Nicolas GAUTIER-KUEA

Master MNIS
2023-2025

ClearSpace SA
Rue de Lausanne, 64, 1020 Renens, Switzerland

Design and development of the electronics for a Solar Trajectory Simulator

From 03/02/25 to 31/07/25

Under the supervision of :

- **Company supervisor : Simon BOTHNER, simon.bothner@clearspace.today**
- **Phelma Tutor : Jonathan MIQUEL, jonathan.miquel@grenoble-inp.fr**

Confidentiality : ☐ yes ■ no

Ecole nationale supérieure de physique, électronique, matériaux

Phelma
Bât. Grenoble INP - Minatec
3 Parvis Louis Néel - CS 50257
F-38016 Grenoble Cedex 01

Tél +33 (0)4 56 52 91 00
Fax +33 (0)4 56 52 91 03

http://phelma.grenoble-inp.fr

Acknowledgments

This internship has been a tremendous opportunity for me as a future engineer, and I am truly grateful to everyone who contributed—directly or indirectly—to making it possible.

First and foremost, I would like to sincerely thank Simon Bothner, my internship supervisor, for trusting me with a project that was both meaningful and fascinating, given the wide range of applications and future possibilities it offers. The freedom he gave me in researching and developing the solution was an incredible opportunity to gain a comprehensive vision of the project. It allowed me not only to thrive within it, but also to face and overcome moments of crisis—valuable learning experiences for both my professional and personal growth.

I also want to thank the ClearSpace team for creating such a positive and supportive work environment. Their kindness and good spirit made me feel welcome and respected from day one. Special thanks go to Mickael Pellet, whose guidance on several key electronic design decisions was especially helpful throughout the development process.

Finally, I would like to thank my family and friends, who have always shown genuine interest in the projects I've undertaken and have been a constant source of motivation and support throughout my academic journey.

Table of contents

Glossary

LEO: Low Earth Orbit. An orbit close to Earth (160–2,000 km altitude), used for satellites and space missions.

ESA: European Space Agency. Europe's main space organization, responsible for space research and missions.

VESPA: VEga Secondary Payload Adapter. A structure on the Vega rocket that allows launching multiple payloads.

PROBA-1: PRoject for Onboard Autonomy. An ESA satellite testing autonomous systems; launched in 2001.

SunSim: Sun Simulator. A robot designed to replicate illumination scenarios similar to those encountered in orbit.

CS-1: ClearSpace-1. Prior mission of ClearSpace aiming to remove the PROBA-1 debris from the LEO.

GNC: Guiding Navigation Control. ClearSpace's computer vision protocol to guide the servicer satellite toward the target.

ADR: Active Debris Removal

KiCAD: Ki Computed Aided Design. Open-source software used to design electronic schematics and PCBs.

SSH: Secure SHell. A secure protocol to remotely control a computer via terminal.

RPi: Raspberry Pi. A compact, low-cost computer used in electronics and programming.

JSON: JavaScript Object Notation. A lightweight text format to store and exchange structured data.

µC: microcontroller. A small chip with a processor, memory, and inputs/outputs, used in embedded systems.

PCB: Printed Circuit Board. A board that connects and supports electronic components.

GPIO : Global Purpose Input Output

List of Figures

Introduction

As part of the final semester of my Master's program in MNIS (Micro and Nanotechnologies for Integrated Systems), I completed an internship at ClearSpace SA, a Swiss startup whose mission is to pave the way for sustainable space operations. Currently, no viable solution exists to handle space debris—such as end-of-life satellites or fragments trapped in low-Earth orbit—which poses significant threats to future space exploration. These risks imply careful mission planning to avoid potential collisions and thus, ClearSpace aims to address this problem by proposing a dual solution: either removing space debris by launching a satellite equipped with a guiding navigation algorithm and arms, or performing life extension operations on still-working spacecraft by refueling them or providing provide in-orbit maintenance operation to make them operational for several more years.

During this internship, I worked on the electronic and software development of a sun simulator, intended to support the training of the company's guidance algorithms. This project involved electronic circuit design, PCB prototyping, and Python programming to create a system capable of simulating various space illumination scenarios.

In this report, I will present the different aspects of my internship, beginning with an introduction to the project's context, including the company, its goals, and my role as a development intern. I will then discuss the state of the art and the preliminary studies that preceded my involvement, as well as the project requirements. Following that, I will describe the component testing and development phases and finally, I will provide an overview of the results along with suggestions for future improvements.

I. Context

a) Presentation of ClearSpace

ClearSpace is a Swiss startup founded in 2018 with the mission of addressing space debris and end-of-life satellites to make future space missions safer and more sustainable. The importance of ClearSpace's mission is significant. Considering the growing amount of space debris over the past decades, it is clear that without actors like ClearSpace, Earth's orbit in the near future could become saturated by debris, making future missions extremely challenging—if not impossible. According to ESA figures, as of May 5, 2025, there are 14,240 satellites in orbit, with around 11,600 still functioning. However, the total number of space objects regularly tracked is approximately 41,670. This much higher number can be explained by the fact that once a spacecraft is no longer in service, it can be destroyed by collisions with other debris, generating smaller fragments. And for this reason, ESA reports that about 54,000 space objects are larger than 10 cm, and around 1.2 million are between 1 cm and 10 cm. Given that these objects can travel at speeds up to 8 km/s, even small fragments pose a significant threat, whereas others of the order of mm can disable satellites.

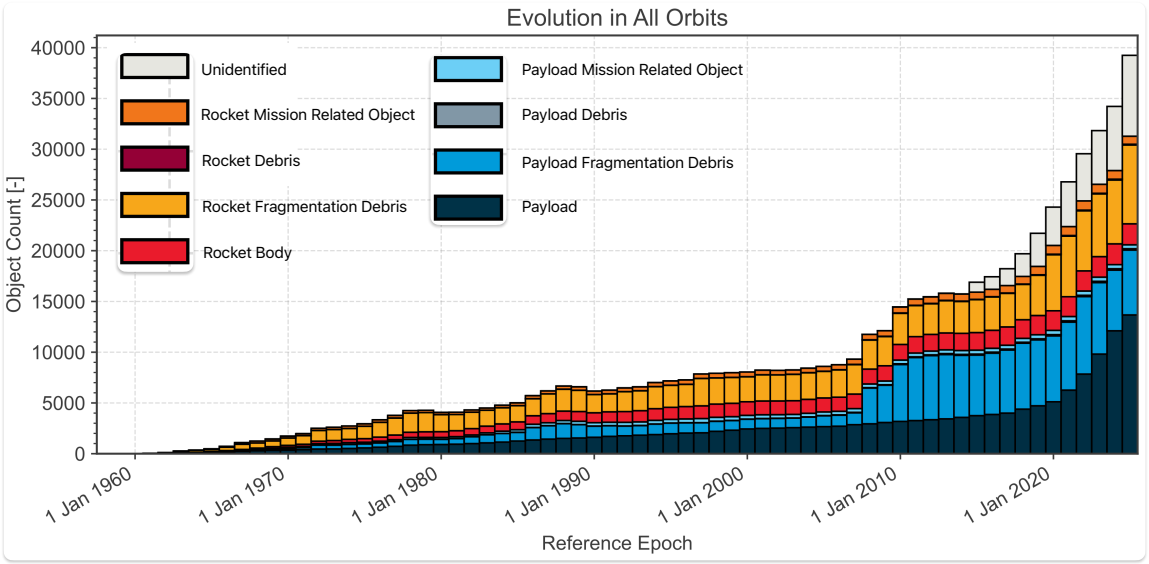

Figure 1: Evolution of in-orbit debris since the beginning of the space era

Eventually, the goal of ClearSpace is to propose In Orbit Servicing (IOS), services that include removing debris to build a safe path for future launch, or bringing maintenance possibilities to in-orbit satellites. ClearSpace's technology focuses mainly on Active Debris Removal (ADR) of non-cooperative spacecraft. This means that the success of the removal operation relies heavily on the guidance algorithm, electronics, and mechanical systems developed by ClearSpace, as the target is an unresponsive object whose trajectory cannot be adjusted.

ClearSpace has several offices: in Renens, Switzerland; London, United Kingdom; and Luxembourg. At the Renens office, where I completed my internship, the primary focus areas are computer vision, Guidance Navigation and Control (GNC) protocols, and mechanical engineering, including the design and testing of the robotic claw that will be mounted on the satellite. In Switzerland, to support the development of their product, ClearSpace has built two key facilities for in-house testing: the CleanRoom and the DarkRoom.

Figure 2: Darkroom (left) and Cleanroom (right)
(Images credit ©ClearSpace)

The CleanRoom is a sterile environment used to assemble and test mechanical claw while minimizing dust and moisture contamination whereas the DarkRoom is a blacked-out facility where two robotic arms face each other under a carbon structure with a mobile LED light source. The purpose of this room is thus to replicate space lighting conditions to test and train the GNC and computer vision systems. One robotic arm represents the satellite equipped with the guiding equipment (cameras, infrared sensors…), the other the target, and both are mounted on one rail to perform translational movement.

While ClearSpace focuses on developing the ADR and GNC protocols, it subcontracts the development of the other parts of the satellite. These subcontractors are responsible for designing or supplying components such as guidance equipment (including cameras and sensors), the propellant tank, the propulsion system, or even the satellite platform itself. Maintaining effective communication with these subcontractors is therefore crucial, as their work directly influences how ClearSpace's overall solution is built.

b) ClearSpace missions

In this subsection, I will present the main missions ClearSpace aims to develop along the next years.

- The ClearSpace-1 mission

The ClearSpace-1 mission (CS-1) is the prior mission for which ClearSpace has been created: the removing of a space debris from the Low Earth Orbit (LEO). This first mission was born after ClearSpace was selected by the European Space Agency (ESA) to remove the upper part of a VESPA (Vega Secondary Payload Adapter) from the VEGA launcher. However, the target since changed as the VESPA entered in collision with another debris, changing its expected trajectory and the target is now the PROBA-1 (Project for On-Board Autonomy) satellites, a 94 kg satellites launched in 2001 for Earth observation.

This first mission aims to be a demonstration of the capture system of ClearSpace. Indeed, during this mission, the satellite equipped with the ClearSpace guiding electronics and claws is supposed to capture the PROBA-1 and then engage a re-entering phase to destroy the debris in the atmosphere, which will also destroy the servicer at the same time. In future mission, the idea would obviously be to deorbit multiple targets at each launch, but this mission would already be a world's first.

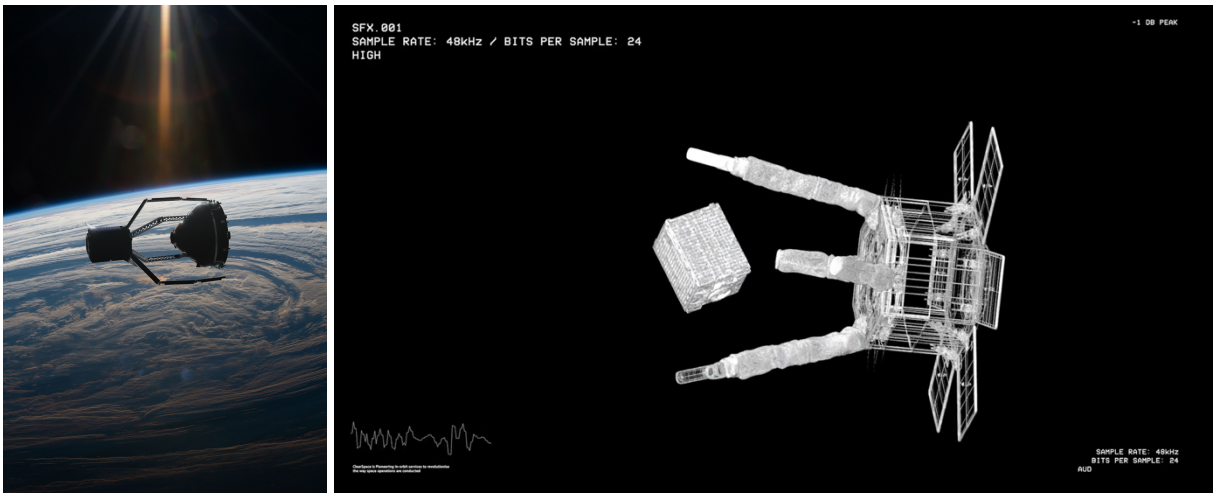

Figure 3: Illustration of CS-1 mission (left: VESPA, right: PROBA-1)
(Images credit ©ClearSpace)

- The CLEAR mission

The CLEAR mission has the same goal as the CS-1: remove a debris from LEO, the TOPSAT satellite (Tactical Operational Satellite). The difference however is that after the deorbitation, the aim would be to get back in orbit to then deorbit another target and even do some refueling of another satellite. This mission follows a call for tenders of the United-Kingdom Space Agency.

Figure 4: Illustration of CLEAR mission
(Image credit ©ClearSpace)

Something worth mentioning, however, is that for both CLEAR and CS-1, since these missions result from calls for tenders issued by ESA and the UK Space Agency, it means that multiple companies are competing to ultimately be selected to launch their solution into space and carry out the mission. Currently, the selection process for both ESA and UKSA is in an advanced stage, with only a few companies still in the running. This phase mainly involves demonstrating development progress to the agencies, showcasing the maturity and feasibility of the proposed technologies. At the end of this stage, the two space agencies will assess which solution appears most promising and will select the final contractor to move forward with the mission.

- The Phoenix mission

The Phoenix mission, by contrast, focuses on satellite life extension. Using a similar rendezvous protocol, a servicing satellite would approach another satellite nearing the end of its operational life to refuel it, perform maintenance operations, or provide a momentum boost, allowing it to remain operational for several more years. In the context of this mission, the idea is for ClearSpace's servicer to dock on the target satellite, move it to a safer orbit, and once the satellite eventually stops functioning, send it further away.

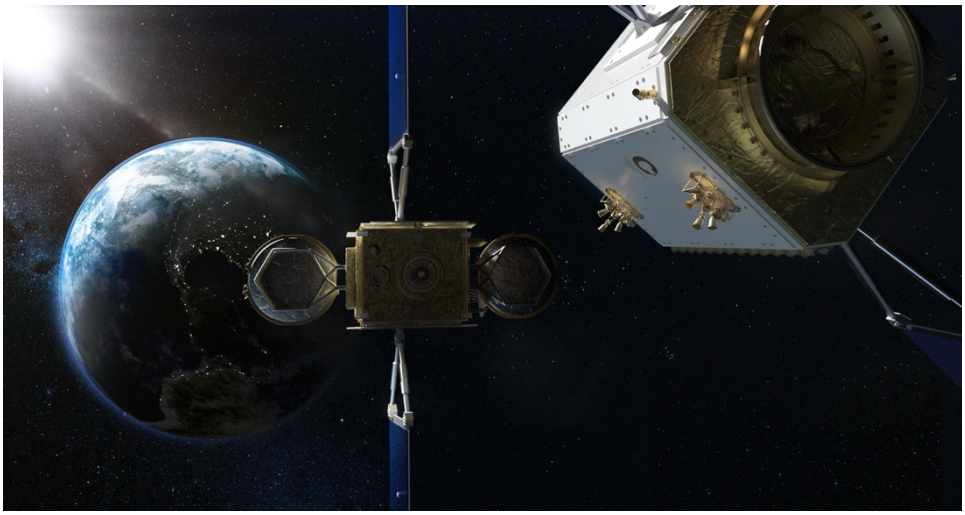

Figure 5: Illustration of the Phoenix mission
(Image credit ©ClearSpace)

This mission would represent ClearSpace's first opportunity to carry out a commercial mission not directly commissioned by a space agency. Unlike the CLEAR and CS-1 missions, which are funded by institutional clients and target already defunct satellites, Phoenix targets an active, commercially valuable satellite. Successfully completing such a mission would generate significant revenue for ClearSpace, supporting both team growth and the development of future missions.

c) My goals at ClearSpace

From the previous section, one can notice how various and complex the missions that ClearSpace might need to carry in the future are. And for this reason, I joined ClearSpace to help developing a simulator that will support the development of their GNC protocol, the SunSimualtor. As a development intern, I had to build the electronics and software controlling the robot and my responsibilities included designing electronic circuits, prototyping PCBs using KiCAD, and programming a Raspberry Pi in Python. The system was intended to control multiple motors simultaneously based on data from a JSON file describing the sun's relative trajectory with respect to the satellite-target system, in order to recreate realistic space illumination conditions. Once the electronic developed, the PCBs and wiring had to be integrated into the existing DarkRoom structure, which includes a steel frame and a carbon cantilever on which the LED light would move.

II. Preliminary studies and existing work

a) System overview

The DarkRoom is a testing facility designed to simulate, on Earth, the approach phase of a servicer satellite toward its target. The setup of this room is illustrated in the figure below:

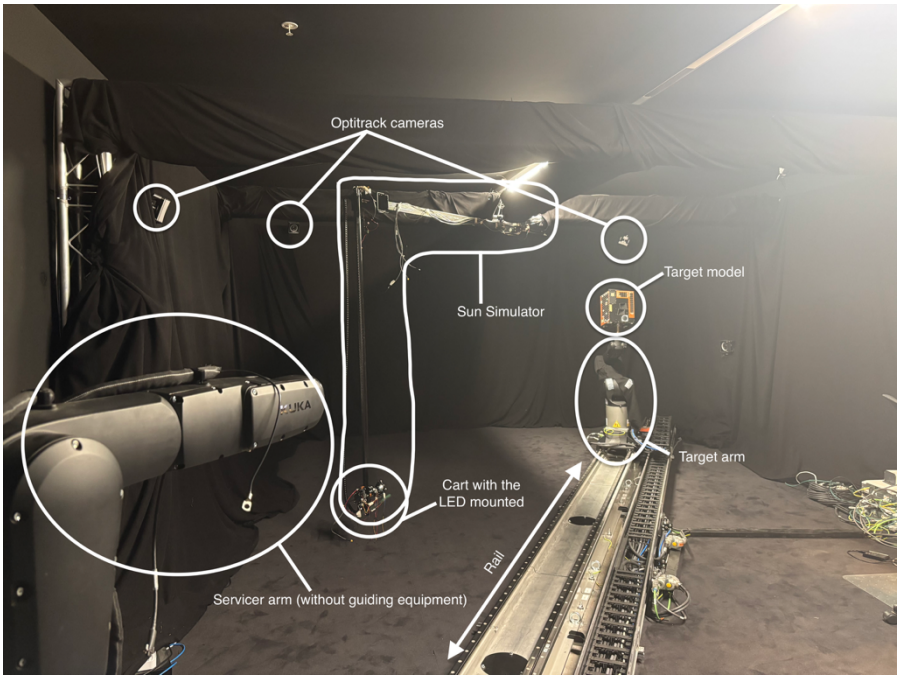

Figure 6: Darkroom facility

It is a completely blacked-out room containing three main elements: the servicer arm, the target arm, and the Sun Simulator. Both the servicer and target are represented as 6-axis robotic arms. At the end of the target arm is fixed a physical replica of the target satellite and the servicer arm holds the guidance electronics, including cameras and sensors that collect and process critical data-these data would normally be used to command the satellite's propulsion system and adjust its trajectory. Thanks to their six degrees of freedom, the arms can perform precise, repeatable, and complex motions. These allow for the realistic mimic of orbital dynamics between the servicer and target. Additionally, both arms are mounted on a rail system that enables translational movement, simulating the approach between the two spacecraft.

A metal structure surrounds the room from the walls to the ceiling and allows important elements to be hung from it, such as the sun simulator or the black curtain covering the perimeter of the room. The latter notably allows cables or electronics to pass through without harming the desired complete darkness by preventing cables or PCBs from reflecting light.

The reason why this room is entirely dark is that the navigation system relies heavily on optical cameras and sensors. This means that for proper testing of the algorithm, it is important to replicate the lighting conditions found in space, where the sun is the only source of light and the surrounding environment is otherwise pitch black

This is where the Sun Simulator comes into play. The DarkRoom alone only supports linear translation between the modules. If a fixed light source were used, it would falsely imply that the servicer and target are moving in a straight line relative to the sun—which does not reflect the reality of orbital mechanics. In actual missions, both satellites are rotating around the Earth and possibly also around each other depending on the scenario. The purpose of the Sun Simulator is therefore to recreate this complex relative motion of the sun by dynamically

adjusting the lighting on the target. This is achieved by moving an LED light source along four different axes around the servicer-target system, simulating a realistic and changing sun position over time.

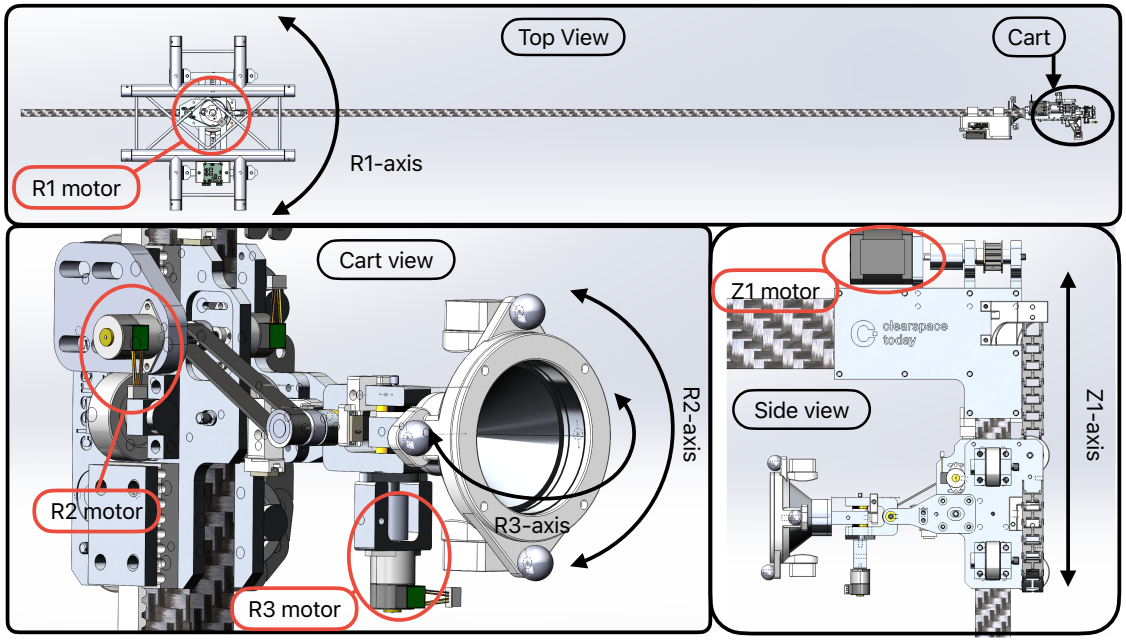

Figure 7: Axis schematic

The SunSimulator dispose of four different axis: the R1 axis that enables the sun to rotate around the servicer-target couple, the Z1 axis that allows to adjust the height and the R2 and R3 which control the respectively the pitch and yaw of the LED for a better focus on the target. The LED itself is mounted on a so called “Cart” that slides onto the Z1 axis. At this point, it is worth mentioning that the size of the robot is a major point to consider: from the R1 rotating point to the end of the cantilever, there is about 3 meters of carbon fiber whereas from top to bottom, the Z1 axis measures more than 2 meters. While this large form factor is advantageous for illuminating the entire dark room, it also introduces significant constraints due to the inertia generated during movement.

To control the position of each axis—both for the robotic arms and the Sun Simulator—a JSON file is used, containing pre-calculated coordinate data. The system I am aiming to develop shall read this data and translates it into commands that drive the motors, ensuring each axis is set to the correct position at the right time to reproduce the intended scenario.

Additionally, an optical tracking system surrounds the so called “arena”. This system called the Optitrack was not used during my internship, but its purpose would eventually be to track the motion of the target, servicer and sun by using multiple cameras equipped with infrared emitters illuminating the arena and reflecting off small retroreflective markers attached to the tracked objects. This would thus allow a closed-loop system allowing to have realtime check of the motion performed.

Figure 8: OptiTrack and infrared reflector

b) State of the art

Upon my arrival, the first major component I studied to familiarize myself with the project was the electronic system previously designed by a former intern. Although the PCBs had been designed, they had not yet been routed or physically manufactured and my objective was therefore to verify the correctness of the schematics and assess whether the design still aligned with the updated project requirements.

The existing work consisted of two PCBs: one designed to control the four motors and host the computational unit, and another dedicated to powering and controlling the LED.

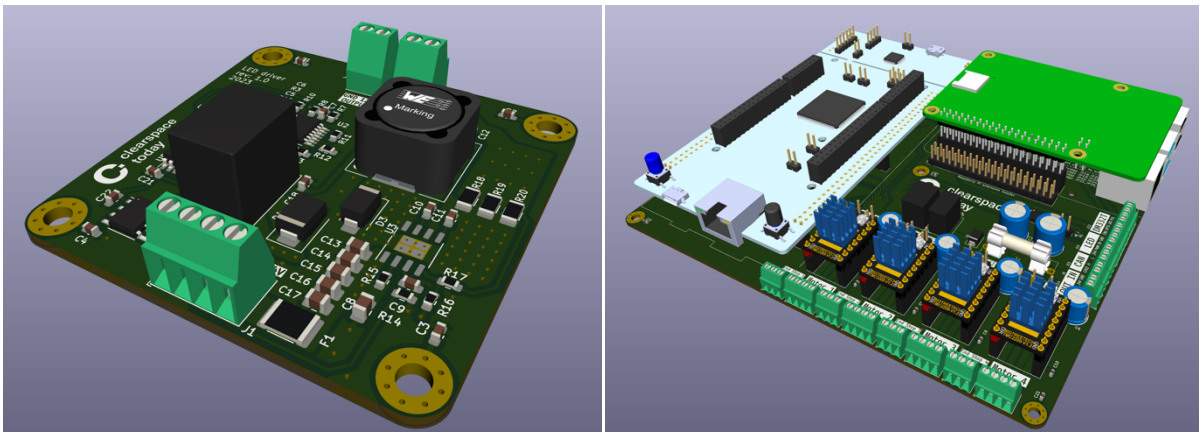

Figure 9: Previous PCBs iteration

While the initial design appeared to be electrically functional, the specific needs of my project required significant modifications—both in terms of control logic and the physical architecture of the PCBs, in order to ensure better integration with the robotic structure.

One of the most notable differences concerned the control system architecture. The original approach involved using both a Raspberry Pi and an STM32 microcontroller, communicating with one another: the Raspberry Pi would receive the input data and send instructions to the STM32, which would then generate the low-level commands for the motor drivers. This design choice was made to leverage the real-time capabilities of the microcontroller, as unlike the

Raspberry Pi, it does not run a full operating system and thus avoids task scheduling delays: the goal was to meet tight timing constraints for precise motor control. However, it became apparent that establishing a reliable communication protocol between the Raspberry Pi and the STM32 was difficult and unstable. Previous tests had yielded inconclusive results, and I was therefore advised to explore a simplified solution using the Raspberry Pi alone as the main controller.

Another key consideration was the mechanical integration of the PCBs. The previous two-board setup did not account for the spatial constraints imposed by the robot's structure. In particular, the first iteration of the ControlPCB implied to run cable from the PCB to each motor, which is quite inconvenient given the distributed placement of the motors on the robot (see Figure 6 and 7). This design lacked the modularity and flexibility required for proper integration and this constraint became a driving factor in the design decisions I made during the development phase.

III. Component investigation and system architecture

The purpose of this section is to summarize the components intended for use in building the solution and to introduce the communication protocol used to control the system. It will therefore provide a brief overview of how the system is to be built and how each component should interact with the others.

a) Hardware components

- The power supply

Along most of the tests, a bench power supply was used, capable of supplying up to 24V-6A, but it was decided that to power the whole robot, a 24V-13A power supply would be used on the final setup.

Figure 10: Bench power supply and HEP-320 PSU

- Stepper motors + drivers

The stepper motors and the drivers are the core of the robot, as they are responsible of creating the motion of the SunSim. Stepper motors are a type of motor which peculiarity compared to regular motors is that they are controlled using pulses instead of a DC voltage. A stepper motor operates by energizing coils in a specific sequence to rotate the rotor in fixed angular steps. It consists of a stator with multiple coils arranged around the rotor. When current flows through the stator coils in a controlled pattern, magnetic fields are generated that attract the rotor teeth or poles, causing the rotor to align step by step. Thanks to their precise motion control capacities, they are often used for CNC or 3D printer.

In contrast, a traditional electric motor—such as a brushed DC motor—has a stator that creates a constant magnetic field and a rotor (the armature) with windings connected to a commutator. As current flows through the rotor coils via brushes, the magnetic interaction between the stator and rotor causes continuous rotation. Unlike stepper motors, which move in discrete increments, standard electric motors rotate smoothly and require feedback systems for precise position control. Stepper motors, by directly controlling the sequence of coil energizing, offer better precision and repeatability without needing external sensors.

Another option would have been to use servomotors. These devices are appealing because they include integrated encoders that provide real-time position feedback, allowing the system to know the motor's exact position at all times—even after power loss. However, in the context of this project, servomotors presented several limitations. On certain axes, gear reduction is used to increase torque, which would decouple the encoder's readings from the actual position of the moving part, making the position feedback unreliable. Furthermore, most standard servomotors are limited to single-turn operation, whereas some of the robot's axes require continuous or multi-turn motion. High-torque servomotors capable of continuous rotation do exist, but they are typically larger and bulkier than stepper motors. This lack of compactness makes them less suitable for integration into the SunSim system, especially on the cart, where space constraints and design flexibility are critical.

In the context of the SunSim project, two types of motors were selected: a motor from the NEMA17 lineup and the 15PM20L02-SG100 motor (referred to as SG100). The NEMA17 is a powerful stepper motor that provides high holding torque, making it suitable for both the R1 and Z1 axes, as they are responsible for moving the entire robot. The SG100, on the other hand, is used as a pointing motor to control the pitch and yaw of the LED. Although it provides significantly less torque, the weight it needs to move is much lower, and its compact form factor made it easy to integrate into the robot.

Figure 11: NEMA17 (left) SG100 (right)

To control the motion of a stepper motor, a driver is required to generate the electrical sequence applied to the coils. The basic principle is that the driver is powered by a voltage source, and this voltage is then applied to the motor's coils according to a pulsed signal (PUL), whose frequency determines the motor's speed. This signal in a way, "shapes" the power supply in order to control the motor. The direction of rotation (clockwise or counterclockwise) is controlled by a HIGH or LOW signal (DIR) applied to another input of the driver. One important parameter for speed control is the number of pulses per revolution (pulse/rev). This value corresponds to the number of pulses required for the motor to complete one full revolution. A commonly used value is 400 pulses/rev, meaning that with a PUL signal of 400 Hz, the motor will rotate at 1 revolution per second. This parameter can be set in different way depending on the driver.

The NEMA17 is powered by 24V, whereas the SG100 operates at 5V. For this reason, they are controlled by two different drivers. The NEMA17 uses the DM320T driver, which requires three input signals in addition to the 24V/GND power supply: PUL, DIR, and OPTO. The OPTO input is simply a 5V signal used to power the driver's optocoupler stage. These are robust industrial-grade drivers capable of delivering high currents of up to 2.2 A. The output current can be configured using dedicated DIP switches (set to 1.6 A per phase for the NEMA17). Microstepping is also configured via another set of three DIP switches, allowing settings from 400 to 12,800 pulses per revolution. The SG100, on the other hand, is driven by the TMC2209 — a smaller, compact driver that is easy to integrate onto a PCB thanks to its pin headers. It requires only the PUL and DIR signals, and microstepping is configured using two dedicated pins, each of which can be set to HIGH or LOW, allowing four possible combinations. Both drivers have four outputs corresponding to the four motor windings, labeled A+, A− and B+, B−. The connections between the motor and the driver must respect these designations to ensure proper operation.

Figure 12: DM320T (left) TMC2209 (right)

- LED + Fans

The LED used is a 3.3 V / 10 W high-power white LED. Thanks to its compact size, it was easy to integrate onto the robot. With a maximum brightness of 1000 lumens, most lighting scenarios can be recreated using a current driver circuit that controls the luminosity by controlling the current flowing through the LED. Due to its small form factor and high-power density, two fans are required to cool it down. These are standard 5 V fans that only need to be powered — they do not require control signals and are simple to implement.

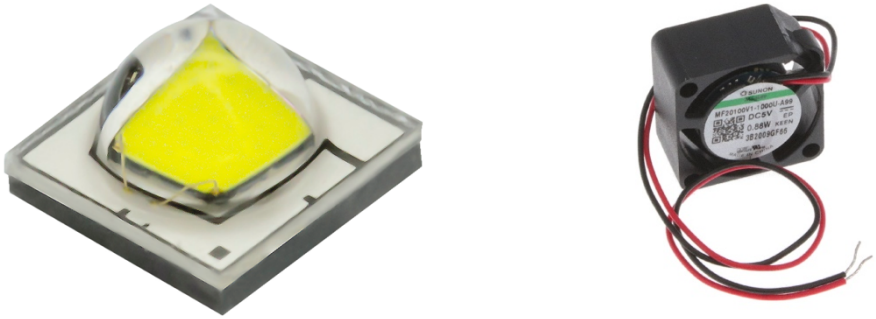

Figure 13: LED and Fan

- The slip ring

As described in section II.a, the robot is attached to a steel frame. Since the power supply is wall-plugged and routed from the top of the structure, a slip ring is necessary to allow full 360° rotation of the carbon arm without the cable restricting movement. A slip ring enables continuous rotation by having fixed wires on one side (connected to the static part) and rotating wires on the other (connected to the moving axis). The selected slip ring features 12 channels rated at 2A per wire, providing enough flexibility to transmit both power and signals between the rotating and static parts of the robot.

Figure 14: Slip ring working principle

- Contact and optical sensors

As the stepper motors used do not have built-in encoders or other position feedback systems, contact and optical sensors are employed. These are digital high-low sensors that act essentially like switches. When triggered, they provide a signal indicating that a motor has reached a predefined position or limit. The contact sensors are basic mechanical switches activated when a physical element presses the button. They are used on the R2, R3, and Z1 axes. On the R1 axis, four optical sensors are installed to divide the rotation into four quadrants. These sensors are fixed to the moving part, and each one is triggered by a screw attached to the metal frame. As the carbon arm rotates, the screw successively activates each sensor, allowing the system to determine in which quarter the arm is currently located.

Figure 15: Contact sensors (left) and optical sensors (right)

b) Control and communication

The entire robot is designed to be controlled by a Raspberry Pi. Raspberry Pi boards are compact, affordable and user-friendly single-board computer. They can perform many tasks similar to a standard computer, such as running a Linux-based operating system, connecting to the internet, or managing peripherals via the Global Purpose Input Output (GPIO) pins they embed. They are also well equipped with useful port such as USB, Ethernet or HDMI making them very versatile for hardware-oriented projects. In the frame of the project, the RPi board was used to execute python script to control the motor. The concept is thus to provide it with a JSON file containing the sun's coordinates at various time intervals, describing a trajectory synchronized with the two robotic arms. The Raspberry Pi interprets these coordinates into motor commands by generating the appropriate PUL and DIR signals and sending them to the drivers. Initial programming and testing were carried out with the Raspberry Pi connected to an external monitor, but the final goal is to operate the system remotely via an SSH connection between a computer and the Raspberry Pi.

Figure 16: Raspberry Pi 5

The following figure summarizes the overall system architecture:

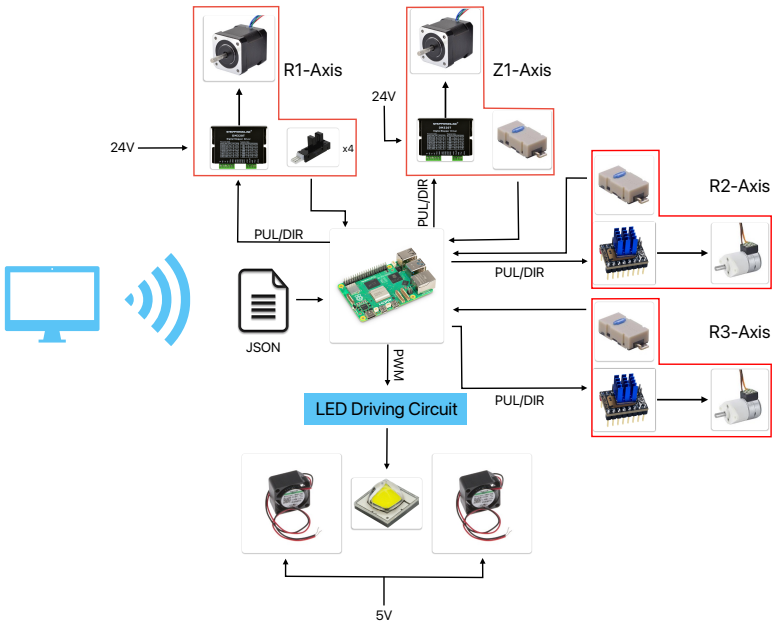

Figure 17: System architecture

IV. System development

a) Initial setup and tests

The first step consisted in becoming familiar with the main components — namely the stepper motors and the Raspberry Pi. The objective was to test and understand how to control the motors by generating PUL and DIR signals using the Raspberry Pi.
The stepper motor tests were relatively simple, as the main properties to verify were speed and direction control. The test setup is shown in the figure below:

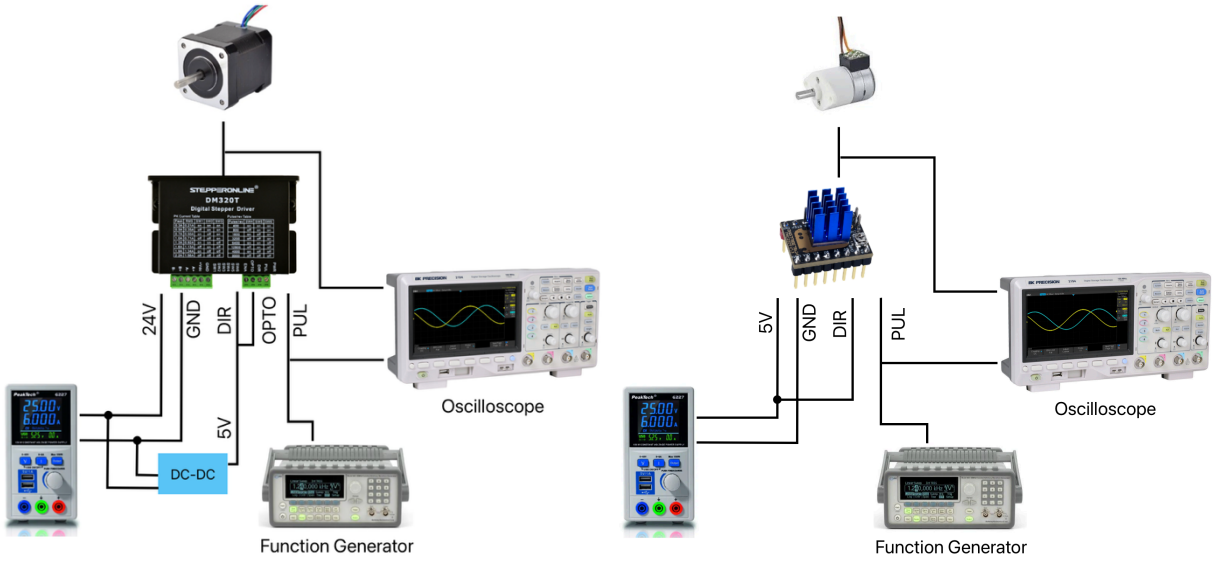

Figure 18: Stepper motors startup test

It consisted of the drivers and motors, both powered by a lab bench power supply, along with a function generator. In the case of the NEMA17, an additional 24V-to-5V DC-DC converter was used to generate the OPTO and DIR signals. An oscilloscope was also used to continuously monitor the input and output signals.

Motor control proved to be quite intuitive: changing the frequency of the PUL signal allowed control over speed, and toggling the DIR pin between 5V and GND changed the rotation direction. Microstepping was also tested, confirming that it affected the driver's resolution — a higher pulse frequency was needed to achieve one full revolution per second at finer microstepping settings.

The next objective was to control the stepper motors using a Raspberry Pi by generating a pulsed signal with a tunable frequency to adjust motor speed. Initially, I worked with a Raspberry Pi 4, as it was the version available at the company. Research indicated that the most suitable library for generating pulse signals was the *pigpio* library. However, after implementing the program, it became clear that this approach had significant limitations. While the library provided several methods for generating pulses, each came with major drawbacks. Some methods allowed precise frequency control but were limited to a single GPIO pin. Others supported simultaneous signal generation on multiple GPIOs but only at a very limited set of predefined frequencies. The company then acquired a Raspberry Pi 5, which was compatible with an alternative library, *gpiozero*. However, this library also presented limitations, with similar trade-offs depending on the method used. Ultimately, the chosen solution was to use an external module dedicated to signal generation, offloading this task entirely from the Raspberry Pi.

The Pulse Train Hat (PTHat) is a module from the "HAT" (Hardware Attached on Top) family, which consists of hardware extensions designed specifically for the Raspberry Pi environment. Most of these modules are simply plugged onto the GPIO pins as extension boards, and communication between the Raspberry Pi and the HAT is generally straightforward. Some modules enhance the Pi's computing power, others add features like cooling fans, and in the case of the PTHat, it enables the generation of control signals for driving stepper motors. The PTHat was originally developed for 3D printer applications, which rely heavily on stepper motors to control the movement of the print head. It is therefore a perfect fit for our project.

Figure 19: PTHat boards

The PTHat can generate up to four independent PUL signals simultaneously while also providing the 5V supply required for the OPTO pin of the DM320T driver. It also features dedicated direction pins for each motor. These signals are available via connector blocks which eases the access and wire making. Communication with the PTHat is handled via a UART connection established between the Raspberry Pi and the board and A dedicated Application Programming Interface (API) is provided, making it easy to send commands to the PTHat. These commands are in ASCII format, simplifying programming by eliminating the need to manually handle protocol configurations such as baud rate or parity bits.

This API is particularly convenient, offering intuitive functions that allow users to set all key parameters for precise stepper motor control. As discussed in section III.a, stepper motor rotation is controlled via pulse signals. For instance, if the driver's microstepping is set to 400 steps per revolution, this corresponds to a resolution of 360°/400 = 0.9° per pulse. The API allows users to specify a predefined number of pulses, which enables fine-grained control of motion — down to tenths of a degree, or even finer when using higher microstepping settings. The pulse frequency can also be adjusted, giving control over movement speed. Additional parameters such as rotation direction and acceleration profiles are also available, allowing full control over motor behavior.

Programming a stepper motor using the PTHat is quite intuitive and typically involves five main steps:

- *Yaxis = Axis("Y", command_id=1)*

This line creates an instance of the Axis class, associated with the Y-axis port of the PTHat. The *command_id* is a unique identifier used to differentiate commands addressed to each axis.

- *Set_yaxis_cmd=yaxis.set_axis(frequency=f, pulse_count=p, direction=1, start_ramp=1, finish_ramp=0,ramp_divide=100, ramp_pause=10, enable_line_polarity=1)*

This line constructs the ASCII command string that configures the Y-axis. It defines the motion profile, including frequency (*f*), number of pulses (*p*), rotation direction, and ramp settings for acceleration and deceleration.

The generated string has a fixed 37-character format, with clearly defined byte segments:

ASCII command	I	01	CY	125000.000	4294967295	1	1	1	100	010	0	1	*
Segment	1	2	3	4	5	6	7	8	9	10	11	12	13

Table 1: ASCII packet nomenclature

In the ASCII data frame, the second segment specifies the axis command ID (must match the one used in the *Axis* instantiation). The third indicates which axis is being set, the fourth sets the pulse frequency, the fifth specifies the number of pulses to send and the remaining characters configure acceleration, ramp pauses, and direction. Each command string must end with an asterisk (*) to indicate the end of the instruction.

- *yaxis.send_command(set_yaxis_cmd)*

This line sends the previously generated ASCII command over the UART interface to the PTHat, which interprets it and prepares the motion parameters.

- *wait_for_responses(yaxis, ["RI02CY*", "CI02CY*"], "Message")*

This step ensures that the PTHat correctly received and interpreted the command. The *wait_for_responses()* function listens to the serial line for specific confirmation messages sent back by the PTHat. It takes as parameters the name of the axis, the response the PTHat sends and print a *"Message"* that the user can custom to have an interpretation of the answer of the PTHat. In this case, the expected responses correspond to confirmation of the command configuration. This step is essential to validate proper synchronization and error-free communication.

- *yaxis.send_command(yaxis.start())*

This command sends the *"start"* instruction to begin motor motion, using a similarly structured ASCII command string. As before, the execution would be followed by a call to the *wait_for_responses()* function to verify that the motor has started and that pulses are being sent as expected.

By following this sequence for each of the four motors, I was able to set them in motion simultaneously while maintaining individual control over speed, direction, and timing. I also successfully implemented staggered start times and conditional stops based on sensors feedback, allowing the robot to respond dynamically when reaching one of the end stops.

Once the control of the motor has been assessed, we decided to perform the first real condition tests of the motor by mounting them on the structure one by one. These individual tests were conducted to better assess the current consumption of each stepper motor, which was a key requirement for correctly sizing the PCB power tracks. Since the current powering the motors flows directly through the PCB, undersized tracks could overheat or even be damaged if the current exceeds safe thresholds. Through these tests, it was determined that each NEMA17 motor could draw up to 1 A, while the two smaller SG100 motors typically stayed below 200 mA.

During these tests, a specific mechanical limitation was identified on the R2 axis. This axis is responsible for pitching the LED from bottom to top and, under nominal operating conditions with a 5 V power supply, the motor could not deliver enough torque to lift the LED past its most demanding position—where gravitational torque is highest. To address this issue, one solution involved increasing the supply voltage above the driver's nominal recommendation, which improved torque output and allowed the LED to complete the motion. However, this came at the cost of higher motor temperatures, which raised long-term reliability concerns. To mitigate this, the problem was taken into account in the following design stages, and in parallel, a member of the mechanical team proposed an improvement to the torque transmission system between the motor shaft and the LED axis. This optimization aimed to reduce the required motor torque by improving mechanical leverage, providing a more sustainable and reliable solution to the R2 motion issue. Apart of this issue, the motion on the other axis appeared to be smooth and easy to control, bringing confidence in the control protocol for the next steps.

b) Hardware development and prototyping

Based on the findings from the initial investigation phase, the next step was to design the electronic architecture of the robot, taking into account both mechanical constraints (such as the placement of the power supply unit, efficient use of the slip ring, and distribution of the electronic boards) and electrical considerations (available power, circuit sizing, powering various components, etc.).

The robot can be divided into three main parts: the fixed base, composed of the metal frame on which the carbon arm rotates; the horizontal carbon arm; and the vertical arm, along which the cart carrying the LED moves. Given the robot's wide and distributed design, it became clear that optimizing the PCB placement would be essential to minimize cable congestion and build a safe, robust, and reliable system.

Considering the location of each motor (see Section II.a), it became evident that three PCBs would be necessary: one dedicated to the R1 motor, one for the Z1 motor, and one to control the two motors mounted on the cart. This architecture would provide greater flexibility, as it allows the placement of PCBs in critical positions, thereby reducing cable length. It also offers more freedom in optimizing connector placement on the boards — a key factor in minimizing congestion and simplifying the design of protective housing for each PCB.

This 3-PCBs architecture would be composed of the so called "PWR_PCB", "Control_PCB" and "Cart_PCB".

- PWR_PCB

As its name suggests, this PCB is responsible for powering the robot. It acts as the primary interface between the power supply unit (PSU) and the rest of the system. For this reason, it would be placed on the non-moving part of the robot since the PSU is fixed on the ground. In addition to distributing power, this board also reroutes the control signals — PUL and DIR — from the PTHat to the R1 motor driver. The OPTO signal required by the driver is generated via a 24V-to-5V DC-DC converter and these three signals are grouped into a dedicated connector, allowing the creation of a single cable for driving the motor.

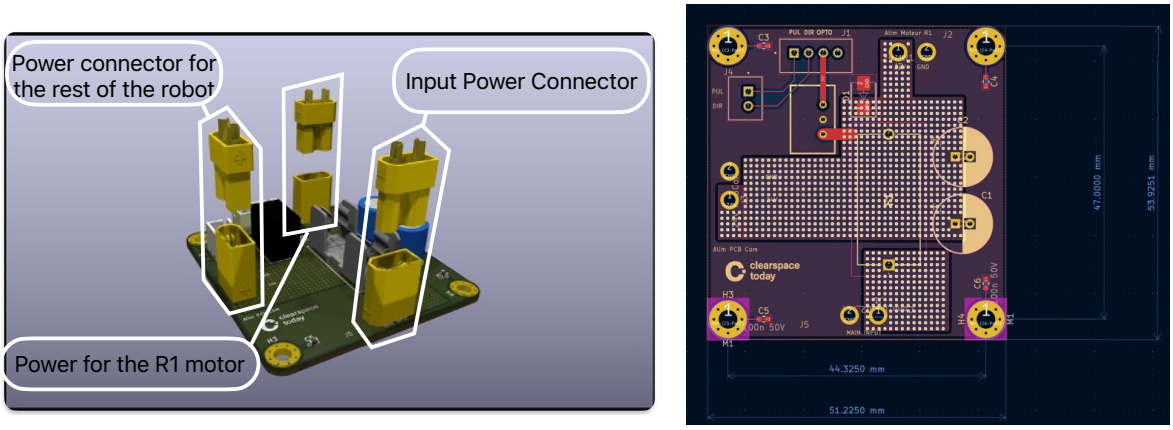

Figure 20: 3D preview and routing of the PWR_PCB

The most critical feature of this PCB is its input stage, which is designed to filter, protect, and stabilize the input power.

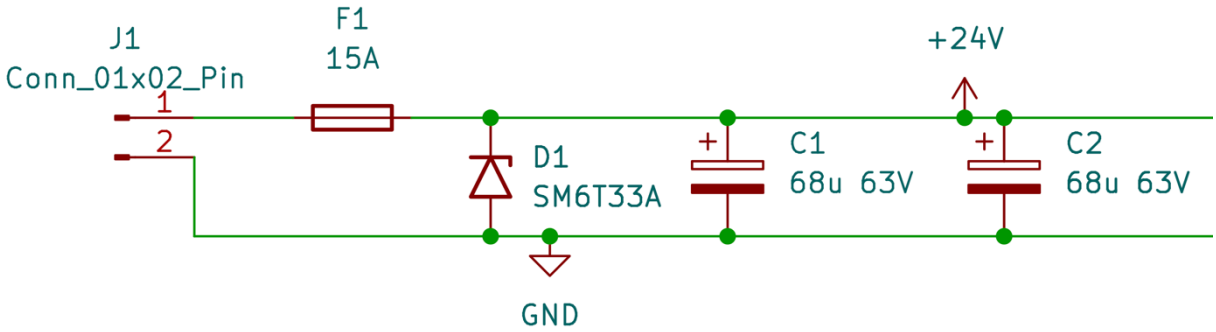

Figure 21: Input power filtering stage

This protection stage safeguards the rest of the system against voltage irregularities that could potentially damage electronic components. It includes:

- A fuse, which prevents overcurrent by physically disconnecting the circuit in case of excessive current draw.
- An SM6T33A TVS diode (Transient Voltage Suppression), which protects against overvoltage by clamping any voltage spike above its threshold and safely diverting it to ground.
- Two electrolytic capacitors connected in parallel, which help filter high-frequency noise and smooth out voltage ripple on the power supply line.

This same protection circuit will be replicated on each of the robot's PCBs to ensure individual protection and improve overall system reliability.

- Control_PCB

The Control_PCB is the main interface between the other PCBs. It embeds the RPi and the PTHat to spread the driving signal to the motors' driver, bring power to the cart and the Z1 motor and collects the signal from the three contact sensors and four optical sensors. I decided to locate it near the end of the horizontal arm to be as close as possible to the cart and the Z1 driver. This allows robust cable management near this location, and only straight wire to travel along the horizontal arm up to the rotation axis where the power and PUL/DIR signal will be transmitted via the slip ring. Same thing goes with the wire that will power the optical sensors and carry their output signals.

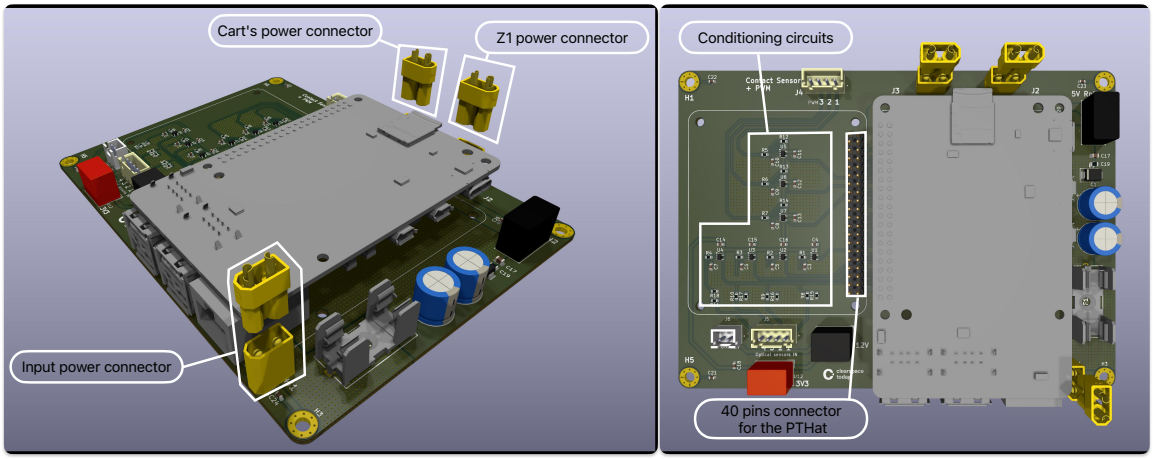

Figure 22: 3D preview of the Control PCB mounted with the RPi

On the schematic, on can notice that this PCB is divided into four parts:

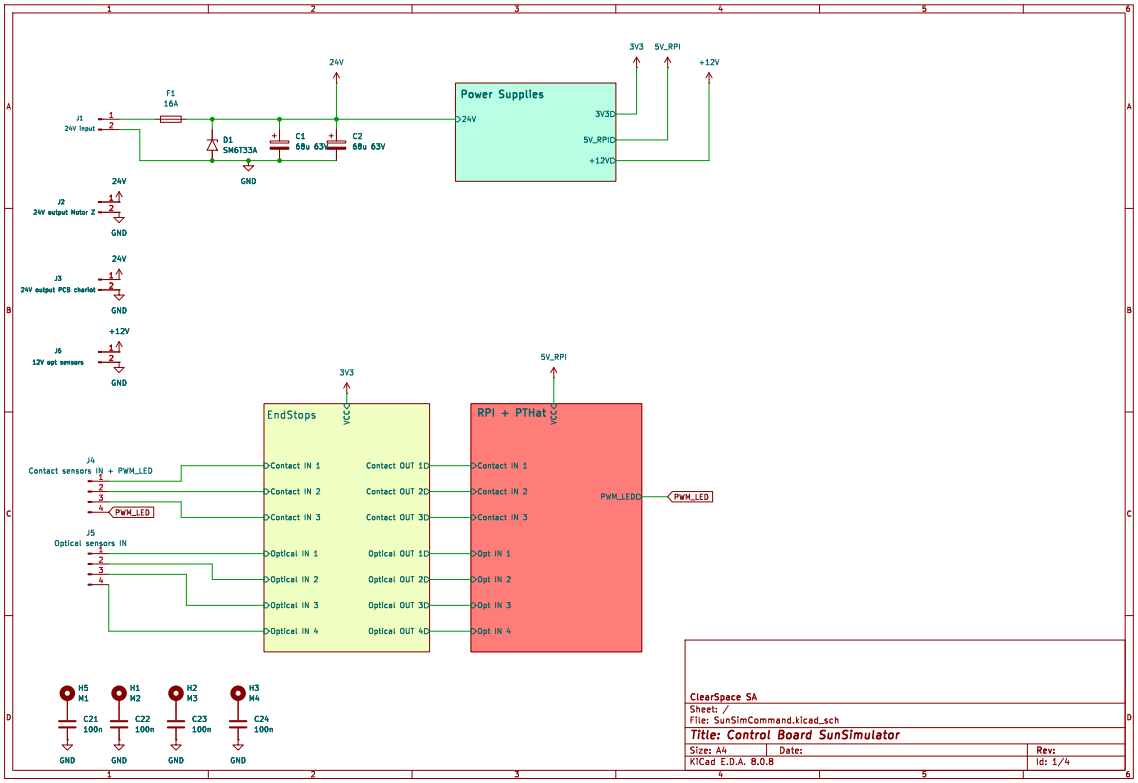

Figure 23: Schematic of the Control PCB

- *Power filtering stage*

As mentioned previously for the PWR_PCB, this section is dedicated to filtering the input power to ensure a safe, clean, and reliable power supply for the rest of the board

- *EndStops*

This section manages the signals from both the contact and optical sensors, using a dedicated Schmitt trigger circuit for each input.

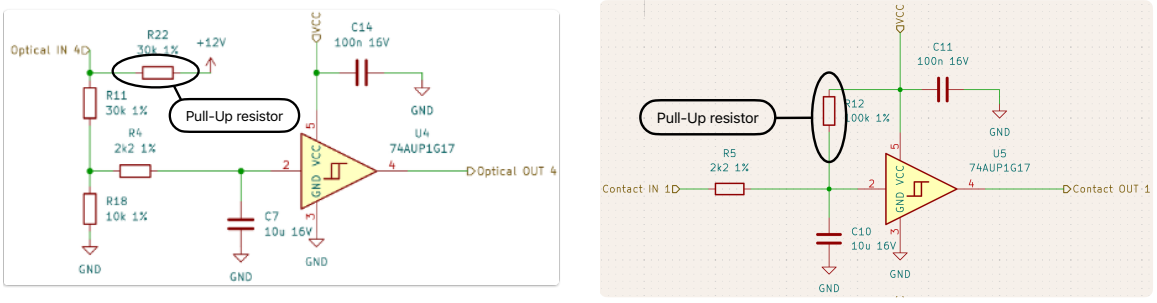

Figure 24: Schematic of the Schmitt trigger circuit

Schmitt triggers are used here as signal conditioning circuits to ensure clean and reliable signals at the Raspberry Pi input. The fundamental principle of a Schmitt trigger is that its output can only have two discrete states — HIGH or LOW — depending on the input voltage. It switches states only when the input crosses defined threshold levels, effectively introducing

hysteresis. This feature is particularly useful for cleaning up noisy signals that may occur during propagation from the sensors to the Raspberry Pi. Additionally, Schmitt triggers serve as a means of electrical isolation between the sensor stage and the Raspberry Pi's GPIO inputs.

For both contact and optical sensors, a pull-up resistor is required to ensure clearly defined logic states. In the case of contact sensors, pressing the sensor connects the input to ground, resulting in a LOW logic level; otherwise, the line would be left floating. To address this, a pull-up resistor is added so that when the sensor is open (not pressed), the input to the trigger is pulled up to 3.3V. When pressed, the voltage drops to 0V, and the Schmitt trigger toggles accordingly.

The same principle is applied to optical sensors, with the distinction that these operate at 12V. The pull-up resistor is placed between the sensor's VCC pin and its output. As a result, the sensor's output toggles between 0V and 12V. Since the input of the Schmitt trigger must not exceed 3.3V (to be compatible with the Raspberry Pi), a voltage divider is added to scale down the 12V signal to approximately 3V, ensuring safe operation.

- *Power Supplies*

This section handles the power conversion from the 24V main supply to the various voltage levels required across the board. It includes DC-DC converters generating 12V for the optical sensors, 3.3V for the Schmitt trigger circuits, and 5V for the Raspberry Pi. Each converter is a standard three-pin component (VIN, GND, VOUT) and is accompanied by input and output filtering capacitors to ensure voltage stability and reduce ripple.

- RPi+PTHat

This final block provides a straightforward interface between the PTHat and the Raspberry Pi, using two 40-pin connectors. A female connector is mounted on the PCB to host the Raspberry Pi vertically, while a male connector allows the PTHat to be connected on top. The GPIO signals are routed from one connector to the other, ensuring a direct and reliable connection between the two boards.

- Cart_PCB

The purpose of this PCB is to control the electronics located on the cart, which includes the R2 and R3 motors, the LED, and the cooling fans. The onboard electronics consist of the LED control circuit, two motor drivers, and multiple DC-DC conversion stages. Indeed, the PCB receives a 24V power supply from the main control PCB but the motors do not operate at 24V but at a lower voltage, between 5V and 10V, as discussed in section IV.a. This voltage had not been precisely determined during testing, especially since the tests were not conducted under fully realistic conditions and to accommodate this uncertainty, I chose to include several

DC-DC converters providing different voltage outputs, along with two rotary switches that allow the user to select the appropriate supply voltage for each motor individually.

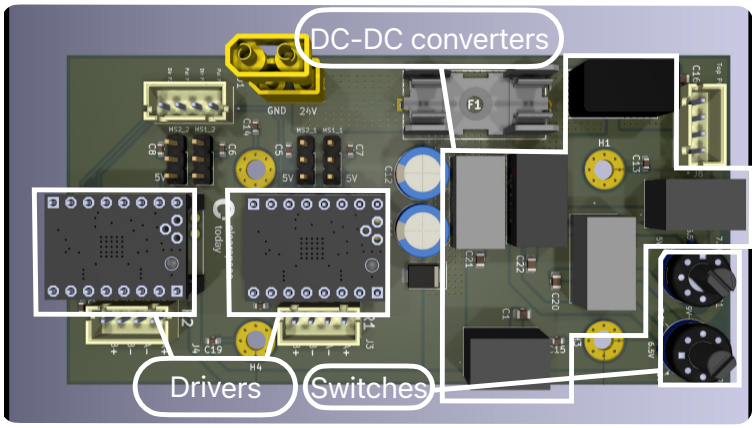

Figure 25: 3D preview of the cart's bottom board

The available voltages are 5V, 6.5V, 7.2V, and 9V, and each DC-DC converter was selected to be capable of supplying enough current—up to 500 mA per motor. The output of each switch is connected to the power input of one motor driver. A 4-pin connector is then used to connect the motor coils to the driver's phase outputs. Microstepping can be configured using jumpers, and the PUL and DIR control signals are injected from the main control PCB via another 4-pin connector. In order to plug the drivers directly into the PCB, two 8-pin female headers are used to allow a simple and secure connection.

Another DC-DC stage is required to power the LED and its control circuit. The LED itself is powered at 3.3V, while the control circuit operates at 12V. Since the LED can draw up to 3A, I selected a 12V–3A power supply to meet this requirement.

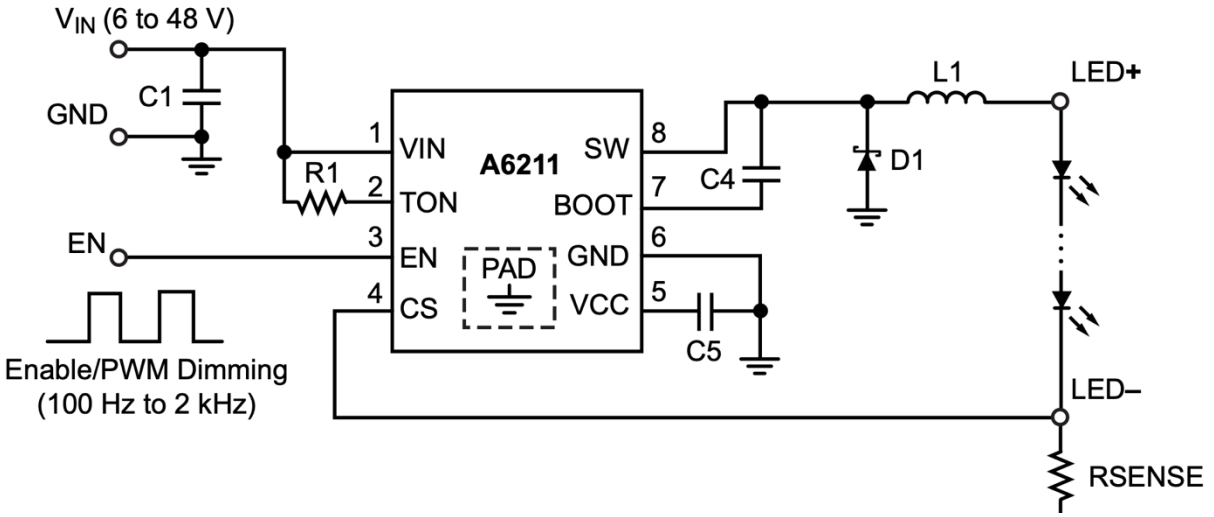

Figure 26: A6211 typical circuit

The LED control circuit is built around the A6211 driver. In theory, this driver can supply up to 3A if properly configured. The configuration is done by selecting component values according to the design guidelines provided in the datasheet. Most importantly, the R_{sense} resistor must be correctly chosen, as it directly sets the output current via the formula: $I = \frac{0.2}{R_{sense}}$.

To achieve a current of around 3A, a 75 mΩ resistor is used, which actually limits the current to approximately 2.6A, providing a safety margin for the LED. The other components are dimensioned to configure additional parameters such as the switching frequency of the internal MOSFET used to regulate the output voltage. The dimming (control over the brightness of the LED) can be performed in two ways : via a PWM signal or an analog voltage. The control is linear which means that the duty cycle of the PWM directly corresponds to the power of the LED whereas for the analog voltage, 0V leads to a maximum current and 5V corresponds to the maximum brightness. To switch between PWM or analog voltage dimming of the LED brightness, I added two jumpers on the PCB.

Finally, it remains the fans that are easily powered by the 5V DC-DC converter used for the motors as the current drawn by the fans is negligible and it does not require any control.

To address integration and space constraints, I chose to split the electronics into two separate PCBs stacked vertically. Combining all components on a single board would have resulted in a large, unbalanced layout that was not compatible with the available space on the cart. Stacking the boards allowed for a more compact and weight-centered design, placing the mass closer to the cart's center of gravity. Another specific mechanical constraints was that the cart was already equipped with four M3 mounting holes, which I integrated into the PCB layout to ensure secure attachment of the PCB on it.

Functionally, the two PCBs were separated according to their roles: the bottom board handles motor control, integrating the DC-DC converters, drivers, and motor connectors, while

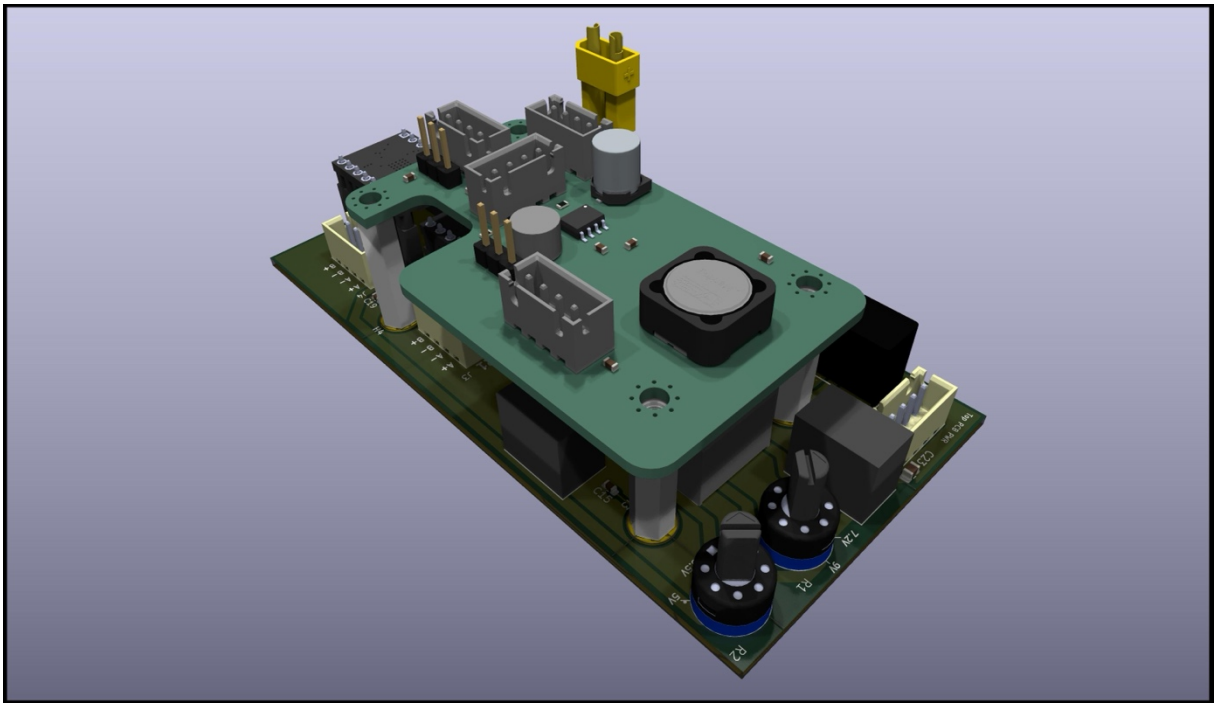

Figure 27: 3D preview of the two Cart-boards stacked

the top board manages signal routing from the contact sensors (mounted on the cart), powers the LED and fans, and handles analog LED control. The bottom board receives the main 24V power input and distributes power to the top board via a dedicated connector and cable carrying GND, 12V for the LED driver, and 5V for the fans and control signals.

Particular attention was paid to signal integrity when designing the wiring. Since the structure is made of carbon fiber, which is conductive, it can behave like an antenna and potentially interfere with the signals transmitted through the wires. To prevent such disturbances, shielded cables were used instead of standard unshielded ones, especially for signals running along the carbon fiber axis.

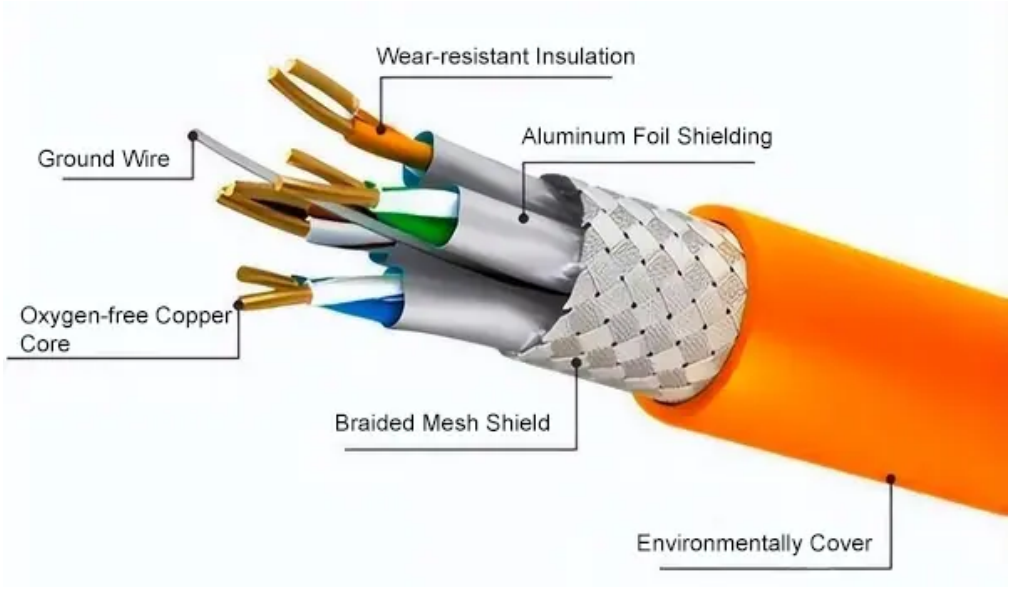

Figure 28: Illustration of a shielded cable

In a shielded cable, a conductive metallic layer is placed between the inner conductors and the outer insulating sheath. This shield is tied to ground, which helps protect the signal or power lines from electromagnetic interference by referencing nearby disturbances to a common ground potential. For areas not exposed to the carbon structure or electromagnetic noise, standard unshielded cables were used, as they are more flexible and easier to route—particularly in tight spaces.

For signal connectors, I selected the JST XH series connectors (BYB-XH, where Y indicates the number of pins). These are widely available, affordable, and offer good reliability for low-power applications. A major advantage is that they can be used with a universal crimping tool, avoiding the need to invest in expensive, brand-specific crimp tools. The JST XH connectors support currents up to 3A and accommodate wire sizes ranging from AWG30 to AWG22, making them versatile enough for various signal types, including PWM, PUL, and DIR signals.

For power transmission, I opted for the XT30 connectors. These gold-plated connectors are compact and robust, capable of handling currents up to 15A, which comfortably meets the power requirements of the motors, LED, and fans. Their secure connection and high current tolerance made them a reliable choice for the robot's power distribution.

Figure 29: B4B-JST connector (left) XT30 power connector (right)

c) Software development

As this was the first software iteration of the project, the expectations were quite basic. The main objective was to create a simple motion from point A to point B, without any constraints on speed or synchronization between the two robotic arms.

The first code I developed was the calibration program. Its purpose was to measure the number of pulses required for each motor to move the LED along its entire axis. For the R1 axis, this meant completing a full 360° rotation. For the Z1 axis, it involved traveling along the entire vertical range. For the R2 and R3 axes, it meant moving across the full range of angles permitted by the cart's mechanical constraints.
This calibration allows the determination of the resolution of each axis in either degrees per pulse (°/pulse) or centimeters per pulse (cm/pulse). With these values, precise motion becomes possible: sending a given number of pulses will move the robot by a defined angle or distance.

The program follows this procedure:

- Prompts the user to choose which axis to calibrate.
- Asks for the desired frequency.
- Asks for the number of pulses to send via the PTHat.
- Asks for the rotation direction.

The motor will then rotate until it hits one of the end-stop sensors. Upon detection, the program enters an interrupt routine that stops the pulse emission and sends a pulse count request to the PTHat. This provides the exact number of pulses emitted right before the stop command was triggered. For the R1 axis, the process is slightly different, since it can perform a full revolution. Four optical sensors are placed around the rotation to map out the full 360°. During calibration, the program retrieves the pulse count every time a sensor is crossed. This provides the pulse count for both a 90° rotation and a full 360° turn, which allows for cross-checking the accuracy of the method: the pulse count for 360° should roughly equal four times that of 90°.

Interrupts are handled using the *gpiozero* library, specifically the Button class. The interrupt setup follows this sequence:

- A Button object is created and linked to a specific Raspberry Pi GPIO. One object is created per sensor. A *bounce_time* is specified to prevent signal bouncing, which could trigger false multiple activations.

```
R1_opt1 = Button(5, bounce_time=0.05)
R1_opt2 = Button(6, bounce_time=0.05)
R1_opt3 = Button(13, bounce_time=0.05)
R1_opt4 = Button(19, bounce_time=0.05)
```

Figure 30: Section of the program creating the "Button" for interruption detection

- Each *Button* is linked to a callback function: when the GPIO detects a change (sensor activation), the program automatically jumps to this function and executes the interrupt routine.
- The interrupt routine is defined for each type of sensor:
    - For contact sensors, the routine stops the motion and sends a pulse count request to the PTHat.

- For optical sensors, the motion continues, but the pulse count is still logged when the sensor is crossed.

The homing program is a crucial step, as it establishes the robot's initial position before executing any trajectory. Because stepper motors do not provide inherent feedback about their position during movement, the only way to determine a known reference is by using the sensors. The principle of the homing strategy is to move each motor along its axis until a sensor is triggered, then set the corresponding coordinate (position or angle) to a defined reference value.

For the Z1, R2, and R3 axes, the homing process is relatively straightforward:

- The PTHat drives the motor until a contact sensor is activated.
- Once triggered, the motor stops, and the system sets the coordinate to either zero or to the initial point of the trajectory.
- The origin is defined as follows:
  - The middle of the vertical axis for Z1.
  - The neutral position (centered) for R2 and R3, based on mechanical limits.

For the R1 axis, the process is more complex because there are no physical end stops. Instead, four optical sensors are used around the rotation axis to determine orientation. Depending on the robot's initial position, a different sensor may be triggered first. Each sensor corresponds to a different angle, which determines the specific sequence of movements required to return the robot to its starting position.

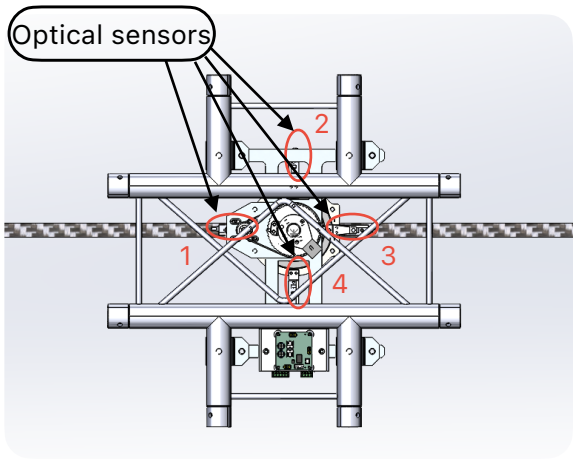

Figure 31: Location of the four optical sensors on the R1 axis

In the figure above, each sensor around the R1 axis has been assigned a number to help define logical mapping between sensor triggers and arm angles. The origin position is located between sensor 1 and sensor 4 as the position where the carbon arm is aligned parallel to the rail of the robotic arms. As such, the homing sequence must adapt depending on the robot's initial position when booting. For instance, if the robot powers up between sensors 2 and 3, the most efficient strategy is to move clockwise until sensor 3 is triggered, and then continue in the same direction by sending a precise number of pulses corresponding to the angular offset required to reach the origin. Conversely, if the robot starts between sensors 1 and 2, the closest path to the origin lies in the counterclockwise direction. In that case, the homing sequence would involve first triggering sensor 2, then rotating counterclockwise toward the initial position, as this path requires less angular displacement than a full clockwise turn. This

conditional logic allows the system to always select the shortest and safest trajectory to reach the origin, regardless of the motor's starting point. It also ensures consistent initialization for accurate motion control throughout the sequence.

The entire trajectory relies on a JSON file containing the positions of each robotic axis at different time intervals. The program begins by reading this file, identifying the relevant data associated with the SunSim position, and extracting it into a dictionary for easier access.

```
def read_json(json_path, time_points):
    with open(json_path, 'r', encoding='utf-8') as f:
        data = json.load(f)

    datas = {}
    for t in time_points:
        for point in data["serializable_data"]:
            if point["time"] == t:
                datas[t] = {
                    "z_sun_sim": point.get("z_sun_sim"),
                    "R1_sun_sim": point.get("R1_sun_sim"),
                    "R2_sun_sim": point.get("R2_sun_sim"),
                    "R3_sun_sim": point.get("R3_sun_sim"),
                }
```

Figure 32: Section of the code reading the JSON file

To read the JSON file, one just needs to give the absolute path to the file. Then the *read_json* function will look for the relevant value needed for the SunSim trajectory, store them inside a dictionary and return it so that it can be used in the next steps.

Once extracted, the program can compute the displacement that occurs between two timestamps, which is then converted into the number of pulses required to move each motor accordingly. In this implementation, I chose to use only the two extreme values (i.e., t = 0s and t = t_end) to define the movement. This decision was based on the observation that intermediate points in the file showed only minor displacement between consecutive timestamps, making the total movement more meaningful when considering only the initial and final positions.

```
def motor_values(datas, time):
    p_z= float (datas["z_displacement"]/z_res)
    p_R1 = float (datas["R1_displacement"]/R1_res)
    p_R2 = float (datas["R2_displacement"]/R2_res)
    p_R3 = float (datas["R3_displacement"]/R3_res)

    delay = 10
    f_z = float (p_z/delay)
    f_R1 = float (p_R1/delay)
    f_R2 = float (p_R2/delay)
    f_R3 = float (p_R3/delay)

    parameters = {
        "z_parameters":{"frequency":f_z, "pulse":p_z},
        "R1_parameters":{"frequency":f_R1, "pulse":p_R1},
        "R2_parameters":{"frequency":f_R2, "pulse":p_R2},
        "R3_parameters":{"frequency":f_R3, "pulse":p_R3},
        }
    return parameters
```

Figure 33: Converting the displacement into pulses for each axis

In this extract of the program, the dictionary generated by the *read_json* function has been sent to the *motor_value* function. This function returns a dictionary containing both the frequency and number of pulses for each axis to perfom the displacement under a duration of delay seconds. One can notice that the resolution constant (z_Res, R1_res…) are here used to compute the number of pulse equivalent to the displacement to perform on each axis.

```
PWM_dim = LED(12) #assign the signal to one of the GPIO

def PWM(frequency, duty_cycle):
    f = frequency
    T = float(1 / f)
    t_on = duty_cycle * T
    t_off = T - t_on

    while True:
        PWM_dim.on()
        sleep(t_on)
        PWM_dim.off()
        sleep(t_off)

frequency = 1000
duty_cycle = 0.3
#create a separate thread
PWM_thread = threading.Thread(target=PWM, args=(frequency, duty_cycle))
PWM_thread.daemon = True
```

Figure 34: Generating the PWM signal via multithreading

This section of the program is dedicated to generating a PWM signal that controls the LED intensity. Since the *gpiozero* library does not offer a direct method to produce a custom PWM signal on arbitrary GPIO pins, I leveraged the "*LED*" object provided by the same library. This object allows toggling a GPIO pin between HIGH and LOW states, simulating a PWM output when combined with controlled timing. To maintain the HIGH or LOW state for a specific duration, I used the *sleep* function from the *time* library. However, this function pauses the entire program, which would interfere with the main execution if used directly. To avoid blocking the rest of the code, I implemented multithreading: the PWM signal generation runs in a separate thread, enabling it to operate independently from the main program flow. This ensures that the LED modulation occurs in parallel with the robot's trajectory execution, without interrupting or delaying other processes.

d) System testing and troubleshooting

The next phase involved assembling and testing the PCBs individually. Out of the four routed PCBs, three were fully functional. The only board that encountered issues was the one embedding the LED driving circuit. Despite the fact that the schematic had been strictly followed—both on the PCB layout and during initial breadboard testing—the current supplied by the LED driver never exceeded a few milliamps. The issue did not stem from routing or component placement but appeared to be related to incompatibility or limitations of the driver itself in the PCB context. To solve this, we decided to switch to a different component. Several alternatives exist for this application, and we selected one that came with an evaluation board to simplify testing. These boards, provided by the manufacturer, offer ideal conditions to test components with proper routing, grounding, and thermal dissipation.

We switched to the MP24833-AGN-P driver, which offers similar functionality to the A6211 but with a different approach to ground distribution. Like the A6211, it uses a sense resistor to regulate output current and supports both PWM and analog dimming. Testing with the evaluation board showed that the LED control was both precise and stable. The board came preinstalled with two 400 mΩ resistors in parallel, resulting in a total sense resistance of

200 mΩ and a corresponding current of 1 A. To match the previous setup, I replaced those with two 150 mΩ resistors in parallel (yielding 75 mΩ), and successfully reached a current of up to 2 A. Although this configuration should have allowed up to 2.6 A, I assumed that parasitic resistance—possibly due to the solder joints or trace impedance—caused a slight drop in performance. At such low resistance values, even minor imperfections in the PCB layout can significantly impact the output current.

Given the successful testing with this new driver, I designed a revised version of the LED driver PCB, applying the routing guidelines recommended in the evaluation board's datasheet, particularly regarding component placement, ground plane distribution, and thermal dissipation. I also ordered a range of sense resistors (from 20 mΩ to 75 mΩ) to explore the full current range supported by the new driver in future tests.

Once this revise version assembled, I managed to control the LED luminosity with a PWM generated by the RPi.

Apart from the LED driver board, the Control PCB delivered conclusive and reliable results during initial testing. It successfully powered the Raspberry Pi as expected, detected sensor inputs through its GPIOs, and provided the necessary 3.3V and 12V supply for the optical and contact sensors using its DC-DC converters. To verify remote access capabilities, I tested the SSH connection to the Raspberry Pi. The connection was successfully established, allowing full remote programming and control of the Raspberry Pi and the PTHat from an external computer. To establish such a connection, the following steps were followed:

- Retrieve the Raspberry Pi's IP address using a monitor or a network scanning tool.
- From a computer connected to the same Wi-Fi network, use the command: *ssh username@ip.address*

This prompts for the Raspberry Pi's password. Once logged in, the Raspberry Pi's file system becomes accessible from the computer. It is then possible to edit, run, or upload scripts, such as the JSON trajectory files used for robot control.

The bottom PCB of the cart, which handles the power distribution and the R2 and R3 stepper drivers, also functioned correctly during bench testing. Power was evenly distributed across the board, and both R2 and R3 motors responded as expected. One key outcome of this test involved validating the motor torque under different supply voltages. Using the rotary switches integrated into the PCB, I was able to easily toggle between the multiple DC-DC outputs (5V, 6.5V, 7.2V, 9V). As anticipated, a 5V supply was insufficient to generate enough torque for the R2 axis to rotate the LED arm smoothly across its full range. Switching to the 6.5V supply resolved the issue and enabled consistent and smooth motion.

The validation of the PWR_PCB was straightforward as the only point to validate was that it would convert properly the 24V into 5V for the OPTO signal, and reroute the drive signals of the R1 axis.

Once all the individual PCBs had been validated, the next phase consisted of integrating the entire system onto the robot's mechanical structure and running tests in real-world conditions. Therefore, 3D enclosures were designed to install the *Control_PCB* and the *PWR_PCB* on the structure. The result of the installation is given in the following pictures:

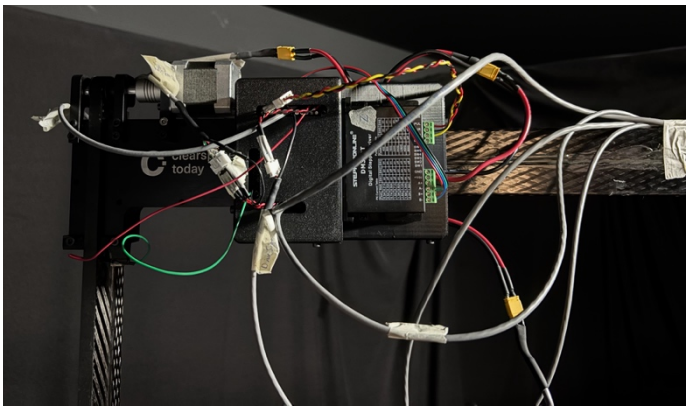

Figure 37: Control PCB in its enclosure mounted on the robot

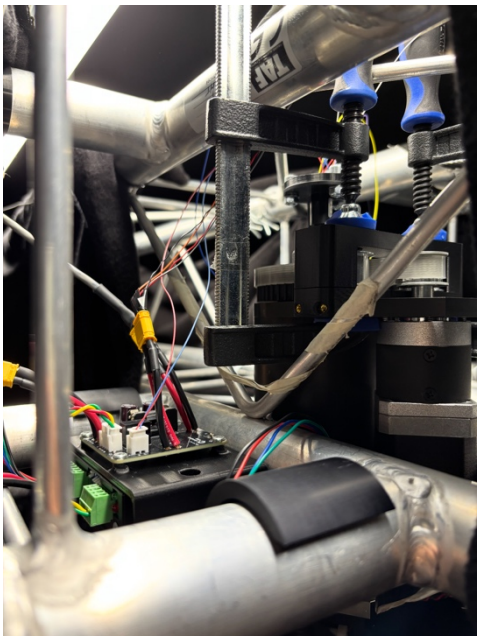

Figure 38: PWR_PCB in its enclosure mounted on the structure

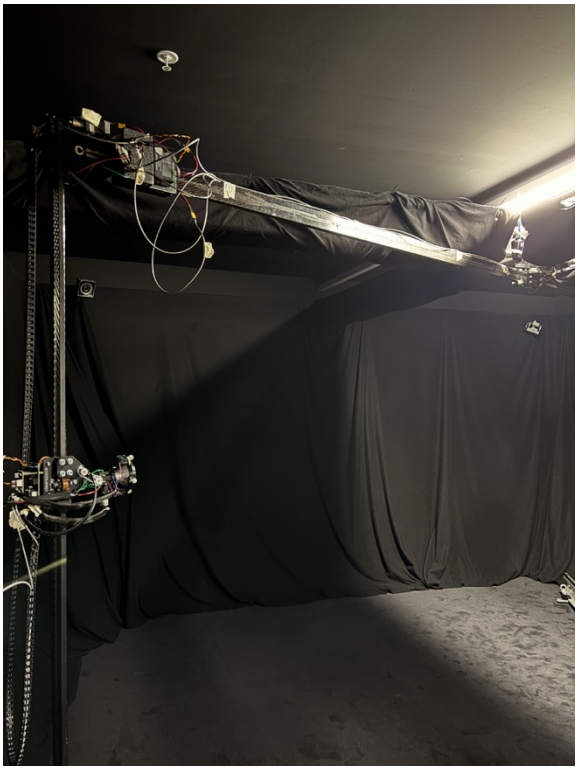

Figure 36: Overall view of the robot equipped with the electronic

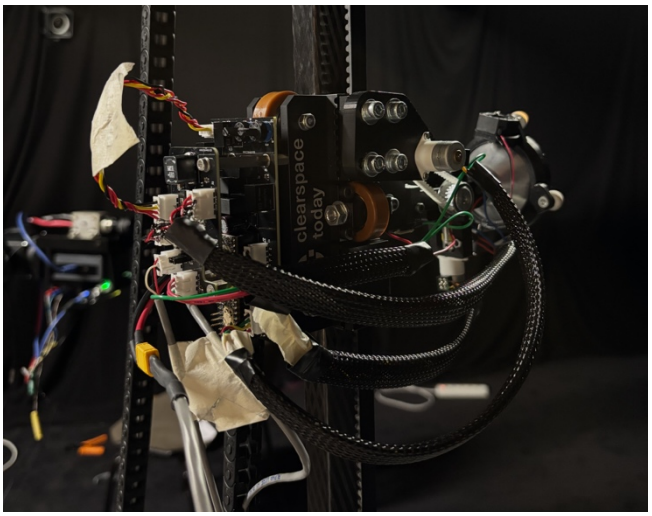

Figure 35:Cart PCBs mounted with wiring

During installation, it became clear that cable congestion remained an issue, even with careful planning. This was especially true for the Control_PCB, which serves as the main interface with the other boards, requiring numerous connections—including motor driving signals, power lines for both the motors and the cart, and optical sensor inputs. On the cart side, the similar issue occured, but the use of braided sleeves helped achieve a cleaner integration with significantly less visible cabling.

The following schematic summarizes the system architecture:

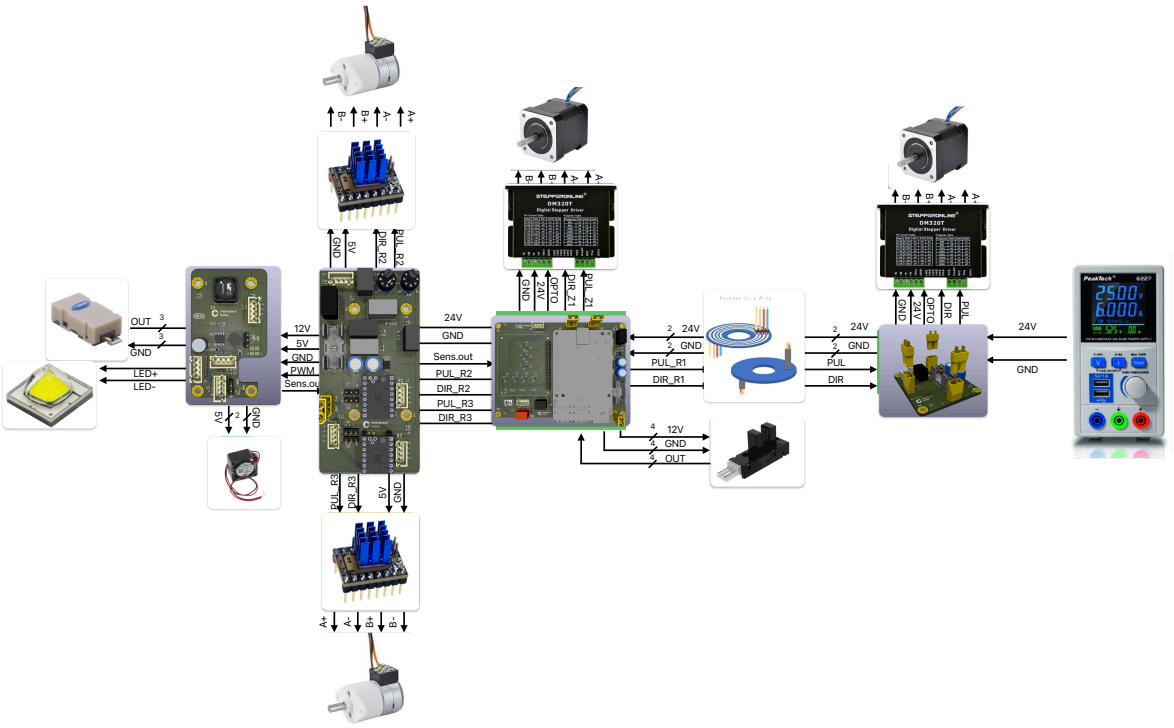

Figure 39: Overall System architecture

At the initial boot-up of the system, a few key indicators helped confirm that the setup was functioning correctly. First, checking whether each motor remained locked (i.e., held in position) by its driver confirmed that the motors were properly powered. Additionally, the immediate startup of the fans and the activity of the Raspberry Pi's (RPi) status LED confirmed that the RPi was successfully booting.

Once these technical checks were complete, the first programming tests could begin, starting with the calibration routine. As expected, when a motion was triggered, pressing an endstop sensor would stop the motion (or not depending on the axis) and return the number of pulses. This provided the pulse count for each axis. However, at the time of writing this report, the actual angular range of the R2 and R3 axes was not yet known. Further testing is required to determine these ranges accurately and thereby calculate the resolution of each axis.

Despite this, initial approximations enabled the validation of a first iteration of the SunSim system, albeit with a few trade-offs.

The validated aspects include:

- The ability to perform simultaneous motion across all four axes.
- Smooth and repeatable movements, confirming the reliability of both the program and the electronics. Even though trajectory extracted from the JSON file is not yet possible, one could consider that the repeatability of a motion from one point to

another represent, in a way already a trajectory, bar the fact that it does not originate from the JSON
- Effective control of the LED via PWM on the RPi, allowing for a wide range of brightness settings, which supports flexible lighting scenarios.

Several limitations were however identified during testing:
- Without an accurate determination of angular ranges, the homing sequence-and by extension, the trajectory planning-lacks precision. As a result, realistic trajectories are not yet achievable.
- Although the LED can be dimmed using PWM, attempts to control it via analog voltage were unsuccessful. A voltage divider using a potentiometer was implemented on the PCB, but the measured voltage output did not match the expected values.
- A significant issue emerged regarding the RPi power supply. Initially, the RPi was powered through the Control PCB using an onboard 24V-to-5V DC-DC converter, with power supplied via the RPi's GPIO pins. However, after several months of test, the RPi eventually crashed. It was later discovered that powering the RPi through the GPIO bypasses the voltage protection circuitry present on the USB-C input. After discussing with ClearSpace electronics engineers, it was agreed that RPis are relatively fragile in terms of power management, and for that reason, they should be considered more as consumables than as robust computing units. As a future improvement, it was proposed to integrate a power delivery board-similar to a PTHat-that would deliver 5V via the USB-C port, offering safer and more reliable power regulation from the main 24V line.

Despite a few technical compromises, the designed and tested system shows strong potential. It safely powers all four motors, supports reliable and coordinated movement, and offers precise LED brightness control. Furthermore, the identified issues appear solvable in the short term, paving the way for more advanced use cases and experimentation with the SunSim in the near future.

V. Project handover and sustainability

This section outlines a non-exhaustive list of possible future improvements to enhance the long-term viability and capabilities of the SunSim robot.

Obviously, the above mentioned trade-off are part of the next major upgrades that this project would benefit of, and the first major update would be to determine the right resolution of each axis which would allow for motion with the same logic that the two robotic arms follow.

Then the most significant opportunity lies in further developing the software. The current implementation enables basic linear displacement between two orbital points but does not support dynamic trajectory adjustments such as real-time changes in speed or direction. As a result, the complexity and realism of simulated trajectories are currently limited. The system is functional for taking snapshots at relatively long intervals, but to support more advanced use cases—such as synchronized operation with the dual robotic arm setup—future development should allow seamless image capture throughout the motion. This would enable more accurate and representative data collection for validating GNC algorithms. Fortunately, the PTHat's API supports real-time control and trajectory modulation, meaning these upgrades could be implemented with a few additional weeks of development.

Another improvement concerns the LED module. While the current LED is quite powerful for its size, using a brighter LED would enable more versatile test scenarios, particularly in high-contrast lighting simulations. However, this would likely require redesigning the cart's driver and upgrading to a more robust DC-DC converter capable of delivering higher current.

Once a wide range of motion scenarios have been tested, a second iteration of the cart's bottom PCB could be considered. The current board includes multiple DC-DC converters to offer flexibility, but test results suggest that a single 6.5V converter performs well across all required movements. Removing the unnecessary converters would reduce both the physical footprint and the cost of the PCB—particularly valuable given that DC-DC converters are among the most expensive components on the board.

Further improvements to the cart could address thermal management. Due to the compact form factor of the R2 and R3 motors, prolonged use leads to noticeable heating, which could shorten their lifespan. To mitigate this, passive heat sinks or small active cooling elements like fans—similar to those already used for the LED—could be integrated into the system.

Finally, adding a status LED directly onto the PCB would provide a quick visual indication of the cart's power state. At present, only the fans indicate whether the system is powered, which is not an ideal or reliable solution.

Beyond the work I directly focused on, a valuable improvement for the DarkRoom setup would be to integrate the OptiTrack motion tracking system. Although this system has already been physically installed on the structure, it has not yet been implemented. Once active, it would enable real-time tracking of both the Sun (LED) movement and the relative displacement of the servicer and target. As previously mentioned, apart from the SunSim endstop sensors, the robot currently lacks a feedback system to determine the real-time position of the LED—aside from the limited pulse count query the PTHat can give which only offers an approximate location. By using reflective markers and infrared cameras, the OptiTrack system could provide accurate, real-time position data. This would allow for the implementation of a feedback control loop, ensuring that the commands sent from the Raspberry Pi to the motors are correctly executed. The same benefit could apply to the two robotic arms: while they already include integrated position sensors, OptiTrack could serve as a redundant system to cross-verify motion accuracy, thereby improving overall reliability.

From a long-term perspective, one key area of improvement would be upgrading the motors currently used in the system. While the present motors are sufficient for most trajectory scenarios, they begin to show limitations when motion speed increases. Specifically, during high-speed movements, the motors often lack the necessary torque to decelerate abruptly and stop precisely, leading to missed steps. This issue is especially noticeable on the Z1 and R1 axes, as discussed in section IV.d. The large size and mass of the carbon arms generate significant inertia during fast motion. As a result, when the motors attempt to stop, the system behaves almost like it is "skidding" due to the excessive momentum—causing a loss of positional accuracy. To address this, more powerful motors capable of handling higher deceleration forces would be needed. These would provide better control during abrupt stops and enable smoother, more reliable operation at higher speeds. Such an upgrade would unlock the full potential of the two robotic arms, which are theoretically capable of executing much faster and more precise motions than what is currently achievable with the existing motors.

Conclusion

At the conclusion of this internship, I believe I successfully delivered a solid first iteration of the SunSim's electronic system. The main goal of the project was to animate a high-intensity LED in accordance with a trajectory extracted from a JSON file. Despite a few remaining limitations, the developed system largely meets the functional expectations initially set out. It allows for the coordinated control of four stepper motors, accurate homing using endstop sensors, and precise motion execution along the defined trajectory. Throughout the development, several technical challenges and unexpected issues significantly slowed down progress. However, overcoming these obstacles proved to be extremely valuable from a learning perspective. As a future engineer, this internship gave me the opportunity to deepen my understanding of system architecture, embedded electronics, and software development, while also training my critical thinking and problem-solving abilities. I had the chance to carry the project end-to-end — from the early design phase to the integration and functional testing. This included defining the electronics and communication interfaces, designing and assembling the PCBs, setting up the system on the physical robot structure, and ensuring the integration of all components. This holistic involvement gave me ownership over the entire system and allowed me to put into practice the skills I've developed throughout my academic journey. While the current prototype can already support interesting experimental use cases, especially in the context of Sun-satellite interaction simulation, there is clear potential for further improvement. With continued refinement — particularly in motor control, software architecture, and synchronization with the robotic arms — the SunSim could become a powerful test platform for ClearSpace's guidance and navigation development. I am confident that the work carried out during this internship lays down a solid technical foundation for future upgrades and experimentation.

Abstract (french version)

Dans le cadre de mon projet de fin d'études en filière Nanotech à Grenoble-INP Phelma, j'ai effectué un stage de six mois au sein de ClearSpace, une startup suisse œuvrant pour une industrie spatiale durable en collaboration avec l'Agence Spatiale Européenne (ESA). L'objectif principal de ClearSpace est le développement de missions de désorbitage de débris en orbite basse, afin de limiter les risques pour les missions spatiales présentes et futures. Durant ce stage, j'ai participé au développement d'un simulateur de trajectoire solaire destiné à la salle d'expérimentation « DarkRoom », dans le cadre du système GNC (Guidance, Navigation and Control). Ce simulateur vise à reproduire les conditions d'éclairage spatiales pour tester et calibrer les capteurs optiques utilisés lors des phases d'approche de satellites de capture. Mon travail a porté sur la conception complète de l'électronique et de l'informatique embarquée pilotant la source lumineuse mobile. J'ai conçu et réalisé les circuits électroniques, sélectionné les composants, routé les cartes sur KICAD, procédé à l'assemblage (soudure, câblage, intégration mécanique) et développé les programmes de pilotage et de calibration sur Raspberry Pi. L'ensemble permet de contrôler précisément la position d'une LED via quatre moteurs pas-à-pas, selon des trajectoires importées en JSON. Ce projet m'a permis de consolider mes compétences en électronique embarquée et en intégration système. La solution livrée constitue une base fonctionnelle pour les futures expérimentations de ClearSpace dans la DarkRoom, permettant dès à présent des tests réalistes de guidage orbital.

Abstract (english version)

As part of my final-year project in the Nanotech program at Grenoble-INP Phelma, I completed a six-month internship at ClearSpace, a Swiss aerospace startup working with the European Space Agency (ESA) to promote a more sustainable space industry. These efforts focus on the deorbiting of space debris in low Earth orbit, which poses a significant risk to current and future missions. During this internship, I contributed to the development of a solar trajectory simulator, part of ClearSpace's Guidance, Navigation, and Control (GNC) system. This simulator is used in the "DarkRoom" test facility to replicate complex space illumination conditions, enabling realistic testing of visual navigation systems involved in satellite-target approach phases. My responsibilities included designing the electronics and embedded software for controlling a robotic system that positions an LED to simulate sunlight. This involved developing schematics (power conversion, signal conditioning, LED control), routing PCBs in KiCAD, selecting and assembling components, and programming a Raspberry Pi interfaced with a PTHat motor controller. The system interprets trajectory data from JSON files to drive four stepper motors. I implemented calibration routines, position initialization, and motion control software, with remote operation via SSH. This project allowed me to manage a full electronics development cycle and deepen my skills in embedded systems integration. The delivered prototype enables precise and realistic solar positioning, providing a solid foundation for future GNC testing in the DarkRoom.

Abstract (italian version)

Nel contesto del mio progetto di fine studi nella specializzazione Nanotech presso Grenoble-INP Phelma, ho svolto un tirocinio di sei mesi presso ClearSpace, startup svizzera del settore aerospaziale che collabora con l'Agenzia Spaziale Europea (ESA) per promuovere un'industria spaziale più sostenibile. Le missioni di ClearSpace sono dedicate alla deorbitazione dei detriti in orbita bassa, per ridurre i rischi legati alle missioni spaziali presenti

e future. Durante il mio tirocinio, ho partecipato allo sviluppo di un simulatore di traiettoria solare, utilizzato nella “DarkRoom” — un ambiente sperimentale oscurato — per riprodurre le complesse condizioni di illuminazione spaziale. Questo simulatore rientra nello sviluppo del sistema GNC (Guidance, Navigation and Control), impiegato durante la fase di avvicinamento di un satellite di cattura verso un bersaglio. Mi sono occupato della progettazione e realizzazione dell’elettronica e del software necessari per controllare una sorgente luminosa mobile basata su LED, guidata da quattro motori passo-passo. Il sistema è gestito tramite una Raspberry Pi e un modulo PTHat, in grado di interpretare dati di traiettoria in formato JSON e convertirli in comandi motore. Il lavoro ha incluso la progettazione degli schemi elettrici, il layout dei circuiti con KiCAD, la scelta e l’assemblaggio dei componenti, oltre allo sviluppo dei programmi per la calibrazione degli assi, l’inizializzazione delle posizioni e l’esecuzione dei movimenti. Questo progetto mi ha permesso di acquisire una solida esperienza nello sviluppo completo di sistemi elettronici embedded. La soluzione finale, completamente operativa, consente movimenti precisi della sorgente luminosa, costituendo una base affidabile per i futuri test del sistema GNC all’interno della DarkRoom.

Internship summary sheet

Nicolas GAUTIER-KUEA

Master MNIS
2023-2025

ClearSpace SA
Rue de Lausanne, 64, 1020 Renens, Switzerland

Design and development of the electronics for a Solar Trajectory Simulator

From 03/02/25 to 31/07/25

Under the supervision of :

- **Company supervisor : Simon BOTHNER, simon.bothner@clearspace.today**
- **Phelma Tutor : Jonathan MIQUEL, jonathan.miquel@grenoble-inp.fr**

Confidentiality : ☐ yes ■ no

Ecole nationale supérieure de physique, électronique, matériaux

Phelma
Bât. Grenoble INP - Minatec
3 Parvis Louis Néel - CS 50257
F-38016 Grenoble Cedex 01

Tél +33 (0)4 56 52 91 00
Fax +33 (0)4 56 52 91 03

http://phelma.grenoble-inp.fr

Internship overview: The main focus of this internship was the electronic and PCB design of a robot embedding motors and sensors. The responsibilities included the design, sourcing and testing of the solution powering and controlling the robot. The work was done in order to be integrated onto an existing mechanical structure, the electronic aiming to make it operational. The tasks also included firmware development for the low-level control of the motor and sensors.

Technical resources used:

- KiCAD
- Mouser/PCBWAY/Digikey suppliers
- Oscilloscope
- Power supplies
- Soldering station (tweezers, soldering iron, tin paste, fume extractor…)
- Raspberry Pi
- UltraLibrarian/Octopart/SnapEDA/ComponentSearchEngine
- Stepper Motor Drivers