Instituto Superior de Engenharia de Lisboa – ISEL
BSc in Electronics, Telecommunications and Computers Engineering

Final Year Project – P26

Antenna Rotor for Satellite Reception

Diogo Maurício nº 49372

Miguel Fonseca nº 49341

BSc. in Electronics, Telecommunications and Computers Engineering

Supervisors: Prof. João Casaleiro

Prof. Tiago Oliveira

2025

We declare that this document is our own work and complies with the ISEL Code of Conduct and Good Practices.

ISEL

Abstract

Our project is based on the use of two BLDC (Brushless DC) motors with Moteus r4.11 controllers, functioning as a rotor system for antenna positioning. The positioning commands are generated in real-time by the application 'Gpredict', which calculates the required elevation and azimuth coordinates for satellite tracking. These data are sent via TCP/IP using the Hamlib library.

However, Moteus controllers are not directly supported by Hamlib (unlike other commercially available controllers).

Thus, the main goal of our project was to create a bridge between the data sent by Hamlib and the Moteus rotors, in order to correctly interpret the received coordinates and apply the corresponding movements. This system allows the rotors to react accurately and synchronously to the positions calculated by Gpredict, ensuring efficient real-time satellite tracking.

Keywords

Rotor, Antennas, Satellites, Tracking, LEO.

Index

List of Figures

List of Acronyms

BLDC	*Brushless Direct Current* (Motor de Corrente Contínua sem Escovas)
FOC	*Field Oriented Control* (Controlo Orientado pelo Campo)
Gpredict	GNOME Predict (*Software* de rastreamento de satélites)
Hamlib	*Ham Radio Control Libraries* (Bib. p/ controlo de rádios e rotores)
TCP	*Transmission Control Protocol*
IP	*Internet Protocol*
CAN-FD	*Controller Area Network with Flexible Data rate*
EME	*Earth-Moon-Earth*
DIY	*Do It Yourself*
TLE	*Two-Line Element*
LEO	*Low Earth Orbit*
MEO	*Medium Earth Orbit*
GEO	*Geostacionary Equatorial Orbit*

Chapter 1

Introduction

In this chapter, we present an overview of our project as well as the motivation and objectives behind its development. In addition, the structure of the report will also be presented.

1.1. General Overview

Satellite tracking systems are essential for various applications such as telecommunications, Earth observation, and space exploration. To ensure efficient communication with non-geostationary satellites, antennas must remain constantly aligned with the satellites by dynamically adjusting the azimuth and elevation angles. This project aims to develop a control system for an antenna rotor capable of automatically tracking a satellite in real time.

The problem under study involves converting the satellite's real-time positional data—specifically its azimuth and elevation—into valid commands for rotors controlled by the *Moteus R4.11* drivers and controllers **[1][2][3]**. Two motors are used to manage both elevation and azimuth, and they must instantly acquire and update their positions according to the data received from *Gpredict* **[4]**, a software that calculates satellite positions in real time based on their orbits and date/time.

Communication between *Gpredict* and the rotor will be performed using the *Hamlib* library [5], which provides standard commands for retrieving azimuth and elevation angles. A Python script reads these values and sends the corresponding instructions to the rotors.

For this project, we acquired two motors: one for azimuth (horizontal coordinates) and another for elevation (vertical coordinates). It is important to note that both azimuth and elevation depend on the satellite's position and the geographical location of the rotor.

.

1.2. Motivation and Objectives

Real-time satellite reception and tracking is an area of growing interest—not only in professional contexts such as telecommunications, meteorology, and scientific research—but also in academic environments and among amateur radio enthusiasts. With the increasing number of satellites in orbit, particularly within LEO (Low Earth Orbit) constellations, the development of systems capable of accurately following their trajectories has become essential.

Commercial antenna rotor systems, although efficient, often rely on brushed motors and gear reduction mechanisms. In contrast, brushless motors are already commonly used in telescope rotors due to their greater efficiency and precision.

This project was motivated by the desire to develop an automatic satellite tracking system that leverages the capabilities of motor controllers like the *Moteus R4.11*, CAN-FD communication, and the open-source tracking software *Gpredict*. The primary goal is to create a complete and functional automatic antenna control system.

2 Report Structure

This report is structured into five chapters. The first chapter introduces the topic, providing an overview of the project, the motivations behind it, and the objectives to be achieved. The second chapter outlines the state of the art, analyzing related works and discussing the advantages and limitations of other solutions, as well as providing a technical explanation of how *Gpredict* and the *Moteus* motors work.

The third chapter focuses on system development, detailing the practical implementation—from receiving data from *Gpredict* to controlling the *MJBots* motors, including the logic for data conversion and communication between components. The fourth chapter addresses the testing scenarios, and the results obtained, demonstrating the system’s functionality and analyzing its performance. Finally, the fifth chapter presents the conclusions of the work carried out, reflecting on the completed stages, evaluating the results, and suggesting possible improvements and future developments.

Capítulo 2

State of the Art

This chapter presents an analysis of related work on satellite tracking systems, technologies used in antenna rotor control, and the main technical concepts that underpin the development of the project.

2.1. State of the Art

In the state of the art, there are commercial solutions for satellite tracking; however, many are costly or do not offer flexibility for customization. Other alternatives include the use of controllers such as the *Yaesu GS-232*, which allow automatic tracking but use older and less adaptable protocols. This project proposes an innovative approach by integrating Moteus motors, which provide greater precision and advanced control, with an accessible and customizable system.

2.1.1. Commercial Solutions

2.1.1.1. Yaesu G-5500

There are several commercial antenna rotors on the market with built-in control systems. One of the most well-known is the *Yaesu G-5500* **[6]**, which allows control of the azimuth and elevation axes and is widely used by amateur radio operators. This system is compatible with external controllers such as the GS-232, which enables interfacing with software like *Gpredict*. However, it has some limitations:

- High acquisition cost;
- Mechanical gears that, over time, accumulate backlash, reducing tracking precision and increasing maintenance costs.

2.1.1.2. SPID Rotators

The SPID Rotators (and their AlfaSpid version) are high-capacity antenna rotors designed to control the orientation of antennas on both the azimuth and elevation axes, making them ideal for satellite tracking, astronomical observation, weather stations, among others.
The best-known models are the SPID RAS, which is a typical azimuth-elevation rotor for satellite tracking, offering a good balance between cost and robustness. There is also the SPID BIG-RAK model, which is mainly used for large parabolic antennas and is widely used in EME (Earth-Moon-Earth) and long-range communications.

2.1.2. *Open Source* and DIY Solutions

Alongside commercial systems, various DIY (Do It Yourself) solutions have emerged, which are very popular in academic works and amateur radio communities. These solutions often use:

- Stepper Motors with drivers such as the A4988 or TMC2209;
- Microcontrollers like Arduino or ESP32;
- Controllers based on relays and limit switches;

Although these solutions have the advantage of being economical and customizable, they present considerable limitations in terms of

- Angular precision;
- Mechanical durability;
- Response speed;
- Calibration complexity.

Our project positions itself between the best of both worlds: precision and robustness comparable to commercial systems, combined with the flexibility and customization of open-source systems.

2.1.3. Adopted Solution

The choice to use *Moteus R4.11* controllers with BLDC motors and CAN-FD communication represents an innovative approach in this type of application. Originally designed for high-performance robotics, the *Moteus* controllers offer:

- Real-time FOC vectorial control;
- Sub-degree precision in position reading (via magnetic absolute encoder);
- Smooth and fast response to commands;
- Communication via CAN-FD, enabling scalability and robustness in data transmission.

Combined with the use of Gpredict as tracking software and the *Hamlib* library for standardized communication, this project offers a robust and economically viable solution, allowing precise and automatic control of an antenna based on real-time orbital calculations of satellites.

2.2. Satellite Tracking Software Gpredict

Gpredict is a real-time satellite tracking open-source application developed for the GNOME environment. It uses graphical elements based on the GTK+ library and offers a robust and extensible platform for tracking satellites in Earth orbit, with a special focus on amateur radio, telecommunications, education, and scientific research applications.

Gpredict's operation is based on orbital TLE data (Two-Line Element sets) obtained from sources such as Celestrak or NORAD, which allow highly accurate prediction of the trajectories of satellites in low Earth orbit (LEO), medium Earth orbit (MEO), or geostationary orbit (GEO).

Based on this data, Gpredict calculates in real-time the relevant orbital parameters for each selected satellite, including:

- Azimuth and Elevation (to move the rotors and steer the antenna);
- Distance to the observation point;
- Relative Velocity;
- Time for the next pass;
- Doppler shift (for radio frequency correction).

Gpredict's modular architecture allows the creation of multiple windows (modules), where each module can track different groups of satellites, offering graphical visualizations (orbital maps, elevation/time graphs) and tables with real-time parameters. This modularity is particularly useful in environments that require simultaneous monitoring of multiple satellites.

One of the most relevant features for this project is Gpredict's ability to integrate with external hardware, such as antenna rotors and full-duplex radios, through the Hamlib library (Ham Radio Control Libraries). For rotor control, Gpredict communicates with the daemon ***rotctld***, which listens on a TCP port (default 4533) and accepts standardized commands to report and set the rotor position. The protocol is simple and text-based, facilitating integration with scripts and custom applications (like ours in Python).

Gpredict can be configured to send azimuth and elevation commands in real-time, dynamically adjusting the antenna's orientation based on the satellite's movement. This functionality is essential for projects requiring satellite tracking automation, as is the case in this work, where the antenna is controlled by Moteus R4.11 BLDC motors commanded via a CAN-FD interface.

Within the scope of this project, Gpredict plays a central role as a calculation and tracking interface, serving as the starting point for system automation. Through its integration with rotctld and our control script, Gpredict provides updated orbital data that are converted into commands for the motors, allowing the antenna to automatically follow the satellite's trajectory across the sky. This integration between tracking software and control hardware enables the construction of a robust, flexible solution adapted to different types of missions and satellites.

2.3. Description of MJBots Motors

The *Moteus R4.11* rotors, developed by the company *MJBots*, is an integrated motor controller with closed-loop control capability for BLDC (Brushless DC) motors, based on the Field Oriented Control (FOC) technique. This controller was designed to provide position control, being especially suitable for robotic applications, mechanical actuation systems, and, in this project's case, for positioning control of antennas in satellite tracking systems.

The *Moteus R4.11* combines a power driver, an ARM Cortex-M4 processor, and a current and voltage measurement circuit into a single compact module. The controller is designed to operate directly with BLDC motors equipped with absolute magnetic encoders, which allow real-time knowledge of the rotor's angular position, even after power interruptions.

Moteus implements vector control using FOC (Field Oriented Control), a technique that allows control of motor torque and speed, even at low speeds. This type of control is essential in positioning applications where a fast response with no vibrations or oscillations is required. Through FOC, the controller can generate rotating magnetic fields synchronized with the rotor's actual position, maximizing torque per ampere consumed.

Besides torque control, Moteus offers simultaneous speed and position control, enabling operation in different modes:

- Absolute position control (in rotations)
- Speed control (rpm or rad/s)
- Torque control (Nm)

These modes can be used individually or combined, offering high flexibility to the system.

Communication and CAN-FD Interface

One of the most distinctive features of *Moteus* is its communication based on CAN-FD (Controller Area Network – Flexible Data Rate), a bidirectional protocol that allows sending commands and receiving feedback with low latency. Each controller can be configured with a unique ID on the CAN network, allowing multiple motors to be controlled on the same bus.

Communication follows a structured binary protocol, supporting commands such as:

- set_position: set the motor target position/angle;
- set_stop: stop the motor and clear faults;
- query: read current status (position, speed, current, etc.).

The controller also supports firmware updates and parameter reconfiguration through dedicated tools like ***moteus_tool*** and ***tview***.

In the context of this project, *Moteus R4.11* motors are used to control the two axes of the tracking system: azimuth (horizontal rotation of the antenna) and elevation (vertical rotation). Their positioning precision, combined with fast communication via CAN-FD, allows the system to quickly respond to position changes calculated by *Gpredict*.

The combination of Moteus controllers with BLDC motors enables smooth, silent movement with high angular precision, essential to ensure the antenna maintains correct alignment with satellites throughout their visible path.

Using motors with absolute encoders eliminates the need for manual initial calibration since the system knows its position even after restarts, making it ideal for autonomous and remote applications like this one.

Chapter 3

Development

This chapter provides a detailed description of the system implementation process, including the integration between the tracking software, motor control, and the communication logic that enables automatic satellite tracking.

3.1. Implementation of the *Tracking* Software *Gpredict*

To enable automatic satellite tracking, the tracking software Gpredict was used, an open-source application widely employed to estimate satellite positions in real time. Gpredict features an intuitive graphical interface and allows integration with external hardware, being compatible with libraries such as Hamlib, which supports communication with rotor controllers and radios via standardized protocols.

Gpredict does not communicate directly with Python scripts or controllers; instead, it uses rotctld, a Hamlib daemon that acts as an intermediary. This controller was configured with the following command:

rotctld -m 1 -T 4533

- -m 1: Generic dummy rotor model or defined as a virtual interface;
- -T 4533: TCP port for communication with Gpredict.

3.2. Control of *MJBots* motors

Communication is done via CAN-FD between the device (computer) and the first motor, then the two motors are linked in parallel through a second CAN-FD cable. After establishing communication, it is necessary to provide power to each motor individually, using two power cables (one per rotor). Finally, the software referred to in the previous section is executed to start system control.

Figure 1 shows a block diagram illustrating the overall system architecture and the communication flow between the different components.

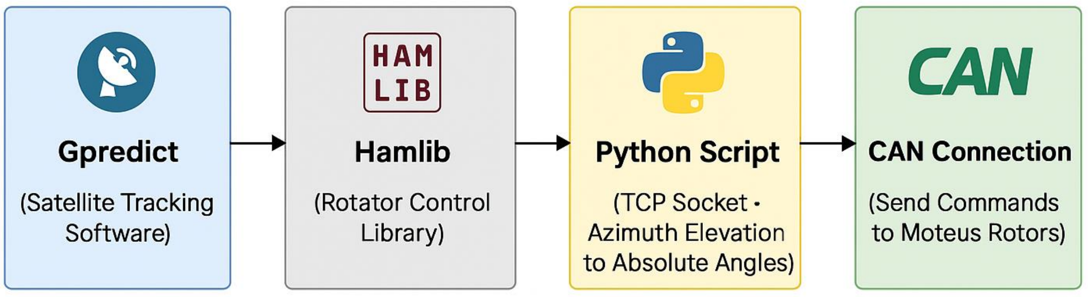

HAM LIB
CAN
Gpredict
(Satellite Tracking Software)
Hamlib
(Rotator Control Library)
Python Script
(TCP Socket • Azimuth Elevation to Absolute Angles)
CAN Connection
(Send Commands to Moteus Rotors)

Figure 1:Block Diagram illustrating the system architecture.

3.3. Interconnection between *Gpredict* and *MJBots*

With the rotors already configured and the tracking software sending data via TCP to port 4533, we only need to read this data and make the rotors perform actions (tracking movements) according to this data.

To enable this automatic tracking based on the received data, a Python script was developed (see Error! Reference source not found.), which establishes communication between the Gpredict tracking software and the Moteus R4.11 motor. Through the Hamlib library, these values are provided via a TCP/IP socket on port 4533.

Here is a quick explanation and quick overview of the Python script and the method used to interconnect the motors with the software.

1) Initial Definitions:

```
import asyncio
import moteus
import socket

HOST = "localhost"
PORT = 4533

MOTEUS_AZIMUTE_ID = 1
MOTEUS_ELEVACAO_ID = 2

azimute_motor = moteus.Controller(id=MOTEUS_AZIMUTE_ID)
elevacao_motor = moteus.Controller(id=MOTEUS_ELEVACAO_ID)
```

The presented code begins by importing three essential libraries for the system's operation. The ***asyncio*** library allows the execution of asynchronous code, meaning it can perform tasks that wait for external events without blocking the rest of the program—this is particularly useful when waiting for motor movement data. The ***moteus*** library is the official interface for communication with the Moteus controllers, used to send position, velocity, or torque control commands to the motors. The ***socket*** library belongs to Python's standard library and enables establishing TCP/IP communication, which in this case is used to connect the script to Gpredict via the rotctld service.

To ensure that the two motors can be controlled independently, each needs a unique identifier on the CAN bus. Therefore, ID 1 was assigned to the motor responsible for azimuth and ID 2 to the elevation motor. The elevation motor's ID was changed using the ***tview*** tool with the command conf write, which saves this configuration to the controller's flash memory. This way, even if the system is powered off or the CAN-FD communication is interrupted, the IDs remain stored and active upon the next startup.

2) Setting rotor parameters:

```
async def set_motor_position(controller, position):
    await controller.set_position(
        position=position,
        velocity=0.2,
        accel_limit=1.0,
        velocity_limit=3.0,
        query=False
    )
```

Starting from this function, which receives as inputs the 'controller' (elevation or azimuth motor) and 'position' (for horizontal (azimuth) or vertical (elevation) coordinates).

With the **set_position** function from the moteus library, it is possible to define various attributes such as torque, acceleration, velocity, position, among others.

3) Estabelecer Ligação TCP/IP e Conversão de Dados (Main)

```
async def monitorar_e_mover():
    try:
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as client:
            client.connect((HOST, PORT))
            await azimute_motor.set_stop()
            await elevacao_motor.set_stop()

            while True:
                client.sendall(b"p\n")
                data = client.recv(1024).decode("utf-8").strip()

                if data:
                    #print("debug cenas", repr(data))
                    try:
                        values = data.split("\n")[0:2]     # Garante que
processa apenas as duas primeiras linhas
                        azimute = float(values[0])
                        elevacao = float(values[1])

                        azimute_rotor = round(azimute / 360.0, 3)    # 0.0 a
1.0 voltas
                        #elevacao_rotor = round(elevacao / 90, 3)   # 0.0 a
1.0 voltas (ate 90 graus)
```

```
                        elevacao_rotor = -0.25 - (elevacao / 180.0) * 0.5

                        # Envia comandos para os dois motores sempre
                        await              set_motor_position(azimute_motor,
azimute_rotor)
                        await             set_motor_position(elevacao_motor,
elevacao_rotor)

                        print(f"Azimute:          {azimute:.1f}°         →
{azimute_rotor:.3f} voltas | Elevação: {elevacao:.1f}° → {elevacao_rotor:.3f}
voltas")

                    except (ValueError, IndexError) as e:
                        print("Erro ao interpretar os dados:", data)

                await asyncio.sleep(0.02)

    except Exception as e:
        print("Erro ao conectar ao rotctld:", e)

asyncio.run(monitorar_e_mover())
```

Connection:

First, communication is established between the system and the *Gpredict* tracking software by creating a TCP/IP connection through port 4533, using the protocol provided by the *rotctld* service from the *Hamlib* library. This connection allows real-time access to the azimuth and elevation values calculated by *Gpredict* based on the satellite's trajectory.

Data Processing:

Once the connection is established, the system continuously sends the command "p\n" to rotctld, requesting the current rotor position. The response is received in byte format and needs to be converted to readable text (UTF-8) using the ***.decode("utf-8")*** method. Then, the ***.strip()*** function is used to remove unnecessary spaces and newline characters, resulting in a clean string. This string is split into two parts: the first value corresponds to azimuth and the second to elevation, which are then stored in separate variables for later conversion and sending to the motors.

Conversion of Received Data:

The Moteus R4.11 controllers do not work directly with angles in degrees, but with positions expressed as motor rotations. Therefore, it is necessary to convert the azimuth and elevation values received in degrees from *Gpredict* to a format compatible with the controller, i.e., fractions of a full rotation (1.0 = 360°). For example, 0.5 corresponds to 180°, 0.25 corresponds to 90°, and so on.

The round function rounds the value to the specified decimal places (in this case, 3 decimals). For azimuth, the conversion is straightforward: the value varies from 0.0 (0°) to 1.0 (360°), representing a full rotation on the horizontal plane. For the elevation motor, an additional adjustment is needed due to the physical orientation of the motor. During assembly, the motor was mounted inverted to facilitate power and CAN-FD connection wiring between the two motors.
After testing, it was observed that 0° corresponds to -0.25 rotations and 90° corresponds to -0.5 rotations.

Sending Commands to the Motors:

Finally, after converting the azimuth and elevation values to rotation format, the commands are sent to the motors via the ***set_motor_position()*** function. Commands are sent asynchronously and continuously, ensuring the rotor moves in real-time according to the satellite's position.

Conclusions of the Developed Script:

The developed script runs a continuous asynchronous loop, ensuring frequent command sending to the Moteus motors. This approach prevents the activation of the controller's internal watchdog, ensuring the motors remain operational throughout the tracking process.

.

Chapter 4

Test Scenarios and Results

During the testing phase, a practical test of the automatic position tracking system with the *Moteus* motors integrated with the *Gpredict* software was conducted. The test demonstrated the smooth and precise movement of the rotors in response to coordinates sent in real time by *Gpredict*, simulating satellite tracking. The system's response confirmed the correct communication between the controller and the motors, as well as the effectiveness of the control logic implemented in *Python*. This test validated the system's core functionality and allowed for identifying minor fine-tuning calibration adjustments to ensure greater tracking accuracy.

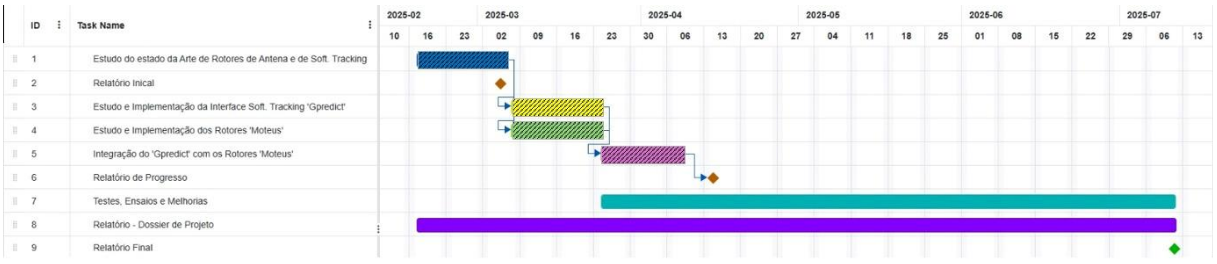
These tests also provide an important foundation for future system improvements. By observing the rotor movements, it became possible to identify opportunities to optimize rotation smoothness, as well as to adjust angle limits and increments to ensure even more precise and continuous tracking. This experimental data will be essential to fine-tune control parameters, minimizing oscillations or abrupt movements, especially during satellite passes with fast trajectories or sharp changes in azimuth/elevation.

Chapter 5

Conclusions

In the conclusions chapter, the steps carried out so far are analysed based on the Gantt chart, an assessment of the work developed is made, and the future tasks necessary until project completion are indicated.

The Gantt chart presented in Figure 2 represents the tasks performed in this project, with the shaded tasks indicating those already completed.



ID	Task Name
1	Estudo do estado da Arte de Rotores de Antena e de Soft. Tracking
2	Relatório Inical
3	Estudo e Implementação da Interface Soft. Tracking 'Gpredict'
4	Estudo e Implementação dos Rotores 'Moteus'
5	Integração do 'Gpredict' com os Rotores 'Moteus'
6	Relatório de Progresso
7	Testes, Ensaios e Melhorias
8	Relatório - Dossier de Projeto
9	Relatório Final

Figure 2 ***Gantt*** Chart with finished tasks (shaded) and remaining tasks.

There is still a period allocated for trials, testing, and improvements during which we will make the system safer, more reliable, and possibly add new features.

So far, the implementation of the automatic satellite tracking system based on real-time reading of azimuth and elevation coordinates provided by Gpredict has been successfully completed. A Python script was developed that continuously reads data from Gpredict, processes the received values, and converts them into commands compatible with the Moteus R4.11 motor controllers. The azimuth motor is already fully operational, showing smooth and precise movement in response to variations in the tracked satellite's azimuth. The implementation of the elevation motor was also completed, including the definition of safety limits.

Before the project's conclusion, it is planned to integrate a Bosch BNO055 sensor with the goal of calibrating the antenna's initial position relative to geographic north. Since the azimuth and elevation coordinates provided by Gpredict are always referenced to north, it is essential to ensure the antenna is correctly aligned with that reference before starting tracking. The BNO055 sensor, which combines accelerometer, gyroscope, and magnetometer with internal sensor fusion, will allow obtaining the system's absolute orientation, automatically identifying the north direction. With this information, it will be possible to correct the initial position of the azimuth motor and ensure the antenna always starts from a calibrated base, avoiding accumulated deviations. This step is crucial to increase tracking accuracy across the entire movement range, especially for satellites with fast trajectories or short passes.

Attachment 1

Script python for Connection between motors and Gpredict:

```python
import asyncio
import moteus
import socket

##rotctld -m 1 -s 19200 #baudrate
#python -m moteus_gui.tview --target 2


HOST = "localhost"
PORT = 4533

MOTEUS_AZIMUTE_ID = 1
MOTEUS_ELEVACAO_ID = 2

azimute_motor = moteus.Controller(id=MOTEUS_AZIMUTE_ID)
elevacao_motor = moteus.Controller(id=MOTEUS_ELEVACAO_ID)

async def set_motor_position(controller, position):
    await controller.set_position(
        position=position,
        velocity=0.2,
        accel_limit=1.0,
        velocity_limit=3.0,
        query=False
    )



async def monitorar_e_mover():
    try:
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as client:
            client.connect((HOST, PORT))
            await azimute_motor.set_stop()
            await elevacao_motor.set_stop()

            while True:
                client.sendall(b"p\n")
                data = client.recv(1024).decode("utf-8").strip()

                if data:
                    #print("debug cenas", repr(data))
                    try:
                        values = data.split("\n")[0:2]  # Garante que processa
```

```
apenas as duas primeiras linhas
                        azimute = float(values[0])
                        elevacao = float(values[1])

                        azimute_rotor = round(azimute / 360.0, 3)   # 0.0 a 1.0
voltas
                        #elevacao_rotor = round(elevacao / 90, 3)  # 0.0 a 1.0
voltas (ate 90 graus)
                        elevacao_rotor = -0.25 - (elevacao / 180.0) * 0.5

                        # Envia comandos para os dois motores sempre
                        await set_motor_position(azimute_motor, azimute_rotor)
                        await                set_motor_position(elevacao_motor,
elevacao_rotor)

                        print(f"Azimute:  {azimute:.1f}° → {azimute_rotor:.3f}
voltas | Elevação: {elevacao:.1f}° → {elevacao_rotor:.3f} voltas")

                    except (ValueError, IndexError) as e:
                        print("Erro ao interpretar os dados:", data)

                await asyncio.sleep(0.02)

    except Exception as e:
        print("Erro ao conectar ao rotctld:", e)

asyncio.run(monitorar_e_mover())
```

References

[1] MJBots, Moteus R4.11 Developer Kit - Official Site. [Online]. Available:
https://mjbots.com/products/moteus-r4-11-developer-kit

[2] MJBots, Moteus Reference Documentation. GitHub. [Online]. Available:
https://github.com/mjbots/moteus/blob/main/docs/reference.md

[3] Joshua Awesome, @awesjosh - YouTube Channel. [Online]. Available:
https://www.youtube.com/@awesjosh

[4] VA3HDL, Gpredict and Hamlib Integration. [Online]. Available:
https://www.va3hdl.com/projects/gpredict-and-hamlib

[5] Hamlib Contributors, Hamlib – Control Software for Radios. GitHub. [Online]. Available:
https://github.com/Hamlib/Hamlib

[6] J. H. Hetherington, Gpredict User Manual 1.3. Oregon State University. [PDF]. Available:
https://sites.science.oregonstate.edu/~hetheriw/whiki/psp/main/base/files/gpredict-user-manual-1.3.pdf