Instituto Superior de Engenharia de Lisboa – ISEL
Lic. Eng. de Eletrónica e Telecomunicações Computadores - LEETC

Projeto Final de Curso – P26

Relatório de Progresso

Rotor de Antena para receção de satélite

Diogo Maurício nº 49372

Miguel Fonseca nº 49341

Licenciatura em Engenharia de Eletrónica e
Telecomunicações e de Computadores

Orientadores: Prof. João Casaleiro

Prof. Tiago Oliveira

2025

Declaramos que este documento é da nossa autoria e está conforme o Código de Conduta e Boas Práticas

ISEL

Resumo

O nosso projeto tem como base a utilização de dois motores BLDC (*brushless* DC *motors*) com controladores Moteus r4.11, que funcionam como rotor para posicionamento de uma antena. Os comandos de posicionamento são gerados em tempo real pela aplicação 'Gpredict', que calcula as coordenadas de elevação e azimute necessárias para o seguimento de satélites. Estes dados são enviados via protocolo TCP/IP utilizando a biblioteca Hamlib.

No entanto, os controladores Moteus não são diretamente suportados pela Hamlib (como outros controladores comercializados atualmente).

Assim, o principal objetivo do nosso projeto foi criar uma ponte entre os dados enviados pela Hamlib e os rotores Moteus, de forma a interpretar corretamente as coordenadas recebidas e aplicar os movimentos correspondentes. Este sistema permite que os rotores reajam de forma precisa e sincronizada às posições calculadas pelo Gpredict, assegurando um seguimento eficiente dos satélites em tempo real.

Palavras-chave

Rotor, Antenas, Satélites, Tracking.

Índice

Lista de Figuras

Lista de Acrónimos

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*

Capítulo 1

Introdução

Neste capítulo apresentamos uma visão geral do nosso projeto e a motivação e objetivos para a realização do mesmo. Para além disso, será apresentada a estrutura do relatório.

1.1. Visão Geral

Os sistemas de *tracking* de satélites são essenciais para diversas aplicações, como telecomunicações, observação da terra e exploração espacial. Para garantir uma comunicação eficiente com satélites que não são geoestacionários, é necessário que as antenas estejam constantemente alinhadas com os satélites, ajustando dinamicamente os ângulos de azimute e elevação. Este projeto tem como objetivo desenvolver um sistema de controlo para um rotor de antena, capaz de seguir automaticamente um satélite em tempo real.

O problema em estudo envolve a conversão dos dados de posição do satélite(em tempo real), mais concretamente o seu azimute e elevação, em comandos válidos para rotores controlados pelo driver e controlador Moteus R4.11 [1][2][3], motores(dois motores pela necessidade de controlar a elevação e o azimute) estes que deverão obter e atualizar instantaneamente as suas posições mediante os dados recebidos pelo Gpredict[4] (software que calcula a posição dos satélites em tempo real com base na suas orbitas e data hora).

A comunicação entre o *Gpredict* e o rotor será realizada utilizando a biblioteca Hamlib'[5]**,** que fornece comandos padrão para obtenção de ângulos de elevação e azimute. Um script escrito em Python faz a leitura desses valores e o respetivo envio para os rotores.

Para a realização deste projeto, adquirimos dois motores: Um motor para o azimute (coordenadas horizontais) e outro para a elevação (coordenadas verticais). Note-se que tantoo o azimute como a elevação depedem da posição do satélite bem como da localização geográfica do rotor.

1.2. Motivação e Objetivos

A receção e seguimento de satélites em tempo real é uma área de crescente interesse, não só no contexto profissional (telecomunicações, meteorologia, investigação científica), mas também no meio académico e entre entusiastas de rádio amadorismo. Com o aumento da quantidade de satélites em órbita, nomeadamente em constelações LEO (Low Earth Orbit), tornou-se essencial o desenvolvimento de sistemas capazes de seguir as trajetórias com precisão.

Os sistemas comerciais de rotores de antenas, apesar de eficientes, baseiam-se em motores com escovas e mecanismos de (des)multiplicação de rotações. Nos rotores de telescópios, porém, já é comum a utilização de motores sem escovas.

Neste projeto, surgiu a motivação para desenvolver um sistema moderno de seguimento automático de satélites, tirando partido das capacidades avançadas de controlo de motores de última geração, como os *'Moteus'* R4.11, da comunicação moderna via CAN-FD, e do software de seguimento open-source *'Gpredict'*. O objetivo principal é criar um sistema completo e funcional de controlo automático

de antena, capaz de receber em tempo real os dados de azimute e elevação do *'Gpredict'*, converter essas coordenadas em comandos de posição para os motores *'Moteus'* e garantir um movimento suave, contínuo e preciso dos rotores ao longo da trajetória do satélite.

2 Estrutura do relatório

O relatório está estruturado em cinco capítulos. O primeiro capítulo apresenta uma introdução ao tema, abordando a visão geral do projeto, as motivações que o impulsionaram e os objetivos a atingir. O segundo capítulo descreve o estado da arte, analisando trabalhos relacionados e discutindo as vantagens e limitações de outras soluções, bem como uma explicação técnica do funcionamento do *Gpredict* e dos motores *Moteus*.

O desenvolvimento do sistema é tratado no terceiro capítulo, onde se detalha a implementação prática, desde a receção dos dados do *Gpredict* até ao controlo dos motores *MJBots*, incluindo a lógica de conversão e comunicação entre os componentes. O quarto capítulo foca-se nos cenários de teste e nos resultados obtidos, demonstrando o funcionamento do sistema e analisando o seu desempenho. Por fim, o quinto capítulo apresenta as conclusões do trabalho desenvolvido, refletindo sobre as etapas cumpridas, avaliando os resultados e apontando possíveis melhorias e desenvolvimentos futuros.

Capítulo 2

Estado da Arte

Este capítulo apresenta uma análise de trabalhos relacionados com sistemas de seguimento de satélites, tecnologias utilizadas no controlo de rotores de antena, e os principais conceitos técnicos que fundamentam o desenvolvimento do projeto.

2.1. Estado da Arte

No estado da arte, existem soluções comerciais para *tracking* de satélites, porém, muitas são de alto custo ou não oferecem flexibilidade para personalização. Outras alternativas incluem o uso de controladores como o Yaesu GS-232, que permitem o seguimento automático, mas com protocolos mais antigos e menos adaptáveis. Este projeto propõe uma abordagem inovadora ao integrar motores Moteus, que oferecem maior precisão e controle avançado, com um sistema acessível e personalizável.

2.1.1. Soluções Comerciais

2.1.1.1. Rotor Yaesu G-5500

Existem no mercado diversos rotores de antena comerciais com sistemas de controlo incorporados. Um dos mais conhecidos é o **Yaesu G-5500 [6]** , que permite o controlo dos eixos de azimute e elevação, sendo amplamente utilizado por radioamadores. Este sistema é compatível com controladores externos como o **GS-232**, que permite a interface com software como o *Gpredict*. No entanto, apresenta algumas limitações:

- Elevado custo de aquisição;
- Engrenagens mecânicas, que com o tempo acumulam folga (backlash), reduzindo a precisão do seguimento e aumentam o custo de manutenção.

2.1.1.2. SPID Rotators

Os SPID Rotators (e sua versão AlfaSpid) são rotores de antena de alta capacidade, projetados para controlar a orientação de antenas tanto no eixo de azimute como de elevação, sendo ideais para *tracking* de satélites, observação astronómica, estações meteorológicas, entre outros.

Os modelos mais conhecidos são os SPID RAS, este que é um rotor azimute-elevação típico para seguimento de satélites, com bom compromisso entre custo e robustez. Também existe o modelo SPID BIG-RAK,modelo mais usado para grandes antenas parabólicas, muito usado em EME (Earth-Moon-Earth) e comunicações de longo alcance.

2.1.2. Soluções *Open Source* e DIY

Paralelamente aos sistemas comerciais, surgiram diversas soluções DIY (*Do It Yourself*), muito populares em trabalhos académicos e comunidades de radioamadores. Estas soluções recorrem frequentemente a:

- Motores de passo com drivers tipo A4988 ou TMC2209;
- Microcontroladores como Arduino ou ESP32;
- Controladores baseados em relés e sensores de fim de curso;

Embora estas soluções tenham a vantagem de serem **económicas e personalizáveis**, apresentam limitações consideráveis em termos de:

- Precisão angular;
- Resistência mecânica;
- Velocidade de resposta;
- Complexidade de calibração.

O nosso projeto posiciona-se entre o melhor dos dois mundos: precisão e robustez comparáveis aos sistemas comerciais, com a flexibilidade e customização dos sistemas open-source.

2.1.3. Solução Adotada

A escolha de utilizar os controladores *Moteus R4.11* com motores BLDC e comunicação via CAN-FD representa uma abordagem inovadora neste tipo de aplicação. Originalmente concebidos para robótica de alto desempenho, os controladores Moteus oferecem:

- Controlo vetorial FOC em tempo real;
- Precisão sub-grau na leitura de posição (via encoder absoluto magnético);
- Resposta suave e rápida aos comandos;
- Comunicação via CAN-FD, permitindo escalabilidade e robustez na transmissão de dados.

Aliado ao uso do *Gpredict* como software de seguimento e da biblioteca *Hamlib* para comunicação normalizada, este projeto oferece uma solução robusta e economicamente viável, permitindo controlo preciso e automático de uma antena com base nos cálculos orbitais de satélites em tempo real.

2.2. *Software* de *tracking* de satélites Gpredict

O *Gpredict* é uma aplicação de seguimento de satélites em tempo real, de código aberto, desenvolvida para o ambiente *GNOME*. Utiliza elementos gráficos baseados na biblioteca *GTK+* e oferece uma plataforma robusta e extensível para seguir satélites em órbita terrestre, com especial foco em aplicações de rádio amador, telecomunicações, educação e pesquisa científica.

O funcionamento do Gpredict baseia-se em dados orbitais *TLE* (Two-Line Element sets), provenientes de fontes como o Celestrak ou NORAD, que permitem prever com elevada precisão a trajetória de satélites em órbita terrestre baixa (LEO), média (MEO) ou geoestacionária (GEO). Com base nesses dados, o Gpredict calcula em tempo real os parâmetros orbitais relevantes para cada satélite selecionado, incluindo:

- Azimute e Elevação (para direcionar as antenas)

- Distância ao ponto de observação
- Velocidade relativa
- Hora da próxima passagem
- Doppler shift (para correção da frequência do rádio)

A arquitetura modular do *Gpredict* permite a criação de várias janelas (módulos), onde cada módulo pode seguir diferentes grupos de satélites, oferecendo visualizações gráficas (mapas orbitais, gráficos de elevação/tempo) e tabelas com parâmetros em tempo real. Esta modularidade é particularmente útil em ambientes que requerem o monitoramento simultâneo de múltiplos satélites.

Uma das funcionalidades mais relevantes para este projeto é a capacidade do Gpredict de se integrar com hardware externo, como rotores de antena e rádios Full-Duplex, através da biblioteca Hamlib (Ham Radio Control Libraries). Para o controlo de rotores, o *Gpredict* comunica com o daemon *rotctld*, que escuta numa porta TCP (por omissão, 4533) e aceita comandos normalizados para reportar e definir a posição do rotor. O protocolo é simples e baseado em texto, facilitando a integração com scripts e aplicações personalizadas (como a nossa em *Python*).

O *Gpredict* pode ser configurado para enviar comandos de azimute e elevação em tempo real, ajustando dinamicamente a orientação da antena com base no movimento do satélite. Esta funcionalidade é essencial para projetos que exigem automação do seguimento de satélites, como é o caso deste trabalho, em que a antena é controlada por motores BLDC *Moteus R4.11*, comandados através de uma interface CAN-FD.

No âmbito deste projeto, o *Gpredict* desempenha um papel central como interface de cálculo e seguimento, servindo como ponto de partida para a automação do sistema. Através da sua integração com o *rotctld* e o nosso script de controlo, o *Gpredict* fornece os dados orbitais atualizados que são convertidos em comandos para os motores, permitindo que a antena siga automaticamente a trajetória do satélite no céu. Esta integração entre software de rastreamento e hardware de controlo permite construir uma solução robusta, flexível e adaptada a diferentes tipos de missões e satélites

2.3. Descrição Motores MJBots

O *Moteus R4.11*, desenvolvido pela empresa *MJBots*, é um controlador de motor integrado com capacidade de controlo em malha fechada para motores *BLDC* (Brushless DC), baseado na técnica de *Field Oriented Control* (FOC). Este controlador foi concebido com o objetivo de fornecer controlo de posição, sendo especialmente adequado para aplicações robóticas, sistemas de atuação mecânica e, no caso deste projeto, para controlo de posicionamento de antenas em sistemas de rastreamento de satélites.

O *Moteus R4.11* combina um driver de potência, um processador ARM Cortex-M4, e um circuito de

medição de corrente e tensão num único módulo compacto. O controlador está preparado para funcionar diretamente com motores *BLDC* equipados com encoders magnéticos absolutos, que permitem conhecer em tempo real a posição angular do rotor, mesmo após cortes de alimentação.

O *Moteus* implementa o controlo vetorial FOC (Field Oriented Control), uma técnica que permite controlar o torque e a velocidade do motor, mesmo a baixas velocidades. Este tipo de controlo é fundamental em aplicações de posicionamento, onde é necessária uma resposta rápida e com ausência de vibrações ou oscilações. Através do FOC, o controlador consegue gerar campos magnéticos rotativos sincronizados com a posição real do rotor, maximizando o torque por ampere consumido.

Além do controlo de torque, o *Moteus* oferece controlo simultâneo de velocidade e posição, possibilitando a operação em diferentes modos de atuação:

- Controlo de posição absoluta (em rotações)
- Controlo de velocidade (rot/min ou rad/s)
- Controlo de torque (Nm)

Estes modos podem ser usados isoladamente ou em combinação, oferecendo elevada flexibilidade ao sistema.

Comunicação e Interface CAN-FD

Um dos elementos mais distintivos do Moteus é a sua comunicação baseada em CAN-FD (Controller Area Network – Flexible Data Rate), um protocolo bidirecional, que permite enviar comandos e receber feedback com baixa latência. Cada controlador pode ser configurado com um ID único na rede CAN, permitindo controlar múltiplos motores no mesmo barramento.

A comunicação segue um protocolo binário estruturado, suportando comandos como:

- set_position: definir posição/ ângulo alvo do motor
- set_stop: parar o motor e eliminar falhas
- query: ler o estado atual (posição, velocidade, corrente, etc.)

O controlador também suporta actualizações de firmware e reconfiguração dos parâmetros através de ferramentas dedicadas como o *moteus_tool* e *tview*.

No contexto deste projeto, os motores *Moteus R4.11* são utilizados para controlar os dois eixos do sistema de rastreamento: o azimute (rotação horizontal da antena) e a elevação (rotação vertical). A sua precisão de posicionamento, aliada à comunicação rápida via CAN-FD, permite que o sistema responda rapidamente às variações de posição calculadas pelo software *Gpredict*.

A combinação de controladores *Moteus* com motores BLDC permite alcançar um movimento fluido, silencioso e com elevada precisão angular, essencial para garantir que a antena mantenha o

alinhamento correto com os satélites durante todo o percurso visível.

O uso de motores com encoder absoluto elimina a necessidade de calibração inicial manual, pois o sistema conhece a posição mesmo após reinicializações, tornando-o ideal para aplicações autónomas e remotas, como esta.

Capítulo 3

Desenvolvimento

Este capítulo descreve detalhadamente o processo de implementação do sistema, incluindo a integração entre o software de seguimento, o controlo dos motores, e a lógica de comunicação que permite o seguimento automático de satélites.

3.1. Implementação do *Software* de *Tracking* ‘Gpredict’

Para permitir o seguimento automático de satélites, foi utilizado o software de seguimento *Gpredict*, uma aplicação open-source amplamente utilizada para estimar a posição de satélites em tempo real. O *Gpredict* possui uma interface gráfica intuitiva e permite integração com hardware externo, sendo compatível com bibliotecas como a Hamlib, que suporta comunicação com controladores de rotores e rádios através de protocolos padronizados.

O *Gpredict* não comunica diretamente com scripts Python ou controladores; em vez disso, utiliza o *rotctld*, um serviço da *Hamlib* que atua como intermediário (Deamon). Este controlador foi configurado com o seguinte comando: *rotctld -m 1 -T 4533*

- -m 1: Modelo genérico de rotor fictício (dummy) ou definido como interface virtual;
- -T 4533: Porta TCP para comunicação com o Gpredict.

3.2. Controlo dos motores ‘MJBots’

A comunicação é por CAN-FD entre o dispositivo (computador) e o primeiro motor, e em seguida interligar os dois motores em paralelo através de um segundo cabo CAN-FD. Após estabelecer a comunicação, é necessário fornecer alimentação a cada um dos motores individualmente, utilizando dois cabos de alimentação (um por rotor). Por fim, executa-se o software referido na secção anterior para iniciar o controlo do sistema.

Na Figura 1 apresenta-se um diagrama de blocos que ilustra a arquitetura geral do sistema e o fluxo de comunicação entre as diferentes componentes

Figura 1:Diagrama de blocos sobre o funcionamento do sistema

3.3. Interligação entre o Gpredict e o MJBots (Script Python)

Tendo os rotores já configurados e o Software de tracking a enviar dados por TCP para porto 4533, apenas necessitamos de ler estes dados e fazer com que os rotores executem ações (movimentos de

rastreamento) de acordo com estes mesmos dados.

Para permitir este seguimento automático de acordo com os dados recebidos, foi desenvolvido um *script* em Python, ver **Error! Reference source not found.**, que estabelece a comunicação entre o *software* de *tracking Gpredict* e o motor *Moteus R4.11*. Através da biblioteca Hamlib, estes valores são disponibilizados por *socket* TCP/IP na porta 4533.

Neste ponto vamos então explicar o script python, método utilizado para a interligação dos motores o *software*.

1) **Definições Iniciais:**

```
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)
```

O código apresentado começa por importar três bibliotecas essenciais para o funcionamento do sistema. A biblioteca `asyncio` permite executar código assíncrono, ou seja, realizar tarefas que podem esperar por eventos externos sem bloquear o restante programa, o que é particularmente útil quando se está à espera de dados do movimento dos motores. A biblioteca `moteus` é a interface oficial para comunicação com os controladores *Moteus*, sendo utilizada para enviar comandos de controlo de posição, velocidade ou torque aos motores. Já a biblioteca `socket` pertence à biblioteca padrão do *Python* e permite estabelecer uma comunicação TCP/IP, que neste caso é usada para ligar o script ao *Gpredict* através do serviço `rotctld`.

Para garantir que os dois motores possam ser controlados de forma independente, cada um precisa de um identificador único no barramento CAN. Por isso, foi atribuído o ID 1 ao motor responsável pelo azimute e o ID 2 ao motor da elevação. A alteração do ID do motor de elevação foi feita através da ferramenta `tview` com o comando `conf write`, permitindo guardar essa configuração na memória flash do controlador. Desta forma, mesmo que o sistema seja desligado ou a comunicação via CAN-

FD seja interrompida, os IDs permanecem armazenados e ativos na próxima inicialização.

2) Definir parâmetros do rotor:

```
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
    )
```

A partir desta função, que recebe como *inputs* do 'controller' (motor de elevação ou azimute) e 'position'(caso coordenadas horizontais(azimute) ou verticais(elevação)

Com a função da biblioteca moteus set_position é possível definir vários atributos tais como o torque,aceleração,velocidade posição entre outros.

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())
```

Conexão:

Primeiramente é estabelecida a comunicação entre o sistema e o software de rastreamento Gpredict, criando uma ligação TCP/IP através da porta 4533, utilizando o protocolo fornecido pelo serviço rotctld da biblioteca Hamlib. Esta ligação permite aceder, em tempo real, aos valores de azimute e elevação calculados pelo Gpredict com base na trajetória do satélite.

Processamento dos dados:

Após estabelecida a ligação, o sistema envia continuamente o comando "p\n" para o rotctld, solicitando a posição atual do rotor. A resposta é recebida em formato de bytes, sendo necessário convertê-la para texto legível (UTF-8) usando o método .decode("utf-8"). Em seguida, a função .strip() é utilizada para remover espaços e quebras de linha desnecessárias, resultando numa string limpa. Esta string é dividida em duas partes: o primeiro valor corresponde ao azimute, e o segundo à elevação, que são então armazenados em variáveis separadas para posterior conversão e envio aos motores.

Conversão dos dados recebidos:

Os controladores *Moteus* R4.11 não trabalham diretamente com ângulos em graus, mas sim com posições expressas em número de rotações do motor. Assim, é necessário converter os valores de azimute e elevação, recebidos em graus pelo *Gpredict*, para o formato compatível com o controlador, ou seja, frações de uma volta completa (1.0 = 360°). Ora, se uma volta completa equivale a 360, então

0.5 equivale a 180º, 0.25 equivale a 90º e assim em diante.

A função round apenas arredonda o valor recebido no primeiro parâmetro (no caso 3 casas decimais).

No caso do azimute a conversão é direta. O valor pode variar de 0.0 (0°) até 1.0 (360°), e representa uma volta completa no plano horizontal.

Para o motor de elevação, a conversão precisa de um ajuste adicional, devido à orientação física do motor. Durante a montagem, o motor foi colocado de forma invertida, de modo a facilitar as ligações de alimentação e de conexão CAN-FD entre os dois motores

Após alguns testes, foi possível observar que no 0º corresponde a -0.25 rotações e 90º corresponde a -0.5.

Envio dos comandos para os motores:

Por fim, após converter os valores de azimute e elevação para o formato de rotações, os comandos são enviados para os motores através da função set_motor_position(). O envio dos comandos é feito de forma assíncrona e contínua, garantindo que o rotor se mova em tempo real de acordo com a posição do satélite.

Conclusões sobre o script desenvolvido:

O script desenvolvido executa um ciclo assíncrono contínuo, garantindo o envio frequente de comandos aos motores Moteus. Esta abordagem permite evitar a ativação do **watchdog interno** do controlador, assegurando que os motores se mantêm operacionais durante todo o processo de rastreamento.

Capítulo 4

Cenários de teste e Resultados

No capítulo dos resultados

Durante a fase de ensaio e testes, foi realizado um teste prático do sistema de seguimento automático de posições com os motores Moteus, integrados com o software Gpredict. O teste em questão

demonstra o movimento fluido e preciso dos rotores em resposta às coordenadas enviadas em tempo real pelo Gpredict, simulando o seguimento de um satélite. A resposta do sistema confirmou a correta comunicação entre o controlador e os motores, bem como a eficácia da lógica de controlo implementada em Python. Este teste validou a funcionalidade base do sistema e permitiu identificar pequenos ajustes de calibração fina para garantir uma maior precisão no seguimento.

Estes testes também fornecem uma base importante para futuras melhorias no sistema. Com a observação dos movimentos dos rotores, tornou-se possível identificar oportunidades de otimização da suavidade da rotação, bem como ajustar os limites e incrementos dos ângulos para garantir um seguimento ainda mais preciso e contínuo. Estes dados experimentais serão fundamentais para afinar os parâmetros de controlo, minimizando oscilações ou movimentos bruscos, especialmente em passagens de satélites com trajetórias rápidas ou mudanças de azimute/elevado acentuadas.

Capítulo 5

Conclusões

No capítulo das conclusões são analisadas as etapas realizadas até ao momento com base no Gantt chart, é feito um balanço do trabalho desenvolvido e são indicadas as tarefas futuras necessárias até à finalização do projeto.

O gráfico *'Gantt'* apresentado na Figura 2 representa as tarefas realizadas neste projeto, sendo que as tarefas sombreadas representam tarefas já realizadas.

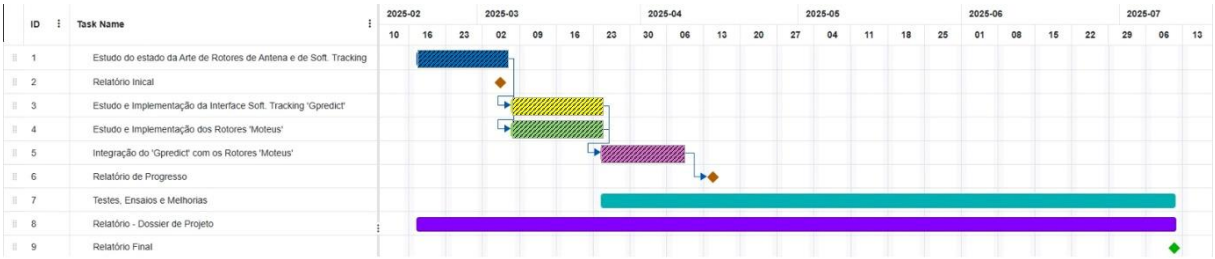

Figura 2 Diagrama de Gantt com as tarefas concluídas (a sombreado) e restantes tarefas por realizar.

Com isto em mente, ainda existe um período de tempo para ensaios, testes e melhorias onde iremos tornar o sistema mais seguro, fiável e adicionar eventualmente novas funcionalidade

Até ao momento, foi concluída com sucesso a implementação do sistema de seguimento automático de satélites com base na leitura em tempo real das coordenadas de azimute e elevação fornecidas pelo Gpredict. Desenvolveu-se um script em Python que lê continuamente os dados fornecidos pelo Gpredict, processa os valores recebidos e converte-os em comandos compatíveis com os controladores de motor Moteus R4.11.O motor de azimute já se encontra totalmente funcional, sendo possível observar o seu movimento suave e preciso em resposta às variações de azimute do satélite rastreado. A implementação do motor de elevação também foi concluída, com a definição de limites de segurança.

Até à conclusão do projeto, está previsto implementar a integração de um sensor Bosch BNO055, com o objetivo de realizar a calibração da posição inicial da antena em relação ao norte geográfico. Como as coordenadas de azimute e elevação fornecidas pelo Gpredict são sempre referidas em função do norte, é essencial garantir que a antena esteja corretamente alinhada com essa referência antes de iniciar o seguimento. O sensor BNO055, que combina acelerómetro, giroscópio e magnetómetro com fusão sensorial interna, permitirá obter a orientação absoluta do sistema, identificando automaticamente a direção do norte. Com esta informação, será possível corrigir a posição inicial do motor de azimute e assegurar que a antena parte sempre de uma base calibrada, evitando desvios acumulados. Este passo é crucial para aumentar a precisão do seguimento em todo o intervalo de movimento, especialmente em satélites com trajetórias rápidas ou passagens curtas.

Anexo 1

Script python para interligação entre os motores e o 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())
```

3Referências

[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