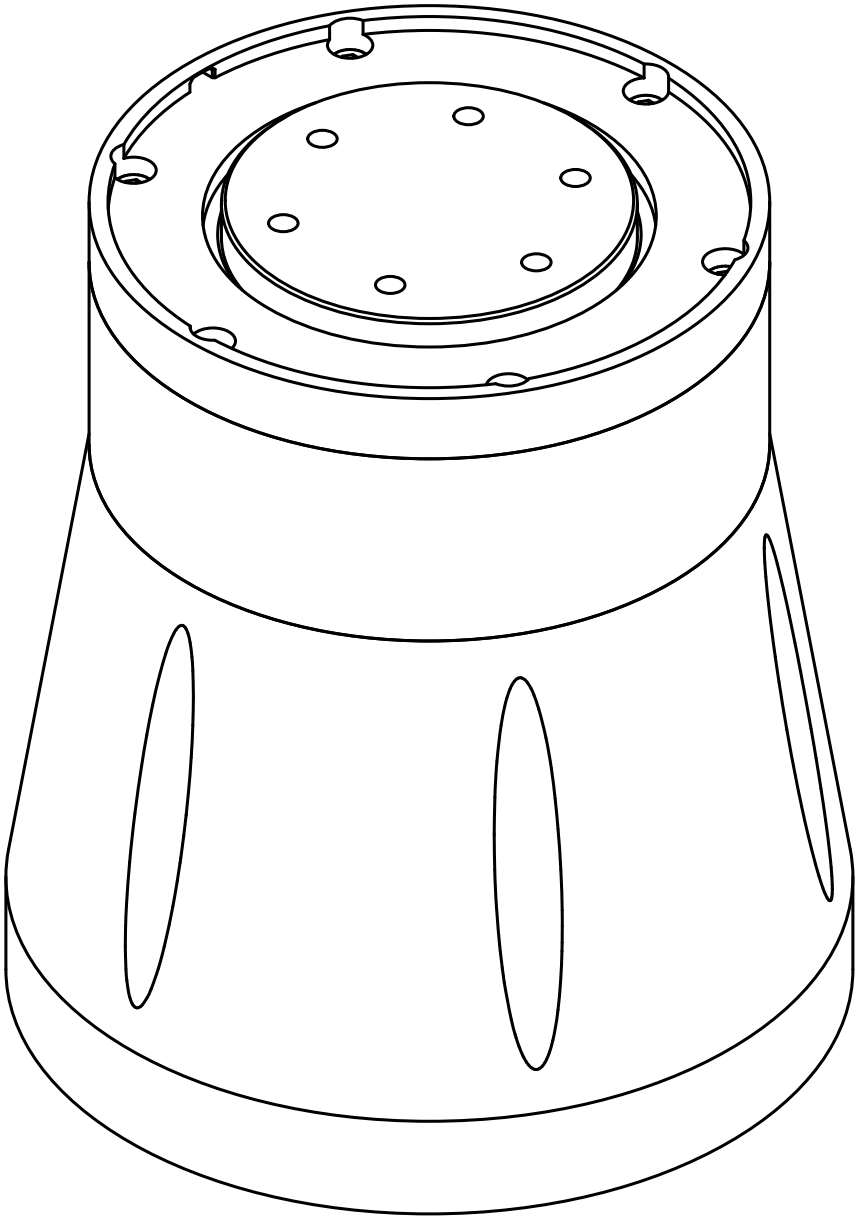
UNLESS OTHERWISE SPECIFIED, DIMENSIONS ARE IN MILLIMETERS		NAME	SIGNATURE	DATE
ANGULAR = ± °	DRAWN	ANDRIN		2025-08-28
SURFACE FINISH	CHECKED			
	APPROVED			
DO NOT SCALE DRAWING				
BREAK ALL SHARP EDGES AND REMOVE BURRS				
FIRST ANGLE PROJECTION	MATERIAL		FINISH	

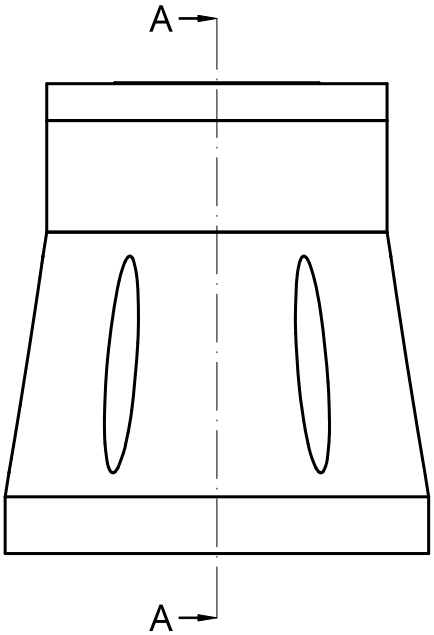
TITLE

SIZE A4	DWG NO.	REV.
SCALE 1:3	WEIGHT	SHEET 1 of 11

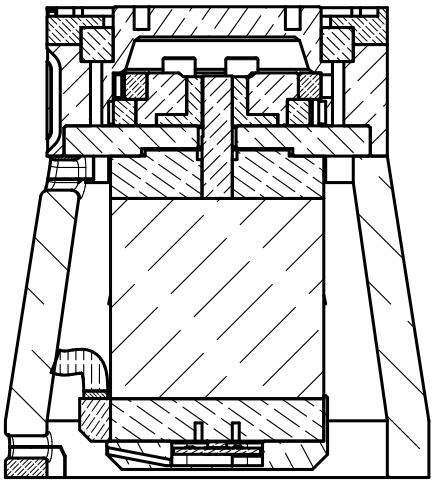


DWG NO.	SIZE	SCALE	SHEET	REV.
	A4	1:3	2 of 11	



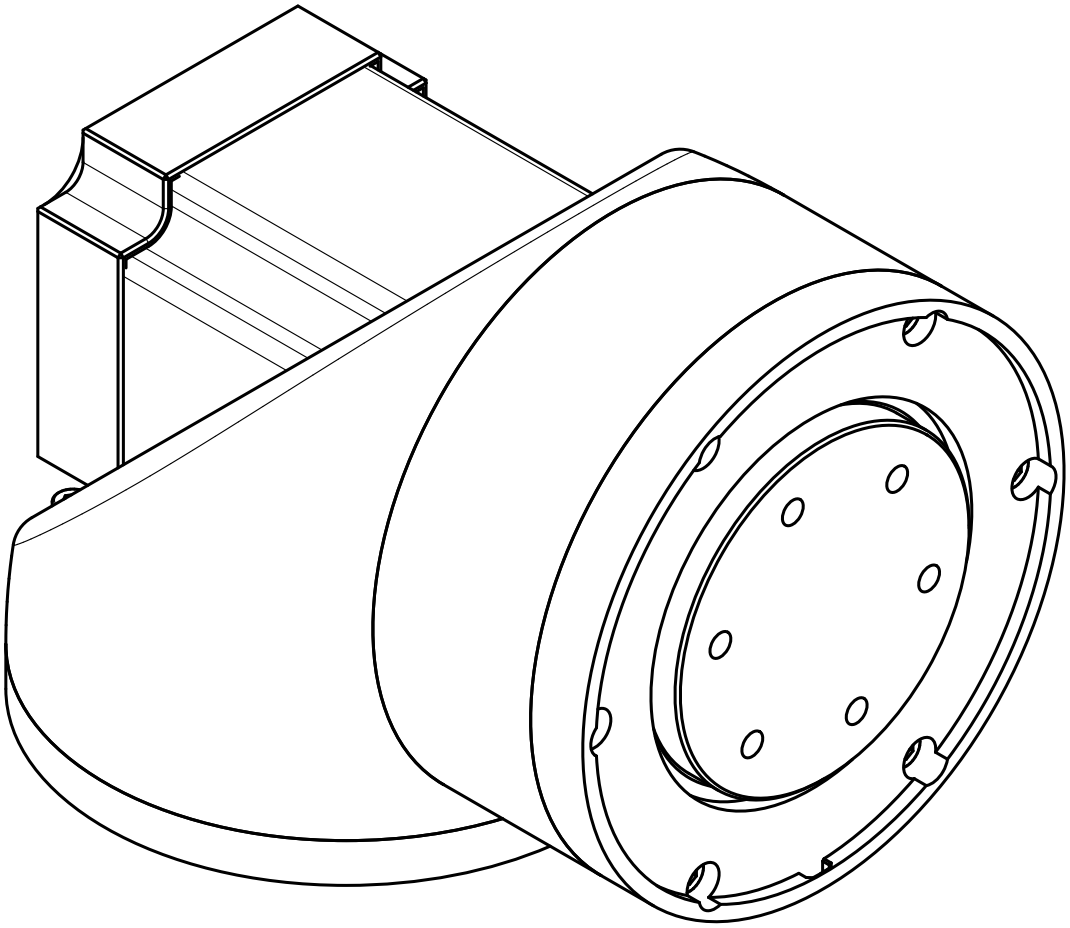


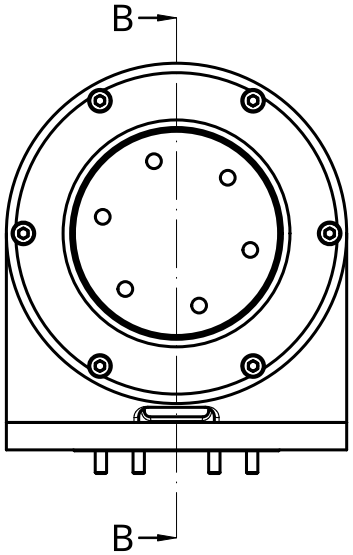
1:2



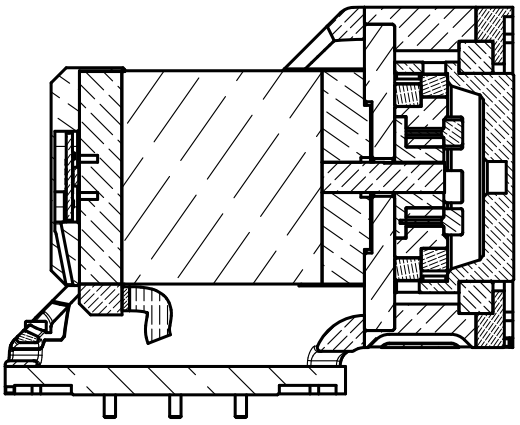
A - A
1:2

DWG NO.	SIZE	SCALE	SHEET	REV.
	A4	1:1	3 of 11	



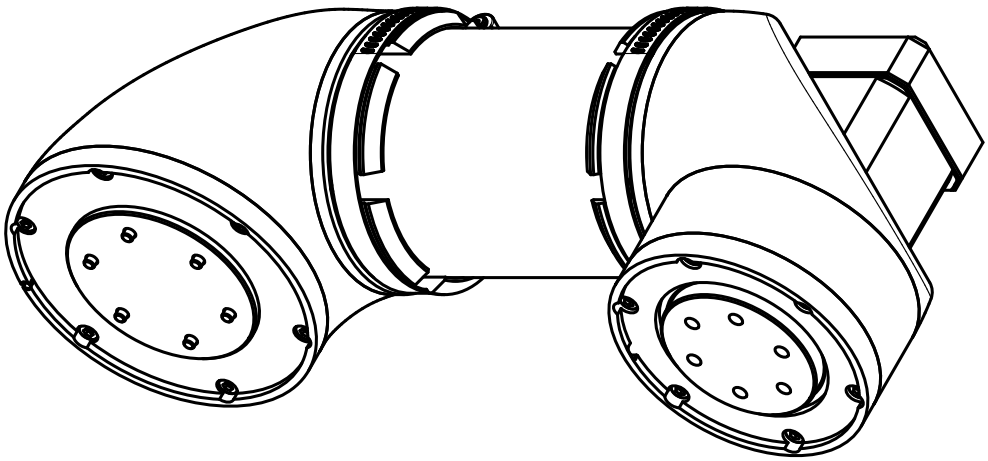


1:2

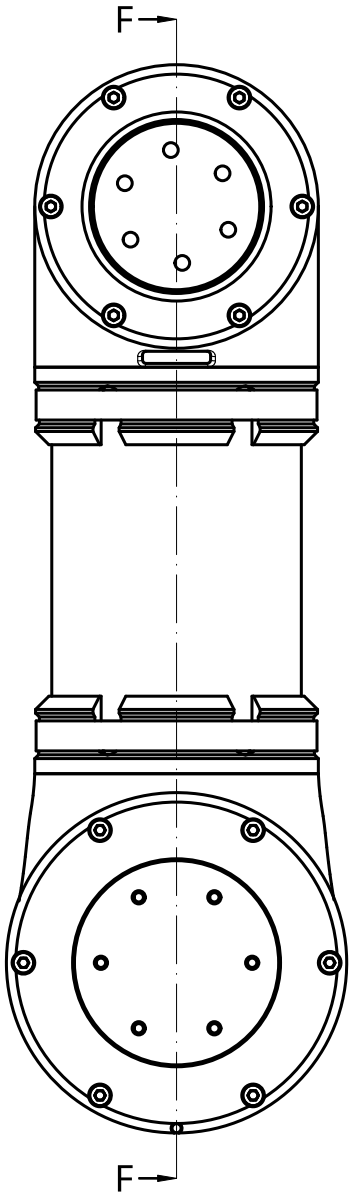


B - B
1:2

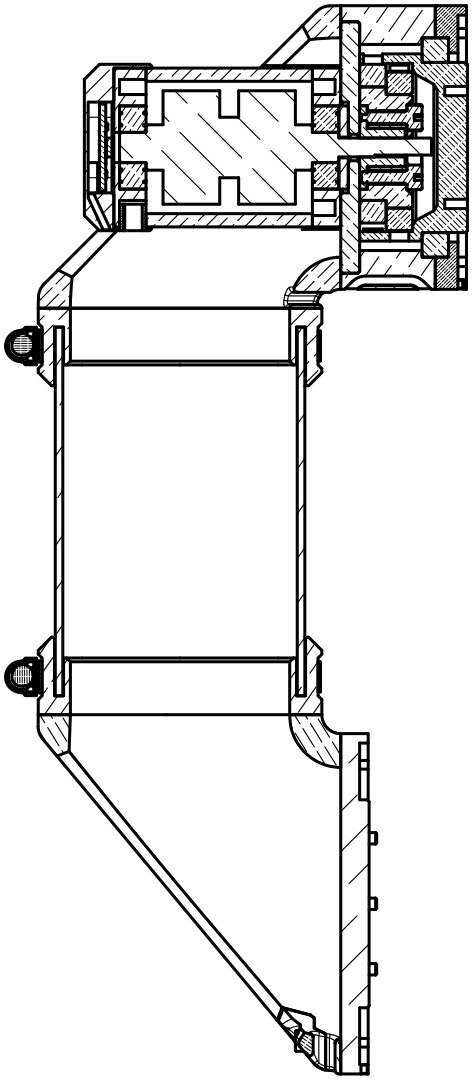
DWG NO.	SIZE	SCALE	SHEET	REV.
	A4	1:1	4 of 11	



1:2

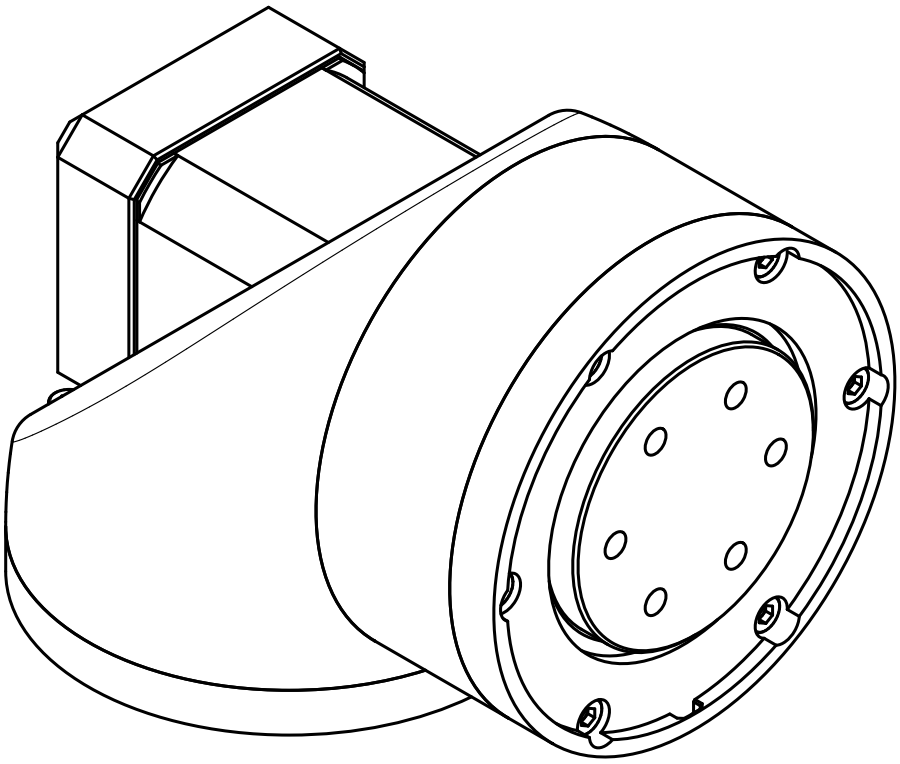


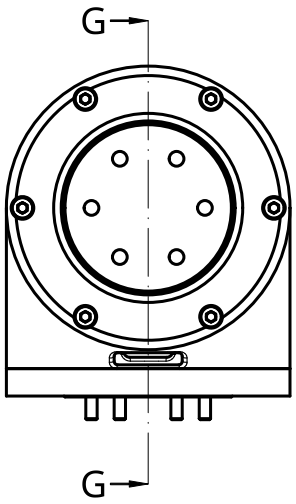
1:2



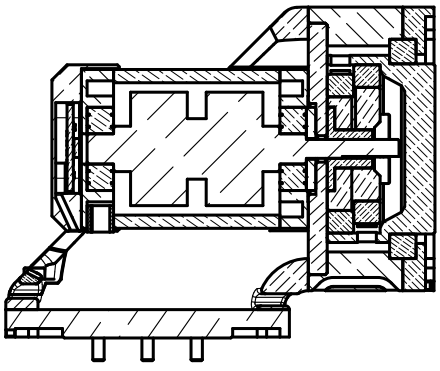
F - F
1:2

DWG NO.	SIZE	SCALE	SHEET	REV.
	A4	1:1	5 of 11	



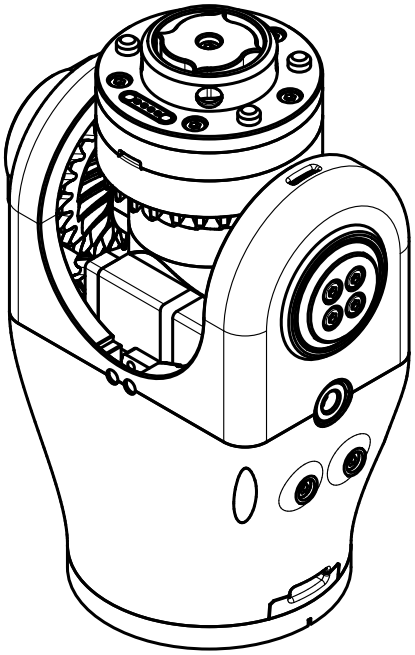


1:2

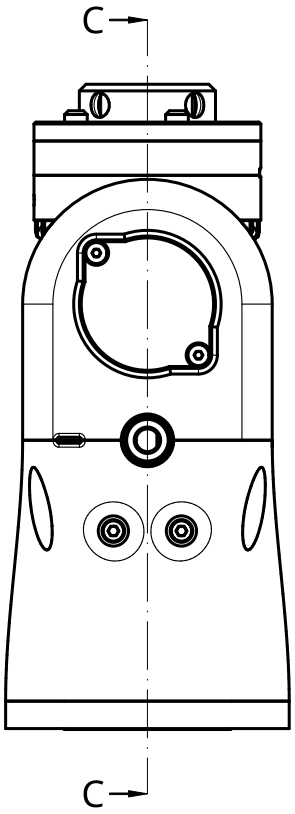


G - G
1:2

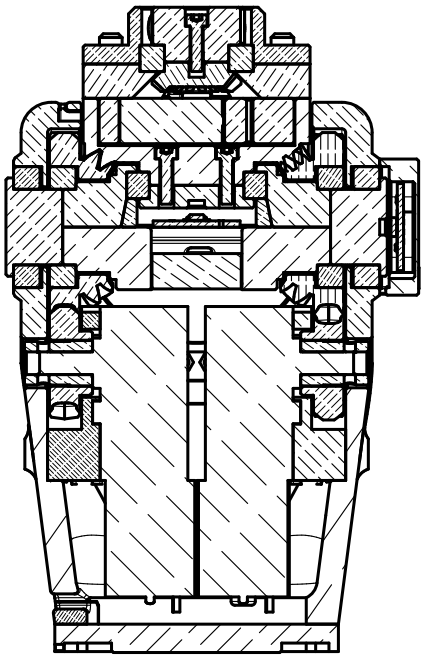
DWG NO.	SIZE	SCALE	SHEET	REV.
	A4	1:1	6 of 11	



1:2

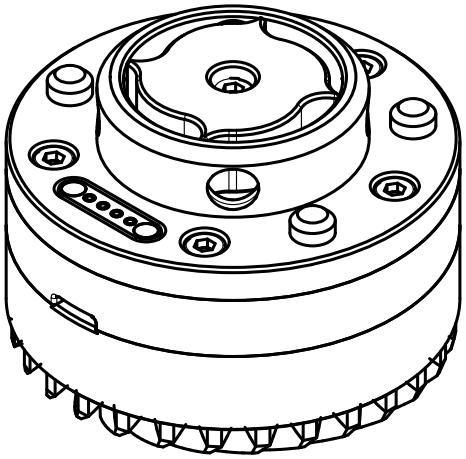


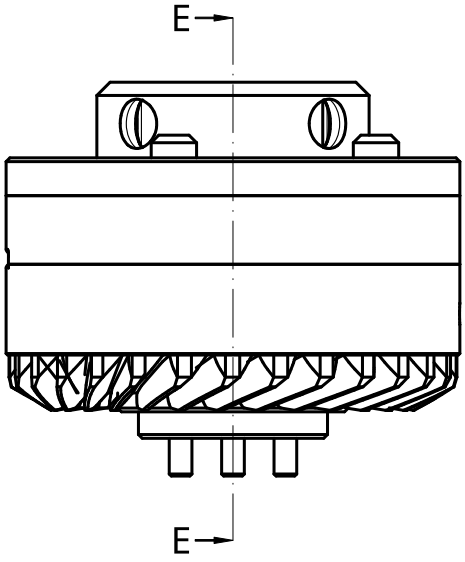
1:2

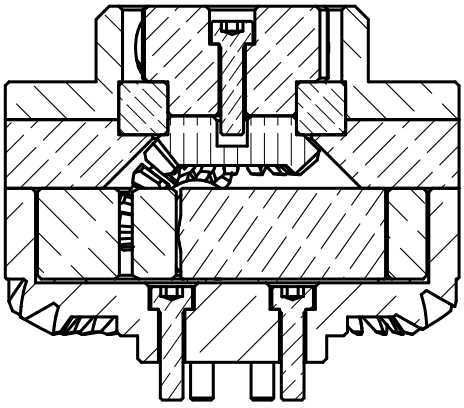


C - C
1:2

DWG NO.	SIZE	SCALE	SHEET	REV.
	A4	1:1	7 of 11	

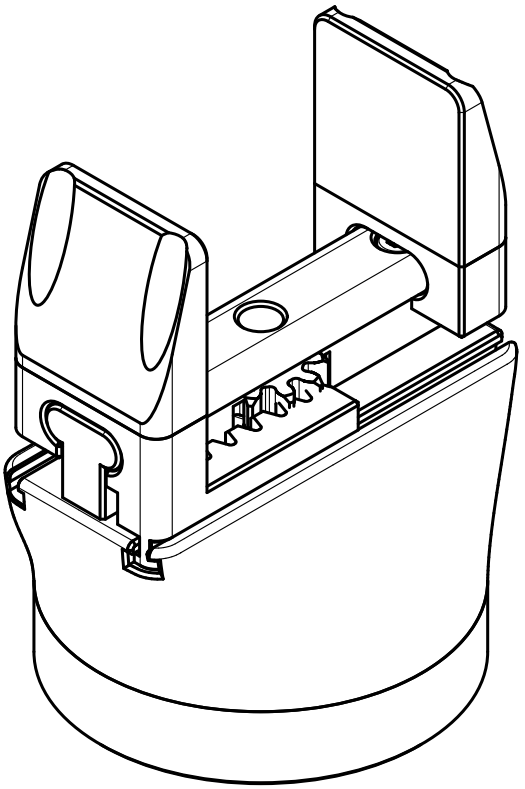


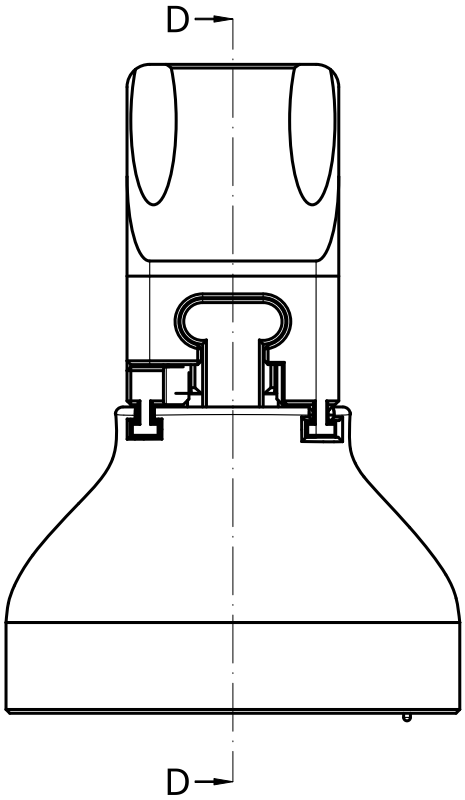


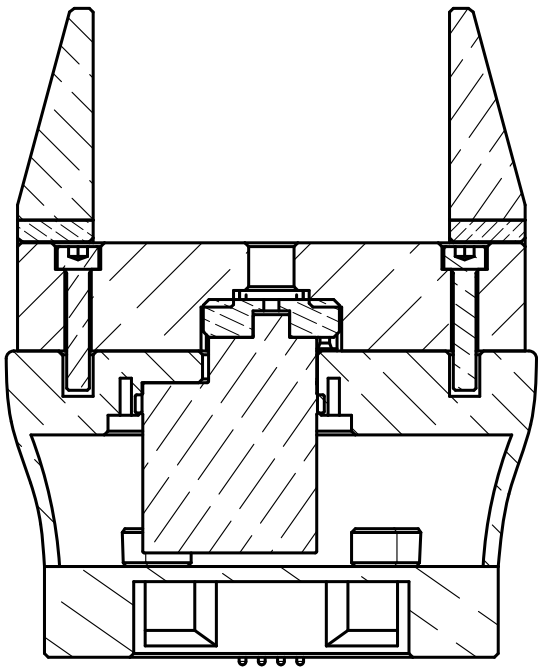


E - E

DWG NO.	SIZE	SCALE	SHEET	REV.
	A4	1:1	8 of 11	

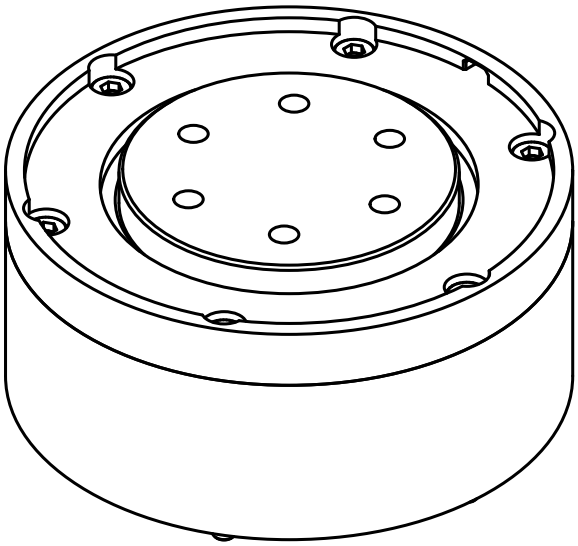


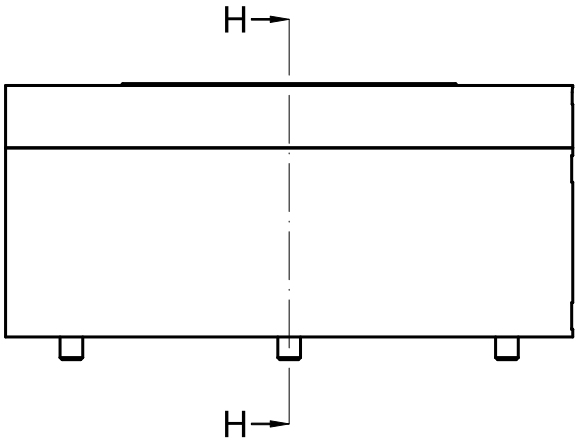


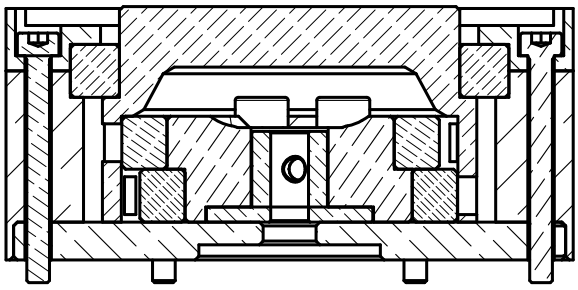


D - D

DWG NO.	SIZE	SCALE	SHEET	REV.
	A4	1:1	9 of 11	

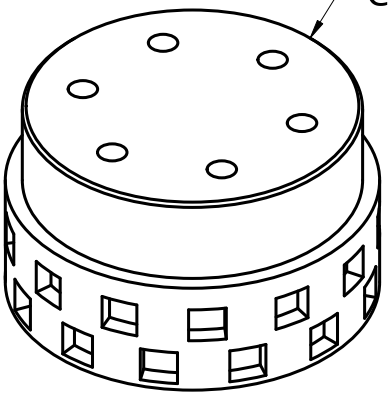


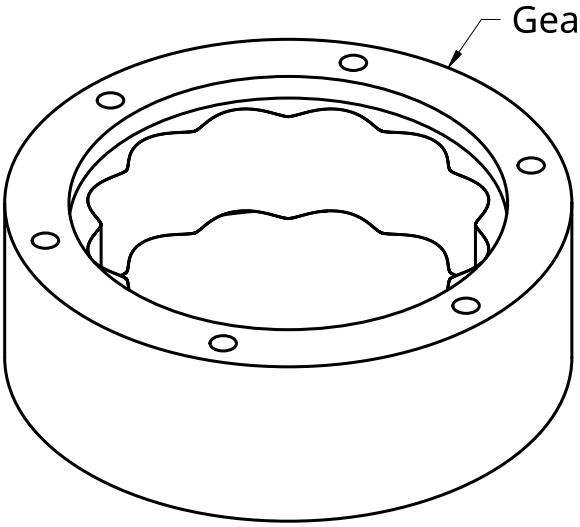


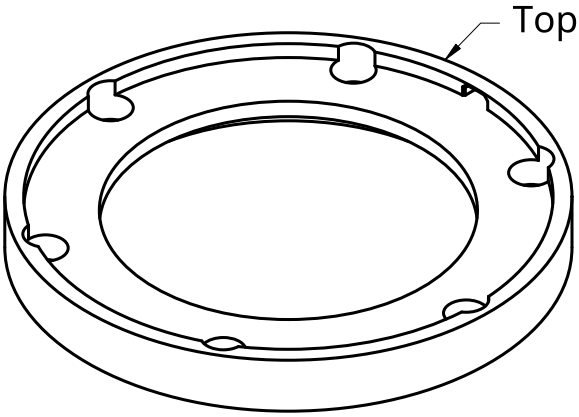


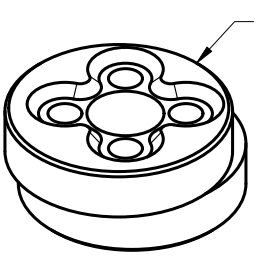
H - H

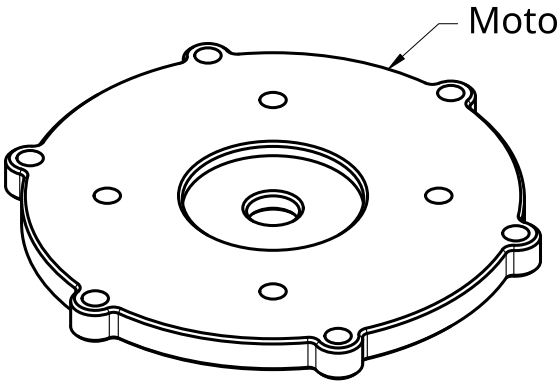
DWG NO.	SIZE	SCALE	SHEET	REV.
	A4	1:1	10 of 11	







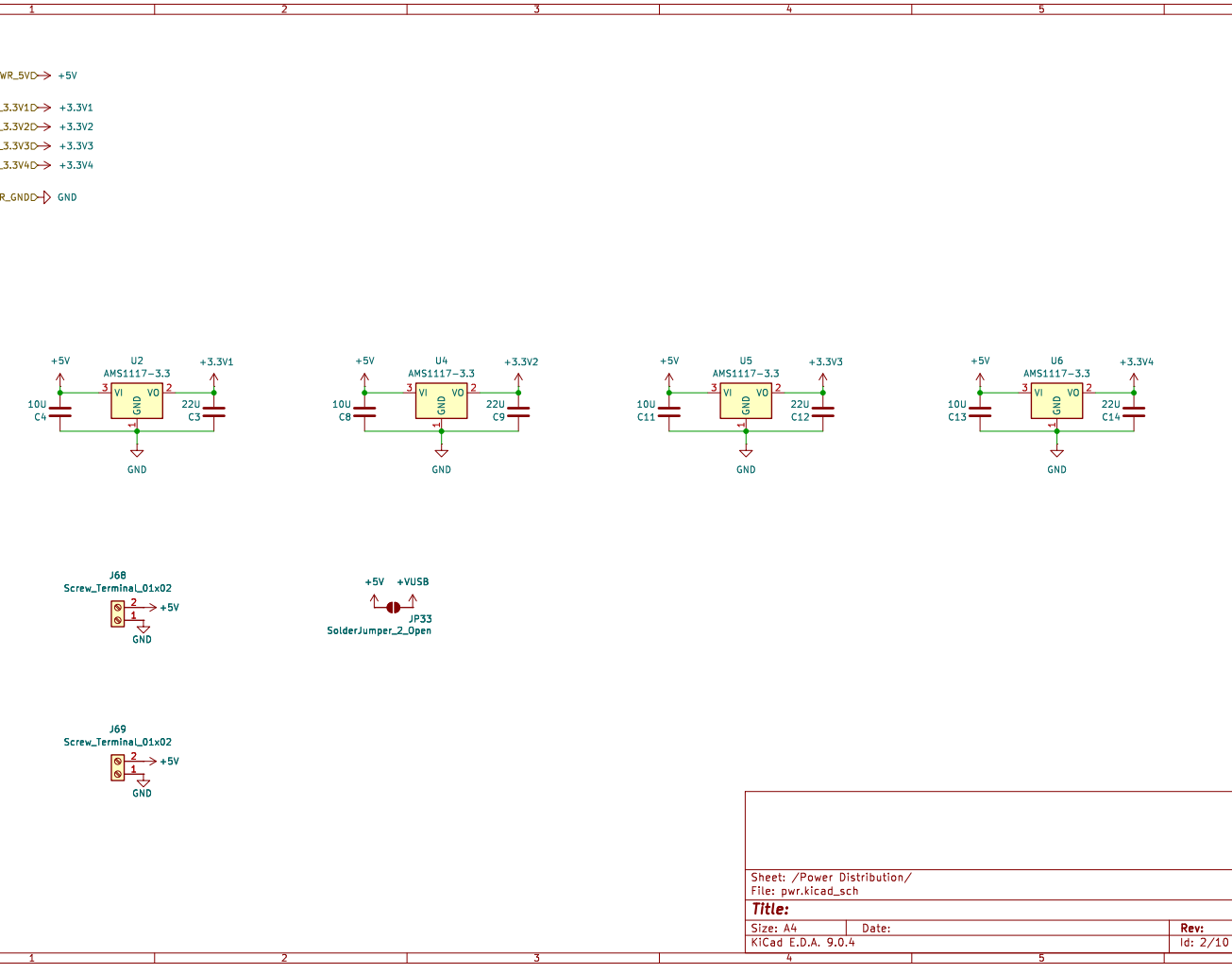


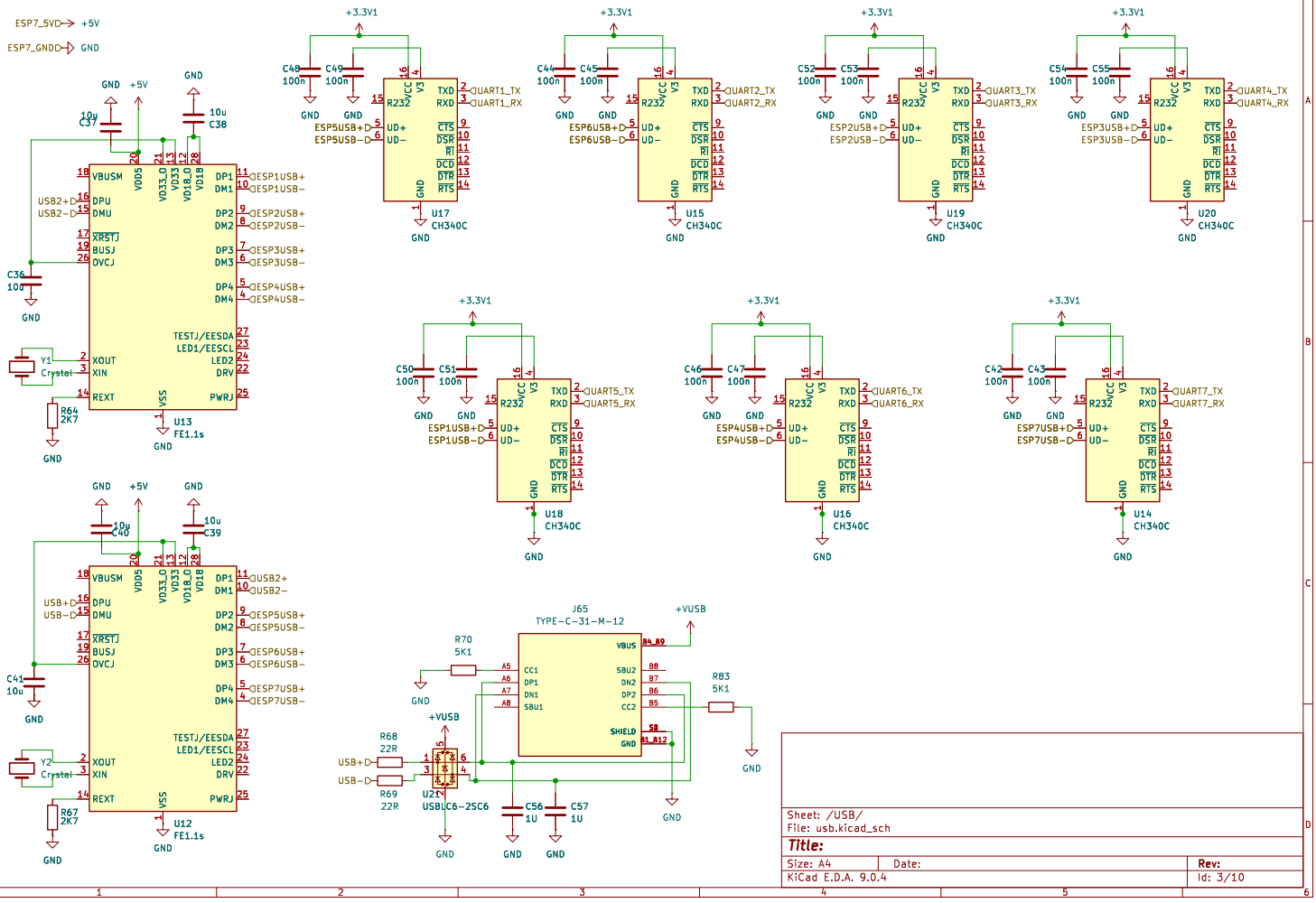


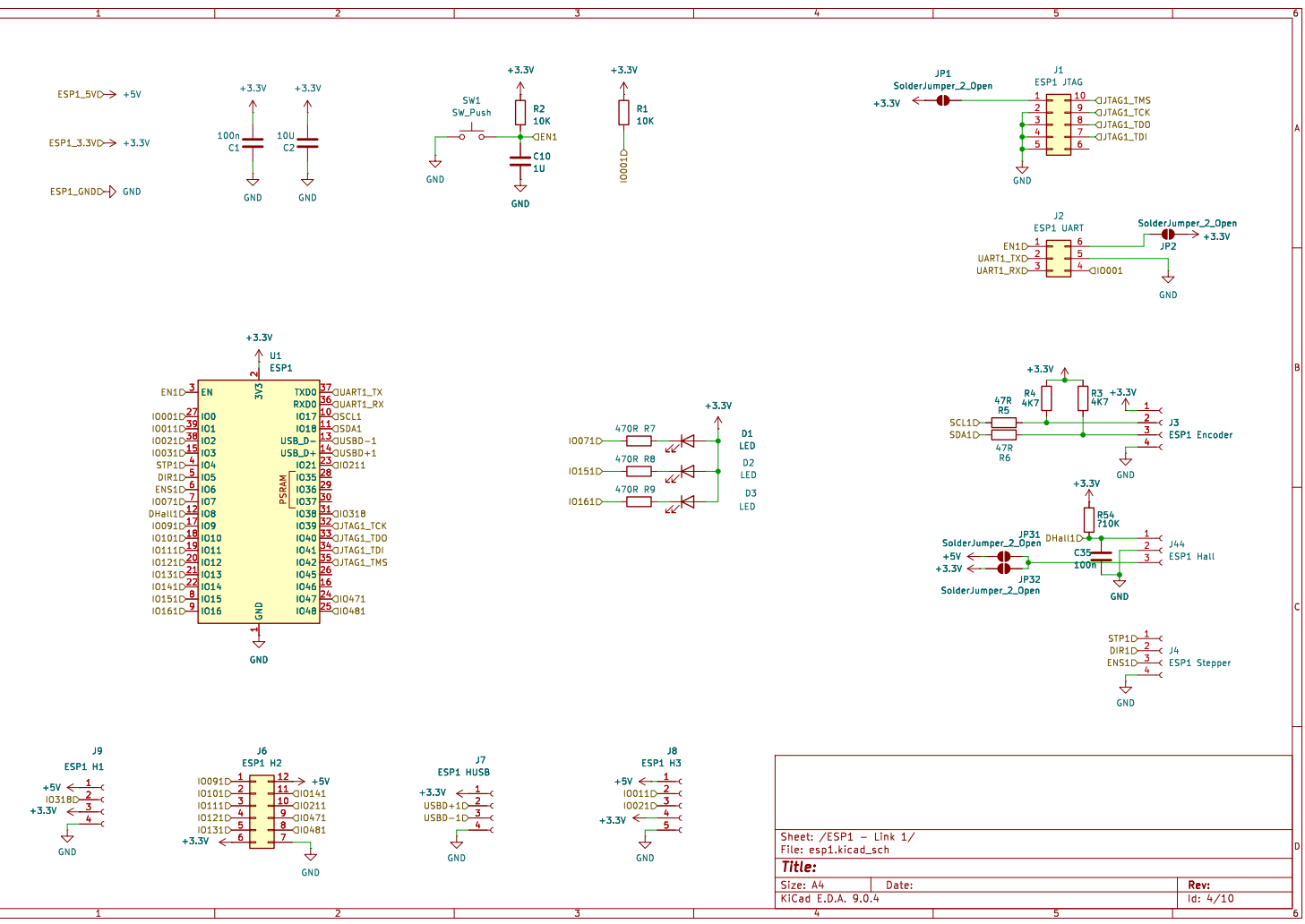
DWG NO.	SIZE	SCALE	SHEET	REV.
	A4	1:1	11 of 11	

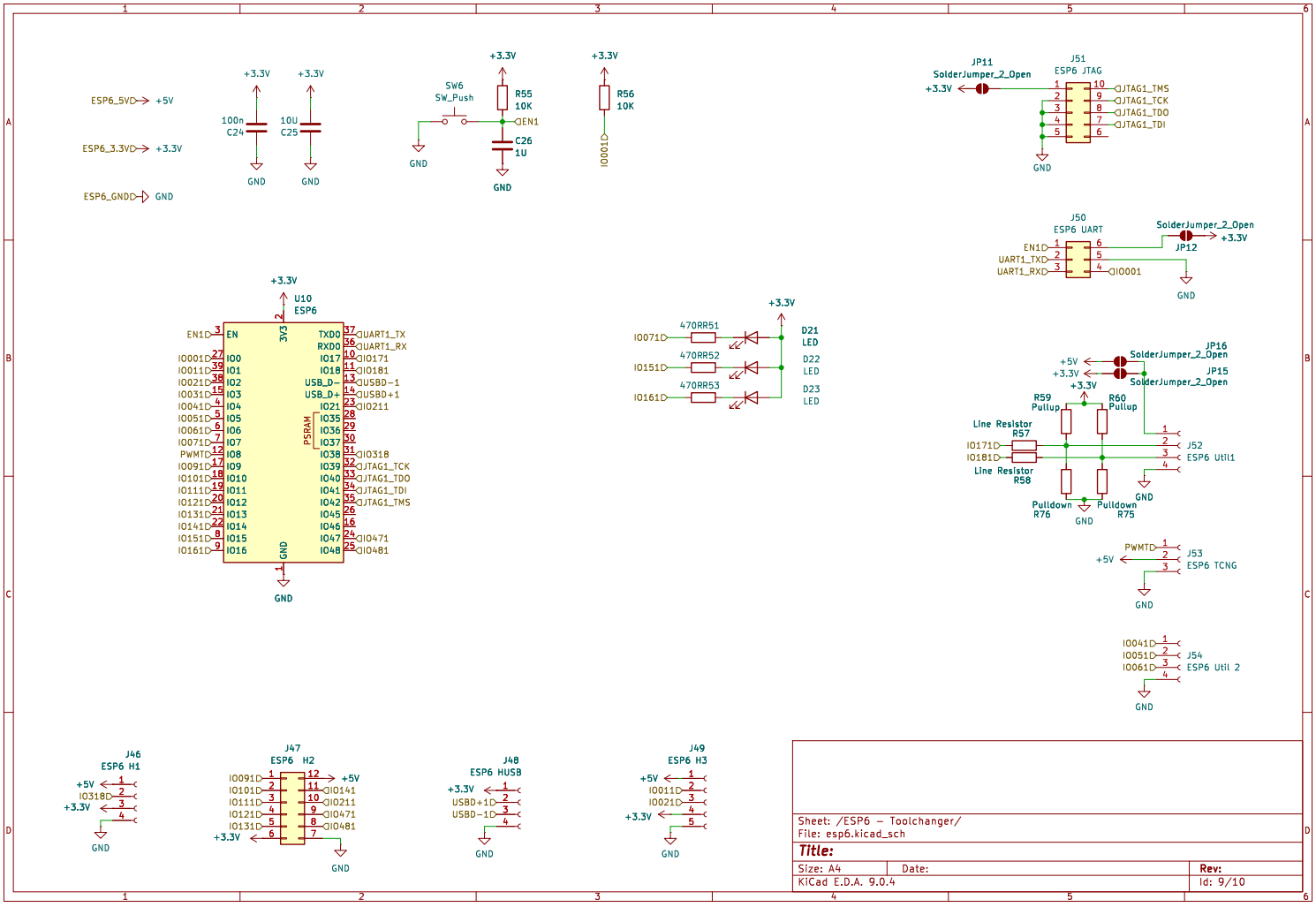
F. Schematics of the First Revision

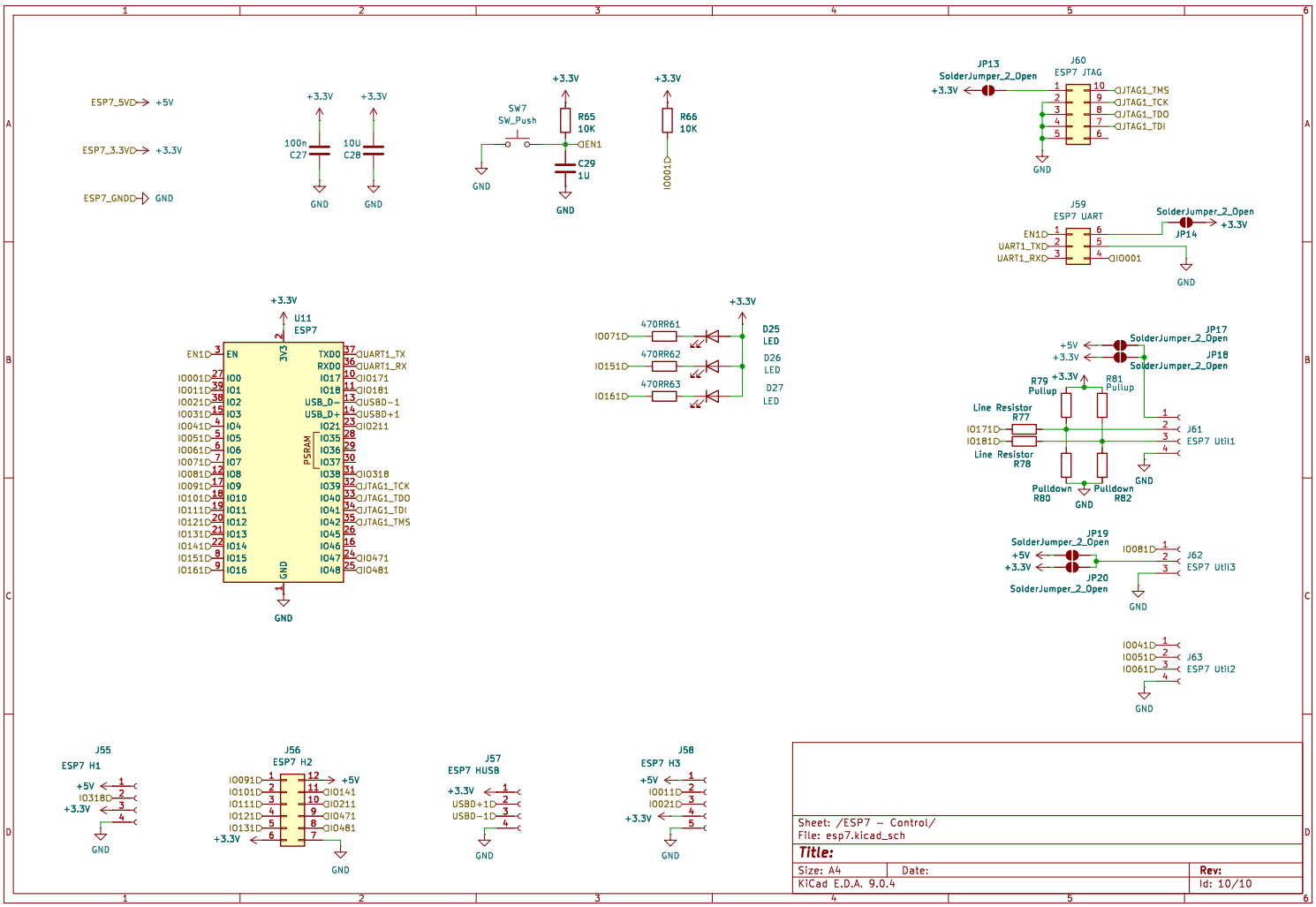

ESP1 – Link 1
+3.3V1
ESP1_3.3V
+5V
ESP1_5V
GND
ESP1_GND
UART1_RX
UART1_TX
ESP1RX
ESP1TX
File: esp1.kicad_sch
ESP2 – Link 2
+3.3V1
ESP2_3.3V
+5V
ESP2_5V
GND
ESP2_GND
UART1_RX
UART1_TX
ESP2RX
ESP2TX
File: esp2.kicad_sch
ESP3 – Link 3
+3.3V2
ESP3_3.3V
+5V
ESP3_5V
GND
ESP3_GND
UART1_RX
UART1_TX
ESP3RX
ESP3TX
File: esp3.kicad_sch
ESP4 – Link 4
+3.3V2
ESP4_3.3V
+5V
ESP4_5V
GND
ESP4_GND
UART1_RX
UART1_TX
ESP4RX
ESP4TX
File: esp4.kicad_sch
ESP5 – Wrist
ESP5_3.3V
+3.3V3
ESP5_5V
+5V
ESP5_GND
GND
ESP5TX
ESP5RX
UART1_TX
UART1_RX
File: esp5.kicad_sch
ESP6 – Toolchanger
+3.3V3
ESP6_3.3V
+5V
ESP6_5V
GND
ESP6_GND
UART1_TX
UART1_RX
ESP6TX
ESP6RX
File: esp6.kicad_sch
ESP7 – Control
+3.3V4
ESP7_3.3V
+5V
ESP7_5V
GND
ESP7_GND
UART1_TX
UART1_RX
ESP7TX
ESP7RX
File: esp7.kicad_sch
USB
+3.3V4
ESP7_3.3V
+5V
ESP7_5V
GND
ESP7_GND
UART1_RX
UART1_TX
ESP1RX
ESP1TX
UART2_RX
UART2_TX
ESP2RX
ESP2TX
UART3_RX
UART3_TX
ESP3RX
ESP3TX
UART4_RX
UART4_TX
ESP4RX
ESP4TX
UART5_RX
UART5_TX
ESP5RX
ESP5TX
UART6_RX
UART6_TX
ESP6RX
ESP6TX
UART7_RX
UART7_TX
ESP7RX
ESP7TX
USB+
USB-
File: usb.kicad_sch
Power Distribution
PWR_3.3V1
+3.3V1
PWR_3.3V2
+3.3V2
PWR_3.3V3
+3.3V3
PWR_3.3V4
+3.3V4
PWR_5V
+5V
PWR_GND
GND
File: pwr.kicad_sch
Sheet: /
File: robotpcb.kicad_sch
Title:
Size: A4
Date:
Rev:
KiCad E.D.A. 9.0.4
Id: 1/10


PWR_5V
+5V
PWR_3.3V1
+3.3V1
PWR_3.3V2
+3.3V2
PWR_3.3V3
+3.3V3
PWR_3.3V4
+3.3V4
PWR_GND
GND
U2
AMS1117-3.3
10U
C4
22U
C3
U4
AMS1117-3.3
10U
C8
22U
C9
U5
AMS1117-3.3
10U
C11
22U
C12
U6
AMS1117-3.3
10U
C13
22U
C14
J68
Screw_Terminal_01x02
J69
Screw_Terminal_01x02
+VUSB
JP33
SolderJumper_2_Open
Sheet: /Power Distribution/
File: pwr.kicad_sch
Title:
Size: A4
Date:
Rev:
KiCad E.D.A. 9.0.4
Id: 2/10

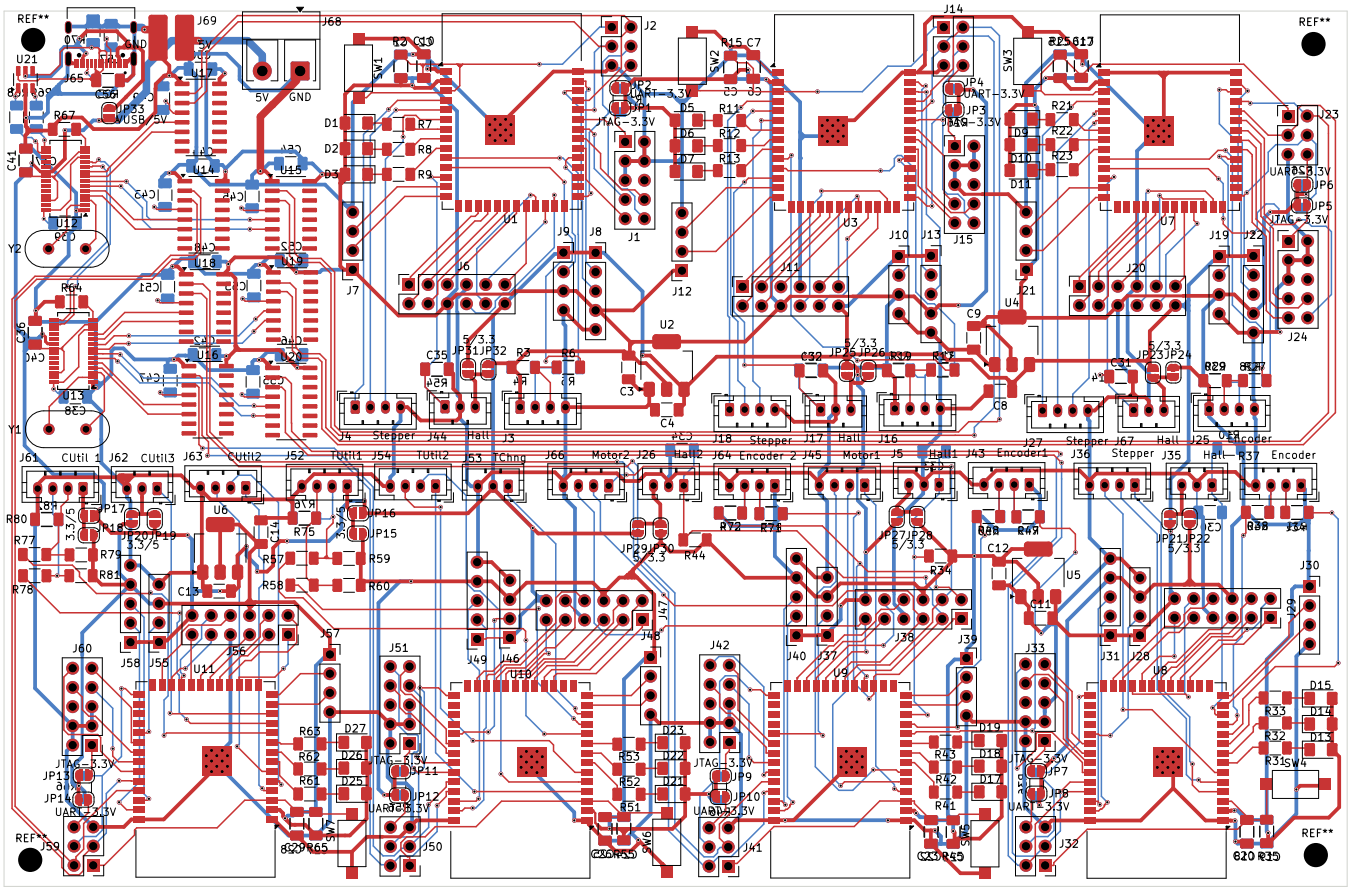

ESP7_5VD
ESP7_GNDD
+5V
GND
+3.3V1
C48 100n
C49 100n
U17 CH340C
ESP5USB+
ESP5USB−
UART1_TX
UART1_RX
C44 100n
C45 100n
U15 CH340C
ESP6USB+
ESP6USB−
UART2_TX
UART2_RX
C52 100n
C53 100n
U19 CH340C
ESP2USB+
ESP2USB−
UART3_TX
UART3_RX
C54 100n
C55 100n
U20 CH340C
ESP3USB+
ESP3USB−
UART4_TX
UART4_RX
C37 10u
C38 10u
U13 FE1.1s
C36 10n
Y1 Crystal
R64 2K7
USB2+
USB2−
ESP1USB+
ESP1USB−
ESP2USB+
ESP2USB−
ESP3USB+
ESP3USB−
ESP4USB+
ESP4USB−
C50 100n
C51 100n
U18 CH340C
ESP1USB+
ESP1USB−
UART5_TX
UART5_RX
C46 100n
C47 100n
U16 CH340C
ESP4USB+
ESP4USB−
UART6_TX
UART6_RX
C42 100n
C43 100n
U14 CH340C
ESP7USB+
ESP7USB−
UART7_TX
UART7_RX
C40 10u
C39 10u
U12 FE1.1s
C41 10u
Y2 Crystal
R67 2K7
USB+
USB−
ESP5USB+
ESP5USB−
ESP6USB+
ESP6USB−
ESP7USB+
ESP7USB−
J65
TYPE-C-31-M-12
+VUSB
R70 5K1
R83 5K1
R68 22R
R69 22R
U21 USBLC6-2SC6
C56 1U
C57 1U
Sheet: /USB/
File: usb.kicad_sch
Title:
Size: A4
Date:
Rev:
KiCad E.D.A. 9.0.4
Id: 3/10


ESP1_5VD
ESP1_3.3VD
ESP1_GNDD
+5V
+3.3V
GND
C1 100n
C2 10U
SW1 SW_Push
R2 10K
C10 1U
EN1
R1 10K
IO001
JP1 SolderJumper_2_Open
J1 ESP1 JTAG
JTAG1_TMS
JTAG1_TCK
JTAG1_TDO
JTAG1_TDI
J2 ESP1 UART
SolderJumper_2_Open
JP2
UART1_TX
UART1_RX
U1 ESP1
UART1_TX
UART1_RX
SCL1
SDA1
USB_D−1
USB_D+1
IO211
IO318
IO471
IO481
PSRAM
D1 LED
D2 LED
D3 LED
470R R7
470R R8
470R R9
R4 4K7
R5 4K7
R3 4K7
R6 47R
J3 ESP1 Encoder
R54 ?10K
JP31 SolderJumper_2_Open
JP32 SolderJumper_2_Open
C35 100n
DHall1D
J44 ESP1 Hall
STP1
DIR1
ENS1
J4 ESP1 Stepper
J9 ESP1 H1
J6 ESP1 H2
J7 ESP1 HUSB
J8 ESP1 H3
Sheet: /ESP1 - Link 1/
File: esp1.kicad_sch
Title:
Size: A4
Date:
Rev:
KiCad E.D.A. 9.0.4
Id: 4/10

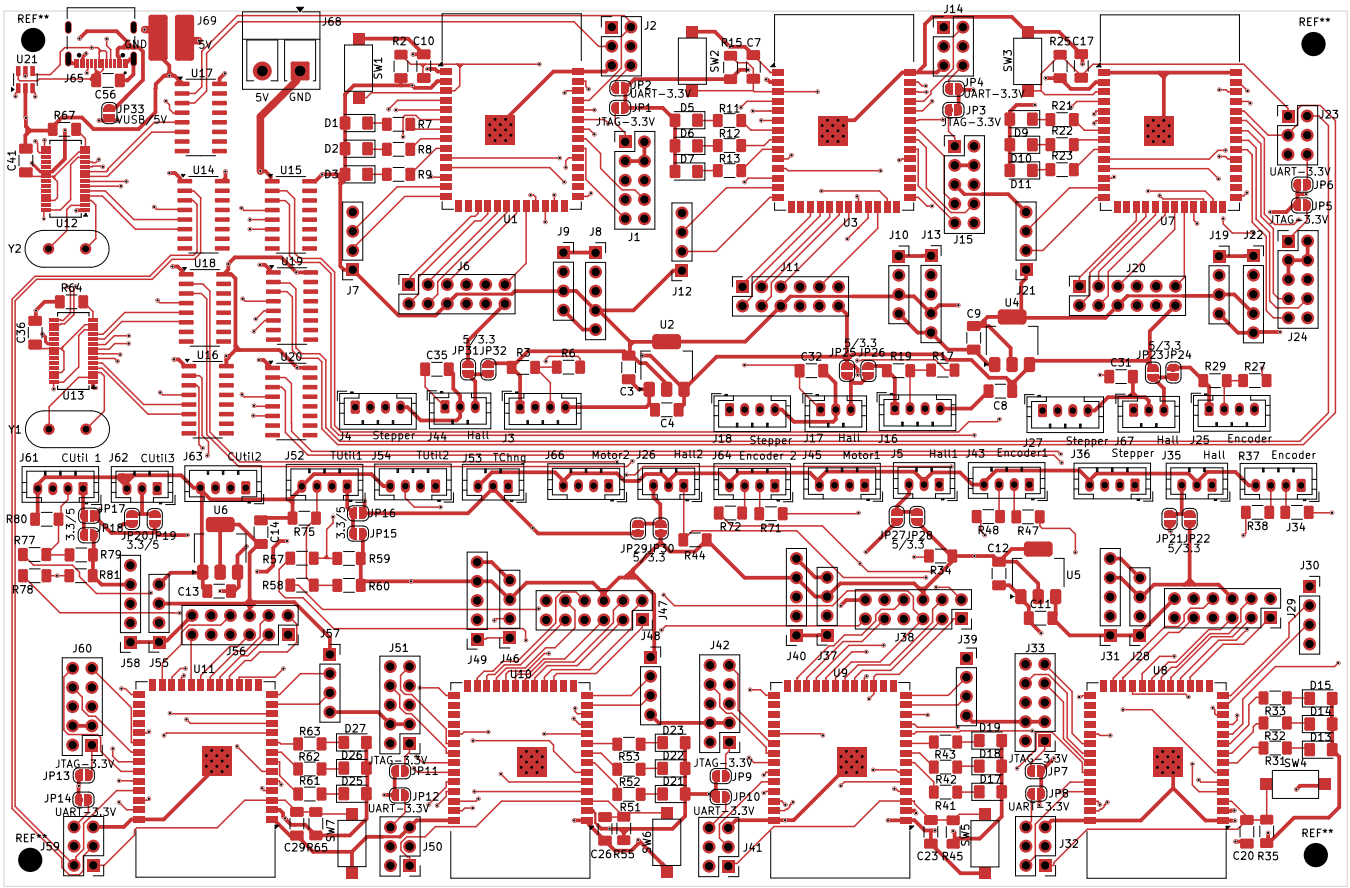

ESP6_5V
ESP6_3.3V
ESP6_GND
+5V
+3.3V
GND
100n
C24
10U
C25
SW6
SW_Push
R55
10K
EN1
C26
1U
R56
10K
IO0001
JP11
SolderJumper_2_Open
J51
ESP6 JTAG
JTAG1_TMS
JTAG1_TCK
JTAG1_TDO
JTAG1_TDI
J50
ESP6 UART
UART1_TXD
UART1_RXD
JP12
U10
ESP6
EN
3V3
TXD0
RXD0
IO00
IO01
IO02
IO03
IO04
IO05
IO06
IO07
IO08
IO09
IO10
IO11
IO12
IO13
IO14
IO15
IO16
IO17
IO18
USB_D-
USB_D+
IO21
IO35
IO36
IO37
IO38
IO39
IO40
IO41
IO42
IO45
IO46
IO47
IO48
PSRAM
UART1_TX
UART1_RX
IO0171
IO0181
USBD-1
USBD+1
IO0211
IO0318
JTAG1_TCK
JTAG1_TDO
JTAG1_TDI
JTAG1_TMS
IO0471
IO0481
PWMTD
470RR51
470RR52
470RR53
D21
D22
D23
LED
JP16
JP15
R59
Pullup
R60
Line Resistor
R57
R58
J52
ESP6 Util1
Pulldown
R76
R75
J53
ESP6 TCNG
J54
ESP6 Util 2
J46
ESP6 H1
J47
ESP6 H2
J48
ESP6 HUSB
J49
ESP6 H3
Sheet: /ESP6 – Toolchanger/
File: esp6.kicad_sch
Title:
Size: A4
Date:
Rev:
KiCad E.D.A. 9.0.4
Id: 9/10


ESP7_5V
ESP7_3.3V
ESP7_GND
+5V
+3.3V
GND
100n
C27
10U
C28
SW7
SW_Push
R65
10K
EN1
C29
1U
R66
10K
IO0001
JP13
SolderJumper_2_Open
J60
ESP7 JTAG
JTAG1_TMS
JTAG1_TCK
JTAG1_TDO
JTAG1_TDI
J59
ESP7 UART
UART1_TXD
UART1_RXD
JP14
U11
ESP7
EN
3V3
TXD0
RXD0
IO00
IO01
IO02
IO03
IO04
IO05
IO06
IO07
IO08
IO09
IO10
IO11
IO12
IO13
IO14
IO15
IO16
IO17
IO18
USB_D-
USB_D+
IO21
IO35
IO36
IO37
IO38
IO39
IO40
IO41
IO42
IO45
IO46
IO47
IO48
PSRAM
UART1_TX
UART1_RX
IO0171
IO0181
USBD-1
USBD+1
IO0211
IO0318
JTAG1_TCK
JTAG1_TDO
JTAG1_TDI
JTAG1_TMS
IO0471
IO0481
470RR61
470RR62
470RR63
D25
D26
D27
LED
JP17
JP18
R79
Pullup
R81
Line Resistor
R77
R78
J61
ESP7 Util1
Pulldown
R80
R82
JP19
JP20
J62
ESP7 Util3
J63
ESP7 Util2
J55
ESP7 H1
J56
ESP7 H2
J57
ESP7 HUSB
J58
ESP7 H3
Sheet: /ESP7 – Control/
File: esp7.kicad_sch
Title:
Size: A4
Date:
Rev:
KiCad E.D.A. 9.0.4
Id: 10/10

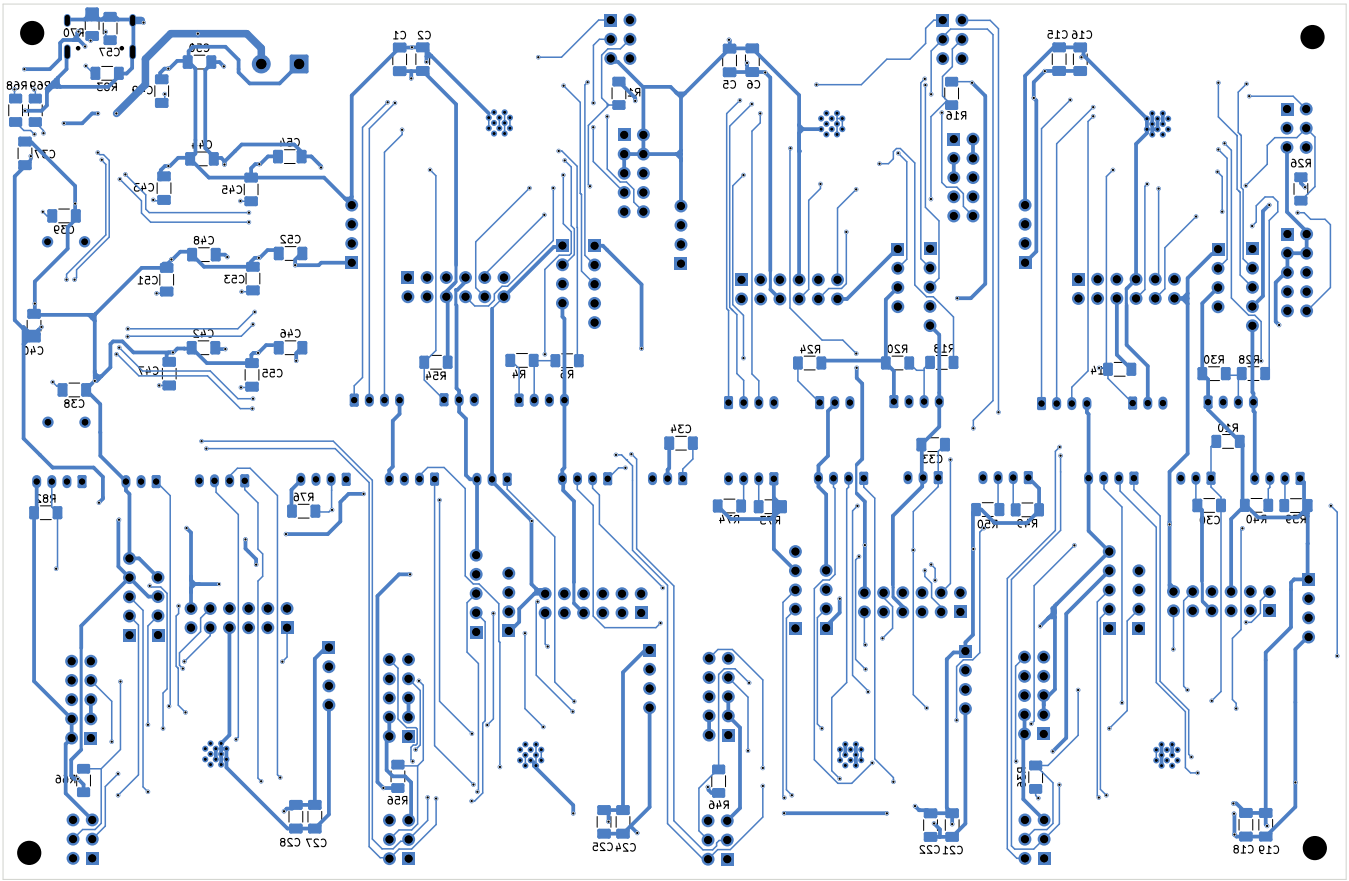
G. Printed Circuit Board Layout of the First Revision



Sheet:		
File: robotpcb.kicad_pcb		
Title:		
Size: A4	Date:	**Rev:**
KiCad E.D.A. 9.0.4		Id: 1/1

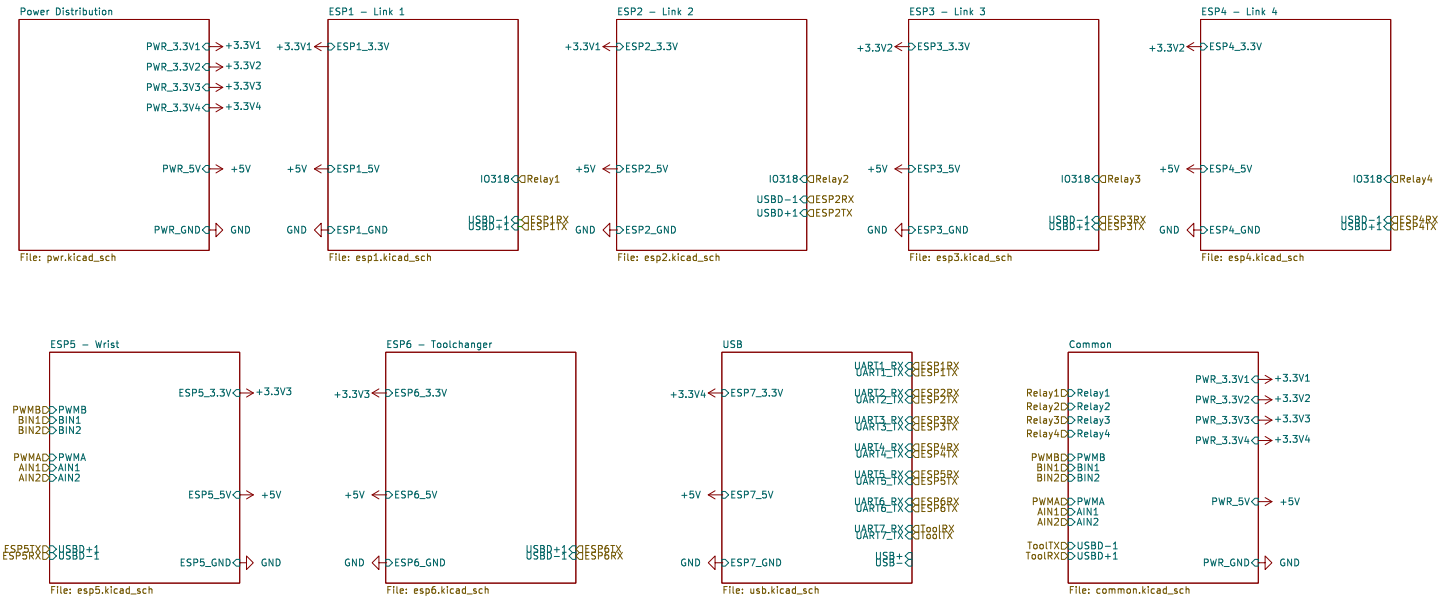


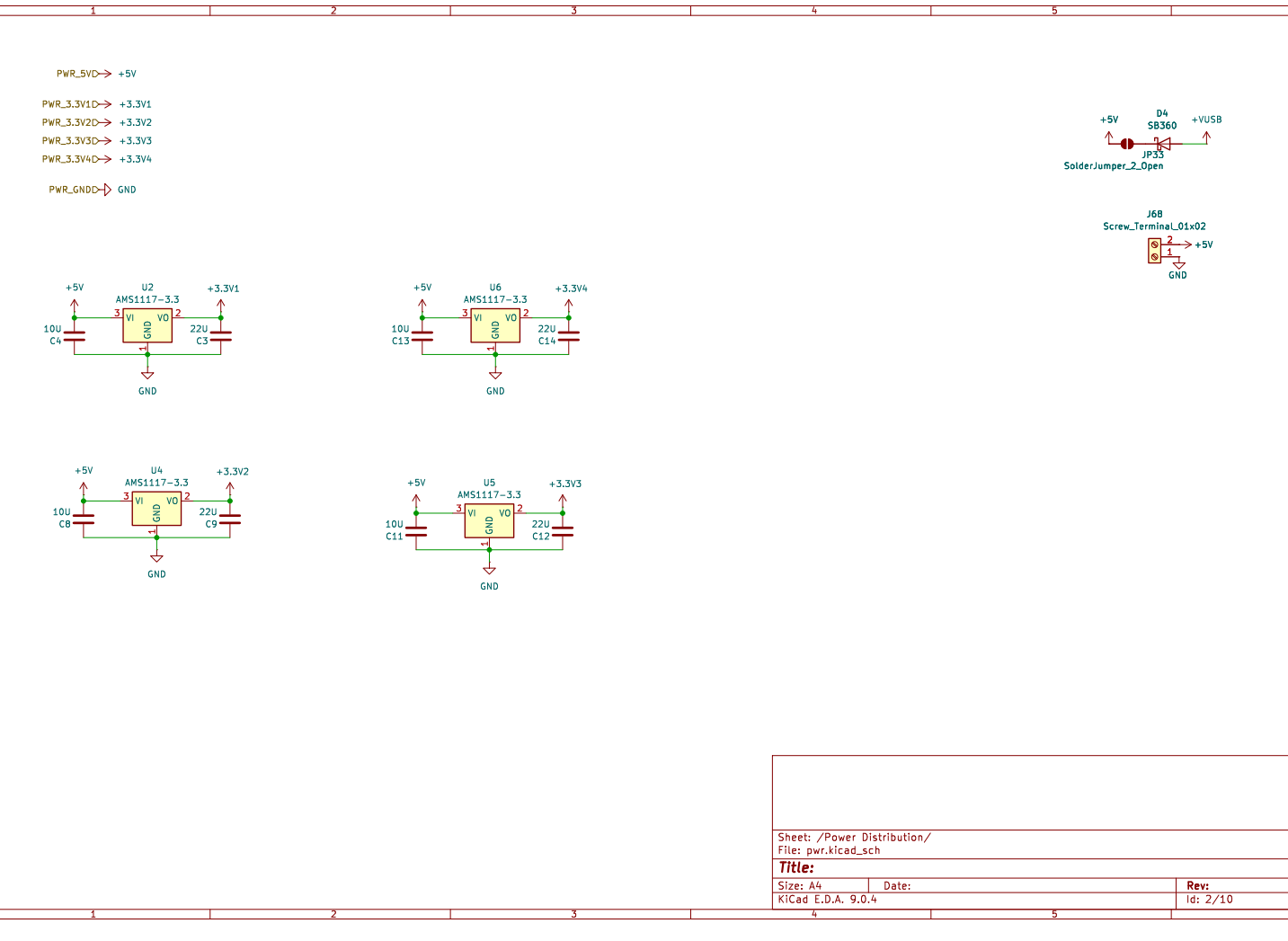
Sheet:		
File: robotpcb.kicad_pcb		
Title:		
Size: A4	Date:	**Rev:**
KiCad E.D.A. 9.0.4		Id: 1/1

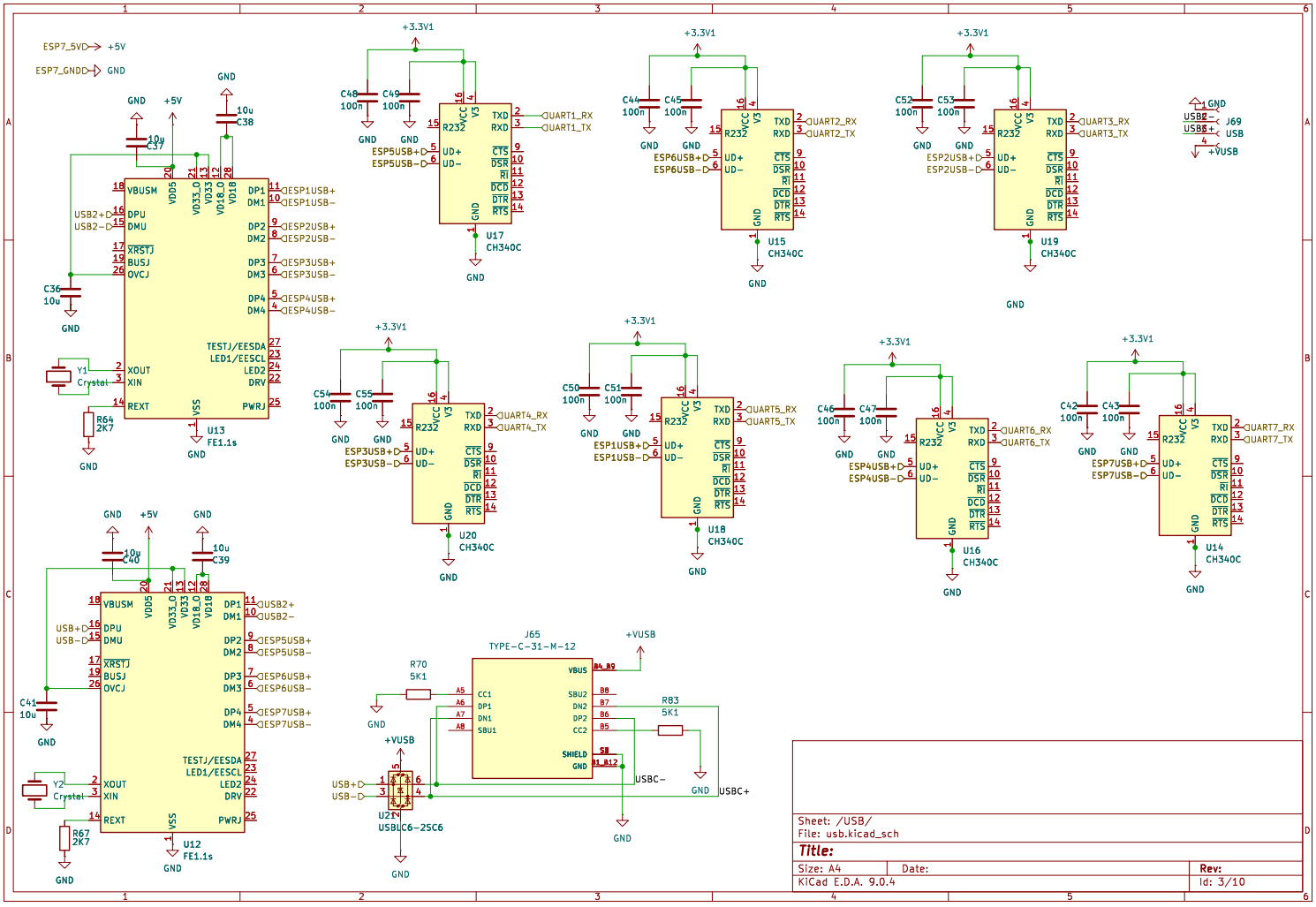


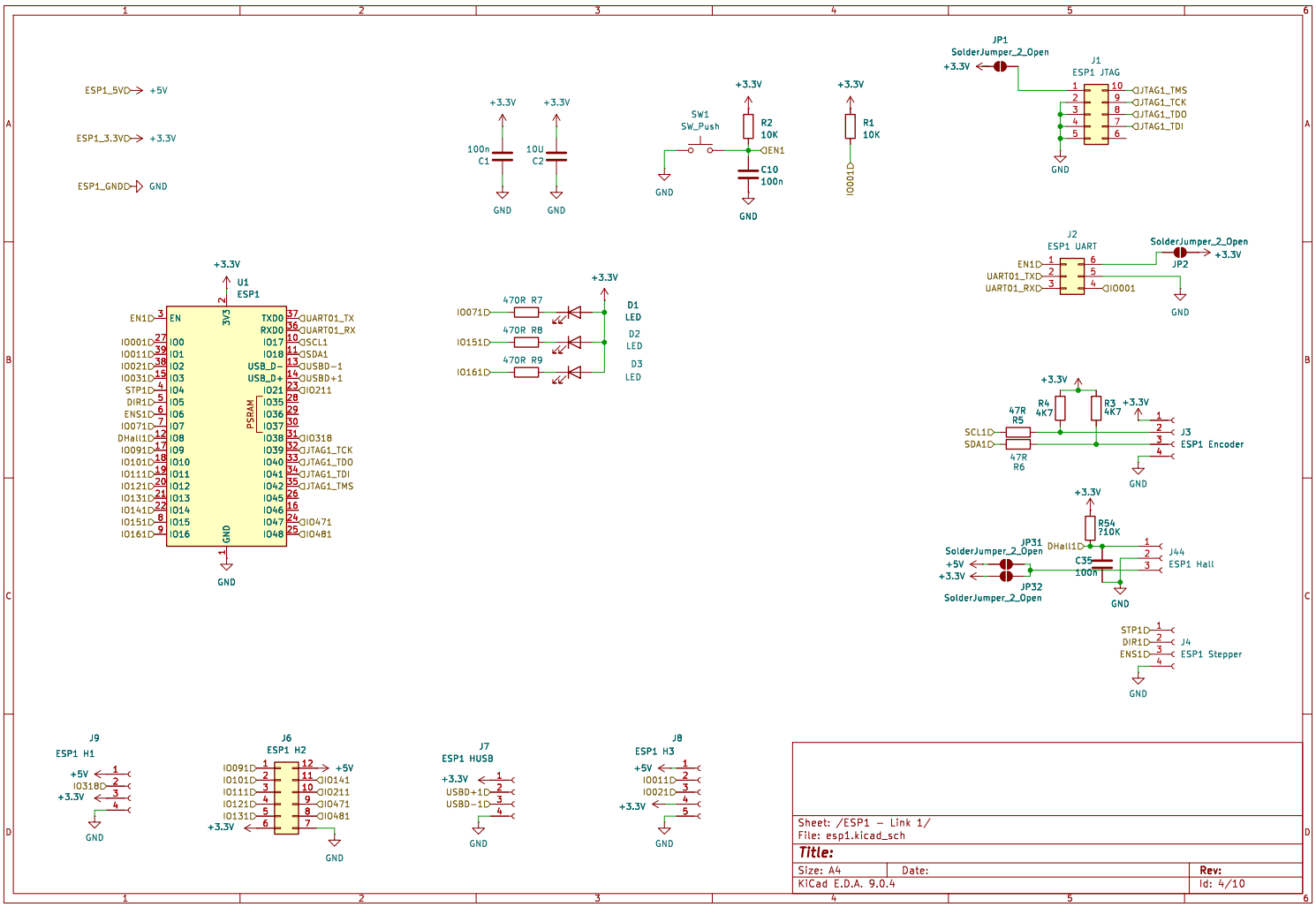
Sheet:		
File: robotpcb.kicad_pcb		
Title:		
Size: A4	Date:	**Rev:**
KiCad E.D.A. 9.0.4		Id: 1/1

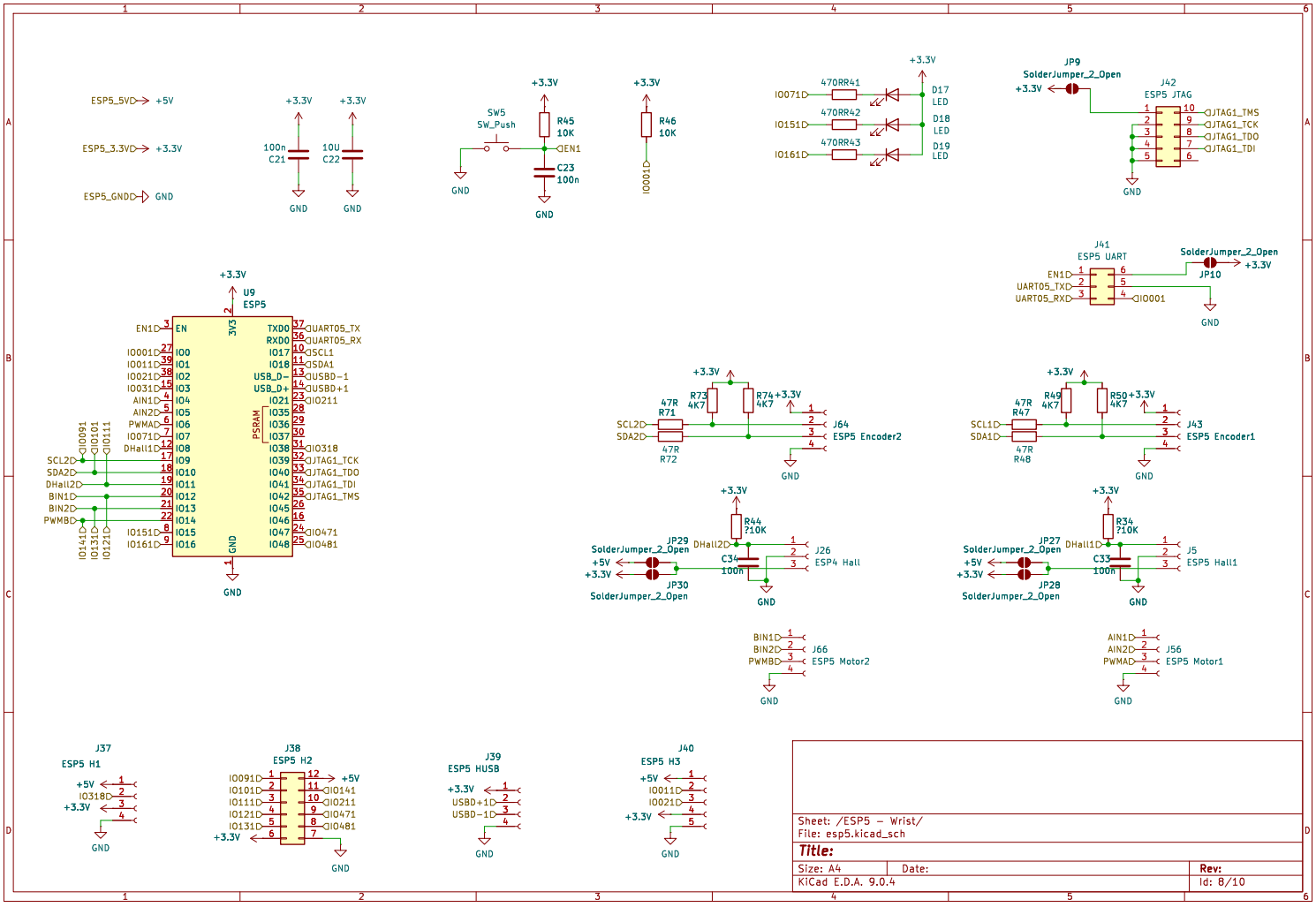
H. Schematics of the Second Revision

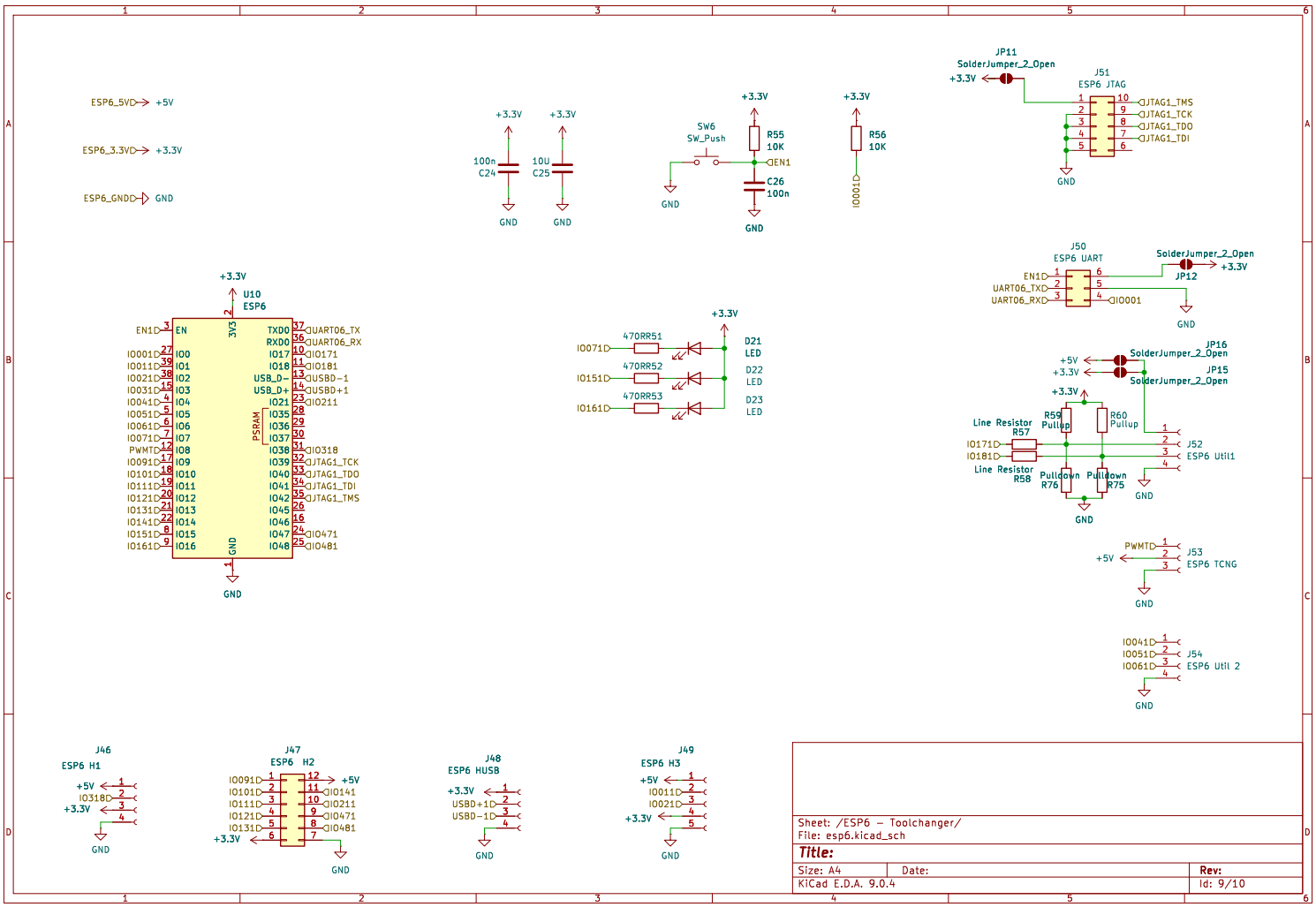

Power Distribution
PWR_3.3V1
PWR_3.3V2
PWR_3.3V3
PWR_3.3V4
PWR_5V
PWR_GND
+3.3V1
+3.3V2
+3.3V3
+3.3V4
+5V
GND
File: pwr.kicad_sch
ESP1 – Link 1
ESP1_3.3V
ESP1_5V
ESP1_GND
IO318
Relay1
USBD-1
USBD+1
ESP1RX
ESP1TX
File: esp1.kicad_sch
ESP2 – Link 2
ESP2_3.3V
ESP2_5V
ESP2_GND
Relay2
ESP2RX
ESP2TX
File: esp2.kicad_sch
ESP3 – Link 3
ESP3_3.3V
ESP3_5V
ESP3_GND
Relay3
ESP3RX
ESP3TX
File: esp3.kicad_sch
ESP4 – Link 4
ESP4_3.3V
ESP4_5V
ESP4_GND
Relay4
ESP4RX
ESP4TX
File: esp4.kicad_sch
ESP5 – Wrist
ESP5_3.3V
ESP5_5V
ESP5_GND
PWMB
BIN1
BIN2
PWMA
AIN1
AIN2
ESP5TX
ESP5RX
File: esp5.kicad_sch
ESP6 – Toolchanger
ESP6_3.3V
ESP6_5V
ESP6_GND
ESP6TX
ESP6RX
File: esp6.kicad_sch
USB
ESP7_3.3V
ESP7_5V
ESP7_GND
UART1_RX
UART1_TX
UART2_RX
UART2_TX
UART3_RX
UART3_TX
UART4_RX
UART4_TX
UART5_RX
UART5_TX
UART6_RX
UART6_TX
UART7_RX
UART7_TX
USB+
USB-
File: usb.kicad_sch
Common
Relay1
Relay2
Relay3
Relay4
ToolTX
ToolRX
File: common.kicad_sch
Sheet: /
File: robotpcb.kicad_sch
Title:
Size: A4
Date:
Rev:
KiCad E.D.A. 9.0.4
Id: 1/10

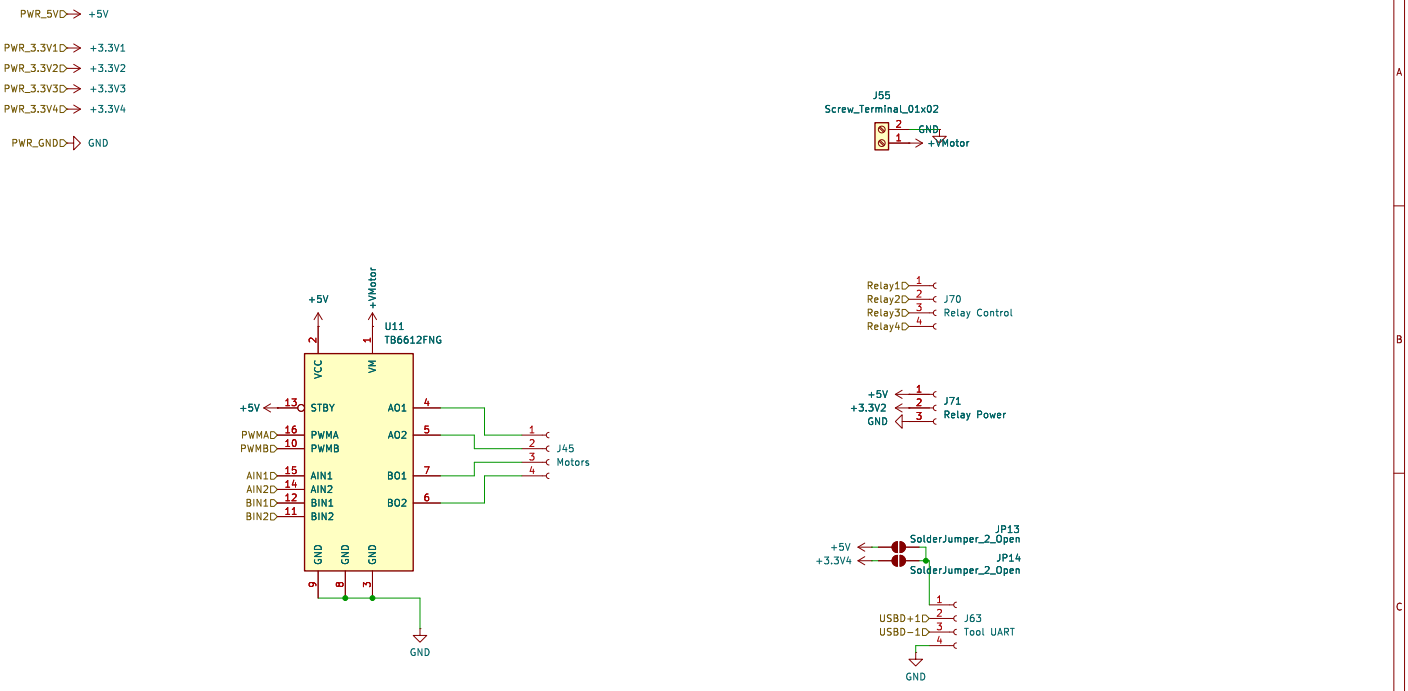

PWR_5VD
PWR_3.3V1D
PWR_3.3V2D
PWR_3.3V3D
PWR_3.3V4D
PWR_GNDD
U2
AMS1117-3.3
U6
U4
U5
10U
22U
C4
C3
C13
C14
C8
C9
C11
C12
D4
SB360
JP33
SolderJumper_2_Open
J6B
Screw_Terminal_01x02
Sheet: /Power Distribution/
File: pwr.kicad_sch
Title:
Size: A4
Date:
Rev:
KiCad E.D.A. 9.0.4
Id: 2/10


ESP7_5VD
+5V
ESP7_GNDD
GND
U13
FE1.1s
U12
FE1.1s
U17
CH340C
U20
CH340C
U15
CH340C
U19
CH340C
U18
CH340C
U16
CH340C
U14
CH340C
J65
TYPE-C-31-M-12
USBLC6-2SC6
J69
USB
Sheet: /USB/
File: usb.kicad_sch
Title:
Size: A4
Date:
Rev:
KiCad E.D.A. 9.0.4
Id: 3/10


ESP1_5VD
+5V
ESP1_3.3VD
+3.3V
ESP1_GNDD
GND
U1
ESP1
JP1
SolderJumper_2_Open
J1
ESP1 JTAG
J2
ESP1 UART
JP2
SW1
SW_Push
D1
LED
D2
LED
D3
LED
J3
ESP1 Encoder
JP31
SolderJumper_2_Open
JP32
SolderJumper_2_Open
J44
ESP1 Hall
J4
ESP1 Stepper
J9
ESP1 H1
J6
ESP1 H2
J7
ESP1 HUSB
J8
ESP1 H3
Sheet: /ESP1 - Link 1/
File: esp1.kicad_sch
Title:
Size: A4
Date:
Rev:
KiCad E.D.A. 9.0.4
Id: 4/10


ESP5_5VD
+5V
ESP5_3.3VD
+3.3V
ESP5_GNDD
GND
100n
C21
10U
C22
SW5
SW_Push
R45
10K
EN1
C23
100n
R46
10K
IO001D
470RR41
470RR42
470RR43
IO071D
IO151D
IO161D
D17
LED
D18
LED
D19
LED
JP9
SolderJumper_2_Open
J42
ESP5 JTAG
JTAG1_TMS
JTAG1_TCK
JTAG1_TDO
JTAG1_TDI
J41
ESP5 UART
UART05_TXD
UART05_RXD
JP10
U9
ESP5
EN1D
SCL2D
SDA2D
DHall2D
BIN1D
BIN2D
PWMBD
IO141D
IO131D
IO121D
IO151D
IO161D
UART05_TX
UART05_RX
SCL1
SDA1
USBD-1
USBD+1
IO211
IO318
IO471
IO481
R73
4K7
R74
4K7
47R
R71
47R
R72
J64
ESP5 Encoder2
R49
4K7
R50
4K7
47R
R47
47R
R48
J43
ESP5 Encoder1
R44
710K
C34
100n
JP29
SolderJumper_2_Open
JP30
SolderJumper_2_Open
J26
ESP4 Hall
R34
710K
C33
100n
JP27
SolderJumper_2_Open
JP28
SolderJumper_2_Open
J5
ESP5 Hall1
BIN1D
BIN2D
PWMBD
J66
ESP5 Motor2
AIN1D
AIN2D
PWMAD
J56
ESP5 Motor1
J37
ESP5 H1
J38
ESP5 H2
J39
ESP5 HUSB
J40
ESP5 H3
Sheet: /ESP5 – Wrist/
File: esp5.kicad_sch
Title:
Size: A4
Date:
Rev:
KiCad E.D.A. 9.0.4
Id: 8/10


ESP6_5VD
+5V
ESP6_3.3VD
+3.3V
ESP6_GNDD
GND
100n
C24
10U
C25
SW6
SW_Push
R55
10K
EN1
C26
100n
R56
10K
IO001D
JP11
SolderJumper_2_Open
J51
ESP6 JTAG
JTAG1_TMS
JTAG1_TCK
JTAG1_TDO
JTAG1_TDI
J50
ESP6 UART
UART06_TXD
UART06_RXD
JP12
U10
ESP6
EN1D
IO001D
IO011D
IO021D
IO031D
IO041D
IO051D
IO061D
IO071D
PWMTD
IO091D
IO101D
IO111D
IO121D
IO131D
IO141D
IO151D
IO161D
UART06_TX
UART06_RX
IO171
IO181
USBD-1
USBD+1
IO211
IO318
IO471
IO481
JTAG1_TCK
JTAG1_TDO
JTAG1_TDI
JTAG1_TMS
470RR51
470RR52
470RR53
D21
LED
D22
LED
D23
LED
JP16
SolderJumper_2_Open
JP15
SolderJumper_2_Open
Line Resistor
R57
Line Resistor
R58
R59
Pullup
R60
Pullup
Pulldown
R76
Pulldown
R75
J52
ESP6 Util1
PWMTD
J53
ESP6 TCNG
IO041D
IO051D
IO061D
J54
ESP6 Util 2
J46
ESP6 H1
J47
ESP6 H2
J48
ESP6 HUSB
J49
ESP6 H3
Sheet: /ESP6 – Toolchanger/
File: esp6.kicad_sch
Title:
Size: A4
Date:
Rev:
KiCad E.D.A. 9.0.4
Id: 9/10

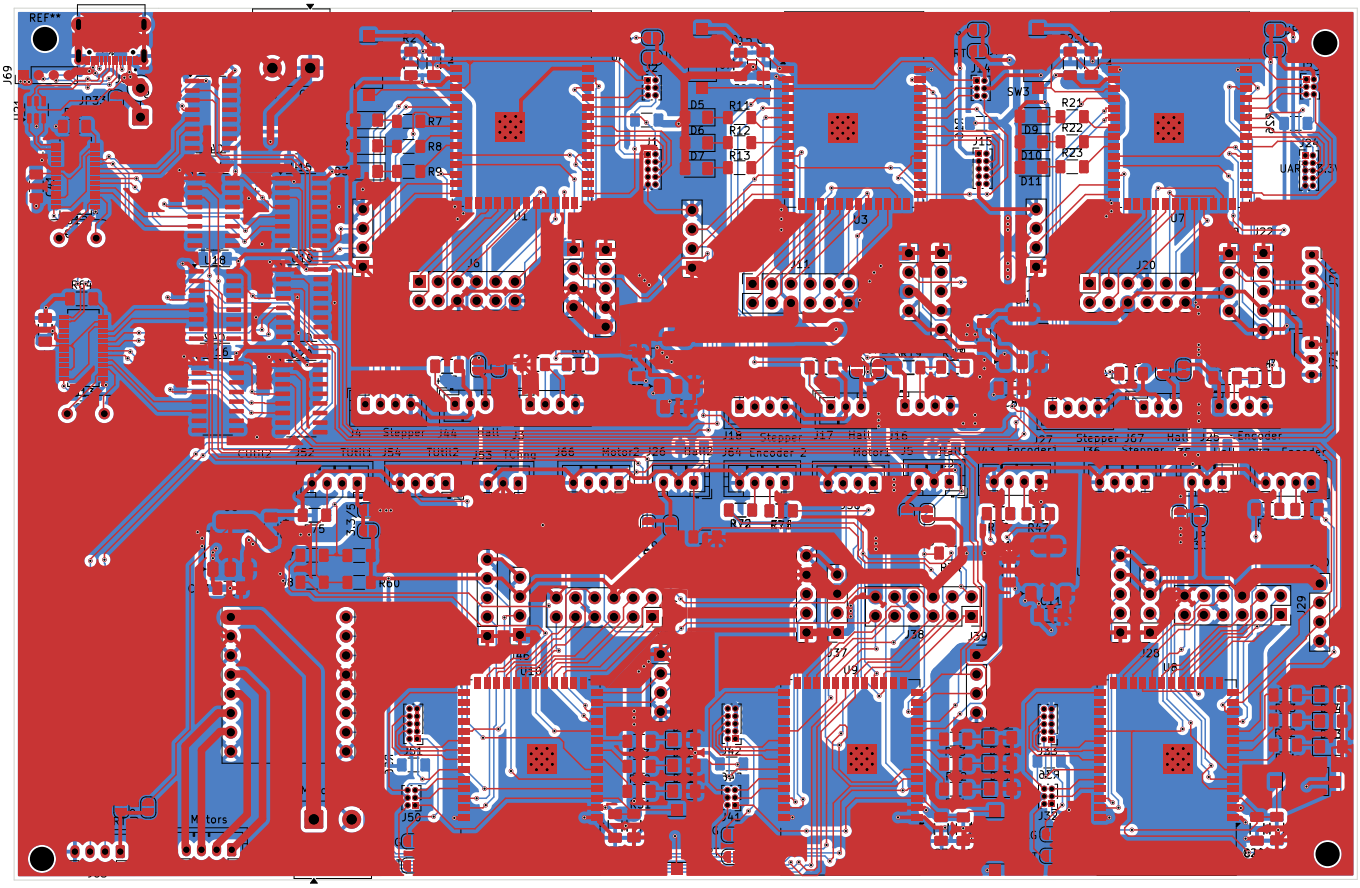


Sheet: /Common/
File: common.kicad_sch

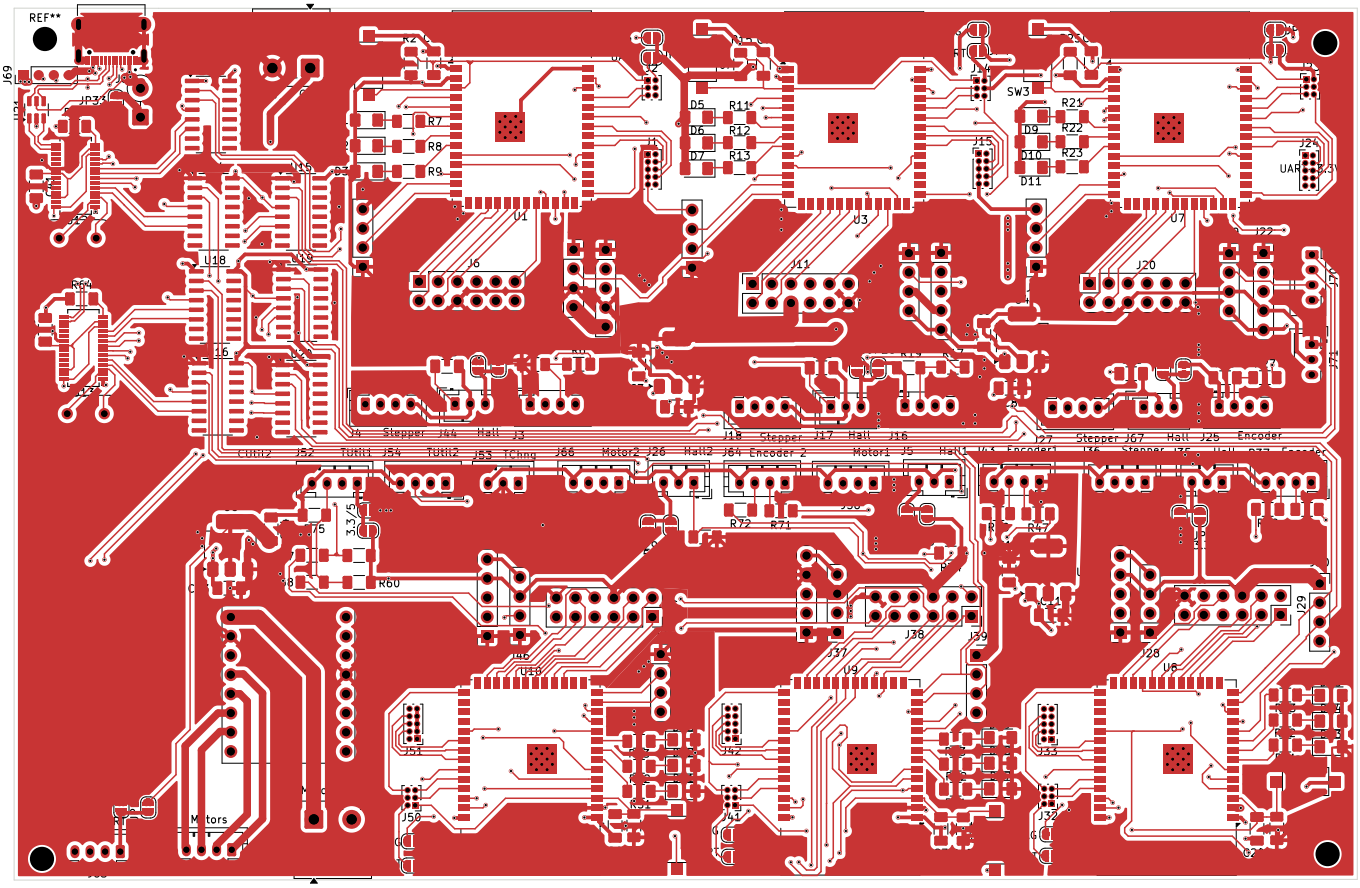
Title:

Size: A4	Date:	**Rev:**
KiCad E.D.A. 9.0.4		Id: 11/10

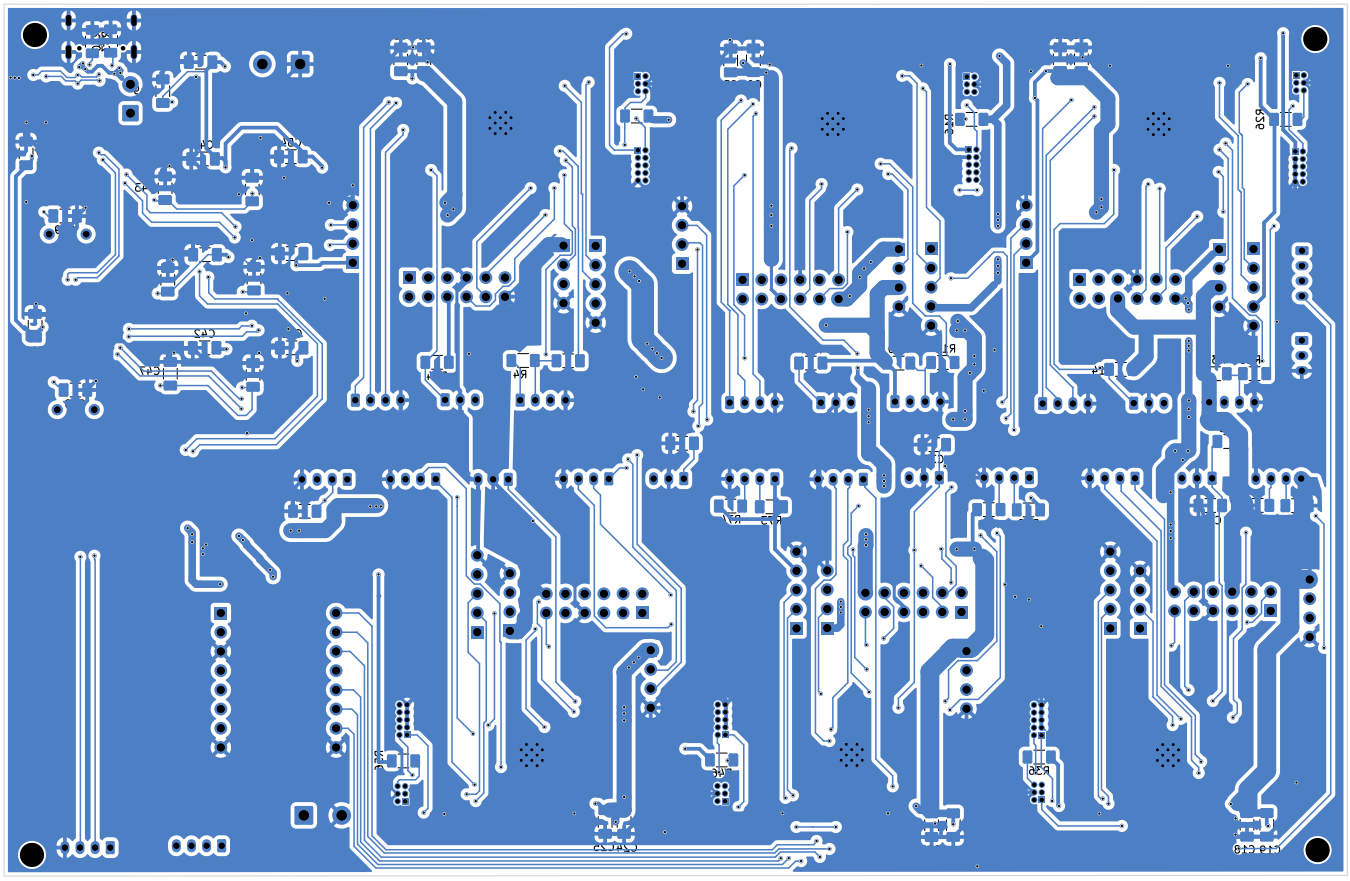
I. Printed Circuit Board Layout of the Second Revision



Sheet:		
File: robotpcb.kicad_pcb		
Title:		
Size: A4	Date:	**Rev:**
KiCad E.D.A. 9.0.4		Id: 1/1



Sheet:		
File: robotpcb.kicad_pcb		
Title:		
Size: A4	Date:	**Rev:**
KiCad E.D.A. 9.0.4		Id: 1/1



Sheet:		
File: robotpcb.kicad_pcb		
Title:		
Size: A4	Date:	**Rev:**
KiCad E.D.A. 9.0.4		Id: 1/1