SCHOOL OF ENGINEERING AND DESIGN
AEROSPACE AND GEODESY

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Mechatronics and Robotics

Towards Dual-Arm Aerial Manipulation: An Exploration in Robotic Arm Design

Samuel Helmut Zeitler

SCHOOL OF ENGINEERING AND DESIGN
AEROSPACE AND GEODESY

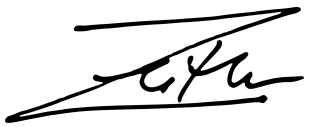
TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Mechatronics and Robotics

Towards Dual-Arm Aerial Manipulation: An Exploration in Robotic Arm Design

Author:	Samuel Helmut Zeitler
Supervisor:	Prof. Markus Ryll
Advisor:	Prof. Markus Ryll
Submission Date:	25.05.2024

I confirm that this master's thesis is my own work and I have documented all sources and material used.



Munich, 25.05.2024 Samuel Helmut Zeitler

Acknowledgments

I am grateful to Prof. Markus Ryll for providing the opportunity to engage with this master thesis project. Developing a robotic arm from the ground up has been an invaluable experience, allowing me to apply my theoretical knowledge in a practical setting. This project has been a significant part of my master's degree, and I thank him for entrusting me with such an innovative task.

My heartfelt thanks also go to my friends and family for their invaluable support. Their assistance during manufacturing and assembly, patience during demonstrations, and understanding of the time commitments involved have been vital.

Abstract

This master's thesis, titled *Towards Dual-Arm Aerial Manipulation: An Exploration in Robotic Arm Design*, is part of a project at the Chair of Autonomous Aerial Systems at the Technical University of Munich, supervised by Prof. Markus Ryll. The project's vision is to develop a prototype of an unmanned aerial vehicle (UAV) equipped with two robotic arms, advancing practical research in dual-arm aerial manipulation.

The focus of this thesis is the design and prototyping of one of these arms, representing the third work package in the project's journey. It builds upon the work of previous students by advancing the earlier prototype to an application-ready state. This thesis serves as a standalone document that details the developed hardware and software, establishing a foundation for future progress. It highlights the use of 3D printing as the primary manufacturing technique, supported by principles of lean product design and modern software development. As an engineering project, the thesis emphasizes learning and practical implementation over in-depth academic analysis, covering many topics, including control theory, software development, and mechanical design. This multidisciplinary approach ensures a comprehensive understanding of the complexities involved in robotic arm development.

Thorough testing of the redesigned prototype confirms its operational feasibility, demonstrating the ability to build a robotic arm from the ground up within the constraints of this student-led initiative. This work not only documents the progress of the project but also helps bridge the gap between theoretical research and practical implementation in robotics.

Contents

1 Introduction

Robotics has become an indispensable part of industrial, commercial, and personal applications, reflecting significant technological strides over the past few decades. A particularly dynamic growth area within this field is aerial robotics, offering versatile, innovative solutions for a variety of complex tasks. Aerial robotics extend the reach and flexibility of robotic applications, enabling novel implementations in inaccessible or hazardous environments where traditional robots cannot operate. The introduction to this thesis explores a sophisticated subset of aerial robotics - dual-arm aerial manipulators.

1.1 Importance of Dual-Arm Aerial Manipulation

The integration of robotic systems on unmanned aerial vehicles (UAVs) has transformed UAVs from primarily observational tools to robust platforms capable of performing advanced, interactive tasks. These UAVs equipped with robotic manipulators have been deployed in various applications, enhancing operational safety, versatility, and driving technical innovation across industries:

- **Construction and Assembly:** Assisting in assembling structures or applying materials in inaccessible places, enhancing precision in tasks such as applying protective coatings on wind turbines [Koc+22].
- **Disaster Response and Emergency Applications:** Performing critical tasks like shutting off gas valves or clearing hazardous debris during emergencies, significantly reducing human risk [Ors+14].
- **Environmental Monitoring and Sample Collection:** Collecting environmental samples from contaminated areas without human exposure, ensuring safe and accurate data collection [GO21].

Although these robotic systems appear in a variety of applications, the use of dual-arm aerial systems remains relatively limited in the industry. Yet, dual-arm manipulators offer enhanced capabilities over their single-arm counterparts, such as performing human-like tasks that require coordinated manipulation and complex

assembly. These capabilities could significantly expand their potential applications, making dual-arm systems a subject of increasing interest in both research and academia. The academic interest is exemplified by the AEROARMS project, a extensive research initiative that explores the capabilities of dual-arm UAVs in depth. The next section provides a brief overview of the project, highlighting current advancements in the field.

1.2 Research in Aerial Robotics: The AEROARMS Project

The AEROARMS project, supported by the Horizon 2020 initiative, represents a significant advancement in aerial robotics, focusing on integrating robotic manipulators on UAVs. The project features comprehensive documentation on the development and application of prototype systems, including the first aerial robotic manipulator with multiple arms for industrial inspection and maintenance. Figure 1.1 showcases a selection of the developed prototypes.

Figure 1.1: Various prototypes developed under the AEROARMS project, including dual-arm configurations [Oll+19].

The primary objective of AEROARMS was to expand UAV capabilities for complex tasks such as assembly, inspection, and contact-based interventions in environments

that are inaccessible or hazardous. To achieve this, the project leveraged and advanced a range of enabling technologies:

- **Advanced Control Systems:** The project developed integrated position and force control algorithms for combining manipulation and flight dynamics. This facilitates stable operations under variable conditions such as high winds and other dynamic disturbances [OHF+20].
- **Teleoperation and Haptic Interfaces:** These systems provide real-time feedback, allowing operators to remotely execute manipulation tasks, thereby enhancing operational safety and precision.
- **Autonomous Perception and Planning Systems:** Sophisticated algorithms for localization and mapping were developed, along with comprehensive planning systems that address the complexities of aerial manipulation.

The AEROARMS project has advanced the field of aerial robotic manipulation by demonstrating the feasibility of prototyping UAVs with dual-arm manipulators. Its broad scope and innovative solutions continue to influence current research and expand the commercial applications of UAVs. The project also addresses current challenges and opportunities in aerial manipulation, which will be explored in the next section.

1.3 Challenges and Opportunities in Aerial Manipulation

Aerial manipulation merges the complexities of robotics with the dynamic environment of UAVs, posing unique challenges that have so far limited its commercial deployment. Integrating manipulation capabilities increases the UAVs' weight and power demands, complicates their control systems, and requires exceptionally stable and precise flight capabilities. Moreover, the operational risks and stringent safety standards, combined with evolving regulatory frameworks, add further complexity. Despite these hurdles, the field offers substantial opportunities for academic innovation, particularly in developing advanced control algorithms and creating sophisticated autonomous perception and planning systems for UAVs [Ors+18]. The availability of technologies such as 3D printing, open-source software, and affordable actuators lowers entry barriers, making aerial manipulation an excellent subject for educational and practical exploration in engineering . This context directly shapes the objectives of this master thesis, selected for the substantial opportunities aerial manipulation offers in practical engineering work and academic research.

1.4 Objectives and Scope of this Study

This master thesis, entitled "Towards Dual-Arm Aerial Manipulation: An Exploration in Robotic Arm Design," represents a continuation and expansion of a project initiated in 2021 at the Chair of Autonomous Aerial Systems at TUM under the supervision of Prof. Markus Ryll. This project is of academic nature with a focus on exploring the field of aerial manipulation from an engineering point of view. The project consists of iterative contributions from individual students at TUM who seek to apply their theoretical knowledge in practice. Managed by a single student at a time, this project stands in contrast to the larger, collaborative effort of the AEROARMS project.

1.4.1 The Projects Vision and Purpose

As depicted in Figure 1.2, the project envisions a UAV equipped with two robotic arms. Designing such a complex system typically involves extensive resources, including a team of specialized research engineers and substantial funding. However, this student-led initiative operates with limited resources and manpower, which means progress may be slow. Nonetheless, the project offers a unique platform for students to delve into the expansive field of aerial robotics within an academic framework. It allows participants to apply their theoretical knowledge to a tangible system, serving as an excellent research topic for aspiring engineers. Given its academic background, the primary aim of this project is exploration and independent learning.

Figure 1.2: Project Vision: UAV with two robotic arms.

1.4.2 Current Thesis Objectives

Being the third work-package within the projects journey towards aerial manipulation, this thesis is focused on the development and prototyping of the robotic-arms for the UAV. It progresses from an initial prototype, developed by previous students Jasper Sindermann and Samuel Zeitler. Figure 1.3 depicts the projects progress so far.

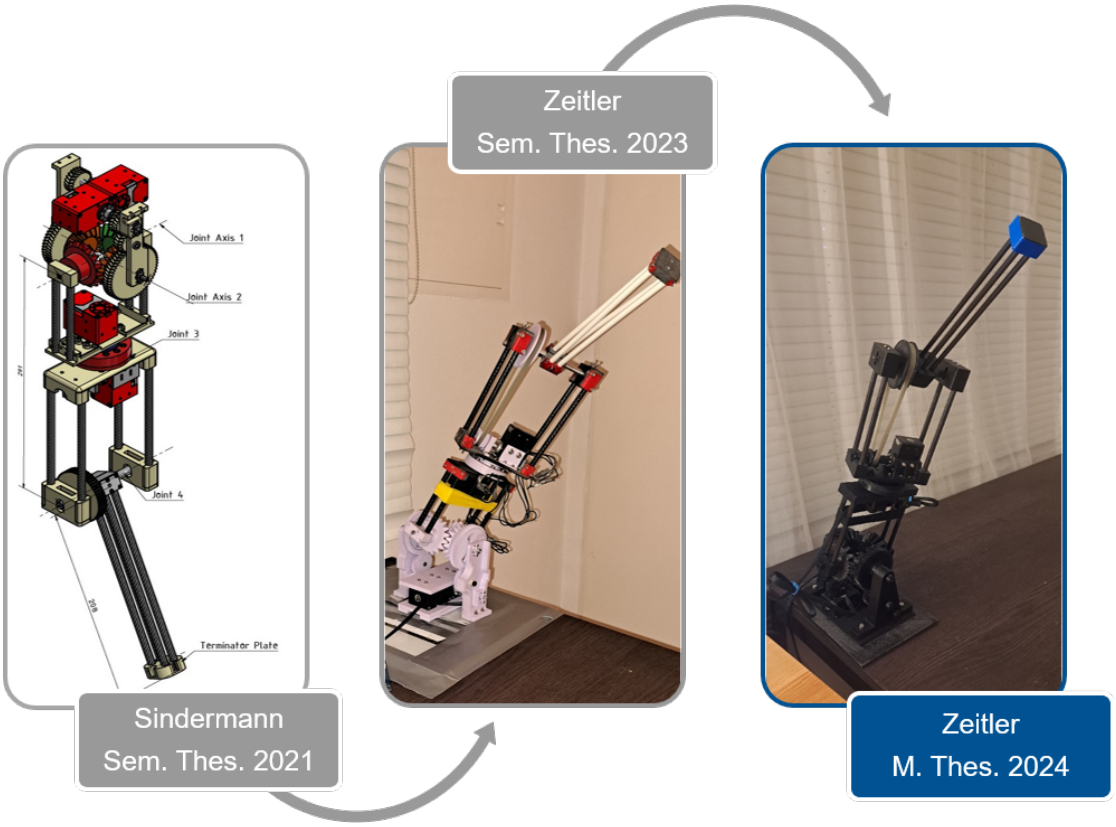


Figure 1.3: The work-packages of the project so far. **Left:** Initial Blueprint [Sin21], **Middle:** First Prototype [Zei23], **Right:** Revisioned Prototype [this].

The initial phase of this project was undertaken by Jasper Sindermann, who focused on the kinematics and mechanical design, setting the foundational parameters and requirements for aerial manipulation tasks [Sin21]. Samuel Zeitler then took this foundation further by developing the first operational prototype [Zei23], which marked a significant step from theory towards practical application. However, this prototype exhibited significant flaws, necessitating a comprehensive redesign to pave the way for the subsequent integration of two arms.

The focus of this master thesis is the redesign of the robotic arm, with the goal of elevating the prototype to an application-ready state. This step is crucial in establishing a foundation for subsequent efforts aimed at advancing the project towards dual-arm aerial manipulation. To ensure a methodical development process, the following key objectives have been established for the redesign:

- **Obj. 1 - Mechanical Resilience:** Enhance the robustness of the mechanical design to withstand operational stresses and allow for increased dynamic capabilities.
- **Obj. 2 - Motion Accuracy:** Enhance the robotic arm's precision in both determining its current pose and in executing motion tasks. Specifically, the arm should accurately achieve desired poses and precisely reach targeted end effector positions within the task space.
- **Obj. 3 - Efficient Manufacturing and Assembly:** Ensure that all parts can be recreated with reduced effort. Aim for a simpler and quicker assembly process.
- **Obj. 4 - Operational Safety:** Avoid self-collision, handle singularity configurations, enforce safety joint limits, handle faulty user input or other edge cases.
- **Obj. 5 - Simplified Interaction:** Develop an intuitive Graphical User Interface and streamline the process for creating movement programs, making it easier to define instructions for the robotic arm.
- **Obj. 6 - Extended Feature Set:** Incorporate Path-Planning, Trajectory Following, Nullspace Utilisation and Kinematic Simulation.
- **Obj. 7 - Documentation:** Produce comprehensive software and hardware documentation to facilitate a smooth handover to future contributors.

The primary aim of the written work of this thesis is to document the hardware and software development for future students participating in or interested in the project (Obj. 7 - Documentation). The documentation is designed to be thorough but concise, enhanced with figures, renderings, and photographs to make the information accessible and intuitive.

The development is presented systematically, detailing the rationale behind each design choice. Chapter 2 covers hardware development, while Chapter 3 focuses on software advancements. The evaluation of the redesigned prototype's suitability for aerial manipulation is conducted in Chapter 4, which showcases the prototype's capabilities in testing scenarios. The thesis concludes by summarizing key findings and insights and outlines future directions and potential enhancements for the robotic arm.

1.4.3 Differentiation and Contributions to Dual-Arm Aerial Manipulation

This project stands apart from extensive efforts like AEROARMS due to its student-led, resource-limited approach. Utilizing 3D printing technology as a key enabling tool, the developed prototype lowers both cost and entry barriers, making advanced aerial manipulation accessible at an educational level. The design philosophy adopted—emphasizing anthropomorphic, lightweight, and compliant features—parallels advanced industry standards but is adapted to be achievable with limited resources.

This master thesis not only builds upon the initiatives by Sindermann and Zeitler but also refines and broadens these efforts into a more robust system while adhering to a key development philosophy: maintaining simplicity to maximize both robustness and understandability. By focusing on this principle, it aims to make a unique contribution to the field of aerial robotics, offering a scalable and educational model that can be seamlessly continued by future students at TUM.

2 Hardware Design

This chapter offers a detailed exploration of the physical components that make up the robotic arm prototype, structured to mirror the chronological phases of its development. It begins with the establishment of design requirements and proceeds to the kinematic analysis, primarily drawing on the seminal work of Jasper Sindermann as documented in his foundational thesis [Sin21]. Subsequent sections delve into the mechanical design, detailing the redesigned CAD model of the arm and pivotal aspects of its manufacturing and assembly processes. The chapter concludes with a discussion of the actuators used and their integration into the system.

2.1 Requirements

The foundational design of the robotic arm, as established in Jasper Sindermann's thesis, is based on requirements aimed at optimizing it for dual-arm aerial manipulation. These requirements will be summarized in the following paragraphs:

- **Req. 1 - Lightweight Design:** The robotic arm shall maintain a lightweight structure to ensure compatibility and operational efficiency when mounted on UAVs. The total system mass, including a changeable end-effector module, shall not exceed 1 kg, with the end-effector module itself weighing no more than 0.1 kg. To reduce torques and enhance controllability, a substantial portion of the mass should be concentrated close to the base, minimizing inertia forces during dynamic maneuvers.
- **Req. 2 - Workspace and Kinematics:** : The kinematics of the robotic arm should be optimized for functionality when deployed on an aerial platform. Each arm shall offer a large individual workspace to enable varied manipulation tasks without requiring drone rotation. The design should also support overlapping workspaces between the two arms to facilitate coordinated activities at multiple points simultaneously. This arrangement shall allow the arms to execute arbitrary movements throughout their workspace while avoiding self-collision. The functional length of each arm is set to approximate the length of a human arm from shoulder to wrist at about half a meter.

- **Req. 3 - Dynamics:** The robotic arm shall manage its own weight plus an additional 0.2 kg payload in any configuration and exert a force of 3N to meet operational demands. These criteria shaped Sindermann's dynamic simulation framework, which utilized a Lissajous trajectory to assess the effects of inertia, centrifugal, and Coriolis forces, as well as gravitational impacts. The simulations resulting torque and speed requirements towards the arm are provided in the following table:

Table 2.1: Torque and operational speed requirement for each joint of the robotic arm as calculated in Sindermann's dynamic analysis [Sin21].

Description	Joint 1	Joint 2	Joint 3	Joint 4	Unit
τ_{total}	3.86	3.61	2.38	3.93	Nm
v_{joint}	2.9	2.9	4.1	6.8	rpm

2.2 Kinematic Design

Kinematic design in robotics refers to the process of defining the motion parameters and joint configurations that enable the robotic arm to achieve desired positions and trajectories. This involves determining the number, type, and range of joints, as well as their relative positions to each other, based on the specific task requirements. Detailed insights on how Sindermann derived a suitable kinematic design with large individual and overlapping workspaces (Req. 2 - Workspace and Kinematics) are available in his work [Sin21]. To maintain conciseness, this section will skip ahead to the resulting kinematic configuration illustrated in Figure 2.1, which showcases the order of joint rotations for the 4-DOF (Degrees of Freedom) robotic arm.

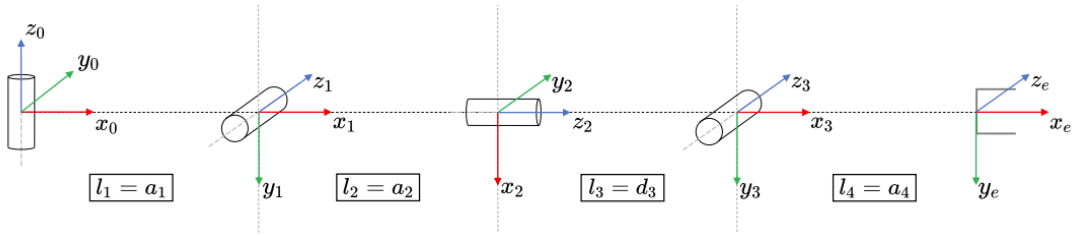


Figure 2.1: Denavit-Hartenberg representation of the chosen 4-DOF configuration. Order of rotation in global coordinates: **y-x-z-x**.

After deciding on the arm's configuration, Sindermann determined the link lengths for the four modules of the robot through numerical optimization of a self-developed kinematic metric, λ.

$$\lambda = V_1 \cdot V_2 \cdot V_i \cdot \bar{c}_{\text{ext,m}} \tag{2.1}$$

This metric evaluates the operational workspace, considering both distinct (V_1, V_2) and intersecting (V_i) workspace volumes of the dual-arm system. Additionally, it includes a manipulability index $\bar{c}_{\text{ext,m}}$, which evaluates the robot's capacity to accurately position and orient its end-effector — a key indicator of its dexterity. For an in-depth explanation of this manipulability index see [HB15]. The resulting link lengths, detailed in Table 2.2, maximize the kinematic metric λ.

Table 2.2: Optimized link lengths of the arm.

Link	l_1	$l_2 + l_3$	l_4	Unit
Length	0	0.31	0.22	m

2.3 Mechanical Design

Upon finalization of the requirements and kinematic parameters, the mechanical design process commenced. The immediate challenge was to embody the kinematic parameters into a concrete CAD design that incorporated all essential elements: joints, links, gears, and actuators. Sindermann laid the groundwork with an initial blueprint in his thesis, which was further refined and brought to life by Zeitler in his semester thesis. Although the kinematic parameters of the arm remained consistent throughout all iterations of the robotic arm, the CAD design underwent a significant evolution, as depicted in Figure 1.3. The design journey progressed from a conceptual blueprint in 2021 to a first prototype in 2023, and finally to an application-ready manipulator in 2024, with each stage increasing in sophistication and feasibility.

This section presents the revised mechanical design of 2024, highlighting key improvements made compared to the previous prototype. It is organized to first provide a comprehensive overview, then delve into detailed examinations of the individual modules of the robotic arm.

2.3.1 Overview

The mechanical design of the robotic arm, depicted in Figure 2.2, features a lightweight architecture achieved through the use of minimalist structural components, such as carbon fiber tubes. These tubes serve as links between joints, forming a stiff and robust body. This design is crucial for ensuring the arm's dynamic stability, particularly when integrated into aerial platforms. By strategically placing heavier components, like the actuators, close to the base, inertia forces are significantly reduced (Req. 1 - Lightweight Design). A belt connects the final link to the centrally-mounted actuator, eliminating the need for an actuator in the elbow joint. The design resembles the human arm, with distinct shoulder, yaw, and elbow joints — a deliberate choice to facilitate tasks traditionally performed by humans.

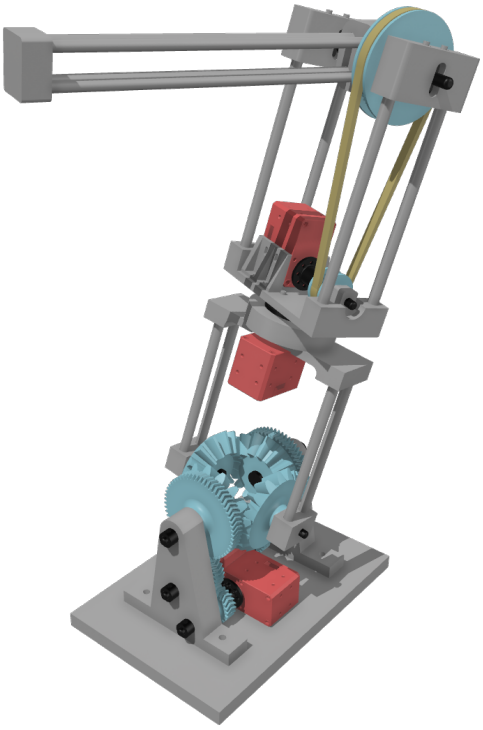


Figure 2.2: Complete CAD render of the robotic arm with highlighted components: actuators in red, gears in blue, and axes in black, while structural parts are grey.

The rectangular shape at the bottom side of Figure 2.2 is the base or mounting plate. Currently, this plate enables the arm to be mounted in an upright position on a table using screw clamps. In the future, it will also facilitate the attachment of the arm

in its designated hanging position to a UAV. Currently, no end-effector module has been added to the arm, allowing for future dedicated development tailored to various applications

The simplicity of the part geometries is a deliberate design choice aimed at maximizing ease of manufacturing and assembly (Obj. 3 - Efficient Manufacturing and Assembly). Several original components were either simplified or merged into composite units. For instance, all rotational axes are mounted within holes in the 3D-printed structural parts (clearance fit), serving as plain bearings with minimal lubrication. Despite their simplicity, these improvised bearings are adequate for the arm's low operational speeds and simplify maintenance.

The subsequent sections will delve deeper into the individual joints of the robotic arm, discussing the various challenges encountered and the innovative solutions implemented throughout its development. The complete CAD model and the code base supporting this prototype are readily accessible for public review and download at the provided GitHub repository[1].

2.3.2 The Shoulder Joint

The shoulder joint of the robotic arm is a critical component integrating two degrees of freedom (DOF) in one assembly, utilizing a differential gear system. This differential consists of two opposing driven bevel gears and two opposing non-driven bevel gears, as shown in Figure 2.3.

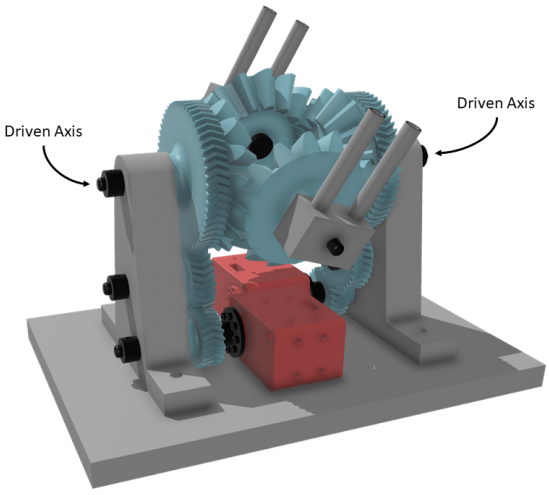


Figure 2.3: The Shoulder Joint.

[1] `https://github.com/Friedsam2000/Robotic-Arm-Prototype.git`

The driven axes are powered by a reduction gear assembly. This symmetric setup comprises three gears per side, achieving a gearing ratio of $i = 2.5$. However, as the differential setup evenly divides the torque over both servos, the total gearing ratio is $i_{shoulder} = 5$. This significantly reduces the servo torque required to manage the shoulder joint's load. Given the required shoulder joint torques of $\tau_1 = 3.86$ Nm and $\tau_2 = 3.61$ Nm, the resulting servo torque is approximately 1.5 Nm, which can be provided by contemporary small servo motors.

A critical insight from the previous prototype was the necessity for sufficient clamping force in the differential gear pairing, as depicted in Figure 2.4. Similar to a human shoulder, the 2-DOF joint can 'dislocate' — meaning the bevel gears can separate, causing system failure. During dynamic testing of the first prototype, this issue frequently occurred, significantly limiting the arm's precision and load-bearing capabilities. The subsequent redesign of the shoulder joint addresses this issue by incorporating larger bevel gears and reinforcing the triangular side braces that mount the driven axes. Additionally, an elastic band was wrapped around the first set of carbon fiber tubes to provide extra support. While not an optimal solution for application readiness, this tensioning mechanism provides enough clamping force on the shoulder joint to function reliably even in highly dynamic situations (Obj. 1 - Mechanical Resilience).

An additional drawback of the 2-DOF design is the limited range of motion (ROM). The shoulder joint must maintain an elevation of at least 40° (spherical coordinates) to avoid self-collision. This results in a ROM of 100° for both axes, significantly less compared to the human shoulder's ROM of about 180°.

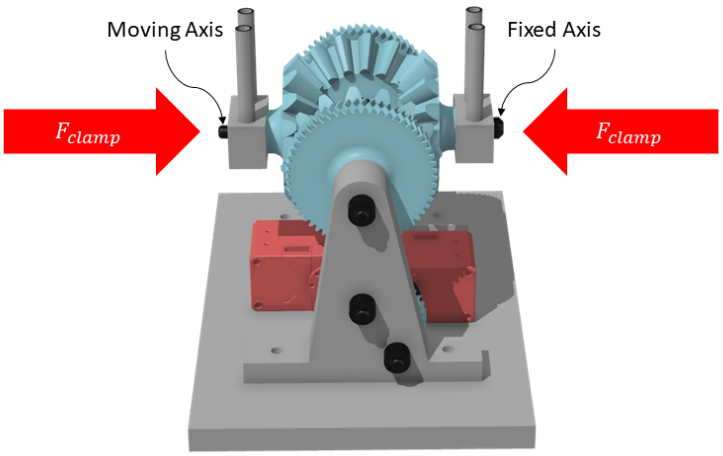


Figure 2.4: Shoulder Joint, red arrows indicating the need for clamping force.

2.3.3 The Yaw Joint

The yaw joint provides the third degree of freedom (DOF) for the robotic arm, functioning as a fully 3D-printed plain bearing that enables the middle link of the arm to rotate around its local z-axis. This setup is illustrated in Figure 2.5, where components moving in unison are color-coded.

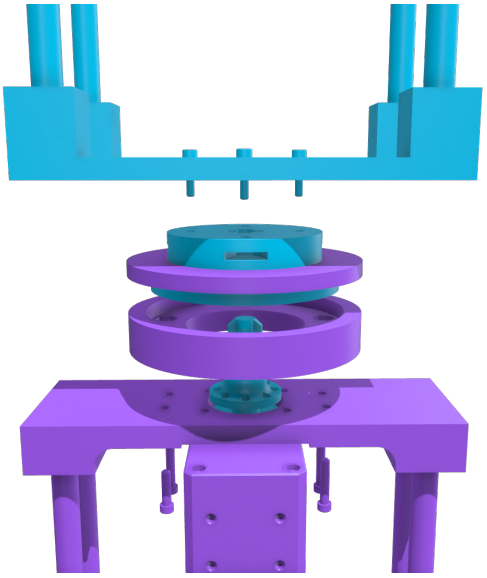


Figure 2.5: The Yaw Joint, with parts moving together highlighted in the same color.

Originally, the yaw joint was designed to include a complex cycloidal gearbox to amplify motor torque, as envisioned by Sindermann [Sin21]. Due to manufacturing challenges with such intricate components, the design was simplified to a basic 3D-printed axial bearing (Obj. 3 - Efficient Manufacturing and Assembly). This modification is compatible with the required joint torque of $\tau_3 = 2.38$ Nm. This torque lies on the edge of the stall torque range specified by contemporary servos, making an external transmission optional.

The simplified yaw joint is easy to manufacture, involving only four parts. The 'Plug-In-Connection' between the actuator and the joint only allows torque transfer, with axial and bending forces being offloaded to the joint's housing, thereby protecting the actuator. The large bearing surface within the joint necessitates lubrication to reduce wear and ensure smooth operation. Silicone grease, applied during assembly, was used for this purpose.

One drawback of the direct drive configuration is that the positional error of the joint equals the actuator's positional error, necessitating higher controller gains and

negatively impacting the stability and accuracy of the joint. Future design iterations might incorporate a planetary gearbox or explore direct position control to mitigate this issue. This control challenge is addressed in section 2.5.

The yaw joint's range of motion is constrained only by the slack in the cable connecting to the actuator of the subsequent elbow joint. In the current prototype, the yaw rotation limit is set at 315° in both directions, a range that does not significantly impair the robotic arm's dexterity, as supported by the control strategies outlined in Chapter 3.

2.3.4 The Elbow Joint

The elbow joint represents the final joint in the kinematic chain of the robotic arm, facilitating flexion and extension movements. The belt drive allows the actuator to be mounted in the middle link of the arm, keeping the weight closer to the base (Req. 1 - Lightweight Design). The elbow joint is illustrated in Figure 2.6.

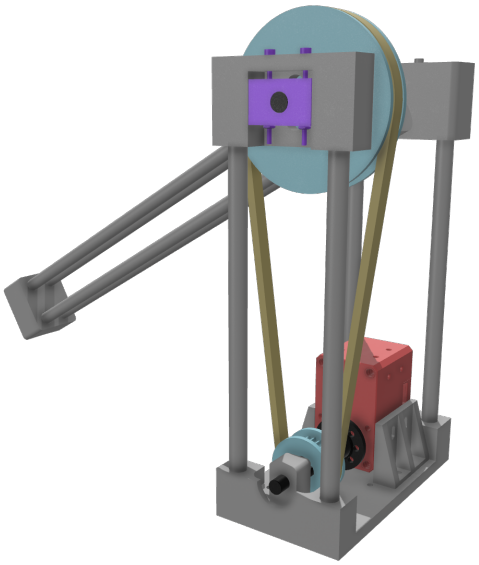


Figure 2.6: Elbow Joint with the belt tensioning mechanism highlighted in purple.

This joint has undergone minimal modifications from its original design. Notably, the size of the pulley gear was increased, and a rim was added to prevent the belt from slipping off, enhancements first implemented in the initial prototype by Zeitler [Zei23].

A critical feature of the elbow joint is its belt tensioning mechanism (highlighted in purple). Initially, after the assembly of the robotic arm, the belt is left to hang loosely between the rims of the pulley gear and the elbow gear. Tension is then applied by turning the screws atop the side braces, which lift the axis of the elbow

joint. This adjustment ensures the belt can effectively transmit torque from the actuator to the elbow joint without slipping. The belt drive provides a transmission ratio of approximately $i_{\text{elbow}} \approx 2.5$ and enables a range of motion up to 300°. This is notably double the range of motion of a typical human elbow, highlighting the joint's enhanced flexibility and functionality.

2.4 Manufacturing and Assembly

The components of the robotic arm are predominantly manufactured using PETG filament on home-scale 3D printers, with the exception of the belt, actuators, and carbon fiber links, which are commercially sourced. PETG was chosen over PLA for its superior strength and durability [Ram+23]. All printed parts were created on a Creality Ender 3 S1 Pro[2].

3D Printed Axes

The axes, originally intended to be milled from aluminum, are also 3D printed to facilitate rapid prototyping and part replacement (Req. 3 - Efficient Manufacturing and Assembly). The Shaft-Hub-Connection, such as between gear and axis, utilizes a LEGO®-inspired design, simplifying the assembly process and strengthening the connection between printed parts. The printing process involves iterative prototyping to ensure precision fits—a time-consuming but critical process. A tight fit between the axes and the mounted gears or structural parts proved crucial for structural stability.

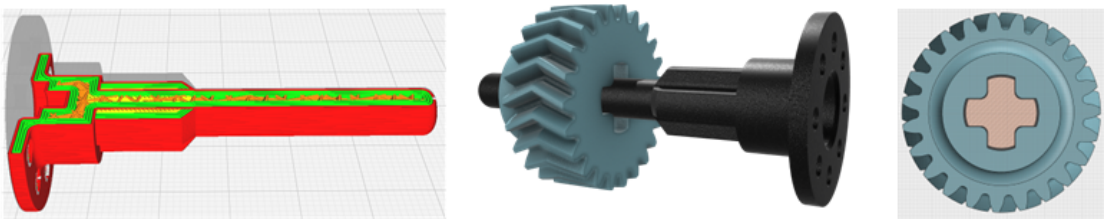


Figure 2.7: LEGO®-inspired Shaft-Hub-Connection. **Left:** Layer orientation of the 3D printed axis, **Middle:** Visualization with a mounted gear, **Right:** Sectional view.

[2]The printing profiles and notes are available in the GitHub repository: `https://github.com/Friedsam2000/Robotic-Arm-Prototype.git`

The layer orientation in all 3D printed axes is planned so that the layers are perpendicular to operational forces, enhancing component durability while slightly complicating the printing process. Figure 2.7 visualizes the layer orientation for the 3D printed axis alongside 3D renderings of the LEGO®-inspired Shaft-Hub-Connection.

Assembly

The assembly of the robotic arm, while not easy and quite time-consuming, has been significantly simplified compared to earlier prototypes (Req. 3 - Efficient Manufacturing and Assembly). The arm consists of four main assembly groups, each of which can be assembled independently before being combined. These groups, illustrated in Figure 2.8, are the Base group 1, Lower group 2, Upper group 3, and the End-effector group 4.

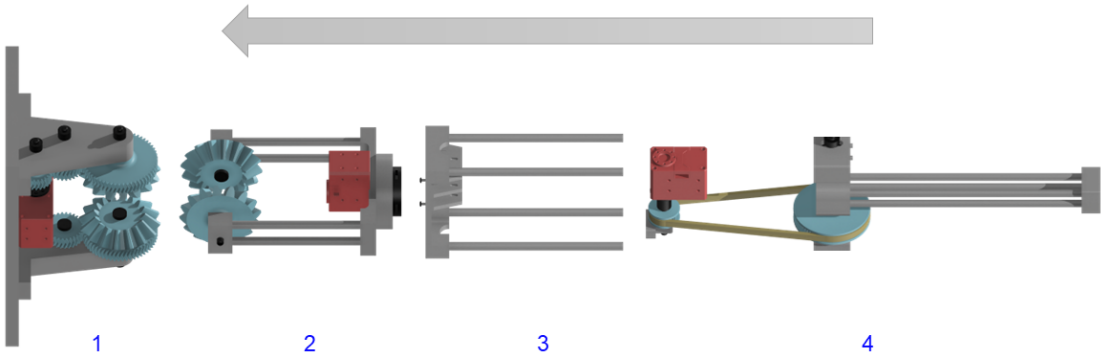


Figure 2.8: Robotic arm separated into assembly groups. The arrow indicates how the four groups fit together.

The Lower group links with the Base group via the 2-DOF shoulder joint, relying on sufficient clamping force to hold the bevel gears together. The Upper group is secured onto the yaw joint using four screws with nuts. Lastly, the End-effector group is lowered onto the Upper group, with the carbon fiber tubes sliding into the side braces. The connection is secured by four screws that attach the actuator to the structural parts and two screws that hold the pulley gear's axis. Tightening the belt provides enough static force to ensure the carbon fiber tubes remain in place. To maintain the conciseness of this thesis, detailed assembly instructions for the four groups are not included here but can be inferred from the CAD design.

2.5 Actuation

The actuation system of the robotic arm employs four Dynamixel XH-430-W210-T servo motors, distributed with one motor each for the elbow and yaw joints, and two for the shoulder joint. These servos are chosen for their compact form factor, high torque-to-weight ratio, and excellent performance characteristics, making them particularly suitable for aerial robotics applications.

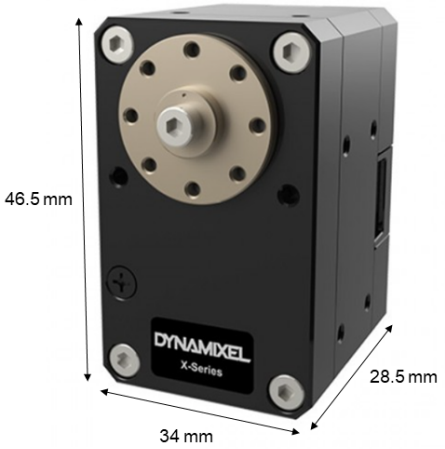


Figure 2.9: Dynamixel XH-430-W210-T

Servo Specifications

The servos used in the robotic arm have the following key specifications:

- Operating Voltage: 10V - 14.8V
- Gear Ratio: 210:1
- Stall Torque: 2.5 N.m at 12.0 V and 1.3 A
- Maximum Velocity: 50 rev/min at 12 V
- Weight: 82 g
- Communication Interfaces: TTL 3-pin (GND, VDD, DATA)

Each dservo in the robotic arm is powered by an ARM CORTEX-M3 microcontroller (72 MHz, 32-bit), featuring integrated controllers over servo angle, velocity and current.

They also incorporate a high-resolution AMS AS5045 Rotary Sensor, offering a 12-bit (4096 positions per revolution) measurement precision. Further information about the servo is available here [24b].

Integrated Velocity Controller

The Dynamixel servos are equipped to support multiple control modes, enhancing their adaptability for different robotic applications. These include the current control mode for managing torque and the position control mode for precise joint positioning within predefined limits. However, the primary control strategy adopted for this prototype is velocity control, which was selected for its simplicity and robustness.

Velocity control is less complex than position or current control systems, requiring fewer sensors and consuming less computational power. This not only enhances system reliability but also reduces the risk of errors during the initial development phases. In this mode, the Main Controller calculates the required joint velocities based on the desired motion of the robotic arm. The servo's integrated microcontroller then focuses on achieving these velocities, utilizing the control scheme illustrated in Figure 2.10.

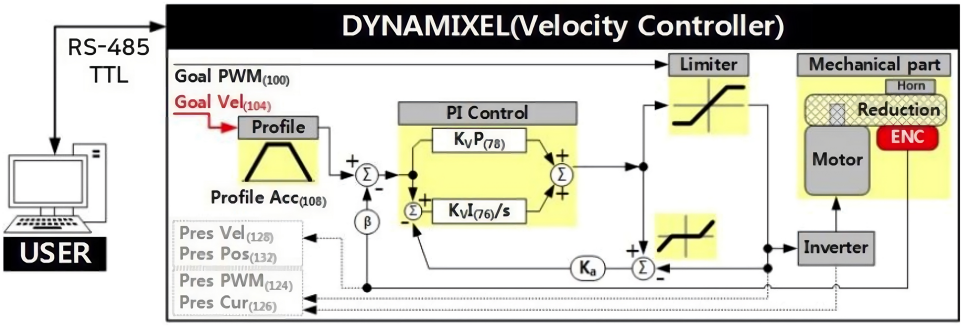


Figure 2.10: Diagram of the Velocity Control used in the Dynamixel servos [24b].

To enhance the system's operational stability, a trapezoidal acceleration profile is used to moderate the acceleration and deceleration phases, thereby reducing mechanical stress and vibration. This profile is particularly effective in creating smoother joint movements. Servo parameters such as profiles and gains can be tuned using the Dynamixel Wizard [24c], a graphical user interface for managing the servos. Despite the general success of the default PI gains across most joints, the yaw joint presents a unique challenge, as referenced in section 2.3.3. This joint lacks a transmission, which directly links servo velocity to joint velocity, leading to controller problems at low speeds due to the high stick-slip friction of its large plain bearing. A substantial torque is required to initiate movement, and because torque is proportional to velocity error

in velocity control mode, small errors necessitate high PI gains. However, excessively high PI gains can destabilize the joint, presenting a trade-off that has required manual adjustments to the PI gains.

Communication between the Servos and Main Controller

Communication between the Dynamixel servos and the Main Controller, which is a PC running MATLAB for this prototype, is facilitated through a TTL Half Duplex Bus. It supports 8-bit, 1-stop, No Parity configurations, suitable for robust and synchronized data transmission. The system diagram, as shown in Figure 2.11, illustrates three Dynamixel servos wired in series, all connected to the Main Controller via a bus.

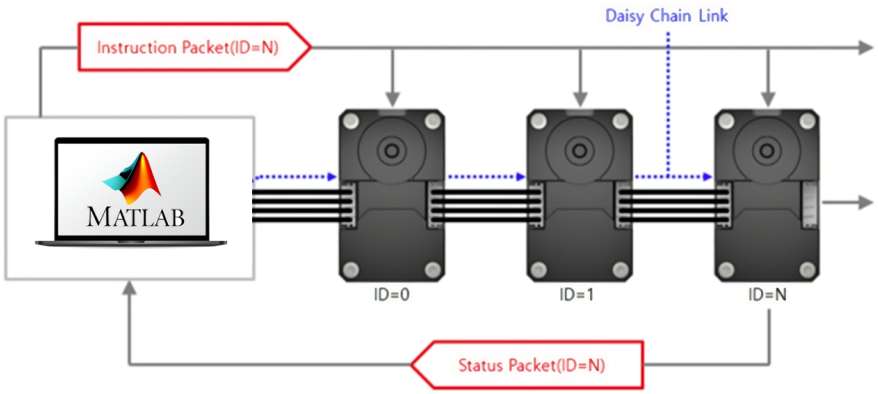


Figure 2.11: Diagram of the Dynamixel servos connected in series on a bus, communicating with a PC as the Main Controller [24e].

The communication is carried out using two types of packets: Instruction Packets and Status Packets. The Main Controller sends Instruction Packets to control the servos by specifying parameters such as the desired velocity. Or it requests operational data from the servos like the current servo angles. In response, the servos send back Status Packets, which provide the requested data, effectively completing a feedback loop necessary for the robotic arms control.

To connect the PC that runs the Main Controller to the Dynamixel bus, a usb-to-serial converter is necessary. For this prototype a properietary solution by Dynamixel was chosen, namely the U2D2 [24d].

Dynamixel Software Development Kit

The software interface for controlling the robotic arm is built around the Dynamixel SDK[3], a comprehensive software development kit that provides control functions for Dynamixel actuators using packet communication. The SDK features a dynamic link library that is written and built in C, facilitating direct control over the servos' built-in features.

In the current prototype, the Main Controller is a PC running MATLAB, which interfaces with the servos via a U2D2 adapter [24d]. This setup enables the use of MATLAB's extensive analytical tools alongside the Dynamixel SDK, providing a powerful platform for developing and testing control algorithms for the robotic arm. However, this integration also introduces significant computational overhead. MATLAB must handle data conversion between its own data types and the native C types used in the SDK's dynamic link library, manage memory, and ensure correct data passage to the underlying C functions each time the 'calllib' function is invoked. This creates a bottleneck for the achievable control loop frequency, as later tests will reveal. To address these challenges and improve system responsiveness, future iterations of the project might consider migrating the Main Controller functions to a compiled language such as C++.

The introduction of the Dynamixel SDK, which serves as the interface between the Main Controller and the servos, sets the stage for the forthcoming chapter on Software Design. This next chapter will focus on the integration and development of the Main Controller, representing a significant portion of the work in this thesis.

[3] `https://github.com/ROBOTIS-GIT/DynamixelSDK`

3 Software Design

This chapter shifts focus from the exploration of individual actuators to a discussion of the software suite that orchestrates all servos, integrating them into a fully operational robotic arm. This suite is developed in MATLAB and includes the Main Controller, tasked with calculating the desired joint velocities based on desired arm motions. Additionally, it adds peripheral functionalities such as visualization, simulation, and user interface, along with numerous smaller features. The entire codebase is accessible for download at the designated GitHub repository[1].

Software development commenced in 2023 alongside the creation of the first prototype by Zeitler [Zei23] and has undergone significant evolution to enhance its capabilities while maintaining efficiency, conciseness, and clarity. The ensuing sections will methodically build up the software, providing insights for future developers and establishing a foundation for further discussion.

Core Development Strategies

As of the time of this master thesis, the software suite was developed by only one developer with limited time and resources. To still be able to deliver reliable software for the robotic arm prototype and fulfill the objectives specified in section 1.4.2, several strategies have proven crucial throughout the development:

- **MATLAB**: The development environment of choice is MATLAB, offering built-in functionalities for visualization and complex calculations that expedite development processes. The project relies exclusively on MATLAB's extensive array of functions, minimizing dependence on external libraries except for the essential Dynamixel SDK, which facilitates communication with the servos.
- **GIT**: The software's version control is managed through a Git repository, which is a fork of the Dynamixel SDK. This setup allows for seamless integration of updates from Dynamixel into the current software framework, ensuring that the project remains up-to-date with the latest servo functionalities.

[1] `https://github.com/Friedsam2000/Robotic-Arm-Prototype.git`

- **OOP**: The software development utilizes Object-Oriented Programming (OOP) to handle code complexity and improve maintainability effectively, allowing the complete codebase to be represented as a UML Diagram in Figure 3.1. The different classes are arranged in colored groups that represent chunks of functionality.
- **Philosophy**: The software design philosophy centers on simplicity and clarity to ensure the system is approachable for developers at all levels. It stresses the importance of keeping the code concise and intelligible, closely aligning calculations with the underlying physical principles, and designing a modular architecture that is straightforward to extend.

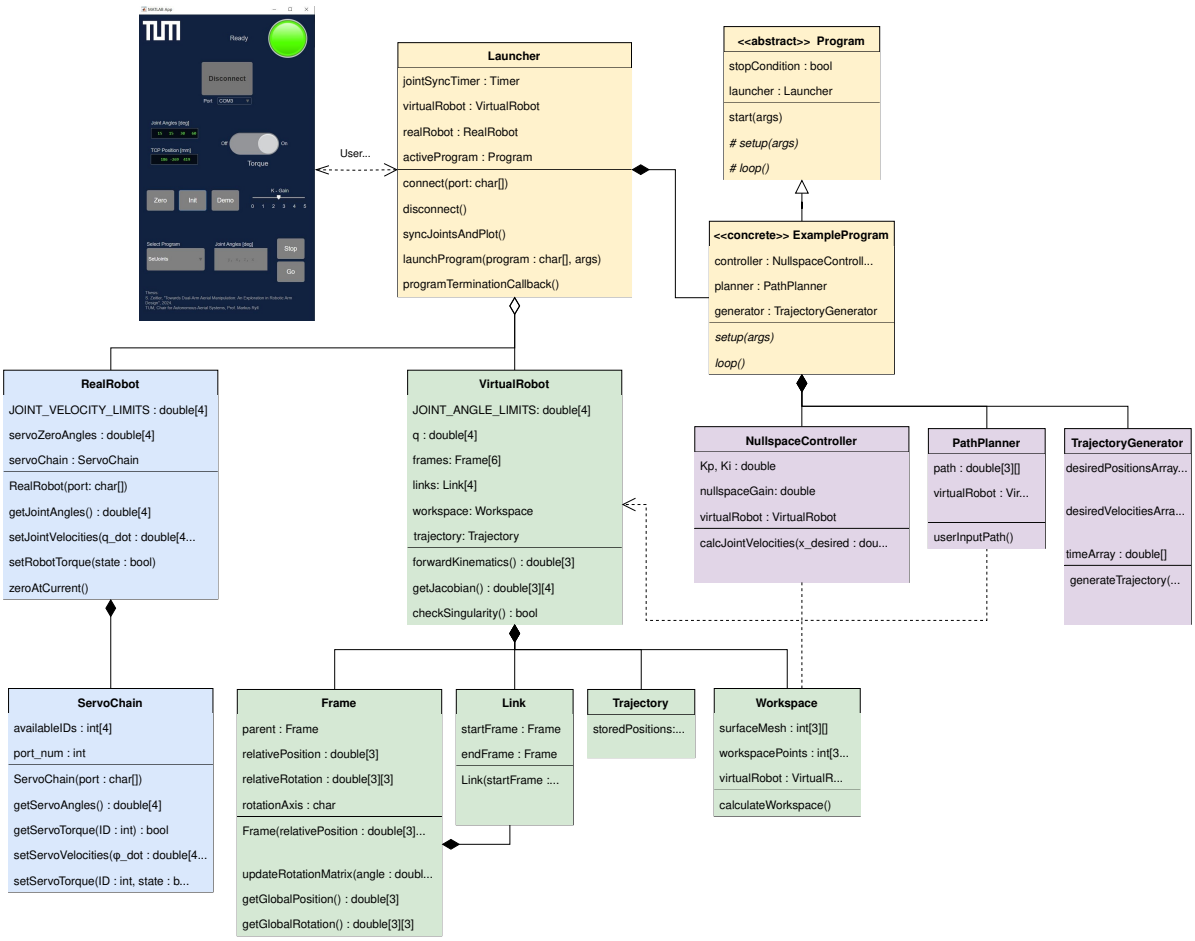


Figure 3.1: UML Diagram representing the developed software suite, segmented into color-coded functional groups.

Software Suite Overview

This section provides an organized walkthrough based on the UML Diagram depicted in Figure 3.1. The diagram is segmented into four color-coded groups, each representing a set of closely related functionalities. Each group will be explored in depth in their respective sections to illustrate how they contribute to the overall functionality of the software suite. Below is a brief overview of what will be discussed:

1. **Virtual Robot Framework (green)**: At the heart of the software suite, a Virtual Robot acts as a kinematic model and visualisation of the robotic arm. It is used to mirror the real arm's configuration and provide kinematic calculations, e.g. forward and inverse kinematics for task space control. The Virtual Robot can also be used as a kinematic simulation to accelerate development.

2. **Real Robot Interface (blue)**: Serving as the crucial link between the dynamixel servos and the software suite, this interface retrieves servo angles to calculate the robot's joint angles. These measurements synchronize the Virtual Robot with the actual robot's configuration. It also processes the desired joint velocities from the Main Controller to manage servo operations accordingly.

3. **Control Systems (purple)**: This group encompasses all components required to plan and execute the desired motions of the robotic arm. It focuses on the establishment of a Nullspace Controller, which navigates the arm through its 3D workspace while utilizing its additional DOF for secondary tasks. The group also covers path planning, trajectory generation, and trajectory tracking control.

4. **User Interface (yellow)**: This group is responsible for integrating all the building blocks and making them accessible to the user. It offers simple interfaces for creating movement programs that the arm will execute on command. A graphical user interface (GUI) is being developed to facilitate easy control over the robotic arm's functionalities.

3.1 Virtual Robot Framework

To establish a digital representation of the robotic arm in MATLAB, it is essential to develop a framework for managing coordinate transforms. While there exist numerous libraries that facilitate the handling of relationships between dependent coordinate frames, a decision was made to develop a custom framework. This approach ensures a code base that is both independent and easy to comprehend, tailored specifically for the prototype. It also allows for deep insights into the inner workings of the kinematic

model, increasing this projects educational value. The ROS transform library served as a conceptual model for this implementation [Foo13].

3.1.1 The `Frame` class

The fundamental concept of transform libraries in robotics involves representing important robot points (e.g. the joints) as coordinate frames. These frames can then be visualized in a 3D plot to illustrate the pose of the robotic arm. For this prototype, each joint is assigned a frame. Additionally, an 'origin' frame at the robot's mounting point and an 'End Effector' frame at the tool center point (TCP) of the robot are defined. Figure 3.2 depicts the arrangement of coordinate frames along the Virtual Robotic arm.

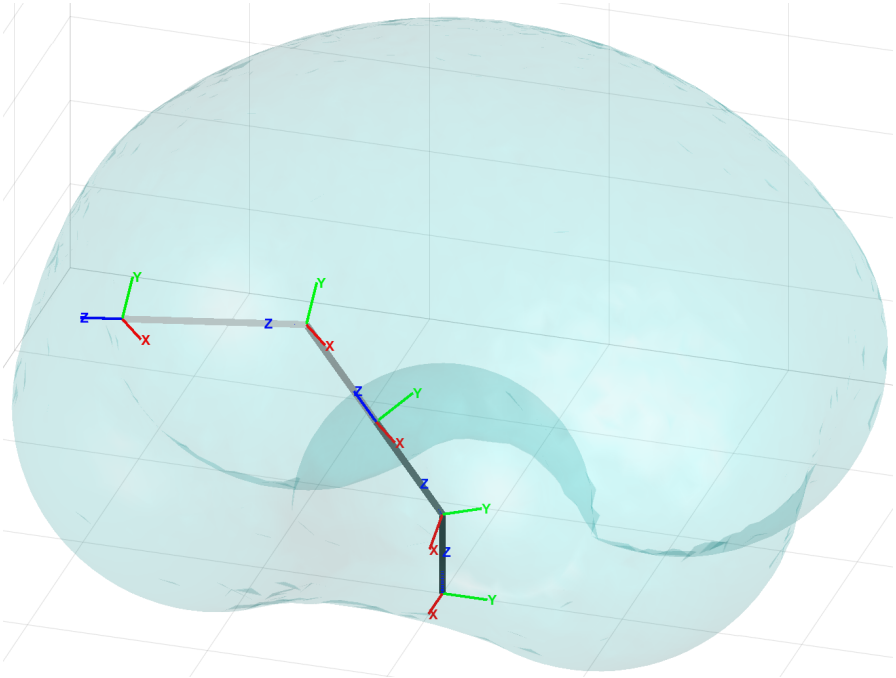


Figure 3.2: The Virtual Robot. Each joint is represented by a coordinate frame. Each link is plotted in grey. The cyan bubble around the robot is it's workspace.

These frames are interconnected through a parent-child relationship, typically referred to as the kinematic chain. The chain begins with the stationary origin frame and progresses through a sequence of kinematically linked joints. Each joint's frame can rotate around a specific axis within its local coordinate system, thereby rotating the descendant frames. This setup facilitates a straightforward implementation of a `Frame` class, as represented in the UML Diagram shown in Figure 3.1.

A frame f, must at least store the following properties:

- A reference to its parent frame p.
- A position vector from p to f, expressed in the coordinate system of p: ${}_p\vec{\mathbf{r}}_{p,f}$. [2]
- A rotation matrix transforming from f to p: ${}_p\mathbf{A}_f$.

A frame can rotate about its local axes by updating its stored rotation matrix:

$$ {}_p\mathbf{A}_f = \mathbf{R}_{x,y,z}(\phi) \tag{3.1} $$

Here, $\mathbf{R}_{x,y,z}$ denotes a basic 3D rotation matrix while ϕ denotes the rotation angle.

The position of frame f in global coordinates is ${}_g\vec{\mathbf{r}}_{g,f}$, computed by iterating backwards through the kinematic chain:

```
{}_g r_{g,f} ← {}_p r_{p,f}
while f has a parent p do
    {}_g r_{g,f} += {}_p̃A_p · {}_g r_{g,f} + {}_p̃ r_p        // p̃ is the parent of p
    f ← p
end
```

Algorithm 1: Compute Global Position

The rotation matrix transforming from the frame f to the global frame g is ${}_g\mathbf{A}_f$, computed in the same way:

```
{}_gA_f ← {}_pA_f
while f has a parent p do
    {}_gA_f ← {}_p̃A_p · {}_gA_f                              // p̃ is the parent of p
    f ← p
end
```

Algorithm 2: Compute Global Rotation

This concise implementation encompasses all essential components for a kinematic model. To tailor the kinematics to the robotic arm prototype, the `VirtualRobot` class is introduced.

[2] The vector notation in this thesis adheres to the conventions established in [Rix23].

3.1.2 The `VirtualRobot` Class

The `VirtualRobot` class facilitates the initialization of a kinematic chain that mirrors the kinematics of the actual robotic arm. This configuration necessitates the specification of each joint's relative positions and their axes of rotation, which are derived from the CAD model discussed in chapter 2.

The `VirtualRobot` class provides the following interface methods:

- `setJointAngles(`$\mathbf{q}$`)`: This method accepts a vector of desired joint angles $\mathbf{q}$ and updates the rotation of the underlying frames accordingly to reflect the new configuration. Per definition, $\mathbf{q}$ is a column vector of four doubles. The vector follows the definition of section 2.2, e.g. $\mathbf{q}(1)$ refers to the first joint, specifically the rotation of the shoulder joint around the global y-Axis in radians.

- `forwardKinematics()`: This method calculates the global position of the End Effector frame ${}_g\vec{\mathbf{r}}_{EE} = \mathbf{f}()$, based on the current configuration using algorithm 1.

- `getJacobian()`: This method numerically computes the translational Jacobian: ${}_g\mathbf{J}_{EE}$, which maps the joint velocities to the velocity of the End Effector in global coordinates. This matrix is crucial for implementing task space control. The Jacobian ${}_g\mathbf{J}_{EE}$ is defined as the derivative of the End Effector's global position vector with respect to the joint angles, formally expressed as:

$$ {}_g\mathbf{J}_{EE} = \frac{d\,{}_g\vec{\mathbf{r}}_{EE}}{d\mathbf{q}} = \frac{d\mathbf{f}()}{d\mathbf{q}} \tag{3.2} $$

  Instead of analytically differentiating the forward kinematics $\mathbf{f}()$, the method uses the finite difference approach to approximate the derivative by perturbing each joint angle by a small increment, Δq, and observing the change in the End Effector's position. Specifically, for each joint i, the method computes:

$$ {}_g\vec{\mathbf{r}}_{EE}{}^{+} = \mathbf{f}(q_i + \Delta q) \quad \text{and} \quad {}_g\vec{\mathbf{r}}_{EE}{}^{-} = \mathbf{f}(q_i - \Delta q) \tag{3.3} $$

  The partial derivative with respect to joint angle i is then approximated using the central difference formula:

$$ {}_g\mathbf{J}_{EE}(:, i) = \frac{{}_g\vec{\mathbf{r}}_{EE}{}^{+} - {}_g\vec{\mathbf{r}}_{EE}{}^{-}}{2\Delta q} \tag{3.4} $$

  As the robotic arm comprises four joints, the forward kinematics have to be calculated eight times to approximate the Jacobian. Despite this, measuring the computation time shows that the numeric approach is approximately 40 times faster than the numeric substitution necessary for the analytic solution.

- `checkSingularity()`: This method evaluates the condition number of the Jacobian, defined as:

$$\text{cond}(\mathbf{J}) = \|\mathbf{J}\| \cdot \|\mathbf{J}^{-1}\| \tag{3.5}$$

where $\|\mathbf{J}\|$ is the norm of the Jacobian (defined as its largest singular value), and $\|\mathbf{J}^{-1}\|$ is the norm of its inverse. A high condition number indicates the robotic arm is near a singularity configuration, risking rank loss in the Jacobian and unstable inverse kinematics. A typical singularity occurs when the arm is fully extended vertically at $\mathbf{q} = \mathbf{0}$, limiting further vertical movement. If `getJacobian()` returns a Jacobian with a condition number above a predefined threshold, `checkSingularity()` returns true, advising the Main Controller to modify or stop motion for stability (Obj. 4 - Operational Safety).

3.1.3 The `Workspace` class

The `Workspace` class is responsible for calculating and visualizing the reachable space, or workspace of the robotic arm . This workspace is represented as a cyan bubble around the Virtual Robot (depicted in Figure 3.2) and illustrates the maximum range of motion attainable by the End Effector across all possible joint configurations.

The class utilizes a reference to the `VirtualRobot`, which stores the joint angle limits $\mathbf{q}_{\text{lim}}$. These limits are used for defining the range within which each joint can safely operate, and are considered when calculating the workspace.

The workspace is calculated by varying the joint angles within their limits and recording the resultant positions of the End Effector. The positions are processed to generate a unique set of points forming the workspace boundary, which is then triangulated to create a surface mesh representing the non-convex hull of these points.

Mathematically, the workspace $\mathbf{W}$ is defined by:

$$\mathbf{W} = \bigcup_{\mathbf{q} \in Q} \{\mathbf{f}(\mathbf{q})\}, \tag{3.6}$$

where Q denotes the set of all possible joint configurations and $\mathbf{f}(\mathbf{q})$ calculates the End Effector's position utilizing the `forwardKinematics()` method.

3.1.4 Kinematic Simulation

Utilizing the kinematic framework provided by the `VirtualRobot` class, a kinematic simulation can be constructed to model the mechanics of robotic movement. A common testing scenario for such simulations is task space position control, which involves iterative adjustments to minimize positional errors between a robot's End Effector and a desired target location in global coordinates.

The positional error $\vec{\mathbf{e}}$ is defined mathematically as:

$$\vec{\mathbf{e}} = \vec{\mathbf{x}}_{\text{desired}} - \mathbf{f}(\mathbf{q}), \tag{3.7}$$

where $\vec{\mathbf{x}}_{\text{desired}}$ represents the desired position, and $\mathbf{f}(\mathbf{q})$ denotes the current position obtained from the `forwardKinematics()` method. The Jacobian matrix ${}_{g}\mathbf{J}_{EE}$, essential for calculating inverse kinematics, is derived using the `getJacobian()` method. Given that the Jacobian matrix ${}_{g}\mathbf{J}_{EE}$ for a 4-DOF robotic arm belongs to $\mathbb{R}^{3\times4}$, the Moore-Penrose pseudo-inverse ${}_{g}\mathbf{J}_{EE}^{+}$ is used to compute the necessary adjustments to the joint angles:

$$\Delta\mathbf{q} = \alpha \cdot {}_{g}\mathbf{J}_{EE}^{+} \cdot \vec{\mathbf{e}}, \tag{3.8}$$

where α is a scaling factor that controls the magnitude of joint angle updates.

These adjustments are applied through the `setJointAngles(q)` method, progressively steering the robot's End Effector towards the target position. This process is visually represented in Figure 3.3.

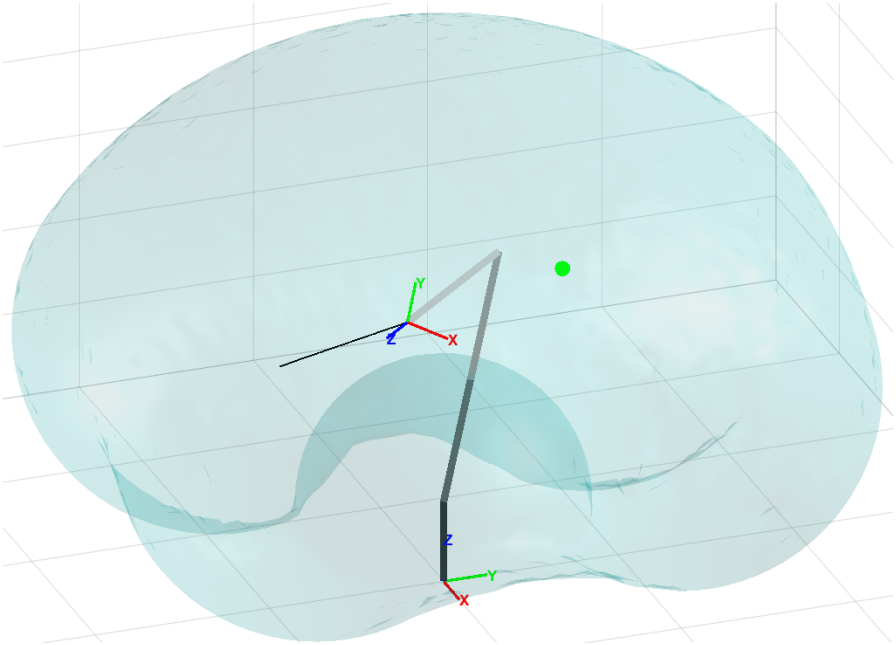


Figure 3.3: The simulated robotic arm approaches a desired position in global coordinates. The desired position is plotted in green, the past trajectory is plotted in black.

This simulation validates the kinematic model and underscores its utility for virtual prototyping of the Main Controller. To adapt this model for the actual robot, the measured joint angles are input into the `setJointAngles(q)` method, aligning the Virtual Robot's configuration with that of the Real Robot. This capability transforms the Virtual Robot object into a visual and computational proxy, extending its functionality to the real prototype.

The forthcoming section, "Real Robot Interface," will develop an interface for extracting the joint angles $\mathbf{q}$ from the robot's servos and for controlling servo velocities. Thus, providing a bridge between the Virtual Robot and the real prototype.

3.2 Real Robot Interface

The Real Robot Interface is specifically designed to facilitate interactions with the servos of the robotic arm. Its primary goal is to provide an abstract interface for directly controlling the robotic arm's joints rather than the servos themselves. This interface is organized into two hierarchical classes:

- `ServoChain`: Handles low-level communications with servos via the Dynamixel SDK. Its primary functions include executing commanded servo velocities and reporting back on servo angles. The `ServoChain` is instantiated by the `RealRobot`, which maintains a reference to it for servo control. Upon initialization, `ServoChain` connects to the dynamixel servos by loading the Dynamixel SDK into MATLAB, opening the designated serial port, identifying the connected servos' IDs, and establishing the connection. It then configures each servo's PI gains, velocity control mode, and acceleration profile. This class adheres closely to the Dynamixel SDK's standard documentation, which is not detailed further here.

- `RealRobot`: Provides high-level functionalities for direct joint control, including reading joint angles and setting joint velocities, rather than controlling individual servos. It acts as the primary interface between the servo measurement system and the robotic arm's kinematic measurement system. `RealRobot` also enforces predefined maximum joint velocities, denoted as $\dot{\mathbf{q}}_{\text{lim}}$, to avoid accidents and potential damage to the joints (Obj. 4 - Operational Safety). Both the `RealRobot` and `ServoChain` promptly call their destructor upon losing connection to the servos, ensuring a safe shutdown.

The information flow through the Real Robot Interface is illustrated in Figure 3.4. The units of measurement will be explained in the following paragraphs.

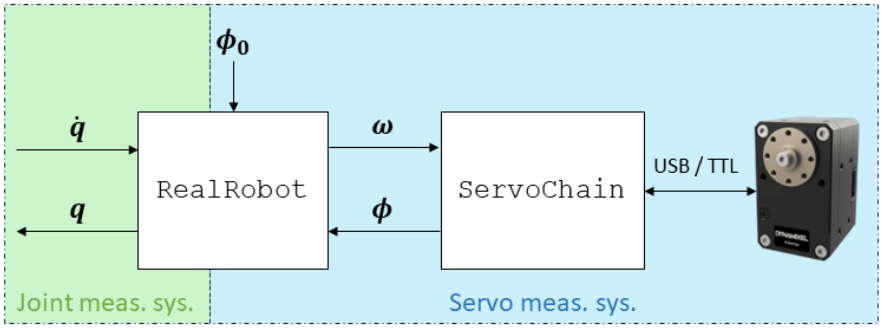


Figure 3.4: Diagram illustrating the information flow within the Real Robot Interface. The `RealRobot` class provides the current joint angles $\mathbf{q}$ and commands the desired joint velocities $\dot{\mathbf{q}}$.

3.2.1 Servo Measurements vs. Joint Measurements

The state of the robotic arm is characterized using joint angles $\mathbf{q}$ and joint velocities $\dot{\mathbf{q}}$, whereas the servo measurements are expressed in terms of servo angles $\boldsymbol{\phi}$ and servo velocities $\boldsymbol{\omega}$. These two measurement systems differ due to the transmission between the servos and the joints, as well as the undefined zero angle of the servos, which is discussed in the following paragraph.

Zero Angle Definition

An additional challenge in accurately translating servo angles into joint angles is that servo encoders do not provide a truly absolute position. Due to mechanical requirements, servos may need to rotate beyond a full turn, necessitating an internal mechanism to track the number of rotations to maintain an absolute position. However, these encoders reset their rotation count after a power loss or system restart, losing all previous positional data. To compensate for this and ensure accurate joint angle calculations, it is necessary to establish a predefined zero position. This prototype proposes the following solution:

- Before each startup, the robotic arm is manually positioned into a fully vertical stance. This position is then defined as the zero configuration ($\mathbf{q} = \mathbf{0}$), ensuring consistency in initial conditions across restarts.

- Angle offsets, referred to as $\phi_{i,0}$, are recorded within the `RealRobot` immediately after this manual calibration. These offsets are used to calculate the relative servo angles from this zero baseline:

$$\Delta\phi_i = \phi_i - \phi_{i,0} \quad \text{for } i = 1, 2, 3, 4 \tag{3.9}$$

This approach enables the system to simulate an absolute position framework by recalibrating to a known starting point, thus ensuring that subsequent measurements and controls reference a known baseline. However, a significant drawback is that the arm must be manually positioned to $\mathbf{q} = \mathbf{0}$ before startup, which, as later tests show, is prone to inaccuracies. The semester thesis on the first prototype [Zei23] defines the coordinate system of the robotic arm, including the zero configuration.

Derivation of Conversion Formulas

Next, it is necessary to establish a mathematical relationship between the servo angles and the joint angles. Given the described limitations of absolute angle measurements, the conversion formulas developed rely solely on the relative angles $\Delta\phi_i$. This method avoids the inaccuracies that could arise from encoder resets and ensures that the derived joint angles and velocities are consistently based on the calibrated zero position established at startup.

Using the defined relative servo angles, joint angles $\mathbf{q}$ are calculated, considering the transmission ratios and the differential gear setup of the shoulder joint.

For the Yaw and Elbow joints (Joint 3 and Joint 4):

$$q_3 = \Delta\phi_3 \tag{3.10}$$

$$q_4 = \frac{\Delta\phi_4}{i_{\text{elbow}}} \tag{3.11}$$

For the Shoulder Joints (Joint 1 and Joint 2), influenced by a differential gear arrangement:

$$q_1 = \frac{\Delta\phi_1 - \Delta\phi_2}{i_{\text{shoulder}}} \tag{3.12}$$

$$q_2 = -\frac{\Delta\phi_1 + \Delta\phi_2}{i_{\text{shoulder}}} \tag{3.13}$$

By differentiating the angle conversion formulas with respect to time, formulas are obtained to convert desired joint velocities $\dot{\mathbf{q}}$ to servo velocities $\boldsymbol{\omega}$:

For the Yaw and Elbow joints:

$$\omega_3 = \dot{\phi}_3 \tag{3.14}$$

$$\omega_4 = \dot{\phi}_4 \cdot i_{\text{elbow}} \tag{3.15}$$

For the Shoulder joints:

$$\omega_1 = \frac{1}{2}(\dot{q}_2 - \dot{q}_1) \cdot i_{\text{shoulder}} \tag{3.16}$$

$$\omega_2 = \frac{1}{2}(\dot{q}_2 + \dot{q}_1) \cdot i_{\text{shoulder}} \tag{3.17}$$

In summary, the `RealRobot` class provides following functions for translating between Servo Measurements and Joint Measurments, based on the defined zero position:

$$\mathbf{q} = \mathbf{f_1}(\boldsymbol{\Delta\phi}) \quad \textit{Convert measured servo angles to joint angles} \tag{3.18}$$

$$\boldsymbol{\omega} = \mathbf{f_2}(\dot{\mathbf{q}}) \quad \textit{Convert desired joint velocities to servo velocities} \tag{3.19}$$

3.2.2 Coupling with the Virtual Robot

The `RealRobot` class seamlessly couples with the `VirtualRobot` class using two primary methods to synchronize their states:

- Retrieving the current joint angles from the Real Robot:

$$\mathbf{q} = \texttt{realRobot.getJointAngles()}$$

- Updating the Virtual Robot's joint angles to match:

$$\texttt{virtualRobot.setJointAngles(}\mathbf{q}\texttt{)}$$

Continuously executing these methods allows the Virtual Robot to accurately reflect the kinematic configuration of the Real Robot. This synchronization not only provides a live visualization of the robot's movements but also extends the application of kinematic computations to the Real Robot. With these capabilities in place, the foundation is set to develop a robust control strategy for the robotic arm, which is detailed in the following section.

3.3 Control Systems

This chapter explores all functionalities necessary for managing the movements of the robotic arm. It specifically focuses on the development of the Main Controller, tasked with navigating the robotic arm within the global coordinate system, referred to as the task space. The control systems also encompass path planning and trajectory generation (Obj. 6 - Extended Feature Set), for conciseness these features will not be covered in this thesis.

3.3.1 Control Structure

This thesis proposes a **cascaded**, **decoupled** control structure for the robotic arm:

- **Cascaded:** This approach refers to a hierarchical setup where the outputs of a higher-level Main Controller serve as the input for subordinate controllers. Specifically for this prototype, the Main Controller calculates the necessary joint velocities $\dot{\mathbf{q}}$ to achieve a desired motion of the robotic arm. The internal velocity controllers of the servos then work to meet these velocity targets. The cascaded approach is chosen for this prototype as it effectively utilizes the capabilities of the servo's internal controllers.
- **Decoupled:** In this context, "decoupled" means that the Main Controller operates independently of the lower-level servo controllers. It does not modify its outputs based on the current behavior of the servo controllers. Instead, the Main Controller assumes that the servos will accurately and instantly achieve the commanded velocities. This assumption allows the main control loop to operate without complex feedback mechanisms from the servo controllers, simplifying the control architecture. It is applicable, as the servo control loop operates at a significantly higher frequency (approximately 1 kHz) compared to the Main Controller (approximately 50 Hz).

3.3.2 Task Space Control

Most movement sequences for the robotic arm are specified in the task space, with a fundamental task being the navigation of the End Effector to a desired position specified in global coordinates. This section will derive the equations necessary for effectively controlling the motion of the robotic arm through three-dimensional space, utilizing all four DOF. The position $\vec{\mathbf{x}}_{\text{desired}}$ and velocity $\vec{\mathbf{v}}_{\text{desired}}$ of the End Effector are generally defined within the global coordinate system; therefore, the subscript g is omitted in subsequent descriptions. Due to the inverse kinematics approach used, task space control is only feasible when the robot is not near a singularity configuration.

PI-Controller

Analogous to the kinematic simulation discussed in section 3.1.4, the positional error $\vec{\mathbf{e}}$ is defined mathematically as:

$$\vec{\mathbf{e}} = \vec{\mathbf{x}}_{\text{desired}} - \mathbf{f}(\mathbf{q}) \tag{3.20}$$

where $\vec{\mathbf{x}}_{\text{desired}}$ represents the desired position, and $\mathbf{f}(\mathbf{q})$ denotes the current position obtained from forward kinematics.

The positional error $\vec{\mathbf{e}}$ is utilized in a PI controller to compute the initial task space velocity $\vec{\mathbf{v}}_{\text{pid}}$:

$$\vec{\mathbf{v}}_{\text{pid}} = \mathbf{K}_p \cdot \vec{\mathbf{e}} + \mathbf{K}_i \cdot \int \vec{\mathbf{e}}\, dt \tag{3.21}$$

where $\mathbf{K}_p$ and $\mathbf{K}_i$ are the proportional and integral gains, respectively. This controller effectively calculates the End Effector's velocity that reduces the positional error. To mitigate the risk of integral windup, which can occur if the integral term accumulates excessive error during saturation periods, an anti-windup mechanism constrains the integral error, maintaining the controller's stability.

Trajectory tracking control

To facilitate trajectory following as well as static position targeting, the control system employs a drift compensation strategy [Rix23]. This involves adding an additive desired velocity term $\vec{\mathbf{v}}_{\text{desired}}$ to the PI controller output, enhancing trajectory adherence. The resulting effective task space velocity $\vec{\mathbf{v}}_{\text{eff}}$, is defined as:

$$\vec{\mathbf{v}}_{\text{eff}} = \vec{\mathbf{v}}_{\text{pid}} + \vec{\mathbf{v}}_{\text{desired}} \tag{3.22}$$

where $\vec{\mathbf{v}}_{\text{desired}}$ aligns the arm's movement with the dynamic trajectory. Without this component, the robot's End Effector would consistently lag behind the intended trajectory due to inherent tracking error. Implementing a 'drift compensation' strategy significantly reduces these tracking errors, enhancing trajectory adherence.

Nullspace Utilization

The next logical step is to formulate the inverse kinematics analogous to section 3.1.4. This approach utilizes the Moore-Penrose pseudoinverse, ${}_g\mathbf{J}_{EE}^{+}$, to calculate the joint velocities $\dot{\mathbf{q}}$ given the desired task space velocity $\vec{\mathbf{v}}_{\text{eff}}$.

Given that the desired task space velocity is a 3×1 vector and the robotic arm has four DOF, the arm is considered 'overactuated'. This results in the existence of a Nullspace, representing a subset of joint velocities that do not influence the velocity of the End Effector. This characteristic allows the robotic arm to perform secondary tasks without affecting the primary task of controlling the End Effector's movement. Secondary tasks can be implemented by explicitly utilizing the Nullspace.

The formula for calculating the desired joint velocities is extended to include an additional component that leverages the Nullspace [Rix23]:

$$\dot{\mathbf{q}} = {}_{g}\mathbf{J}_{EE}^{+} \cdot \vec{\mathbf{v}}_{\text{eff}} + \alpha \cdot \mathbf{N} \cdot (-\frac{\partial H}{\partial \mathbf{q}}) \tag{3.23}$$

This additional term includes the following components:

- $\mathbf{N}$, the Nullspace projection matrix, is defined as $\mathbf{N} = \mathbf{I} - {}_{g}\mathbf{J}_{EE}^{+} \cdot {}_{g}\mathbf{J}_{EE}$, where $\mathbf{I}$ is a 4×4 identity matrix. $\mathbf{N}$ itself is of dimension 4×4 but has a rank of 1, ensuring it projects joint space velocities into a one-dimensional subspace that does not affect the task space velocity.
- $(-\frac{\partial H}{\partial \mathbf{q}})$ is the negative gradient of the cost function $H(\mathbf{q})$, enabling the definition of 'cost' or 'undesirability' for joint configurations. This gradient directs adjustments in joint velocities towards the steepest decrease in $H(\mathbf{q})$.
- α acts as a scaling factor, modulating the influence of the Nullspace component and serving effectively as a proportional gain for the Nullspace task.

This strategy enables the robotic arm to follow a desired End Effector velocity, while optimizing it's joint configuration according to $H(\mathbf{q})$.

The Cost Function $H(\mathbf{q})$

The cost function $H(\mathbf{q})$ is employed to define secondary tasks for the robotic arm, making effective use of the Nullspace to enhance the system's overall functionality. Common applications of this cost function in robotics include:

- Minimizing the energy consumption of the system, thereby improving efficiency and reducing operational costs.
- Avoiding self-collisions, which enhances safety and prevents damage to the robotic system.
- Maintaining a specific End Effector orientation, crucial for tasks requiring precise manipulation and orientation.
- Keeping the robotic arm near a comfortable 'neutral' position, which helps in reducing wear and extending the lifespan of the system.

This thesis proposes leveraging the Nullspace to maintain the robotic arm near the zero configuration $\mathbf{q} = 0$, where the arm is vertically extended with all joints situated midway through their range of motion. This strategic positioning minimizes the risk of encountering joint limits during task execution. Joint limits, if reached, can severely

restrict the arm's capability to achieve certain positions within its workspace, potentially trapping it in suboptimal configurations.

To actively encourage an optimal joint configuration, the cost function $H(\mathbf{q})$ is specifically formulated to penalize deviations from $\mathbf{q} = 0$. The function is defined as:

$$H(\mathbf{q}) = \frac{1}{2} \left(\frac{\mathbf{q}}{\mathbf{q}_{\text{lim}}} \right)^T \left(\frac{\mathbf{q}}{\mathbf{q}_{\text{lim}}} \right) \tag{3.24}$$

This quadratic formulation increases the cost as joint angles deviate from zero, effectively discouraging excessive movement toward the arm's joint limits. The normalization by $\mathbf{q}_{\text{lim}}$, the vector of joint limits, ensures that each joint's contribution to the cost is scaled appropriately, reflecting its allowable range of motion. This adaptation makes the cost function sensitive to the physical constraints of each joint, promoting an equitable penalization across all joints regardless of their individual motion capabilities.

The gradient of this normalized cost function, which indicates the direction and rate of the fastest increase in cost, is given by:

$$\frac{\partial H}{\partial \mathbf{q}} = \frac{\mathbf{q}}{\mathbf{q}_{\text{lim}}^2} \tag{3.25}$$

This gradient is used to adjust the joint velocities, guiding the arm towards the desired position while attempting to avoid joint limits. Relying on this method, it is possible to avoid joint limits as long as the primary task permits it. However, if the primary task necessitates exceeding the limits, the optimization becomes ineffective.

3.3.3 Enforcing Joint Limits

To protect the robotic arm from movements that exceed safe operational joint limits, the control system must enforce these limits (Req. 4 - Operational Safety). The task space control equations output a vector of desired joint velocities, $\dot{\mathbf{q}}$. Before these velocities are commanded to the servo, post-processing is necessary to ensure compliance with the specified angle and velocity limits, detailed in Table 3.1.

Table 3.1: Angle and velocity limits for each joint of the robotic arm.

Description	Joint 1	Joint 2	Joint 3	Joint 4	Unit
$\mathbf{q}_{\text{lim}}[i]$	45	45	315	150	deg
$\dot{\mathbf{q}}_{\text{lim}}[i]$	5.7	5.7	19.1	19.1	rpm

Joint Angle Limits

The angle limits of the joints are stored in the `VirtualRobot`, which the Main Controller references. These limits are enforced by the Main Controller using the following algorithm:

for *each joint angle* q_i *in* $\mathbf{q}$ **do**
  if $q_i \leq -\mathbf{q}_{lim}[i]$ ***and*** $\dot{q}_i < 0$ **then**
    $\dot{q}_i \leftarrow 0$
  else if $q_i \geq \mathbf{q}_{lim}[i]$ ***and*** $\dot{q}_i > 0$ **then**
    $\dot{q}_i \leftarrow 0$
  end
end

Algorithm 3: Enforce Joint Angle Limits

This algorithm checks if any joint has reached its limit and is continuing to move in that direction. If so, that joint's velocity is set to zero.

Joint Velocity Limits

The velocity limits of the joints are stored in the instance of the `RealRobot` class and are directly enforced there using the following algorithm:

for *each joint velocity* $\dot{q}_i$ *in* $\dot{\mathbf{q}}$ **do**
  if $|\dot{q}_i| > \dot{\mathbf{q}}_{lim}[i]$ **then**
    $\dot{q}_i \leftarrow \dot{\mathbf{q}}_{\text{lim}}[i] \cdot \text{sgn}(\dot{q}_i)$
  end
end

Algorithm 4: Enforce Joint Velocity Limits

This algorithm simply caps the magnitude of the joint velocities at their respective limits, ensuring that the velocities remain within safe operational bounds.

3.3.4 Integration of the Main Controller

Following the detailed exploration of the internal equations of the Main Controller, this section provides a broader system-level view, as depicted in the UML diagram shown in Figure 3.1. The integration of the Main Controller with the `VirtualRobot` and `RealRobot` classes is crucial for the effective operation of the robotic system. Figure 3.5 illustrates the information flow between these modules. To better understand how

these core modules cooperate effectively, their application is showcased through an exemplary scenario.

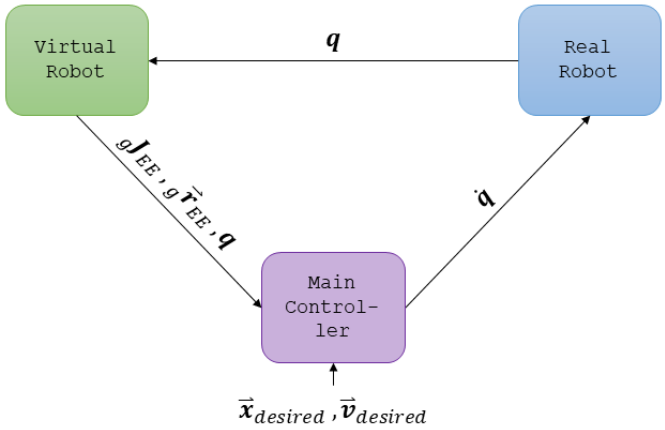


Figure 3.5: Information flow between the VirtualRobot, RealRobot, and Main Controller.

Exemplary Scenario: Trajectory Following

The robotic arm is programmed to follow a predefined trajectory with its End Effector. This trajectory is defined using a set of discretized points in global coordinates $\vec{\mathbf{x}}_{\text{desired}}$, accompanied by the desired End Effector velocity $\vec{\mathbf{v}}_{\text{desired}}$.

The following steps are taken to accomplish this task:

1. The robotic arm prototype begins in its zero configuration ($\mathbf{q} = \mathbf{0}$), fully extended vertically.
2. The `RealRobot` and `VirtualRobot` are instantiated. The `RealRobot` connects to the servos and records their zero position.
3. The `RealRobot` continuously measures the current joint configuration $\mathbf{q}$, updating the `VirtualRobot` with these measurements.
4. The `VirtualRobot`, mirroring the configuration of the `RealRobot`, provides a visualization on the screen. It calculates and relays the current End Effector position ${}_g\vec{\mathbf{r}}_{EE}$, joint configuration $\mathbf{q}$, and Jacobian ${}_g\mathbf{J}_{EE}$ to the Main Controller.
5. The Main Controller computes the desired joint velocities $\dot{\mathbf{q}}$ based on $\vec{\mathbf{x}}_{\text{desired}}$ and $\vec{\mathbf{v}}_{\text{desired}}$.

6. The joint velocities $\dot{\mathbf{q}}$ are checked for compliance with the joint limits and then translated into servo velocities by the `RealRobot`.
7. The `RealRobot` commands the Dynamixel servos to execute these velocities using the Dynamixel SDK.
8. This process repeats from step 3 for the duration of the trajectory.

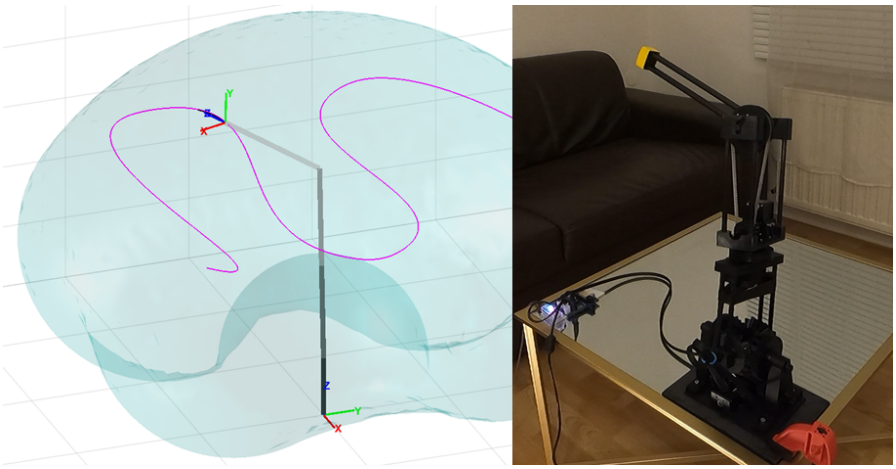


Figure 3.6: The robotic arm following a trajectory with its End Effector. Left: Visualization, Right: Photograph.

This results in the arm successfully following the trajectory, as shown in Figure 3.6. This scenario illustrates how the Main Controller interfaces with `RealRobot` and `VirtualRobot` to execute motion tasks, demonstrating the system's ability to translate theoretical control strategies into the real world.

3.4 User Interface

As demonstrated in the previous scenario, the software suite contains all the necessary components to perform a variety of motion tasks with the robotic arm. However, defining these tasks within a standalone script requires the initial setup and synchronization of all software components, which introduces significant overhead and complexity. This thesis introduces the `Launcher` class as an effective solution to streamline this process. Positioned atop all the software modules developed thus far, this class provides a straightforward API that enables users to focus on crafting motion tasks without being encumbered by the underlying software intricacies (Obj. 5 - Simplified Interaction).

Figure 3.7 provides a zoomed-in view of the UML Diagram to better illustrate the integration of the `Launcher` class into the software suite.

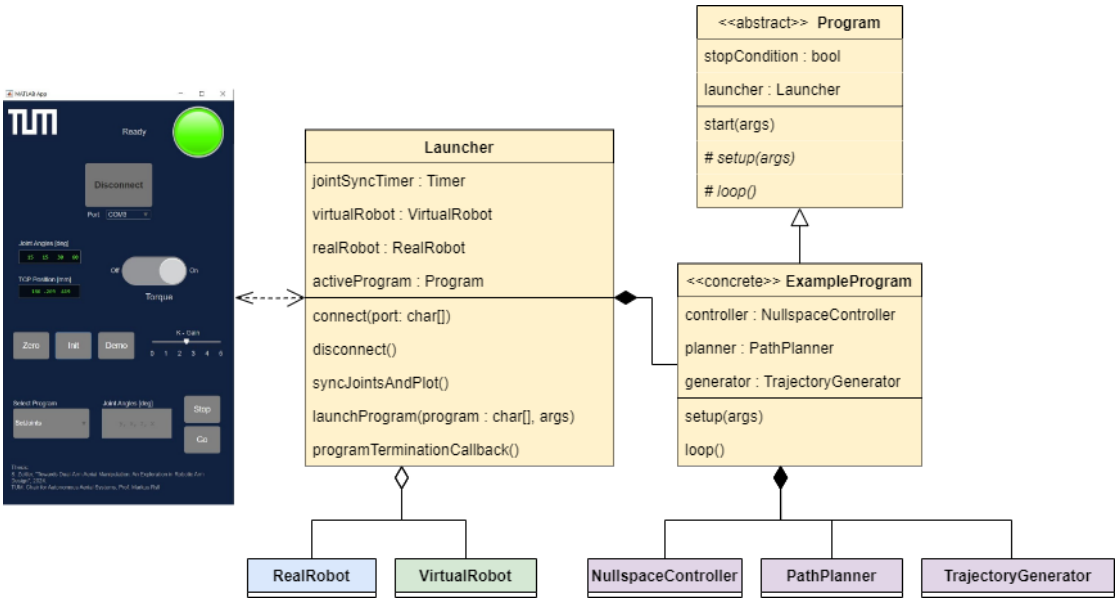


Figure 3.7: Zoomed-in view of the UML Diagram highlighting the `Launcher` class

3.4.1 The `Launcher` class

The `Launcher` class is designed to streamline interaction with the robotic arm by providing an abstract software interface. This thesis proposes the following core concepts to achieve this:

- **Centralized Management:** The central `Launcher` takes care of all the required overhead. This includes synchronizing the underlying `VirtualRobot` and `RealRobot` and monitoring the connection to the servos. The launcher provides a simple interface for executing motion programs while handling all peripheral functionality.
- **Launchable Programs:** Motion tasks are separately defined in concise program files that adhere to a common structure. This modularity enables users to concentrate exclusively on programming movement instructions for the robotic arm, such as navigating to specific End Effector positions or adjusting joint configurations.

Launchable Programs

The implementation of launchable programs empowers users to define specific motion tasks for the robotic arm. This thesis introduces the abstract `Program` class, which serves as a template for all custom programs. The structure and behavior of programs adhering to this template are defined as follows:

- **Method Structure:** programs must implement a `setup(args)` method, called once at the program's launch, and a `loop()` method, called repeatedly until the program terminates. This design mirrors the familiar programming model of the ArduinoIDE, which uses a similar setup for sketch programs [24a].
- **Termination Logic:** Programs must define a boolean `stopCondition`. This condition is evaluated after each iteration of the `loop()` method, terminating the program when the specified condition is fulfilled.
- **Object Access:** Programs have access to the `VirtualRobot` and `RealRobot`. This enables interaction with the broader robotic system. Depending on their specific motion task, programs may also utilize the Main Controller, path planner, or trajectory generator.

Several example programs have been developed to demonstrate the capabilities of this framework:

- `SetJoints.m`: Employs a PI Controller to reach a specified joint configuration.
- `SetPosition.m`: Utilizes the Main Controller to maneuver the end-effector to a desired position.
- `Trajectory2D.m`: Allows users to define a trajectory for the robotic arm by drawing a path in a 2D slice of the workspace.

3.4.2 MATLAB App

To further enhance usability, this thesis also presents a MATLAB App designed to abstract the complexities of direct code interaction. This user interface offers a visual method for controlling the `Launcher` , useful for demonstrating the robot's capabilities and simplifying testing processes. This interface is engineered to minimize the need for in-depth knowledge of the extensive underlying software suite, thereby making it accessible even to users without programming expertise.

GUI Design

The GUI is organized into two main sections: the control interface on the right side of the screen, and the live visualization of the `VirtualRobot` on the left. Figure 3.8 illustrates the interface.

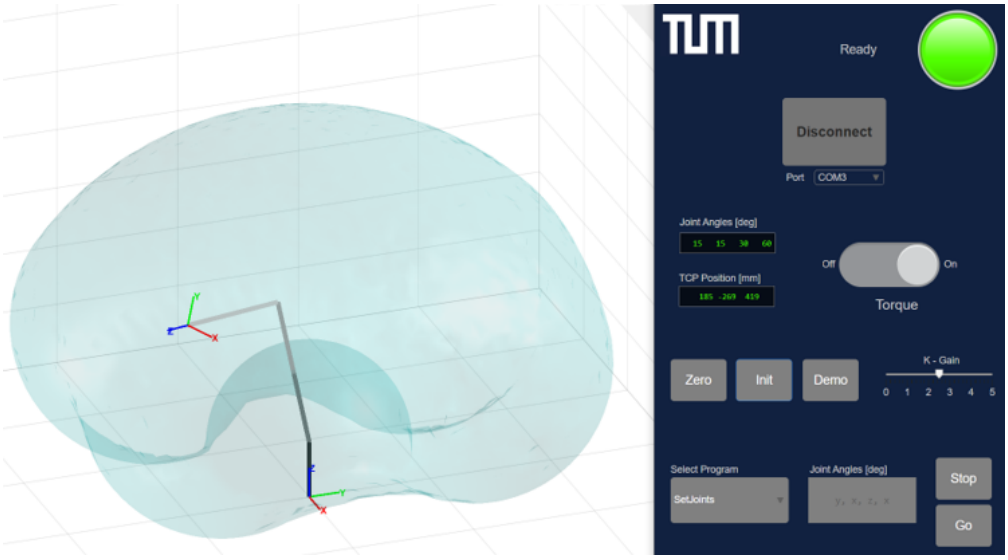


Figure 3.8: Screenshot of the interface for controlling the robotic arm prototype. **Left:** Visualization of the Virtual Robot, **Right:** Control Interface.

Key Features

The MATLAB app, which controls the `Launcher`, integrates various interactive elements such as buttons, dropdown menus, and sliders for interacting with the robotic arm:

- **Status Indicators:** A status lamp within the GUI displays the connection status, signaling whether the robotic arm is executing a motion task, caught in a singularity, or ready to initiate a new program.
- **Program Selection:** Programs can be selected from a dropdown menu or directly initiated using shortcut buttons. For example, the 'Zero' button activates the `SetJoints.m` program, positioning the arm in the zero configuration ($\mathbf{q} = \mathbf{0}$). The 'Stop' button immediately halts the program, freezing the arm in its current position.

- **Torque Control:** This slider enables users to quickly enable or disable servo torque, crucial for managing safety-critical behaviors of the robotic arm (Obj. 4 - Operational Safety).
- **Connection Management:** A prominent central button simplifies the connection process by toggling the robotic arm's connection status. Pressing the disconnect button or closing the GUI triggers the arm to automatically maneuver to the zero configuration, facilitating easy shutdown procedures.

This interface concludes the discussion on software design, covering all essential software modules depicted in the UML diagram in Figure 3.1. The established software suite will be utilized in the testing scenarios presented in the final chapter on evaluating the prototype.

4 Evaluation

This chapter concludes the Master Thesis titled "Towards Dual-Arm Aerial Manipulation: An Exploration in Robotic Arm Design," bringing into focus the broader objectives of dual-arm aerial manipulation. A brief recapitulation of the project's journey provides a backdrop for the comprehensive evaluation that follows.

The initiative began in 2021 at the chair of autonomous aerial systems at TUM, with the vision of equipping a UAV with two robotic arms, a project entirely developed by students. Jasper Sindermann laid the groundwork by creating the first blueprint for one of these arms. This blueprint was realized into a first prototype by Samuel Zeitler, which, although operational, was deemed not suitable for application without substantial revisions. This thesis represents the third contribution to the project, presenting a second prototype iteration designed to address and rectify the major shortcomings of its predecessor.

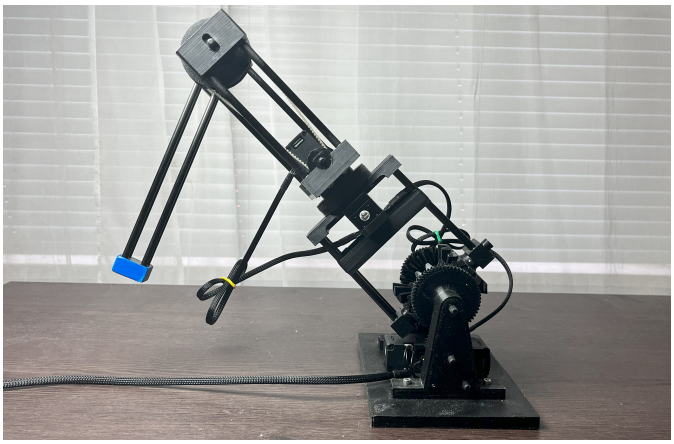


Figure 4.1: Photograph of the finished prototype.

Demonstration videos of the finished prototype are available and linked in the GitHub repository[1]. This final chapter assesses the system's suitability for aerial manipulation applications, grounded in the objectives and requirements specified in sections 1.4.2 and 2.1.

[1] `https://github.com/Friedsam2000/Robotic-Arm-Prototype.git`

Evaluation was carried out through two primary testing scenarios. The first tested the accuracy of the arm's movements, while the second examined the arm's capacity to exert and endure forces. These tests provided a quantitative assessment of the prototype's performance and demonstrated the functionality of the developed hardware and software design. The chapter concludes with a synthesis of the findings, summarizing the progress and key insights gained throughout this thesis. It will also provide an outlook for the future of the project and suggestions for possible improvements to the robotic arm prototype.

4.1 Motion Accuracy Testing

A common task in aerial manipulation, such as applying a protective coating to a wind turbine [Koc+22], requires the robotic arm to accurately follow a designated trajectory with its End Effector. Therefore, the first test assessed the prototype's ability to adhere to a specified trajectory, using precise measurements of the End Effector's position.

4.1.1 Test Setup

This test was conducted in the Flight Lab at the Chair of Autonomous Aerial Systems, taking advantage of the available motion capturing system. Infrared markers were attached to the prototype to localize the End Effector within a fixed coordinate system, as shown in Figure 4.2.

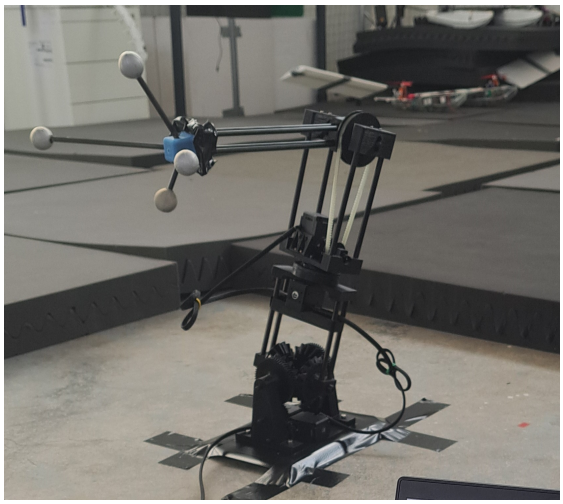


Figure 4.2: The robotic arm equipped with infrared markers on its End Effector.

The robotic arm was tasked with following a desired trajectory using the trajectory tracking control developed and detailed in sections 3.3.2 and 3.3.4. Specifically, it was programmed to follow a predefined circular trajectory with a diameter of 30 cm within 10 seconds. The recording of the arm's End Effector position is superimposed with the desired trajectory and the calculated trajectory, which is the prototype's estimate of the End Effector position based on the measured servo angles.

4.1.2 Test Data Analysis

The resulting trajectories are depicted in Figure 4.3. The plot shows three colored trajectories of the robotic arm's End Effector: the desired trajectory (magenta), the trajectory calculated by the prototype (blue), and the recorded trajectory (orange). In an ideal setup, the three trajectories would align perfectly, indicating perfect trajectory adherence. However, given that the robotic arm is a prototype acting in the real world, discrepancies between these paths are observed. The following paragraphs will analyze the causes of these deviations and the conclusions that can be drawn from them.

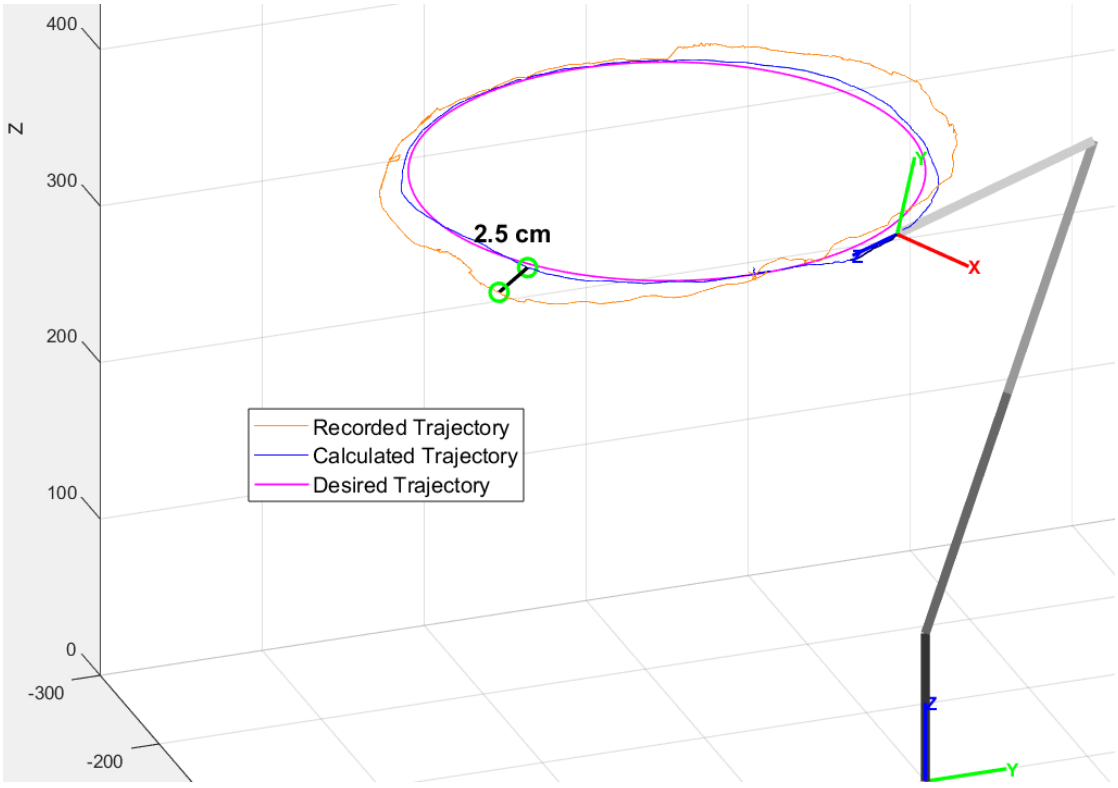


Figure 4.3: The resulting End Effector trajectories.

Deviation Between the Calculated and Desired Trajectory

The minor deviation between the calculated and desired trajectories represents the task space control error, or tracking error. During trajectory tracking control, this error is utilized to compute the desired servo velocities aimed at minimizing the deviation. In simulated environments, this error typically decreases exponentially, ultimately leading to perfect alignment. However, in the actual prototype, transient effects adversely impact performance. Specifically, there is an approximate 20 ms delay between measuring current servo angles and commanding the desired servo velocities, resulting in a control loop frequency of only 50 Hz. This delay, along with the integral relationship between the End Effector position and the servo velocities, is likely responsible for the observed deviation. Supporting this interpretation, allowing the prototype more time to complete the trajectory reduces the average tracking error, suggesting that, with more time, the impact of the 20 ms delay becomes less significant.

Deviation Between the Calculated and Recorded Trajectory

The deviation between the calculated and recorded trajectories peaks at 2.5 cm and averages at around 1 cm, requiring further analysis.

At present, the prototype lacks direct methods, such as visual feedback mechanisms like cameras, to accurately measure the End Effector's position. Instead, it relies solely on the measured servo angles combined with the kinematic model to produce an estimation. This reliance on estimation without feedback control means that inaccuracies and other sources of error can accumulate. The potential sources of error are:

- **Model Inaccuracies:** Potential errors in the specifications of segment lengths and gearing ratios in the kinematic model or the underlying equations.
- **Manual Calibration:** Issues arising from the imprecise manual calibration process, as detailed in section 3.2.1.
- **Mechanical Slack:** Angular inaccuracies due to slack in the joints and gears.
- **Structural Part Elasticity:** Positional inaccuracies due to elastic deformation of structural components under load.

To better understand these sources of error, a zoomed-in version of the results is provided in Figure 4.4, focusing on the discrepancies between the calculated and recorded trajectory.

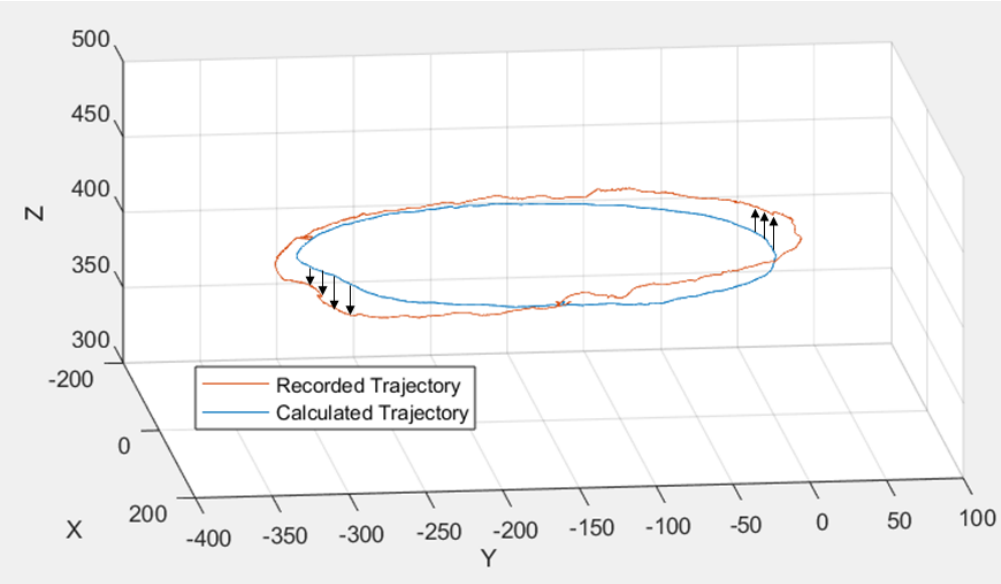


Figure 4.4: Close-up view of the calculated End Effector trajectory (blue) and the recorded trajectory (orange). The recorded trajectory appears to be slightly rotated as indicated by the black arrows.

This close-up view reveals that the recorded trajectory is rotated approximately 4° about the robotic arm's global x-axis. This strongly suggests that the arm's coordinate system itself is rotated due to inaccuracies in the manual calibration process, as discussed in section 3.2.1. If the arm is not perfectly vertical at $\mathbf{q} = \mathbf{0}$ during calibration, the arm's coordinate system will be misaligned. Since the desired trajectory is specified in the arm's coordinate system, any misalignment with the global coordinate system results in the observed skewing of the recorded trajectory during testing.

The remaining deviations between the recorded and calculated trajectories are most likely attributable to hardware imprecisions. Mechanical slack in the shoulder joint is the primary contributor to this error, as even slight angular discrepancies can result in significant displacements of the End Effector. Despite enhancements from the previous prototype, the 2-DOF shoulder joint still exhibits mechanical slack due to the numerous gears involved.

4.1.3 Test Findings

The functional testing of the robotic arm prototype using motion tracking cameras confirms its ability to follow specified trajectories, albeit with minor deviations. Over

time, the trajectory tracking control error remains minimal, indicating the effective implementation of the trajectory tracking controller (Obj. 6 - Extended Feature Set). The relatively low frequency of the Main Controller is most likely responsible for these deviations.

The measured average deviation between the calculated and recorded trajectories is approximately 1 cm, which is less than 2% of the total arm length. This relatively small and consistent positional error strongly suggests that the underlying software modules and equations are correctly implemented; otherwise, the calculated trajectory would completely diverge from the ground truth. The primary sources of this deviation are likely the manual calibration process and mechanical slack in the shoulder joint, both of which are areas identified for future improvements. The overall precision significantly improved compared to the previous prototype (Obj. 2 - Motion Accuracy).

4.2 Force Exertion Testing

The second testing scenario evaluated the robotic arm's ability to exert forces at its End Effector, essential for interacting with the environment. Simultaneously, the test assessed the prototype's resilience against the resulting mechanical stresses — a crucial quality considering that the arm is designed to be mounted on a drone.

Figure 4.5: Force Gauge measuring the static force at the prototype's End Effector.

4.2.1 Test Setup

The force gauge was mechanically linked to the End Effector of the robotic arm and held stationary by hand, as shown in Figure 4.5. The prototype was then programmed to apply force in the blocked direction using the task space controller, detailed in section 3.3.2. The servo torque was gradually increased until the first servo reached its overload state.

The resulting counterforce subjected the hardware to mechanical stresses significantly greater than those in the previous testing scenario, which involved only gravitational and inertial forces. The prototype was closely monitored for any signs of damage to the hardware during the test.

The test was repeated for distinct configurations, designed to subject one specific joint to maximum torque, simulating worst-case scenarios for force exertion. These configurations are illustrated in Figure 4.6.

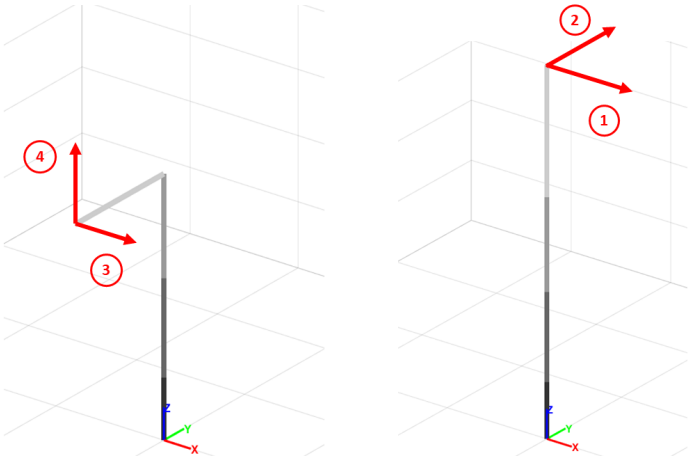


Figure 4.6: Configurations used for measuring maximum force exertion. Red arrows show the direction of force applied by the arm.

4.2.2 Test Data Analysis

The maximum End Effector forces for each configuration are detailed in Table 4.1, representing the highest force measured just before a servo reached the overload state.

Table 4.1: Maximum force measured at the End Effector.

Description	Conf. 1	Conf. 2	Conf. 3	Conf. 4	Unit
F_{max}	7.05	5.45	4.3	6.15	N

The torque at which each servo entered the overload state was calculated by factoring out the transmission ratio and the lever-arm. For configuration 3, this analysis revealed a stall torque of 0.95 Nm for the servo in the yaw joint, considerably lower than the specified 2.5 Nm at 12V for the Dynamixel XH430-W210. Details on the internal mechanism that triggers the overload state are undisclosed by Dynamixel.

4.2.3 Test Findings

The dynamic testing validated that the robotic arm meets the specified dynamic requirements for the End Effector force. The minimum force recorded was 4.3 N in configuration 3, slightly exceeding the required minimum of 3 N (Req. 3 - Dynamics).

The test did not include the 0.2 kg payload specified in the requirement. Therefore, the additional gravitational force of about 2 N, which would primarily affect configuration 4, was not accounted for. Although configuration 4 achieved a margin of 2.45 N relative to the required force, this suggests that the prototype was operating near its limit with respect to servo torques. This limitation is attributed to the realized stall torque of the servos, which reached only about half of the specified value.

The prototype's hardware withstood the operational forces during the tests without sustaining any damage. A notable observation was the stability of the previously volatile shoulder joint, which remained intact and operational, marking a significant improvement in mechanical durability (Obj. 1 - Mechanical Durability).

4.3 Conclusion

This master thesis marks a significant milestone in the project's progression toward dual-arm aerial manipulation. It successfully redesigns the existing robotic arm prototype, elevating it to an application-ready state. The conclusions drawn from this study not only confirm the prototype's suitability for advancing the project but also highlight the progress made and the valuable insights gained throughout the development process.

4.3.1 Suitability For Aerial-Manipulation

The suitability of the prototype for aerial-manipulation is validated by the requirements established in Sindermann's foundational thesis [Sin21]. A key requirement for aerial application, Req. 1 - Lightweight Design, states that the robotic arm's weight must not exceed 1 kg. The developed prototype, weighing 951 grams excluding the mounting plate, successfully meets this requirement. All additional requirements have been fulfilled as referenced throughout this thesis, except for one aspect: the joint torques. During testing, the servos achieved only about half of their specified stall torque, failing to meet the required joint torques outlined in Table 2.1. This issue will be further addressed in the upcoming outlook, exploring potential improvements.

Functionality-wise, the prototype successfully integrates all essential features of a robotic arm, including task-space control and trajectory tracking control, achieving these with satisfactory precision. The prototype's observed behavior throughout the testing phase is promising: its motion is smooth and natural, with no self-collisions and safe handling of singularity configurations. The hardware has proven reliable and resilient under operational stresses, qualities vital for the demanding field of aerial manipulation. The accompanying software suite offers an accessible, modular framework for programming the arm. Its clear structure and comprehensive documentation make it easy for future developers to continue the project. Additionally, the graphical user interface enhances interaction with the prototype, making it ideal for demonstration and testing purposes.

In summary, the developed software and hardware of the prototype is suitable for progressing the project towards aerial manipulation with only minor adjustments needed. This prototype stands as a concrete proof of concept, demonstrating the feasibility of designing and constructing a functional robotic arm, even when limited to the resources available to an individual student.

4.3.2 Progress Made and Key Learnings

Significant advancements were achieved over the initial prototype, both in terms of hardware and software. All objectives outlined in section 1.4.2 were met, resulting in a prototype that is significantly more capable and reliable than its predecessor.

In terms of hardware, the most notable improvement is the increased mechanical stability. By simplifying the components, optimizing joint designs, and transitioning from PLA to PETG filament, the new prototype endured much higher mechanical stresses, enhancing its reliability during testing and expanding its dynamic capabilities. The software suite also saw extensive development from the foundational implementation used in the previous prototype. Significant enhancements include new features such

as trajectory tracking control, Nullspace utilization, and workspace calculation. Core modules like the Virtual Robot Framework and Real Robot Interface were completely reengineered for enhanced efficiency and clarity. The addition of a user interface enables interaction with the robotic arm without requiring in-depth knowledge of the underlying software, a capability absent in the previous prototype.

Several key learnings have proven critical throughout the development of this second prototype iteration:

- **Streamlining Manufacturing and Assembly:** Reducing assembly time and limiting reliance on external manufacturing resources proved essential. The frequent redesigns and need for part replacements, inherent in prototyping, would not have been possible without a commitment to 3D printing and a design philosophy focused on essential functionality only.
- **Reducing Complexity:** Reducing the complexity of both hardware and software was vital for success. Simplified hardware removed potential failure points, while the software's design emphasized clarity and modularity. This method not only kept the project manageable but also enables future collaboration by other developers.
- **Documentation and Rigorous Assessment:** Meticulous documentation and rigorous assessment of the prototype was crucial for keeping the development explainable and goal-oriented. This aspect is particularly important given that the project involves iterative contributions from individual students with diverse backgrounds.

The improved prototype serves as an ideal platform to advance knowledge across various engineering disciplines, including software development, hardware design, robotics, and manufacturing. The practical application of theoretical knowledge provides a satisfying and enriching experience, making participation in this project and ideal choice for future engineers.

4.4 Outlook

This final section of the thesis outlines potential enhancements and future research directions derived from the insights gained during the development of the prototype. Due to the time constraints of this master thesis, it was not possible to implement these suggestions. However, they offer a solid foundation for future theses and aim to advance the project towards dual-arm aerial manipulation. This section is intended for future students who wish to continue this work.

To strategically advance the project, the following specific enhancements and research directions are proposed:

- **Migration of the Main Controller:** The main control loop, currently implemented on a PC running MATLAB, operates at a limited frequency of 50 Hz due to significant overhead. This overhead arises from MATLAB's interpreted nature, the PC's operating system, calls to the Dynamixel SDK, and the USB-to-serial conversion process. Migrating the Main Controller to an embedded system, such as a Raspberry Pi or a microcontroller running ROS, would not only enhance controller performance but also support future integration with UAV systems.
- **Improving Zero Calibration:** The manual zero calibration process, detailed in section 3.2.1, currently leads to inaccuracies in the robot's coordinate system. Developing an automated solution could significantly enhance the motion accuracy of the prototype.
- **Increasing Joint Torques:** Dynamic tests indicated that the utilized servos do not meet the required joint torques. Incorporating a planetary gearset in the yaw joint or increasing the voltage of the servos' power supply could provide a substantial enhancement in torque output.
- **Reducing the Weight:** Although the prototype's weight is just under the 1 kg limit, there is potential for significant weight reduction by optimizing the mechanical design, such as reducing the infill density of 3D printed parts or optimizing the hardware design.
- **Exploring Torque Control:** Implementing torque control instead of velocity control for the servos could enhance the arm's compliance and effectiveness in contact tasks, presenting a substantial area for further research in control theory and robotics.
- **Exploring Dual-Arm Control:** The project could expand to include a second prototype for exploring dual-arm control strategies, or alternatively, simulations could be developed for this purpose without the need for additional physical prototypes. This approach offers significant potential for collaborative strategies and advanced motion planning.
- **Exploring On-Drone Integration:** Assessing the feasibility of mounting the prototype on a UAV would provide valuable insights into the physical demands and necessary optimizations for aerial applications.

List of Figures

Bibliography

[24a] *Arduino IDE Documentation*. Online Documentation. Accessed: 2024-05-12. 2024. URL: https://docs.arduino.cc/software/ide/#ide-v2.

[24b] *DYNAMIXEL XM430-W210*. Online Manual. Accessed: date. 2024. URL: https://emanual.robotis.com/docs/en/dxl/x/xm430-w210/.

[24c] *The Dynamixel Wizard*. Online Guide. Accessed: date. 2024. URL: https://emanual.robotis.com/docs/en/software/dynamixel/dynamixel_wizard2/.

[24d] *The U2D2 USB to Serial Converter by Dynamixel*. Online Documentation. Accessed: date. 2024. URL: https://emanual.robotis.com/docs/en/parts/interface/u2d2/.

[24e] *What is DYNAMIXEL*. Online Guide. Accessed: date. 2024. URL: https://www.dynamixel.com/whatisdxl.php.

[Foo13] T. Foote. "tf: The Transform Library." In: *2013 IEEE Conference on Technologies for Practical Robot Applications (TePRA)*. Woburn, MA, USA, 2013, pp. 1–6.

[GO21] G. V. Garcia and A. Ollero. "On Aerial Robots with Grasping and Perching Capabilities: A Comprehensive Review." In: *Journal of Intelligent & Robotic Systems* 99.3-4 (2021), pp. 789–805.

[HB15] A. Harish and G. S. Babu. "Manipulability index of a Parallel robot Manipulator." In: *International Journal of Mechanical Engineering and Technology* 6.6 (2015). ISSN 0976 – 6340 (Print), ISSN 0976 – 6359 (Online), pp. 09–17.

[Koc+22] B. B. Kocer, L. Orr, B. Stephens, Y. F. Kaya, T. Buzykina, A. Khan, and M. Kovac. "An Intelligent Aerial Manipulator for Wind Turbine Inspection and Repair." In: *2022 UKACC 13th International Conference on Control (CONTROL)*. Plymouth, UK, 2022, p. 226.

[OHF+20] A. Ollero, G. Heredia, A. Franchi, et al. *AEROARMS Project Final Report*. Technical Report. Horizon 2020 Research and Innovation Programme. European Commission, 2020.

[Oll+19] A. Ollero et al. *AEROARMS: Aerial Robotics for Inspection with Contact*. Online Resource. Available Online. 2019. URL: https://example.com/AeroarmsInspection.

[Ors+14] M. Orsag, C. Korpela, S. Bogdan, and P. Oh. "Valve Turning using a Dual-Arm Aerial Manipulator." In: *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*. Orlando, FL, USA, 2014, p. 836.

[Ors+18] M. Orsag, C. Korpela, P. Oh, and S. Bogdan. *Aerial Manipulation*. 1st ed. Advances in Industrial Control. Published October 5, 2017. Springer Cham, 2018. ISBN: 978-3-319-61020-7. URL: https://doi.org/10.1007/978-3-319-61022-1.

[Ram+23] S. Ramírez-Revilla, D. Camacho-Valencia, E. G. Gonzales-Condori, and G. Márquez. "Evaluation and comparison of the degradability and compressive and tensile properties of 3D printing polymeric materials: PLA, PETG, PC, and ASA." In: *MRS Communications* 13 (2023), pp. 55–62. URL: https://doi.org/10.1557/s43579-022-00311-4.

[Rix23] D. J. Rixen. *Skript zur Vorlesung Roboterdynamik: Kinematik und Dynamik*. Lecture Notes. Sommersemester 2023. Munich, Germany, 2023.

[Sin21] J. Sindermann. "Design and Construction of a Robotic Arm Towards Aerial Manipulation." Chair of Autonomous Aerial Systems, Department of Aerospace and Geodesy. Semester Thesis. Technical University of Munich, 2021.

[Zei23] S. Zeitler. "Development of a Robotic Arm Towards Aerial Manipulation." Chair of Autonomous Aerial Systems, Department of Aerospace and Geodesy. Semester Thesis. Technical University of Munich, 2023.