Universidade de Brasília
Faculdade de Tecnologia

Bancada de Ensaios de Controle para Drones Utilizando Arquiteturas ARM e FPGA

Leonardo Teixeira Alves

Rosana Santos Ribeiro

TRABALHO DE GRADUAÇÃO

ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Brasília

2022

Universidade de Brasília
Faculdade de Tecnologia

Bancada de Ensaios de Controle para Drones Utilizando Arquiteturas ARM e FPGA

Leonardo Teixeira Alves
Rosana Santos Ribeiro

Trabalho de Graduação submetido como requisito parcial para obtenção do grau de Engenheiro de Controle e Automação

Orientador: Prof. Jones Yudi Mori Alves da Silva

Brasília
2022

Alves, Leonardo Teixeira.
A474b Bancada de Ensaios de Controle para Drones Utilizando Arquiteturas ARM e FPGA / Leonardo Teixeira Alves; Rosana Santos Ribeiro; orientador Jones Yudi Mori Alves da Silva. –– Brasília, 2022.
104 p.

Trabalho de Graduação em Engenharia de Controle e Automação –– Universidade de Brasília, 2022.

1. ARM. 2. FPGA. 3. *drone*. 4. controle. I. Ribeiro, Rosana Santos. II. Silva, Jones Yudi Mori Alves da, orient. III. Título

Universidade de Brasília
Faculdade de Tecnologia

Bancada de Ensaios de Controle para Drones Utilizando Arquiteturas ARM e FPGA

Leonardo Teixeira Alves
Rosana Santos Ribeiro

Trabalho de Graduação submetido como requisito parcial para obtenção do grau de Engenheiro de Controle e Automação

Trabalho aprovado. Brasília, 02 de maio de 2022:

Prof. Jones Yudi Mori Alves,
UnB/FT/ENM
Orientador

Prof. Renato Coral Sampaio, UnB/FGA
Examinador interno

Prof. Andre Carmona Hernandes,
UFSCar/DEE
Examinador externo

Brasília
2022

Este trabalho é dedicado a minha família,
que me apoiou durante toda minha jornada acadêmica
e permitiram que eu chegasse onde estou hoje 💚.

Leonardo Teixeira Alves

Dedico este trabalho à minha
família, em especial, à minha vó Socorro
e aos pequenos Zazá, Lulu e Davi.

Rosana Santos Ribeiro

Agradecimentos

Gostaria de agradecer a todos e todas que me acompanharam durante minha trajetória na UnB, em especial a meus pais, Ludmila e Darci, que sempre me deram apoio e me ajudaram a conquistar meus sonhos. Agradeço também a meus irmãos, Gabriela e Eduardo, que mantiveram sempre o bom humor e mostraram seu companheirismo. Meu maior agradecimento é a Rosana, sempre prestativa e uma verdadeira amiga desde que a conheci quando ingressamos na UnB. Sua colaboração durante este trabalho foi imprescindível para nossa graduação. Obrigado! 💜 Deixo também meu agradecimento especial a Júlio Eduardo, cuja disponibilidade para ajudar durante o projeto foi essencial para que conseguíssemos concluir este trabalho. Obrigado também a Estêvão e a José Henrique, que fizeram o possível para me ajudar nos momentos de dificuldade com a execução do projeto. Também não posso deixar de agradecer a nosso orientador, professor Jones, cujos conhecimento e aconselhamento foram de suma importância para este trabalho. Por fim, mas não menos importante, a todos meus colegas de curso deixo meu "Obrigado!", sem os quais estudar Engenharia Mecatrônica seria uma jornada muito mais árdua e maçante.

Leonardo Teixeira Alves

Primeiramente gostaria de agradecer a Deus por mais essa vitória, e por tudo na minha vida. Agradeço à minha família por todo o amor. Agradeço aos meus pais pela força e carinho: à minha mãe, Lívia, meu alicerce, pelo esforço para que eu me tornasse uma mulher forte e ao meu pai, Gilberto, pelo entusiasmo e incentivo que me deu durante todos esses anos até aqui. Agredeço à Rosimar e às minhas irmãs Letícia, Jaqueline, Sara e Isadora por me apoiarem, me ajudarem e estarem por mim, sempre. Não poderia deixar de agradecer ao meu amor Rafael, por nos momentos mais difíceis estar aqui ao meu lado, me acalmando, dando suporte, tendo muita paciência comigo, sempre torcendo e acreditando em mim. Ao Leonardo, só tenho a agradecer a honra que foi tê-lo como dupla não só neste projeto, mas em diversas ocasiões. Agradeço por tudo, por ser uma pessoa maravilhosa que tenho o privilégio de ter na minha vida. Ao Júlio Eduardo, agradeço pela contribuição essencial para nosso projeto, e, ao nosso orientador, Jones, por todo o suporte durante o desenvolvimento. Por fim, agradeço a todos os meus amigos de curso, do CATRON e da DROID, pelo companheirismo durante esses anos e pelas experiências memoráveis compartilhadas.

Rosana Santos Ribeiro

“Everything’s got to end sometime.
Otherwise, nothing would ever get started.”
– The Doctor

Resumo

Este trabalho consiste no desenvolvimento, construção e testes de bancada de ensaios para *drones*, simulando o movimento de um dos eixos da aeronave. Com o controle do movimento a partir de arquiteturas ARM e FPGA, aplicadas em controlador MiniZed da Avnet, busca-se trazer resposta mais rápida na estabilidade da estrutura na forma de gangorra, aproveitando a característica de *hardware* dedicado das arquiteturas de processamento utilizadas. Tem-se como objetivo, também, permitir que a estrutura e o código de controle desenvolvidos sejam utilizados em trabalhos futuros e/ou em disciplinas do curso de Engenharia Mecatrônica de forma didática e para pesquisas. Ademais, buscou-se também atender a critérios de robustez, segurança e documentação, para a entrega de um equipamento laboratorial.

Palavras-chave: ARM. FPGA. *drone*. controle.

Abstract

This work consists of the development, construction and tests of a test bench for *drones*, simulating the movement of one of the aircraft axes. With the motion control from ARM and FPGA architectures, applied in MiniZed controller from Avnet, it is sought to bring faster response in the stability of the structure in the form of a seesaw, taking advantage of the dedicated *hardware* characteristic of the processing architectures used. It is also intended to allow the developed structure and control code to be used in future works and/or in disciplines of the Mechatronic Engineering course in a didactic way and for research. Besides, it was also sought to meet the criteria of robustness, safety, and documentation for the delivery of a lab grade piece of equipment.

Keywords: ARM. FPGA. drone. control.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

AGV Veículo Guiado Automatizado – *Automated Guided Vehicles* 16

ANAC Agência Nacional de Aviação Civil 16

Anatel Agência Nacional de Telecomunicações 16

ARM Máquinas RISC Avançadas – *Advanced RISC Machines* 18

AXI Interface Extensível Avançada – *Advanced eXtensible Interface* 54

BLDC Sem Escovas de Corrente Contínua – *Brushless Direct Current* 25

CPU Unidades Central de Processamento – *Central Porcessing Unit* 23

DECEA Departamento de Controle do Espaço Aéreo 16

ESC Controlador Eletrônico de Velocidade – *Electronic Speed Controller* 21

FPGA Arranjo de Portas Programáveis em Campo – *Field-Programmable Gate Array* 18

GPU Unidade de Processamento Gráfico – *Graphics Processing Unit* 23

I^2C/IIC Circuito Inter-Integrado – *Inter-Integrated Circuit* 54

IMU Unidade de Medição Inercial – *Inertial Measurement Unit* 21

IP Propriedade Intelectual – *Intellectual Property* 54

MEMS Sistemas Microeletromecânicos – *Micro-Electromechanical System* 29

MIT Instituto de Tecnologia de Massachusetts – *Massachusetts Institute of Technology* 25

PID Proporcional Integral Derivativo 42

PWM Modulação por Largura de Pulso – *Pulse Width Modulation* 33

RISC Computador de Conjunto de Instruções Reduzido — *Reduced Instruction Set Computer* 19

ROV Veículo Remotamente Operado – *Remotely Operated Vehicle* 16

SDK *Kit* de Desenvolvimento de *Software* – *Software Development Kit* 52

SI Sistema Internacional de Medidas 33

UAV Veículo Aéreo não Tripulado – *Unmanned Aerial Vehicle* 16

UFSCar Universidade Federal de São Carlos 40

VHDL Linguagem de Descrição de Hardware VHSIC – *VHSIC Hardware Description Language* 52

VHSIC Circuito Integrado de Muito Alta Velocidade – *Very High Speed Integrated Circuit*
52

Sumário

1 Introdução

Este capítulo introduz o problema a ser resolvido neste trabalho, explanando detalhes sobre o uso de veículo aéreos, bancadas de teste, plataformas de desenvolvimento e arquiteturas de computador

1.1 Contextualização

1.1.1 Veículos não tripulados

As aplicações da automação são das mais diversas e seu uso vem se popularizando. Uma delas vem na forma de veículos não tripulados, como veículos guiados automatizados (AGVs - *Automated Guided Vehicles*), veículos remotamente operados (ROVs - *Remotely Operated Vehicles*) e veículos aéreos não tripulados (UAVs - *Unmanned Aerial Vehicle*), que são utilizados tanto para uso pessoal – como é o caso de *drones* para filmagem e fotografia por exemplo – quanto na indústria – como, por exemplo, em veículos transportadores de paletes em armazéns – ou no meio acadêmico – com uso de ROVs para pesquisas em ambientes submarinos.

São múltiplas as possibilidades de uso de tais veículos que existem atualmente e, com o crescimento e a popularização da automação, há perspectivas de tornar sua utilização mais comum para realização de diversos tipos de tarefas, como a limpeza de janelas em prédios elevados (ROGERS, 2019).

Já um *drone* é considerado um UAV, podendo ser definido como um sistema embarcado capaz de ser pilotado à distância ou usando um plano de voo automatizado, fazendo uso de um conjunto de sensores, podendo incluir inclusive um sistema de posicionamento global (LUTKEVICH, 2021).

Apesar da generalidade do termo *drone*, este trabalho foca nos chamados *drones* de asa rotativa, isto é, desconsiderando modelos de asa fixa, que baseiam-se na aeroplanagem para movimento.

1.1.2 Legislação para *drones*

A utilização de *drones* tem diversos objetivos, tornando-os uma ferramenta poderosa para quem os possui. Essa característica somada à possibilidade de danos ao ambiente em que estão acionados e às pessoas que estão ao redor faz necessária a criação de normas e regras em suas utilizações. Normalmente, cada país possui um agente que é responsável por essa regulamentação (HERRMANN; MARKERT, 2020).

No Brasil, a regulamentação dos *drones* é regida por três entidades, sendo eles: a Agência Nacional de Aviação Civil (ANAC), a Agência Nacional de Telecomunicações (Anatel) e o Departamento de Controle do Espaço Aéreo (DECEA) (GOVBR, 2016). São descritas regras que definem desde a categoria do *drone*, caracterizada pelo peso máximo para decolagem, a altura e a velocidades máximas, a forma de comunicação, até o perfil e comportamento dos usuários, sejam o piloto, com a necessidade de uma habilitação especial, ou alguém que apenas observa e auxilia o funcionamento da aeronave. Ressalta-se que, no espaço aéreo brasileiro, o voo de aeronaves completamente automatizadas não é permitido. Assim, seu uso ocorre apenas em países que permitam seu voo.

1.1.3 Sistemas auxiliares

Em sistemas embarcados, o uso de sistemas auxiliares de controle ou automação do movimento tem função de colaborar com o piloto de forma a tornar a navegação do veículo mais fácil e, possivelmente, substituir o piloto completamente. Estes sistemas auxiliares vêm na forma de malhas de controle e de instrumentação para garantir comportamento desejado. No caso de *drones*, é necessário, por exemplo, estabilizar o movimento de forma a possibilitar que o veículo paire no ar sem cair ou rotacionar ou realize um deslocamento de maneira conhecida e/ou controlada.

É também necessária atenção ao desempenho do sistema embarcado, pois deve-se também cumprir requisitos de tempo real, de forma a evitar falhas catastróficas. Exemplos de falhas catastróficas incluem a danificação do aparelho ou o ferimento de pessoas. As causas de tais falhas podem acontecer devido a leituras dos instrumentos indevidas (dados não suficientemente recentes, por exemplo) ou a atrasos na atuação do controlador desenvolvido (OLIVEIRA, 2018).

Por se tratar de sistemas embarcados, a quantidade de recursos disponíveis para atender todos os requisitos é limitada, devendo ainda ser levadas em consideração características como peso, tamanho, consumo de energia, entre outros aspectos. Com isso em mente, é interessante considerar plataformas heterogêneas, isto é, com o uso de duas ou mais arquiteturas de processamento combinadas, para a realização desses sistemas auxiliares em um *drone*. Assim, os benefícios de cada uma das arquiteturas utilizadas podem ser aproveitados.

1.1.4 Sistemas embarcados críticos

Sistemas são considerados críticos quando, em caso de falha ou não cumprimento de requisitos, podem gerar falhas consideradas catastróficas (OLIVEIRA, 2018). *Drones* podem ser considerados sistemas embarcados críticos no sentido de que, caso falhem, podem, além de ser danificados e terem custo relativamente alto para reparo e substituição, colocar pessoas e animais em risco.

É fácil encontrar na internet relatos de acidentes com drones (KUKSOV, 2019), reforçando ainda mais a criticidade desse tipo de sistema. Assim, faz-se necessário que, sejam eles guiados remotamente ou completamente autônomos, os *drones* sejam devidamente controlados para manter a estabilidade e cumprimento de requisitos de segurança durante o voo. Uma forma de garantir seu correto funcionamento é pela testagem de seus diversos componentes em um ambiente seguro, como é o caso de bancadas de teste, foco deste trabalho.

1.1.5 Bancadas de Teste

Com as diversas especificações e legislações a serem seguidas para permitir os uso de *drones* de asa rotativa e outros UAVs, torna-se necessário garantir o correto funcionamento dos diversos equipamentos utilizados na construção da aeronave. Além disso, dado o alto custo de construção de um *drone*, é interessante que, durante os processos de desenvolvimento e prototipagem, os equipamentos sejam testados com segurança, minimizando custos de acidentes e falhas. Para tal, é possível encontrar na literatura, formas de bancada de ensaios que permitem a realização de testes de forma segura.

As bancadas podem variar das mais simples às mais complexas, sendo de uso genérico, isto é, permitindo ensaios do funcionamento de diversos componentes, ou específico, ou seja, focando em um grupo menor de componentes a serem avaliados.

Neste trabalho, são exploradas versões de bancadas encontradas na literatura que embasaram a construção neste projeto. Algumas versões encontradas permitem ensaios como de sensores, de conjuntos propulsores, de ambos em conjunto ou de técnicas de controle, por exemplo. A variedade é numerosa, havendo desde versões pequenas e feitas em casa com materiais facilmente encontrados na internet a versões maiores usando tecnologia de ponta para testes que requerem maior precisão e acurácia em seus resultados. Em todos os casos, é preciso averiguar as necessidades de projeto antes de tomar a decisão de qual(is) tipo(s) de bancada é(são) mais apropriada(s).

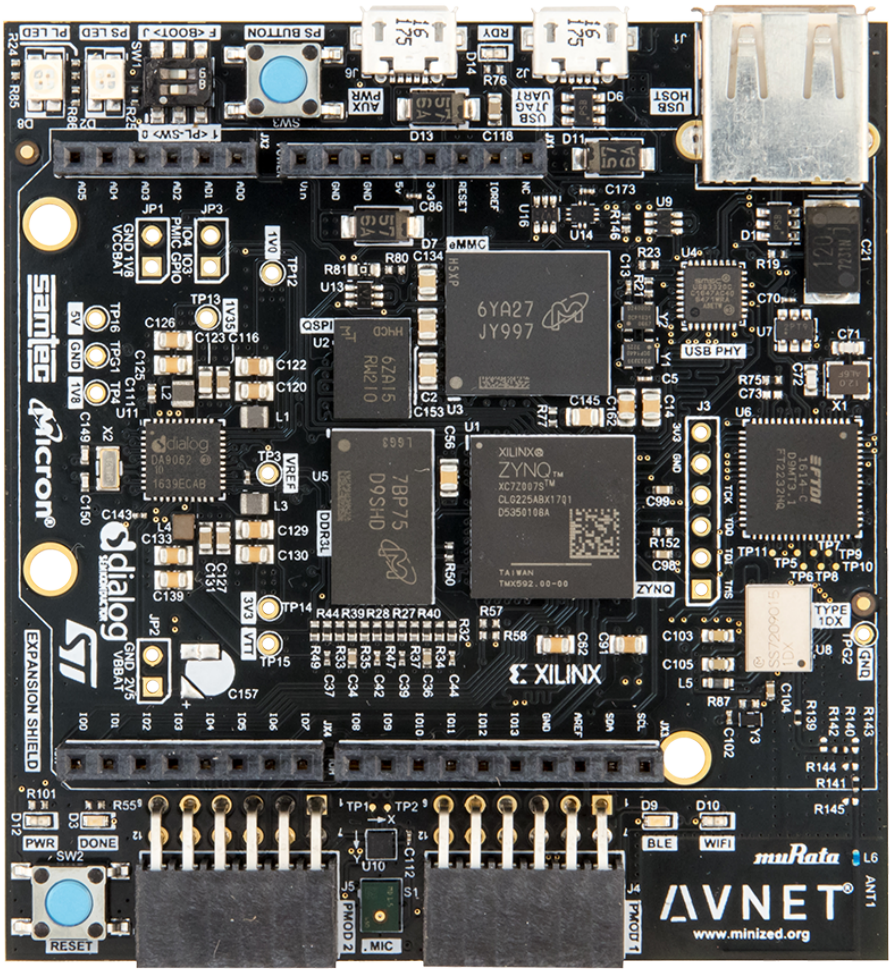
1.1.6 Plataforma Zynq e placa de desenvolvimento MiniZed

Uma das possibilidades de plataformas heterogêneas, em foco neste trabalho, é a série de *chips* Zynq 7000 da Xilinx (XILINX, 2011), que combina arquiteturas ARM (Máquinas RISC Avançadas – do inglês, *Advanced RISC Machines*) e FPGA (Arranjo de Portas Programáveis em Campo – do inglês, *Field-Programmable Gate Array*), possibilitando diversas formas de implementação dos sistemas auxiliares. A combinação de arquiteturas permite que sejam utilizadas as vantagens de cada uma para as finalidades desejadas, possivelmente otimizando o sistema auxiliar desenvolvido.

A MiniZed (AVNET, 2016), ilustrada na Figura 1, é uma placa de desenvolvimento

que utiliza um *chip* Zynq da série 7000, mais especificamente, o *chip* Zynq 7Z007S. A placa permite a personalização completa de suas funções, desde as entradas do sistema, a forma que serão interpretadas e principalmente tratadas, até as saídas, dando ao projetista o controle completo do fluxo de dados dentro da placa de desenvolvimento.

Figura 1 – Placa de desenvolvimento Avnet MiniZed



Fonte: (AVNET, 2016)

1.1.7 Arquiteturas ARM e FPGA

A arquitetura ARM utiliza a filosofia de *design* RISC (também do inglês, *Reduced Instruction Set Computer*, isto é, Computador de Conjunto de Instruções Reduzido), que busca entregar instruções de máquina simples e poderosas capazes de serem executadas em ciclos de *clock* únicos. As motivações da criação da arquitetura ARM – como baixo consumo de energia, alta densidade de código, baixo custo de produção, pequena área ocupada pelo processador, por exemplo – fazem dela uma ótima candidata para uso em sistemas embarcados (SLOSS; SYMES; WRIGHT, 2004), como é o caso com processadores em UAVs.

Já a arquitetura FPGA surgiu de uma necessidade para programação de interconecti-

vidade, quando especificações de componentes eram trocadas frequentemente e os padrões de desenvolvimento evoluíam constantemente, precisando ser atendidos. Assim, a FPGA permite a programação do *hardware* do componente após sua fabricação, possibilitando atingir melhores resultados na adequação aos requisitos de algoritmos e alta performance em quesitos como área, velocidade e potência (WOODS et al., 2008). Esses benefícios também podem ser aproveitados para uso em sistemas embarcados.

1.1.8 Motivação

Seja em ambientes fabris ou em situações cotidianas, a tecnologia tem evoluído para equipamentos cada vez mais autônomos, diminuindo a interferência e o trabalho humano. Sistemas são projetados e implementados a fim de garantir um ambiente seguro e um funcionamento desejado das máquinas.

Além disso, os conceitos de inovação e criatividade estão intimamente relacionados ao desenvolvimento econômico de países e suas regiões. Para tal, o advento de novas tecnologias é fundamental para o crescimento econômico (DEVEZAS; LEITÃO; SARYGULOV, 2017). Assim, a visualização de aplicações distintas de sistemas já existentes torna-se um possível propulsor da inovação tecnológica.

No caso dos UAVs, a implementação de sistemas auxiliares para o controle de um *drone* de asa rotativa tem o objetivo de torná-los cada vez mais autônomos e seguros. Sendo assim, a placa de desenvolvimento da Avnet, MiniZed, por possuir um processador ARM e uma FPGA, permite ao projetista a implementação de uma malha de controle que proporciona rapidez e robustez ao sistema, sendo possível configurar detalhadamente cada módulo e periférico da placa.

A vantagem do uso de uma plataforma heterogênea, neste caso com a implementação em arquiteturas de computador ARM e FPGA, se dá pela possibilidade de combinação das qualidades de cada arquitetura. O baixo custo, a alta velocidade de processamento, o baixo consumo energético e a baixa latência podem ser listados como exemplos dessas qualidades (SAXENA, 2020; PLOEG, 2018).

A utilização da MiniZed para comando e controle de uma bancada de teste é uma maneira educacional de experimentar a implementação de um *hardware* dedicado enquanto são respeitados os requisitos temporais do sistema com maior folga, trazendo maior segurança e completude ao sistema.

1.2 Definição do problema

Sistemas embarcados necessitam de um detalhamento muito específico de seus componentes. O consumo de energia, o peso, o tamanho e a durabilidade são alguns aspectos

importantíssimos e totalmente influenciados por essas especificações. O funcionamento lógico desses componentes também é restrito e próprio, requerendo um controle dedicado e totalmente personalizado para o sistema em questão.

No caso dos *drones*, é necessária certa robustez de seus componentes, de forma a garantir o funcionamento adequado do equipamento e a harmonia do sistema. Sabendo que uma falha pode gerar catástrofes – sejam prejuízos materiais, ao próprio UAV ou a propriedade alheia, sejam danos a pessoas e animais próximos ao trajeto da aeronave –, garantir seu funcionamento correto é essencial para minimizar os riscos advindos de sua utilização. Sendo assim, a implementação de um *hardware* dedicado possibilitaria atingir os requisitos necessários para a robustez desejada e evitar o acontecimento de tais falhas.

Considerando a criticidade do sistema que é um *drone*, a existência de um ambiente de testes seguro possibilita aplicações educacionais e simulações de ambientes reais para testes e prototipagem. Dessa forma, a bancada de testes precisa atender a esses requisitos com robustez, e tornando esse ambiente mais seguro.

1.3 Objetivos do projeto

O objetivo geral deste trabalho é o projeto e a construção de uma bancada de estudos de baixo custo, portável e segura sobre UAVs. A bancada deve ser uma representação simplificada de um *drone*, possuindo inicialmente apenas 01 grau de liberdade.

Sua montagem deve permitir ao usuário que diversos testes sejam realizados, sendo eles gerais ou de um subsistema específico, podendo ser:

- Estrutura mecânica;
- Baterias e seu gerenciamento;
- Sensores e condicionamento de sinais;
- Eletrônica de potência;
- Motores e propulsores;
- Sistemas embarcados de processamento.

Este trabalho também possui por objetivo específico a manipulação da bancada com a MiniZed, uma plataforma de processamento heterogêneo, com processador ARM e FPGA. Sendo assim, podem ser listados os seguintes objetivos específicos:

- Integração da plataforma com ARM e FPGA;

- Integração de sensores, mais especificamente, a Unidade de Medição Inercial (IMU - *Inertial Measurement Unit*);
- Integração do sistema de propulsores e Controlador Eletrônico de Velocidade (ESC - *Electronic Speed Controller*);
- Caracterização da bancada, envolvendo:
  - Caracterização dos sinais de saída da placa de desenvolvimento;
  - Calibração do conjunto propulsor;
  - Calibração da IMU.

1.4 Apresentação do manuscrito

Este trabalho possui quatro capítulos: Introdução; Revisão de Literatura; Desenvolvimento e Resultados e Conclusão. O manuscrito também apresenta dois anexos: a Descrição do Conteúdo do CD; e os Programas Utilizados.

O primeiro capítulo faz uma breve introdução ao tema, apresentando alguns tópicos, tais como: o que são UAVs, a legislação dos *drones*, o que são os sistemas auxiliares, sistemas embarcados críticos, a apresentação da MiniZed, introdução às arquiteturas de computador utilizadas e a motivação do trabalho, assim como o problema apresentado e os objetivos deste trabalho.

O segundo capítulo adentra o conteúdo sobre *drones* e sobre as bancadas de teste para *drones*, apresentando alguns modelos comerciais desses UAVs, algumas bancadas já existentes que serviram de inspiração para este trabalho e uma caracterização mais específica dos sistemas embarcados e seus componentes. O capítulo foca em detalhar conteúdos já existentes e que serviram como base para este trabalho.

O terceiro capítulo descreve o desenvolvimento deste projeto, desde a contribuição de trabalhos anteriores, a escolha dos materiais para a bancada, os movimentos de rotação de uma aeronave, a caracterização da bancada deste projeto e a implementação de técnicas de controle para uso e experimentação na bancada. Além disso, o capítulo traz os resultados obtidos ao longo do desenvolvimento do trabalho.

O quarto e último capítulo descreve as próximas etapas a serem seguidas em trabalhos futuros bem como sugestões de melhorias para futuras implementações, assim como algumas dificuldades encontradas no desenvolvimento deste projeto.

O anexo de descrição de conteúdo do repositório *online* descreve os diretórios e a organização dos arquivos utilizados e desenvolvidos durante o projeto.

O último anexo traz a listagem dos programas utilizados durante o trabalho para seu desenvolvimento.

2 Revisão de Literatura

Este capítulo faz o levantamento a respeito de aeronaves e bancadas de teste existentes no mercado e na literatura, com foco nos tópicos relevantes ao projeto.

2.1 *Drones* comerciais

Existem à venda diversos tipos de *drones* para diferentes usos, desde fotografia aérea, tanto amadora quanto profissional, à manutenção de redes de alta tensão ou combate a incêndios. Assim, *drones* de asa rotativa tomam diferentes formas e tamanhos, a depender de sua aplicação, sendo mais comum o formato de quadrirrotor (quadricóptero), mostrado na Figura 2.

Figura 2 – *Drone* quadrirrotor

Fonte: (BUCCO-LECHAT, 2016)

Segundo a revista *Global Brands*, a companhia Dji é líder na produção desses equipamentos, sendo uma das mais conhecidas (GLOBAL BRANDS, 2020). Seus produtos utilizam *hardware*, como unidades centrais de processamento (CPUs - do inglês, *Central Porcessing Units*) e unidades de processamento gráfico (GPUs - do inglês, Graphics Processing Units), fabricado por grandes empresas. A Intel, para CPUs, e a NVIDIA, para GPUs, são exemplos (DJI, 2019) que utilizam processadores, no geral, mais poderosos e caracterizados por arquiteturas IAx86 e ARM, respectivamente.

Existem outros formatos de *drones* como hexacópteros e octocópteros, como mostram as Figuras 3 e 4. Por possuírem mais motores, conseguem gerar maior força de empuxo e,

com isso, servem para carregamento de equipamentos de maior massa. Em contrapartida, requerem mais energia para alimentar todos os motores, necessitando de mais recargas ou de baterias de maior capacidade para continuarem em operação.

Figura 3 – *Drone* hexacóptero

Fonte: (EHRHARDT, 2018)

Figura 4 – *Drone* octocóptero

Fonte: (STONE, 2017)

2.2 *Drones* customizados

Com o crescimento da área de UAVs, tanto no Brasil quanto no exterior, a disponibilidade de peças e componentes para *drones* também vem aumentando. A internet, além

de facilitar a aquisição de diversas partes para diferentes necessidades de acordo com o projeto individual, tem sido um meio para a propagação do uso amador de *drones*, resultando em um crescente número de adeptos à prática e de fãs. Exemplos de comunidades dedicadas à construção de UAVs por pessoas de diversos níveis de expertise podem ser encontradas, mostrando como a área tem ganhado seguidores. A comunidade "DIY Drones" (`https://diydrones.com/`) é um exemplo que permite a participação de qualquer usuário, inclusive na criação de publicações sobre a construção de *drones* ou sobre projetos relacionados.

O ganho de popularidade por UAVs caseiros também trouxe à tona a expansão das possibilidades de aplicações de seu uso. Há empresas, como o caso da Lumenier (`https://www.lumenier.com/`), focadas em criar equipamentos para usos específicos, sejam eles voltados à amadores, sejam para uso industrial, científico ou militar. Outro exemplo é uma plataforma desenvolvida pelo Instituo de Tecnologia de Massachusetts (MIT - do inglês, *Massachusetts Institute of Technology*), que permite a criação de *drones* customizados, permitindo aplicações distintas, principalmente estudos científicos a respeito das aeronaves (CONNER-SIMONS, 2016).

Dessa forma, *drones* customizados precisam de atenção especial a detalhes para garantir seu correto funcionamento. Por se tratar de equipamentos criados para casos específicos a partir de necessidades diversas, a funcionalidade correta de cada componente precisa ser garantida. Para atingir essas necessidades, uma possibilidade é o uso de bancadas de teste dos equipamentos.

2.3 Física de veículos aéreos

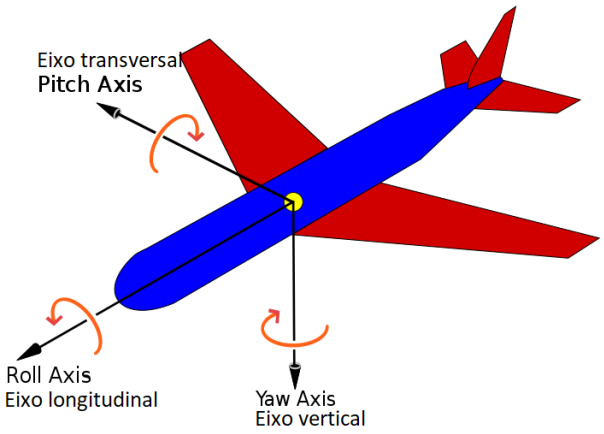
A modelagem matemática do movimento de um quadrirrotor não é elementar, sendo necessário levar em consideração diversos fatores como a força de rotação das hélices, o empuxo gerado, o momento resultante de forças de empuxo distintas, entre outros. O modelo de Newton-Euler pode ser usado para modelar essa dinâmica, como faz Lima (PAULA LIMA, 2013). Tomando certas condições do sistema, como a invariabilidade da matriz de inércia e a simetria do quadrirrotor, Lima realiza a modelagem do movimento de um *drone* genérico com motores trifásicos sem escovas de corrente contínua (BLDC - do inglês, *Brushless Direct Current*) (PAULA LIMA, 2013). Entretanto, a bancada de testes montada tem apenas um grau de liberdade, podendo ser considerado o movimento de rolagem ou arfagem, tornando mais simples sua modelagem, abordada a seguir.

2.3.1 Modelagem matemática de uma aeronave genérica

Uma aeronave possui três eixos de possível rotação, que, juntos, ajudam a definir a trajetória de voo. Esses três eixos são denominados: lateral ou transversal – em torno do

qual acontece o movimento de arfagem ou *pitch*; longitudinal – em torno do qual acontece o movimento de rolagem ou *roll*; e vertical – em torno do qual acontece o movimento de guinada ou *yaw*, e são ilustrados na Figura 5.

Figura 5 – Eixos de rotação de uma aeronave



Fonte: Adaptado de (JRVZ, 2010)

Em um UAV não é diferente: a composição dessas três rotações é um fator importante na caracterização do voo. É importante que esses movimentos estejam bem controlados, para que o *drone* de asa rotativa quadrirrotor, no caso, realize uma trajetória bem definida e com maior estabilidade.

Um dos principais objetivos no sistema de controle desses movimentos é a minimização das oscilações nesses sentidos de rotação, ou seja, a obtenção de baixos sobressinais e tempos de acomodação.

2.3.2 Sentido de rotação dos motores em um *drone* quadrirrotor

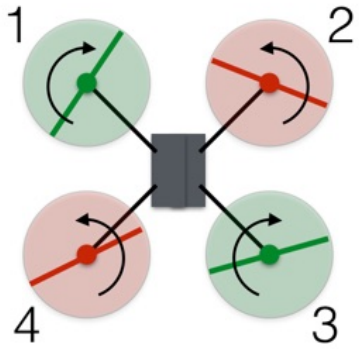
Uma possível explicação do funcionamento da asa de uma aeronave é baseada no princípio de Bernoulli, em que, ao se movimentar pelo ar (ou outro fluido, se for o caso), este se movimenta com velocidade maior em sua parte superior e menor na parte inferior. Essa diferença de velocidade gera uma diferença de pressão, sendo a pressão na parte de baixo da asa superior à pressão na parte de cima. Isso faz com que haja uma força de empuxo que atua para cima, mantendo a aeronave em suspensão (SCIENCE LEARNING HUB, 2011).

Uma hélice pode ser considerada como um conjunto de asas que, em vez de realizar movimento de translação para gerar empuxo, usa o movimento de rotação. Entretanto, esse movimento de rotação pode afetar o UAV em questão.

Pela terceira lei de Newton – lei da ação e reação –, toda ação possui uma reação de mesma direção e sentido oposto. Pode-se aplicar esse conceito no movimento das hélices gerado pelos motores, ou seja, os motores geram uma força nas hélices, e assim, as hélices

geram uma força nos motores. Ao alternar o sentido de rotação para pares de motores, como demonstra a Figura 6, a soma dos momentos angulares gerados no corpo do equipamento é minimizada, reduzindo ou até mesmo anulando o movimento indesejado em torno do eixo vertical.

Figura 6 – Sentido de rotação das hélices de um quadrirrotor



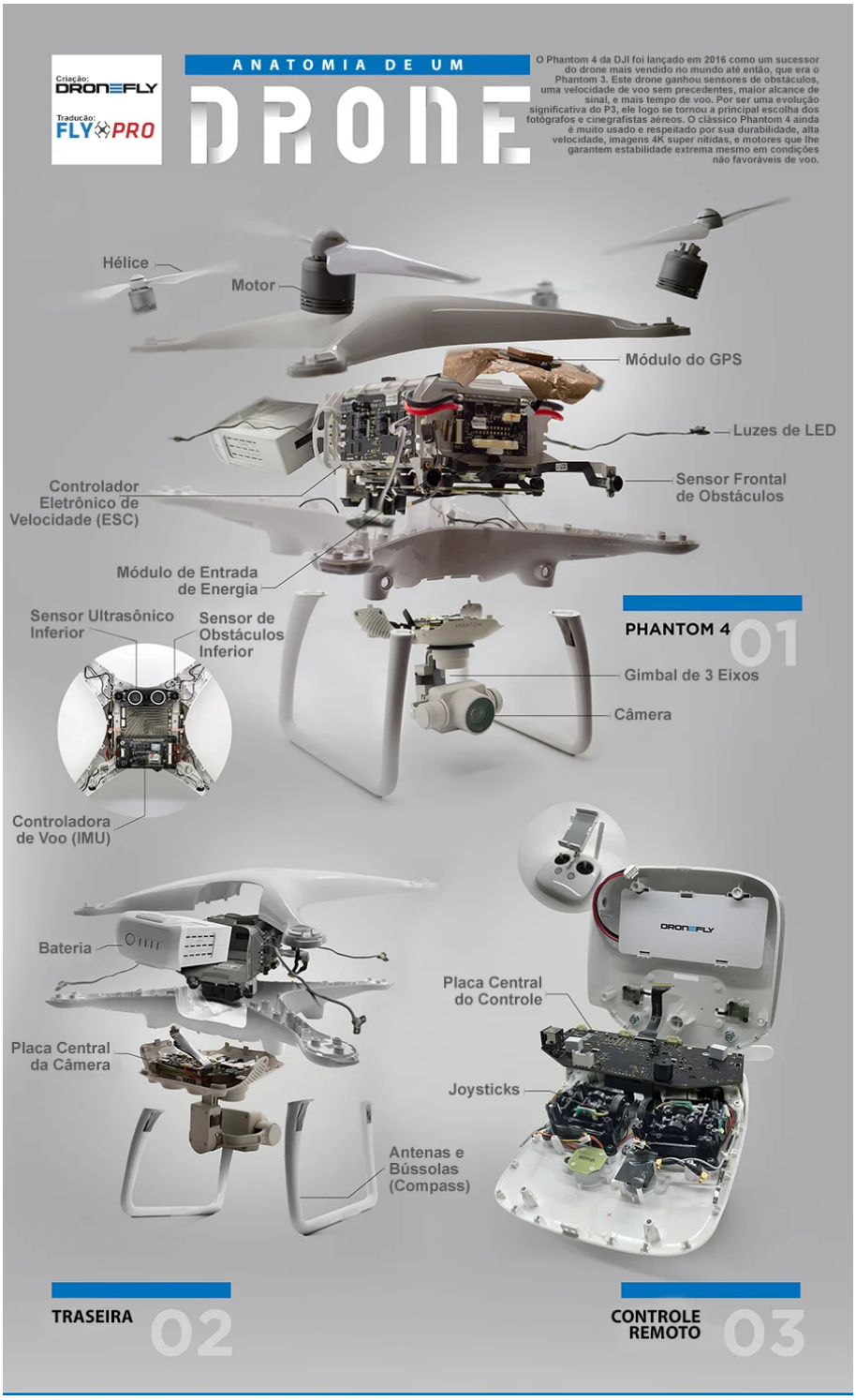
Fonte: (ALLAIN, 2017)

Com base no mesmo princípio, para a bancada, que simula apenas um eixo de rotação com um par de motores, basta que estes girem em sentidos opostos para que haja diminuição do movimento rotacional em torno do eixo vertical da bancada, isto é, reduz-se o efeito do movimento de guinada na gangorra, que deve ser, idealmente, inexistente.

2.4 Sistemas embarcados

Diversos dos sistemas embarcados utilizados em UAVs se baseiam em um mesmo conjunto de componentes, sendo um controlador eletrônico de velocidade (ESC) para acionamento dos motores, os instrumentos de medição para posição angular e aceleração, um processador para o controle do sistema, além de componentes intrínsecos ao sistema, como motores e hélices. A Figura 7 ilustra os componentes comumente encontrados em *drones* comerciais.

Figura 7 – Detalhamento das partes de um *drone*



Fonte: (FLYPRO, 2020)

2.4.1 Conjunto Sensor

Para o correto funcionamento da aeronave, é necessária a presença de elementos sensores. Assim, grande parte dos *drones* possuem, como parte de seu sistema embarcado, sensores como acelerômetros, giroscópios, geolocalização, bússolas, entre outros (WINKLER,

2016).

No caso da bancada de testes, por possuir um único grau de liberdade, apenas a medição da sua posição angular é necessária. Tal medida pode ser realizada apenas com o uso de sensores do tipo giroscópio e acelerômetro pois, quando devidamente calibrados, esses sensores permitem a aferição da posição angular na bancada.

2.4.1.1 Acelerômetros

Os acelerômetros mais simples e comuns classificam-se como do tipo sistema microeletromecânico (MEMS - do inglês, Micro-Electromechanical System), o mesmo utilizado na bancada de testes. Seu funcionamento se dá por meio de um eletrodo com massa capaz de se mover ao longo de um eixo. Ao se mover, a massa faz contato com eletrodos estacionários (a partir do referencial do encapsulamento do acelerômetro), que enviam o sinal elétrico para medição. Os vãos entre a massa e os outros eletrodos atuam como capacitores e a movimentação altera seu valor de capacitância, permitindo que seja medido o valor da aceleração atuando sobre a massa (WOODFORD, 2022).

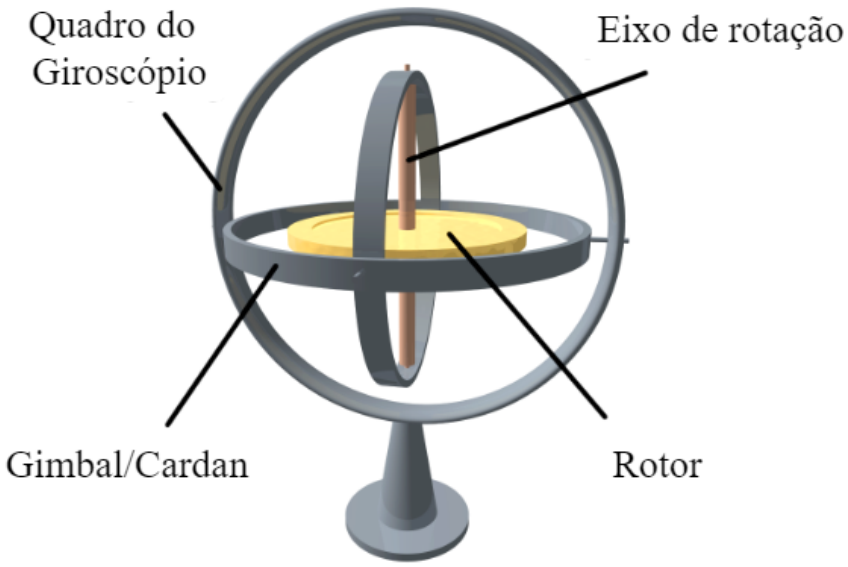
Acoplando três unidades de medição perpendiculares entre si, é possível medir o valor da aceleração em três eixos distintos, isto é, um sistema de medição com três gruas de liberdade. O acelerômetro utilizado na bancada faz uso dessa possibilidade. É necessário realizar também o ajuste das medidas caso o eixo de movimento não esteja alinhado com o eixo de rotação do sensor.

2.4.1.2 Giroscópios

Um sensor giroscópico tem como base de funcionamento o efeito Coriolis (WATELECTRONICS, 2020). Esse efeito atua em corpos em rotação, gerando uma força aparente devido à inércia dos corpos em movimento (SEARA DA CIÊNCIA, 2019).

A partir do movimento de um rotor em uma estrutura de cardan – ou *gimbal*, isto é, que permite o isolamento dos três eixos de rotação do rotor –, é realizada a leitura da aceleração angular à que o giroscópio está submetido. Essa aferição é possível devido à reação que o giroscópio possui ao movimento, sendo realizada a leitura nos três eixos distintos (WATELECTRONICS, 2020). A Figura 8 apresenta o elemento principal de um sensor giroscópico tradicional.

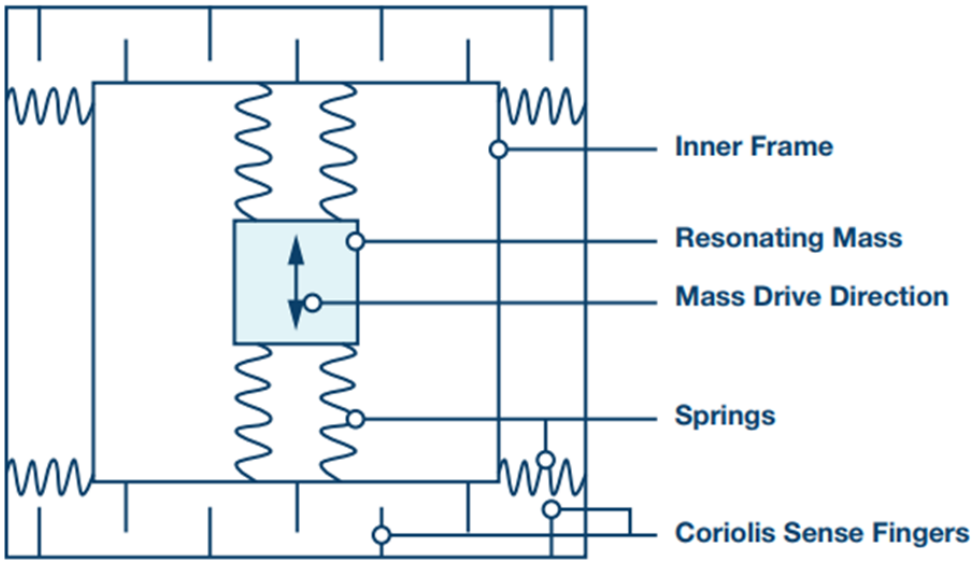
Figura 8 – Estrutura principal de um giroscópio



Fonte: Adaptado de (VIEIRA, 2006)

Já o giroscópio MEMS, utilizado nos sensores da bancada, utiliza uma estrutura como a da figura 9, cujo princípio de funcionamento simula o comportamento de um giroscópio tradicional com uso de molas.

Figura 9 – Sensor de um giroscópio MEMS



Fonte: (WATSON, 2016)

A bancada utiliza o sensor giroscópico em conjunto com o acelerômetro, sendo ambos disponíveis na IMU utilizada, a fim de obter a estimação da posição angular da gangorra.

2.4.1.3 Fusão de Sensores

Os sensores utilizados na bancada – acelerômetro e giroscópio – fornecem as leituras da aceleração e da velocidade angular, respectivamente. Entretanto, para que os ensaios sejam bem sucedidos, é necessário saber a posição angular em que a gangorra se encontra.

Sabe-se que a posição, velocidade e aceleração angulares estão relacionadas de acordo com a Equação 2.1.

$$\theta(t) = \int \omega(t)\,dt = \int\int \alpha(t)\,dt^2 \tag{2.1}$$

Tendo em vista essa relação, é possível estimar a posição angular a partir da aceleração angular dada pela leitura dos sensores. Entretanto, a estimação é propensa a erros devido a integral discretizada realizada em *software*. Buscando mitigar esses erros, é feita a fusão dos sensores acelerômetro e giroscópio.

A fusão de sensores traz o junção de dados distintos disponibilizados por dois ou mais sensores que permitam medir o mesmo sinal. Com a união dos dados, as estimativas do valor real da medição tornam-se mais confiáveis e próximas do valor real (NEVES, 2017).

Uma maneira relativamente robusta de realizar a fusão de sensores é por meio de um filtro de Kalman, que tenta predizer medições futuras a partir de atuais além de fundir os sensores para obter uma medida acurada e confiável (NEVES, 2017).

2.4.2 Conjunto Propulsor

O conjunto propulsor é responsável pela geração de empuxo e de movimento no *drone*. Composto pela hélice, por um motor sem escovas e seu ESC ou por um motor com escovas, é o atuador principal do *drone*. Na bancada de testes, serão utilizados dois conjuntos para que possam ser realizados testes menos complexos envolvendo apenas 1 grau de liberdade, enquanto que em um sistema real são encontrados de 4 ou 8 conjuntos, aumentando a complexidade para o estudo de controle.

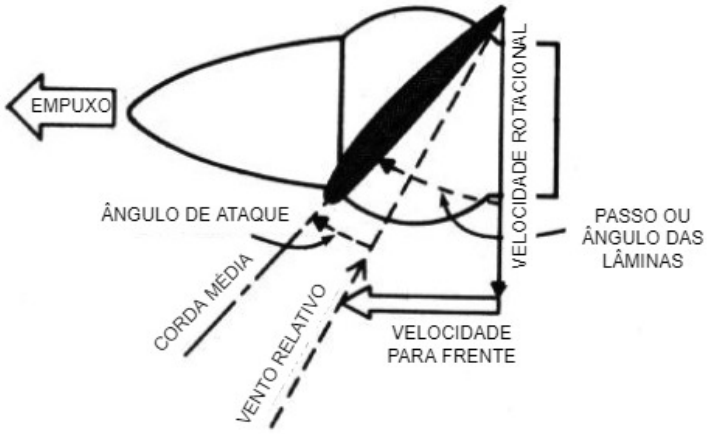
2.4.2.1 Hélices

A hélices são responsáveis pela geração de empuxo, a partir da energia entregue pelo motor, a fim de gerar o movimento em fluídos como água, no caso de barcos, ou no ar, no caso de aeronaves tais quais os UAVs também se inserem.

Diversas das características de uma hélice, como ângulo de ataque, material e seu grau de deformação, passo, quantidade de lâminas, formato, dimensões e seção transversal por exemplo, influenciam o gasto de energia para movimentar a hélice bem como o empuxo máximo que pode ser gerado. No caso de *drones*, use-se geração ativa de empuxo para movimentar o equipamento, isto é, há um gasto energético para rotacionar as hélices e gerar

força de empuxo. Em contrapartida, as hélices de uma turbina eólica, por exemplo, utilizam empuxo passivo, gerado pelo movimento do ar, para gerar energia (LANDELL-MILLS, 2022). A Figura 10 ilustra alguns dos componentes de funcionamento de uma hélice de aeronave.

Figura 10 – Componentes de uma hélice em movimento



Fonte: Adaptado de (U.S. DEPARTMENT OF TRANSPORTATION, 1965)

A caracterização de uma hélice é feita a partir de diversos fatores que determinarão o empuxo gerado pela sua rotação. Esses fatores são relacionados às suas pás, envolvendo (LANDELL-MILLS, 2022):

- Quantidade;
- Ângulo de ataque;
- Comprimento;
- Espessura;
- Largura;
- Seção transversal;
- Material;
- Espaçamento.

2.4.2.2 Motores

Os motores são os responsáveis pela conversão de energia elétrica em energia mecânica. Características como presença ou ausência de escovas; alimentação em corrente contínua ou alternada; número de fases; quantidade de polos; e rotor externo ou interno ao estator, por exemplo, são intrínsecas à definição do motor utilizado.

As constantes dos motores estão fortemente relacionadas à sua caracterização. São elas:

- *Kv*: constante de velocidade;
- *Kt*: constante de torque;
- *Ke*: constante de força contraeletromotriz;
- *Km*: constante de tamanho do motor.

Kv é a constante de velocidade do motor, relacionando a vazio do motor com a tensão de pico aplicada a seus enrolamentos; *Km* representa a constante de tamanho do motor, sendo usada para a escolha do tamanho do motor para dada operação; *Kt* é a constante de torque do motor, sendo utilizada para calcular a corrente de armadura quando o motor aplica certo valor de torque; e *Ke* representa a constante de força contraeletromotriz do motor, tendo a mesma utilidade de *Kt*, geralmente utilizando unidades do Sistema Internacional de Medidas (SI) (WIKIPEDIA, 2022).

As Equações 2.2 e 2.3 relacionam as constantes em valores no SI.

$$Km = \frac{\tau}{\sqrt{P}} = \frac{Kt\ I}{\sqrt{P}} \tag{2.2}$$

$$Kv = \frac{\omega_{vazio}}{V_P} = \frac{1}{Ke} = \frac{1}{Kt} = \frac{I_a}{\tau} \tag{2.3}$$

onde τ é o torque gerado pelo motor; P é a potência elétrica (perda por efeito Joule) do motor; I_a é a corrente de armadura no motor; ω_{vazio} é a velocidade angular do motor sem carga; V_P é a tensão de pico no motor (WIKIPEDIA, 2022).

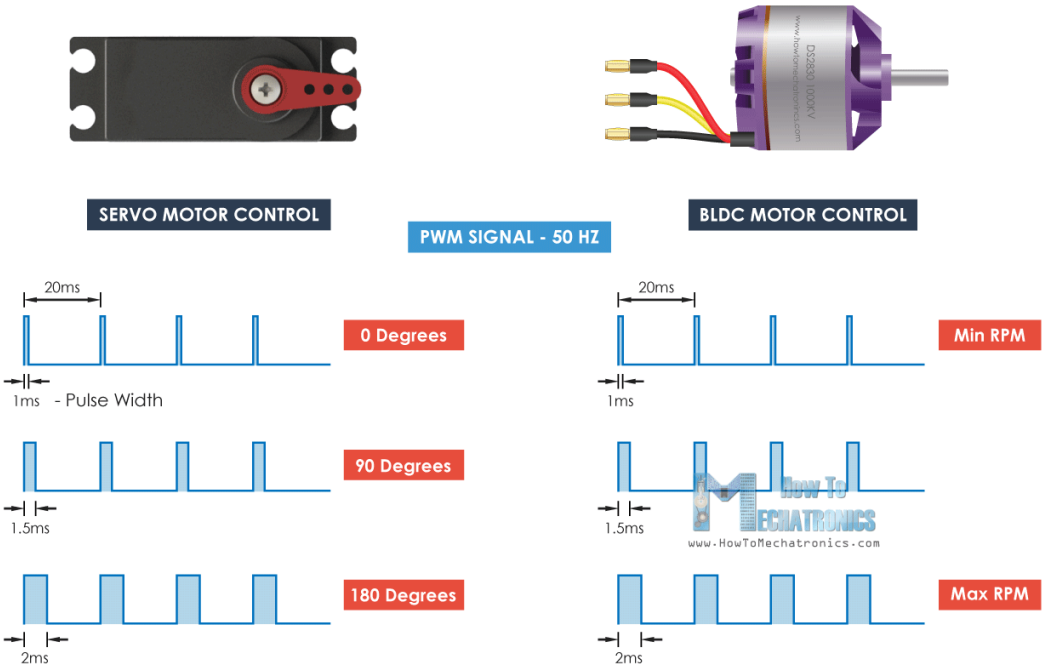
Em *drones*, são comumente utilizados motores trifásicos de corrente contínua, preferencialmente com índices de Kv altos para que sejam geradas altas velocidades nas hélices e, com isso, maior empuxo para a movimentação da aeronave.

2.4.2.3 ESC

O controlador eletrônico de velocidade é o responsável de enviar os sinais para o motor sem escovas. Ele recebe como entrada um sinal modulado por largura de pulso (PWM - do inglês, Pulse Width Modulation) e possui como saída o sinal trifásico que configura a velocidade de rotação dos motores.

O sinal PWM é semelhante ao de controle de um servomotor, com frequência nominal de 50 Hz, e intervalo válido de 5% a 10% de *duty cycle*, com nível típico de tensão lógica de 3.3 V ou 5 V que configuram desde o acionamento em repouso até a velocidade máxima do motor. A Figura 11 compara os efeitos de um mesmo sinal PWM enviado a um servomotor e a um ESC conectado a um motor BLDC.

Figura 11 – Sinal PWM para servomotor e ESC



Fonte: (DEJAN, 2019a)

O ESC também é responsável por prover a potência necessária para os motores, que necessitam de correntes de alimentação com ordem de grandeza duas vezes maior do que a que percorre o sistema de controle implementado na MiniZed (com máximo em torno de 0.3 A). Isso é feito com o auxílio de uma fonte de tensão externa que possa fornecer essa corrente para o ESC.

2.4.3 Processamento Embarcado

A placa de desenvolvimento MiniZed, utilizada na bancada de ensaios, conta com um *chip* Zynq 7Z007S da linha Zynq-7000, que integra as arquitetura ARM e FPGA empregadas no controlador desenvolvido.

O *chip* Zynq permite a integração da unidade central de processamento com processamento de sinais digitais e o desenvolvimento de partes padronizadas para aplicações específicas. Além disso, por possuir um único núcleo, o consumo de energia é baixo, tornando as placas que contam com a Zynq boas escolhas para uso em sistemas embarcados. Outro benefício trazido pelo *chip* é o elevado grau de controle permitido, o que possibilita que as aplicações desenvolvidas atendam a diversos critérios de projeto (XILINX, 2011).

Entretanto, o único núcleo do *chip* reduz consideravelmente seu poder de processamento. Em aplicações com grande quantidade de dados ou com requisitos de processamento elevados, pode não ser uma solução adequada. Ademais, a Zynq traz grau de complexidade mais elevado ao desenvolvimento, principalmente quando comparado a soluções baseadas

em Arduino (ARDUINO, 2015), que possui um número maior de usuários e há mais material para embasamento seja em fóruns *online* ou em canais oficiais, com soluções prontas para diversas aplicações.

2.4.4 Sensores externos

Quando se fala em *drones*, a ideia de sensores capturando dados sobre posicionamento angular, velocidade ou altitude fixados no solo ou em outros objetos que não sejam o próprio UAV não parece ser viável. No entanto, é importante que esses dados sejam bem monitorados e tratados, visto que qualquer distúrbio não capturado pode gerar falhas catastróficas, como discutido anteriormente.

Em uma bancada de testes, cujo objetivo é o estudo de diversos componentes e comportamentos de um *drone*, a presença de sensores com referências externas é uma alternativa para verificar a acurácia dos dados obtidos pelos sensores internos ao sistema, propensos a erros.

Esses sensores externos podem ser eletrônicos, como um potenciômetro, ou mais analógicos, como mecanismo envolvendo um transferidor e um ponteiro, por exemplo. Esses são exemplos de formas para que o pesquisador obtenha dados de posição angular no caso de uma bancada de testes para *drones* com 1 grau de liberdade, como a bancada desenvolvida neste trabalho.

2.4.5 Consumo de energia

Quadricópteros comerciais comumente possuem tempo de voo de aproximadamente vinte minutos (CIOBANU, 2021) quando não usados competitivamente em corridas aéreas, que podem reduzir a duração do voo devido ao elevado uso dos motores (GETFPV, 2018a). Assim, com as baterias comerciais tendo especificação de carga em torno de 1300 mA-h (GETFPV, 2018a), é possível notar que o consumo de energia é relativamente alto, sendo necessárias diversas baterias para permitir sua troca de forma a permitir o uso contínuo do UAV.

Para a bancada de testes, optou-se pelo não uso de baterias para a alimentação dos componentes. Dado que o projeto busca trazer uma plataforma para teste de controle, o uso de uma fonte de tensão ligada à tomada fez-se mais conveniente, não havendo a necessidade de trocar ou recarregar baterias durante o uso da bancada. Dessa forma, caso haja a necessidade de tornar os experimentos mais realistas, exibindo o efeito da descarga das baterias, pode-se fazer a substituição da fonte por um conjunto de baterias.

2.5 Bancadas de teste

Como discutido anteriormente, o uso de *drones* necessita de cuidados tanto com o equipamento utilizado quanto com pessoas e animais que possam ser atingidos pelo UAV de forma a seguir a legislação vigente a respeito do seu voo, seja qual for sua aplicação. Assim, uma malha de controle faz-se necessária para garantir que tais cuidados sejam tomados.

O controle, por sua vez, é possível por meio da medição e da atuação no sistema, isto é, são necessários sensores e atuadores, que devem ser corretamente estudados. O estudo de cada uma das partes pode ser realizado simultaneamente ou de forma separada. Os tópicos a seguir abordam diferentes técnicas para estudar o controle, a atuação e a medição de sistemas similares, simulando o voo de *drones*.

2.5.1 Elementos propulsores

O conjunto propulsor, composto por motor, comumente um motor trifásico BLDC, e hélice, representa o atuador principal de um *drone*. Devido à sua importância, existem equipamentos comerciais para testá-los. Um exemplo de tal equipamento é o encontrado em `https://www.wingflyingtech.com/motor-test/Drone-Test-Equipment.html`, responsável por medir informações de propulsão, como torque, empuxo e velocidade angular, além de outras informações relevantes, como corrente, tensão e eficiência do motor e da hélice testados.

Além disso, existem projetos no estilo "Faça você mesmo", que permitem obter de forma caseira resultados similares a equipamentos comerciais. Um exemplo de tal projeto pode ser visto em (IFORCE2D, 2016), que utiliza sensores e uma placa de desenvolvimento, entre outros materiais, de baixo custo, permitindo a realização de testes de forma mais barata.

2.5.2 Elementos sensores

Um UAV, assim como grande parte dos sistemas embarcados, utiliza elementos sensores para garantir suas condições de estabilidade e voo além de receber comandos de um controle remoto, se for o caso. Assim, uma IMU, com seus acelerômetro e giroscópio, permite a estabilização da aeronave. Um exemplo de teste para estudo e calibração de tal sensor pode ser visto na Figura 14 do artigo de Ranky et al. (RANKY et al., 2014).

Com o conjunto de teste, é possível também realizar a configuração de dinâmicas distintas, permitindo que sejam analisados eixos de rotação diferentes – dado que o sensor permite a leitura dos seus dados em três eixos perpendiculares entre si – e em diferentes condições, como com a alteração da velocidade angular ou com métodos variados de leitura dos dados do sensor.

2.5.3 Testes conjuntos

Equipamentos para estudos e testes com *drones* são essenciais para que o desenvolvimento desses UAVs seja cada vez mais preciso, seguro e eficaz. Sendo assim, a criação e o investimento nessas ferramentas têm se mostrado cada vez mais necessários. Um exemplo dessas plataformas de teste é o FFT GYRO 600 PRO (EUREKA DYNAMICS, 2018) da empresa mexicana Eureka Dynamics, que permite testes nos 3 graus de liberdade, utilizando uma estrutura circular rotatória. Apesar de não ser possível realizar translação com esse produto, os estudos se tornam já bem próximos de um funcionamento real de um *drone*, pelo fato das limitações em se tratando de rotação serem bastante reduzidas.

Figura 12 – Bancada de teste FFT Gyro 600 PRO

Fonte: (EUREKA DYNAMICS, 2018)

Não foram encontrados exemplos de plataformas que permitissem o teste total em translação. No entanto, há projetos acadêmicos para estudo de 4 graus de liberdade, com translação na vertical, a partir da construção de uma bancada de teste. Exemplos disso são os trabalhos *Development of the Test Platform for Rotary Wing Unmanned Air Vehicle* (YÜZGEÇ et al., 2016), da Universidade de Bilecik, na Turquia, ilustrado pela Figura 13, e *Quadricóptero 4DOF: desenvolvimento, modelagem e controle* (BARBOSA, 2017), ilustrado pela Figura 14, da Escola Politécnica da Universidade de São Paulo.

Figura 13 – Bancada de teste da Universidade de Bilecik, com translação na vertical

Fonte: (YÜZGEÇ et al., 2016)

Figura 14 – Bancada de testes com quatro graus de liberdade

Fonte: (BARBOSA, 2017)

Há também sistemas completamente embarcados em um *drone* comercial, realizando o controle dos movimentos de rolagem, arfagem e guinada, bem como o movimento horizontal e vertical da aeronave. Exemplo de tal sistema embarcado pode ser encontrado na dissertação de Madruga (MADRUGA, 2018). Madruga utiliza a tecnologia do módulo Raspberry Pi para o processamento dos dados e controle do quadricóptero visto na Figura 15.

Figura 15 – Aeronave de testes utilizada pro Madruga

Fonte: (MADRUGA, 2018)

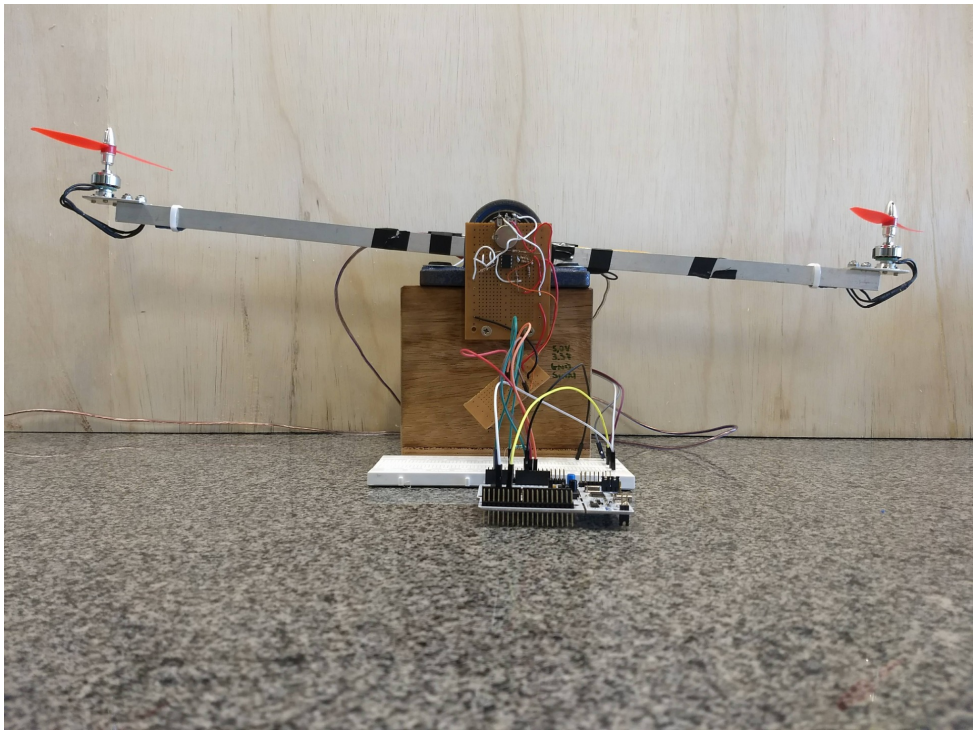
Por motivos de simplicidade, custo e evolução contínua do projeto, foi construída uma bancada de ensaios com apenas um grau de liberdade.

2.5.4 Trabalhos relacionados

Tomando um par de hélices de um *drone*, é possível realizar uma montagem que simule um dos eixos de rotação do equipamento, isto é, com um único grau de liberdade. Com esse tipo de montagem, pode-se testar o movimento do *drone* de forma simplificada, além de mais segura e barata.

A Figura 16 mostra esse tipo de montagem, realizada em um projeto da Universidade Federal de São Carlos (UFSCar) por Campos e Hernandes (CAMPOS; HERNANDES, 2018). Em sua montagem, foram utilizados materiais simples, de forma a obter um produto final de baixo custo e replicável, com foco na possibilidade de utilização educacionalmente.

Figura 16 – Bancada de testes utilizada na UFSCar



Fonte: (CAMPOS; HERNANDES, 2018)

A modelagem utilizada por Campos e Hernandes segue a Equação 2.4.

$$\left(I_{xx} + 2mL^2\right)\ddot{\Phi} + B_{loss}\dot{\Phi} = L\left(T_1 - T_2\right) \tag{2.4}$$

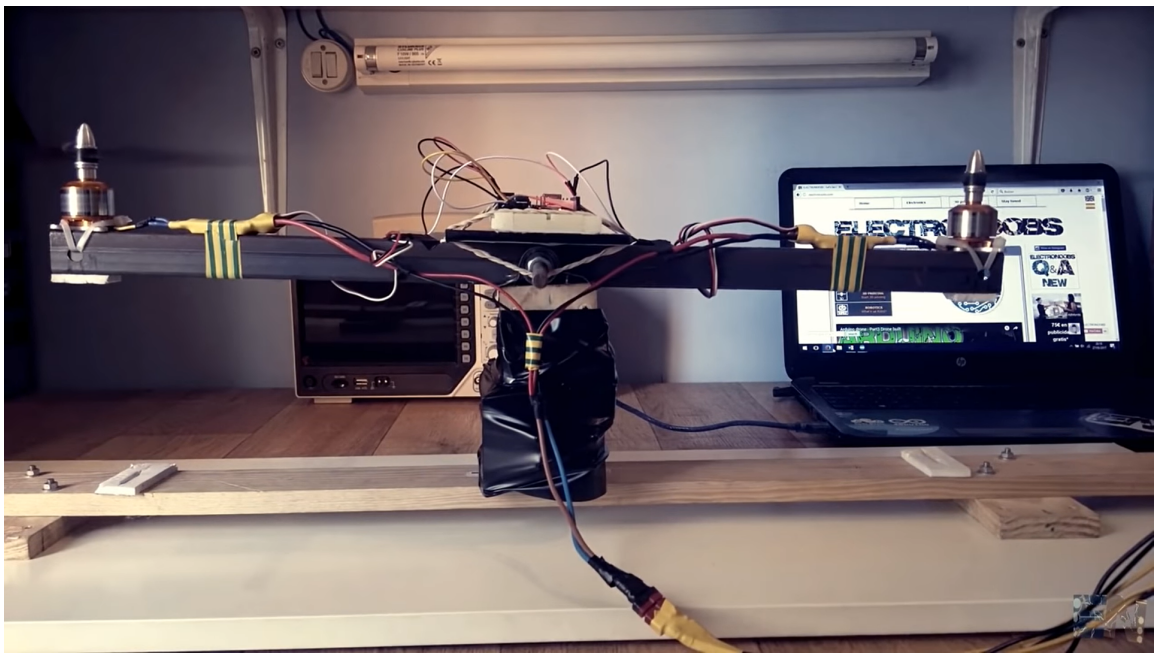
onde I_{xx} representa o momento de inércia da viga da gangorra; m é a massa de cada conjunto propulsor (motor e hélice); L é a distância do centro de rotação aos propulsores; Φ é a posição angular da gangorra – sendo $\dot{\Phi}$ e $\ddot{\Phi}$ a velocidade e a aceleração angulares, respectivamente –; B_{loss} é uma componente que representa as perdas do sistema, como o atrito nos mancais de rolamento e a resistência do ar ao movimento; e T_1 e T_2 são as forças de empuxo geradas por cada conjunto de propulsão.

Os materiais utilizados nessa construção foram:

- Motores BLDC AX-1806 com índice $Kv = 2500\,{}^{rpm}/_{V}$
- Hélices ABS 5030
- ESC RC06666 de corrente máxima $I_{max} = 12\,A$
- Placa de desenvolvimento ST Nucleo F411RE
- Potenciômetro para leitura da posição angular

É possível encontrar também vídeos instrucionais de montagens similares, como a da Figura 17 (ELECTRONOOBS, 2017). Assim como no caso da UFSCar, é utilizada arquitetura simples, neste caso, baseada em Arduino, para realizar o controle do eixo simulado.

Figura 17 – Bancada de testes de vídeo tutorial



Fonte: (ELECTRONOOBS, 2017)

Nessa bancada de ensaios, o autor foca o controle de posição angular por tentativa e erro, alterando os parâmetros de um controlador proporcional integral derivativo (PID), verificando o efeito de cada alteração, sem fornecer a modelagem matemática da dinâmica utilizada.

Para essa construção, os materiais utilizados foram:

- Motores BLDC A2212/13T com índice $Kv = 1000\,{}^{rpm}/_{V}$
- Hélices plásticas 1245
- ESC de corrente máxima $I_{max} = 20\,A$
- Placa de desenvolvimento Arduino Uno
- IMU MPU-6050 para leitura da posição angular

Em todos os casos apresentados, o controle dos motores das bancadas é feito pelo uso de controladores de arquiteturas comuns, como Arduino, que possui, relativamente, mais simples implementação. Com muitos exemplares de códigos prontos na linguagem de Arduino e diversos cursos e tutoriais disponíveis, trata-se, de fato, de uma arquitetura acessível e didática. Entretanto, por se tratar de um componente de uso geral, pode não

trazer as melhores soluções para as bancadas de ensaio. O uso da arquitetura FGPA visa a trazer os benefícios do *hardware* programável para a criação de uma solução específica.

A construção proposta pelo trabalho de Barbosa, mostrado na Seção 2.5.3, também utiliza elementos como motores BLDC de $Kv = 2300^{rpm}/_{V}$ e hélices plásticas (BARBOSA, 2017). Assim como nas bancadas com um único grau de liberdade, a placa de desenvolvimento também é baseada em Arduino, mais especificamente, uma placa Arduino Due. Entretanto, por se tratar de uma construção de elevado grau de complexidade e com ensaios de controle mais avançados do que o necessário para uma bancada educacional, seu escopo pode ser melhor abordado em trabalhos futuros.

3 Desenvolvimento e Resultados

Este capítulo descreve o projeto e o desenvolvimento da bancada de testes, desde sua estruturação mecânica até os periféricos em hardware de acionamento dos motores e o desenvolvimento do firmware utilizado para os ensaios de controle. Ademais, este capítulo traz os resultados dos experimentos realizados.

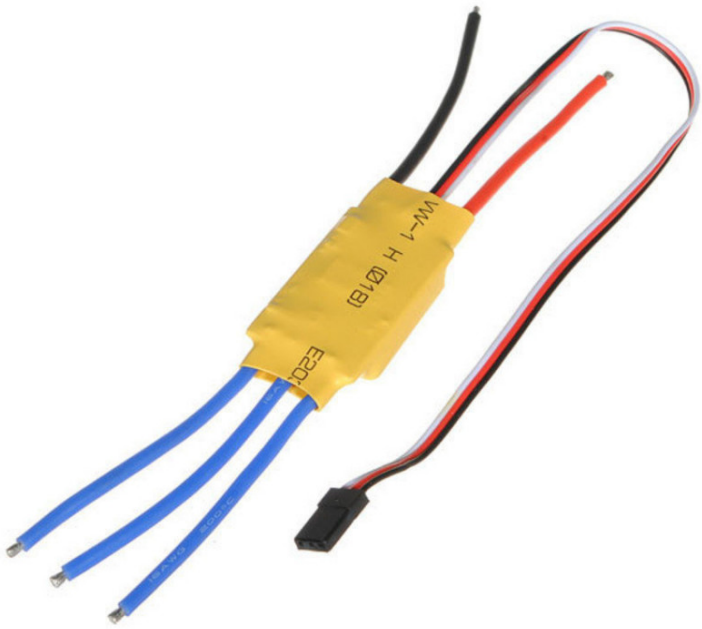
3.1 Seleção do conjunto propulsor

A primeira etapa para a construção da bancada de teste foi a escolha do conjunto de controlador eletrônico de velocidade (ESC), motores e propulsores. O custo e a disponibilidade no mercado somados às especificações do modelo apresentado pelo artigo de Campos e Hernandes (CAMPOS; HERNANDES, 2018) foram fatores essenciais na escolha dos componentes. A Figura 18 mostra o ESC originalmente escolhido para uso na bancada. Entretanto, devido a falhas no aparelho, foi necessário realizar a troca pelo produto da Figura 19.

Figura 18 – ESC escolhido para uso na bancada

Fonte: (ELECTROYA, 2017)

Figura 19 – ESC utilizado na bancada



Fonte: (BAÚ DA ELETRÔNICA, 2022)

Os atuadores da bancada, conjunto formado por motor e hélice, estão mostrados na Figura 20.

Figura 20 – Conjunto propulsor utilizado na bancada

Fonte: dos autores

Com o objetivo de reduzir custos dos componentes, todos os materiais escolhidos buscaram, além de atender às especificações da bancada, trazer baixo impacto no valor monetário total. [1]

Para a escolha dos motores, foram preferidos aqueles que possuíam índice Kv mais alto, pois permitem a rotação em mais altas velocidades com uma mesma tensão de alimentação quando comparado a motores com Kv de melhor valor. Os motores escolhidos foram os motores trifásicos sem escovas RS2205 da ReadyTosky com $Kv = 2300\,rpm/V$. Os motores trazem a vantagem de conseguirem atingir maiores velocidades do que motores escovados ou de corrente contínua, permitindo que a aeronave também atinja velocidades maiores e que as hélices gerem maior empuxo, dado que rotacionam mais rapidamente. Além disso, por não possuírem escovas, sua manutenção torna-se menos recorrente, pois há a redução do atrito entre partes mecânicas. Em contrapartida, os motores trifásicos sem escovas requerem o uso de controladores de velocidade (ESCs), o que aumenta o seu custo para uso em adição ao custo mais elevado do motor em si (MILLETT, 2020).

Já os ESCs foram escolhidos de forma a fornecer controle de velocidade robusto e compatível com os motores escolhidos. Apesar da possibilidade de construção de um ESC (como um exemplo pelo site Instructables: `https://www.instructables.com/Make-Your-Own-ESC/`), a aquisição de um controlador de velocidade comercial garante a repetibilidade da construção da bancada, além de reduzir o tempo utilizado em sua construção. Entretanto, os ESCs comerciais podem apresentar um custo relativamente elevado quando comparados aos valores de componentes eletrônicos usados na construção de um controlador de velocidade caseiro. Cada par composto por motor e ESC teve custo aproximado de R$ 92.50, dado que foram adquiridos em conjunto.

A escolha das hélices levou em consideração fatores como: compatibilidade com o eixo do motor; material de sua composição; passo; e número de pás. Com isso, as hélices escolhidas foram de plástico – que as torna leves, de baixo custo e menos perigosas caso se destrocem –, com diâmetro de furo de 5 mm – para permitir acoplamento com os motores – passo de 4 mm e com duas pás. As hélices utilizadas, portanto, estão entre as mais comumente encontradas em *drones* (GETFPV, 2018b), aproximando a bancada de um equipamento real. Hélices com mais pás trazem maior estabilidade ao voo e hélices de maior passo podem trazer maior velocidade. Porém, em ambos os casos, o seu uso pode aumentar o consumo de energia ou reduzir a eficiência dos motores. Com isso, a escolha de hélices 5040 de plástico traz um equilíbrio entre vantagens e desvantagens de cada possibilidade para esse componente. O custo de cada hélice chegou a menos de R$ 4.00, cumprindo o objetivo de reduzir custos da bancada.

[1] Devido à pandemia do vírus SARS-COV-2, todas as aquisições de materiais foram feitas pela internet para evitar o possível contato com o vírus em lojas físicas, restringindo a busca de materiais a vendedores *online*.

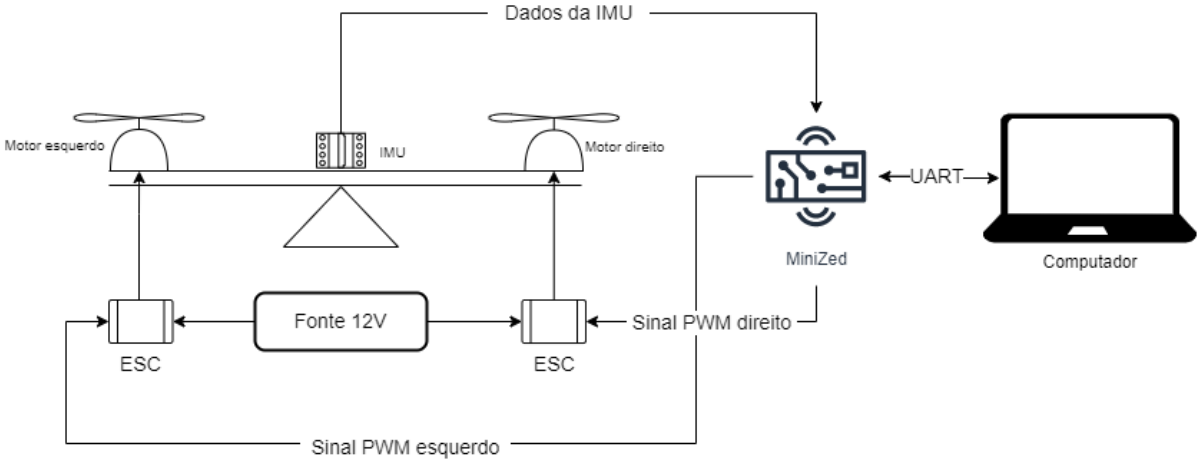
3.1.1 Componentes elétricos e eletrônicos

Para a construção da bancada, os principais componentes escolhidos podem ser condensados em:

- 1 MiniZed;
- 1 IMU MPU-6050 GY-521;
- 1 motor BLDC ReadyTosky Rs2205 Kv = 2300 sentido horário;
- 1 motor BLDC ReadyTosky Rs2205 Kv = 2300 sentido anti-horário;
- 1 hélice para *drone* no sentido horário;
- 1 hélice espelhadas para *drone* no sentido anti-horário;
- 2 ESCs 30 A;
- 1 fonte de tensão 12V, 50A 600W;
- conectores do tipo *jumper*;
- cabos elétricos.

A Figura 21 mostra um diagrama de como os componentes estão conectados entre si para funcionamento da bancada.

Figura 21 – Esquemático de conexão dos componentes



Fonte: dos autores

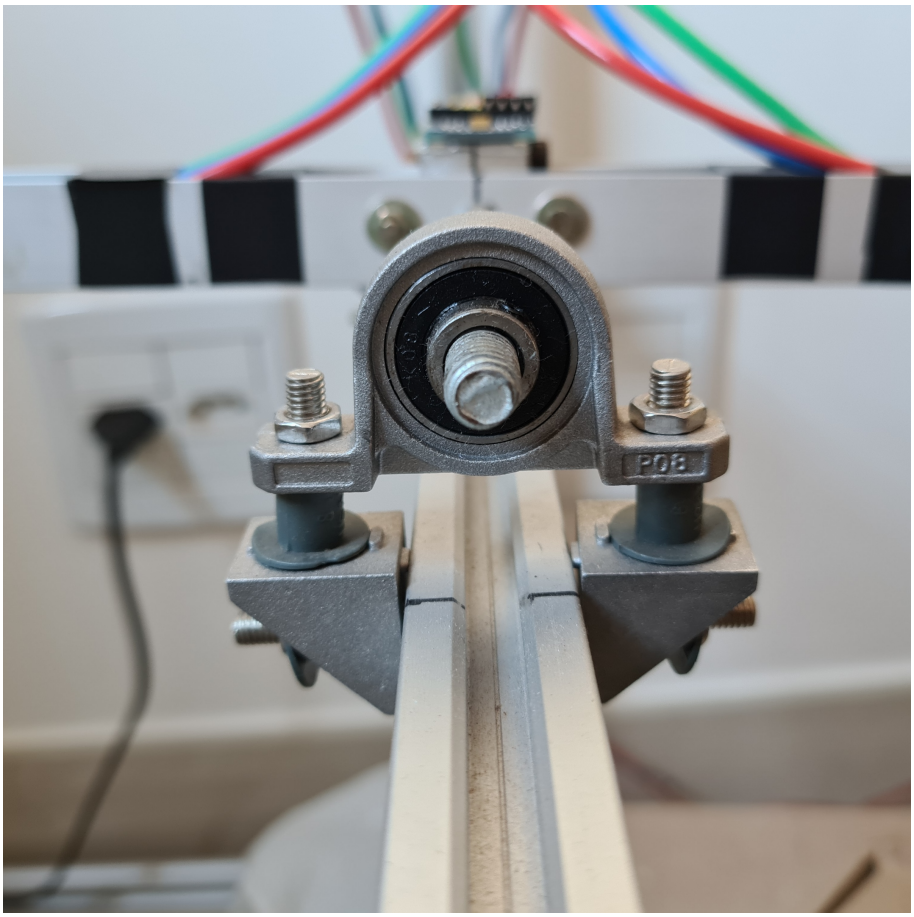
3.2 Projeto mecânico da bancada

A estrutura da bancada possui dimensões de 1×0.8×0.5 m, sendo a gangorra de aproximadamente 0.45 m de comprimento. Tais dimensões foram escolhidas para permitir que o movimento angular da gangorra atingisse amplitude de 90°, sendo 45°para cada lado a partir de sua posição horizontal de repouso. Além disso, as dimensões buscam reduzir

efeitos do vento produzido pelas hélices interagindo com o solo e com a tela de segurança colocada em volta, o que pode alterar a dinâmica do movimento da gangorra.

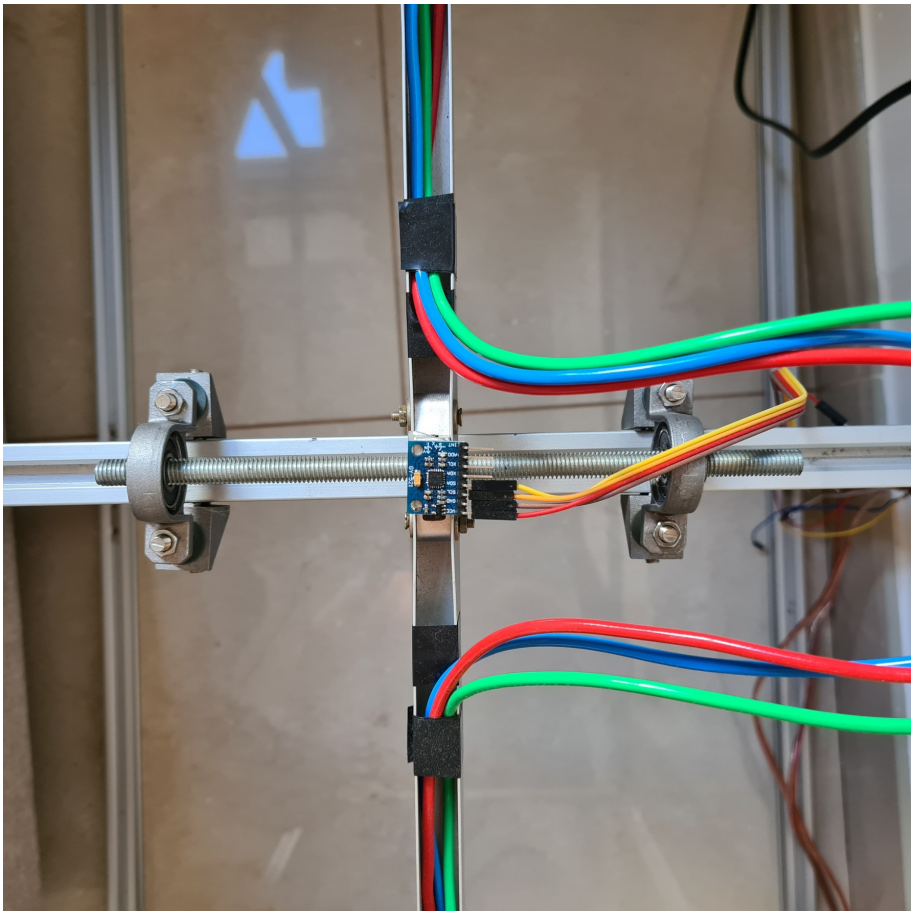
Para facilitar a fixação dos motores, além de comportar os cabos elétricos e oferecer resistência a um possível envergamento devido à força produzida pelos propulsores, a viga central da gangorra foi projetada com seção transversal em formato de “U”. Através dessa viga, passa uma barra roscada sustentada por mancais de rolamento, agindo como o eixo de rotação da bancada. Nota-se também que é importante ter cuidado ao escolher os cabos elétricos, pois eles influenciam o comportamento da gangorra, principalmente em se tratando de causar resistência a seu movimento. Além disso, tanto os motores quanto os ESCs possuem referências para a grossura dos cabos a serem escolhidos. Os cabos utilizados neste projeto foram escolhidos de acordo com essas referências, porém, inicialmente, foram utilizados os cabos das Figuras 22 e 23, que ilustram a montagem dos componentes supracitados.

Figura 22 – Detalhamento do mancal de rolamento utilizado



Fonte: dos autores

Figura 23 – Detalhamento do eixo de rotação da bancada



Fonte: dos autores

De modo a adicionar requisitos de segurança, foi criada uma tela de proteção em volta do equipamento principal. A tela busca trazer proteção contra eventuais partes que se desprendam e/ou quebrem, evitando que projéteis atinjam pessoas, animais ou equipamentos próximos à bancada. Ela tem o objetivo de formar uma gaiola e proteger os usuários da bancada de eventuais acidentes, tendo em vista que os motores giram a velocidades altíssimas, podendo causar, tais como *drones* reais, acidentes catastróficos. Além disso, a estrutura também atua protegendo contra incidentes como encostar nas hélices em movimento ou derrubar itens sobre o equipamento, permitindo que eventuais testes e medições sejam realizados com segurança.

Figura 24 – Estrutura da bancada com motores e hélices

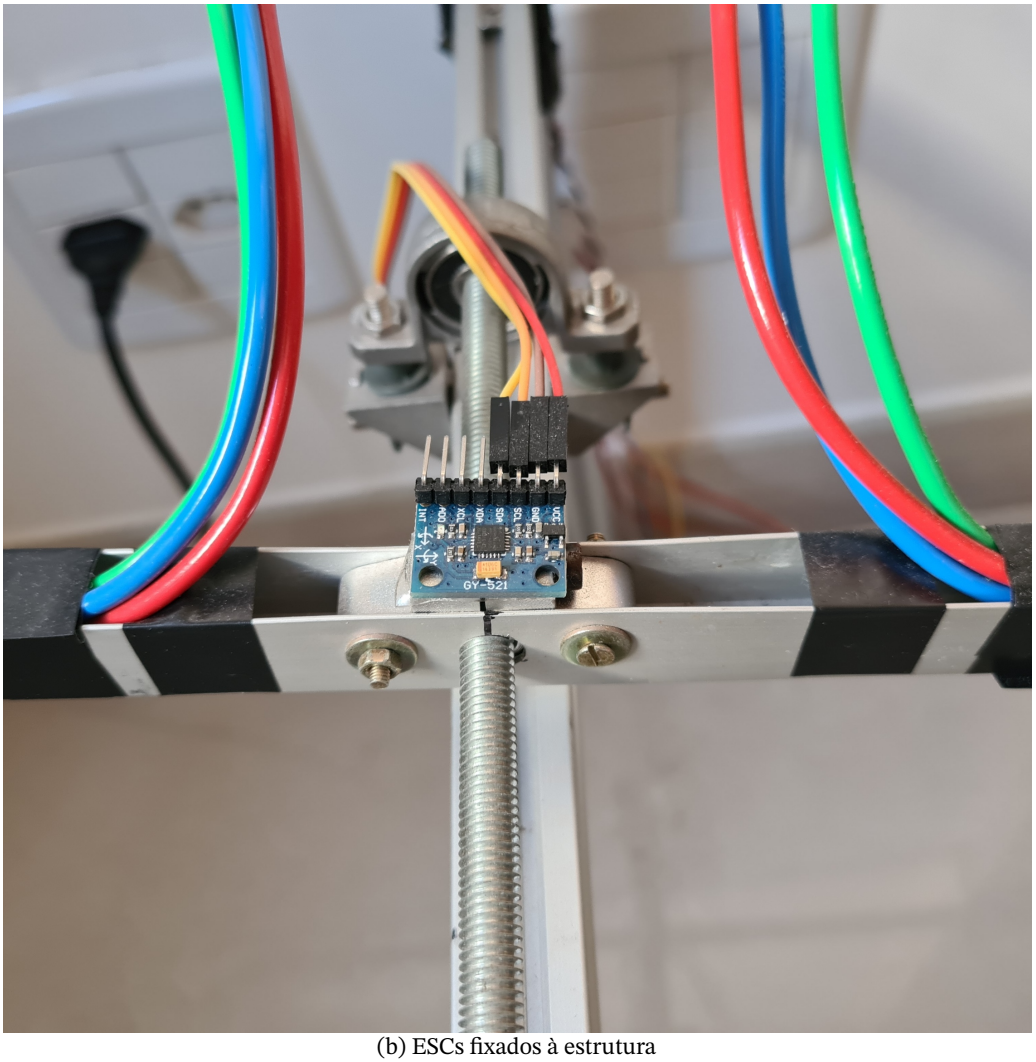
Fonte: dos autores

A gangorra é formada pelo perfil de alumínio em U, onde estão os motores, e é presa na bancada a partir de uma barra roscada, transpassando o perfil de alumínio e posicionada entre dois mancais de rolamento.

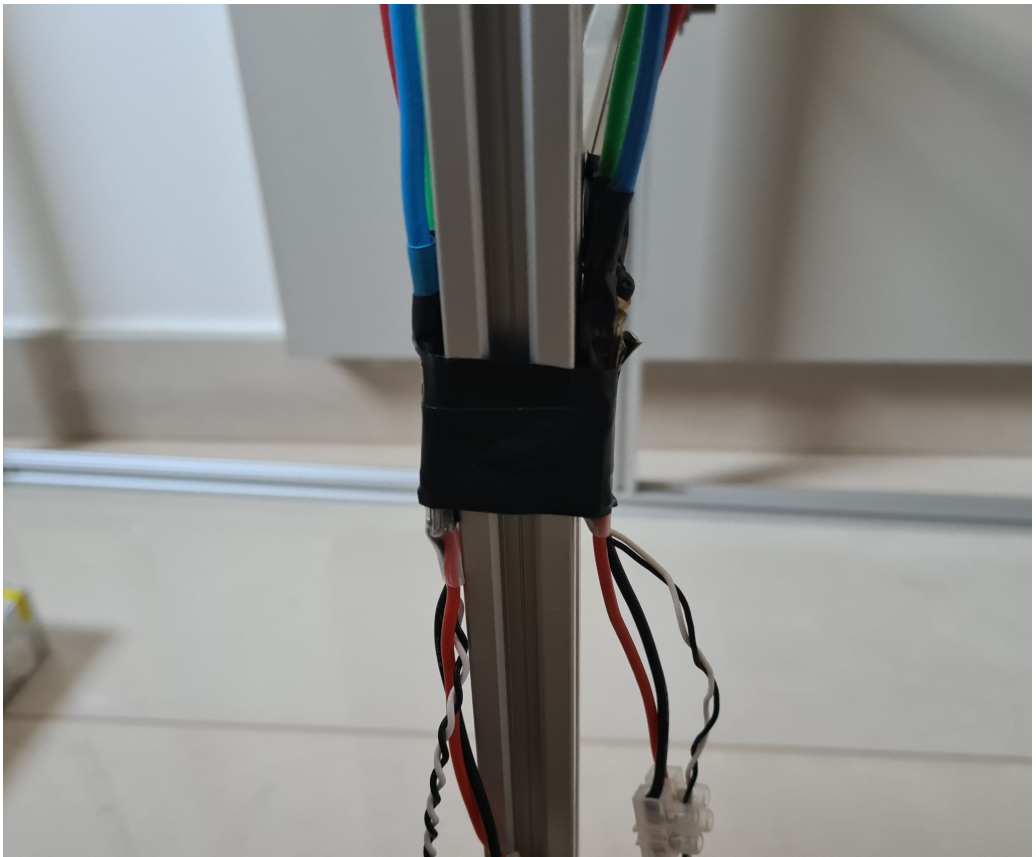
O sensor IMU, com giroscópio e acelerômetro, é acoplado ao centro do perfil em U a fim de informar a posição e a aceleração da gangorra; os ESCs são posicionados na estrutura vertical da bancada para acesso fácil, visto que são conectados à MiniZed para o controle do sistema. As Figuras 25a e 25b mostram como esses elementos foram agregados à bancada.

Figura 25 – Componentes acoplados à bancada

(a) IMU fixada à gangorra



(b) ESCs fixados à estrutura



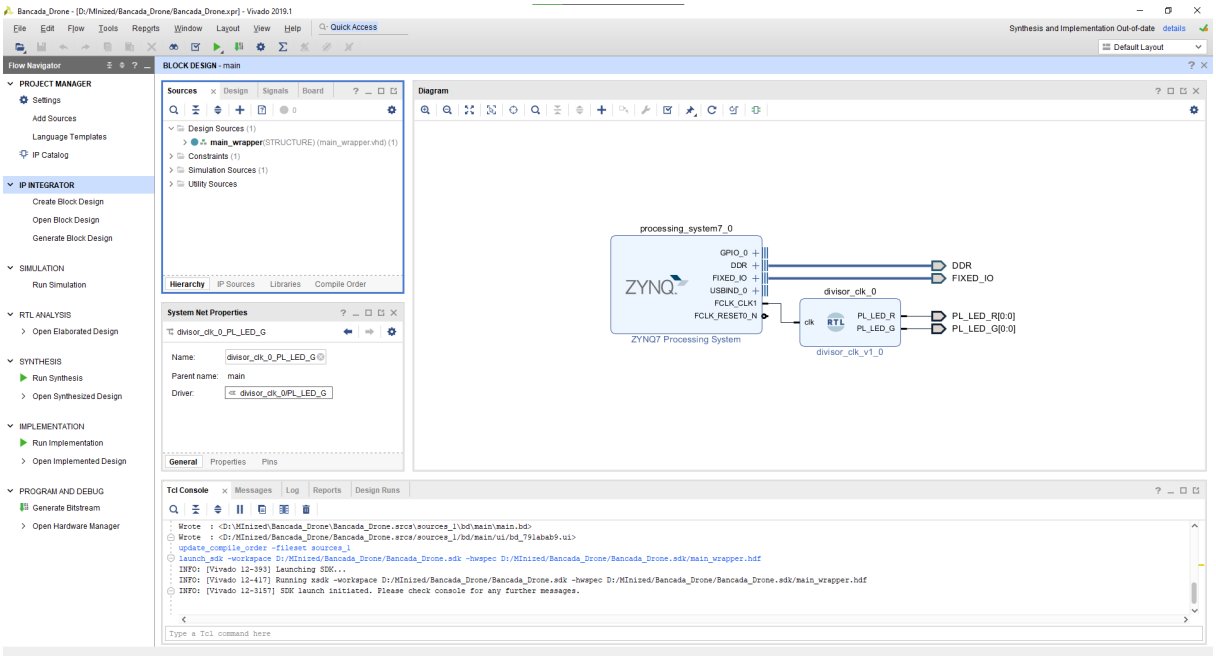
Os principais materiais utilizados na construção da estrutura da bancada de ensaios são:

1. 15 perfis de alumínio estrutural, dos quais:
    - 4 perfis de 1000 mm de comprimento;
    - 4 perfis de 500 mm de comprimento;
    - 6 perfis de 750 mm de comprimento;
    - 1 perfil de 470 mm de comprimento.
2. 1 perfil de alumínio em “U”;
3. porcas e parafusos;
4. prendedores para acoplamento das peças;
5. 2 mancais de rolamento;
6. 1 barra roscada;
7. tela mosquiteira;
8. velcro autoadesivo.

3.3 Desenvolvimento do Sistema Embarcado

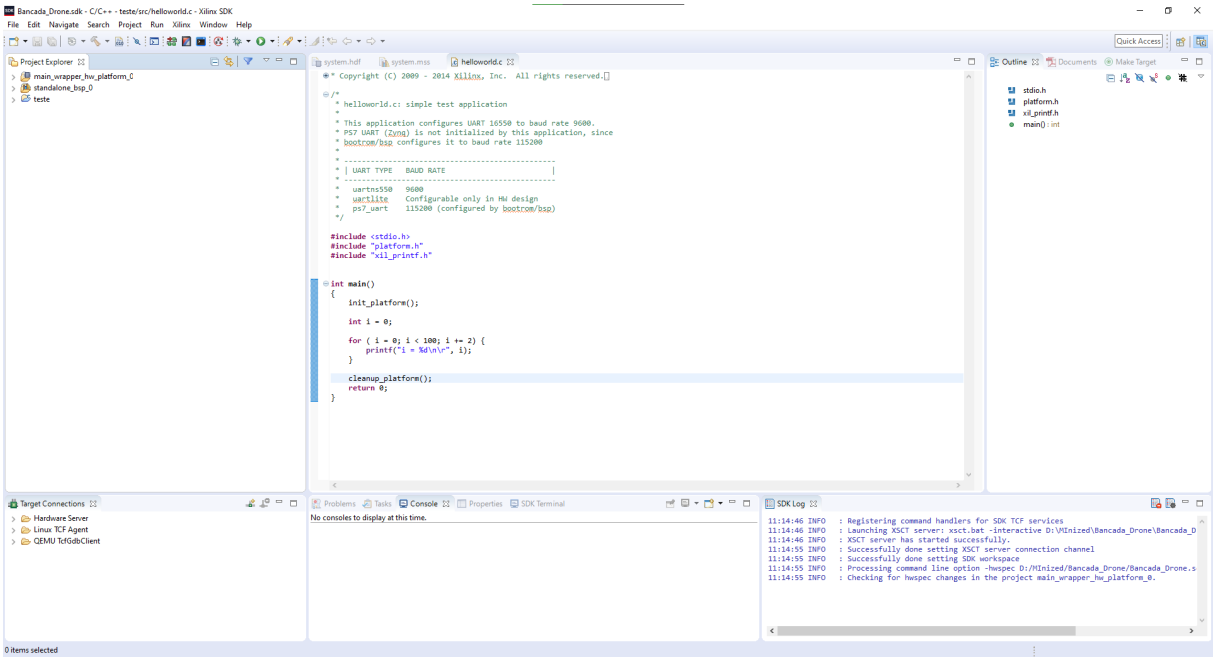
Para realizar as configurações do controlador, utilizaram-se os *softwares* de desenvolvimento da Xilinx Vivado e Xilinx SDK (*Kit* de Desenvolvimento de *Software* - do inglês, *Software Development Kit*). O primeiro é responsável principalmente pelas configurações gerais e permite o desenvolvimento em linguagens Verilog e VHDL (Linguagem de Descrição de Hardware VHSIC - do inglês, *VHSIC Hardware Description Language*, send VHSIC sigla para Circuito Integrado de Muito Alta Velocidade - do inglês, *Very High Speed Integrated Circuit*) para a arquitetura FPGA disponível na MiniZed; já o segundo é responsável pela programação, principalmente em linguagens C e C++, do processador ARM da placa de desenvolvimento, integrando as arquiteturas. Os programas utilizados podem ser vistos nas Figuras 26 e 27.

Figura 26 – Ambiente de desenvolvimento Vivado



Fonte: dos autores

Figura 27 – Ambiente Xilinx SDK

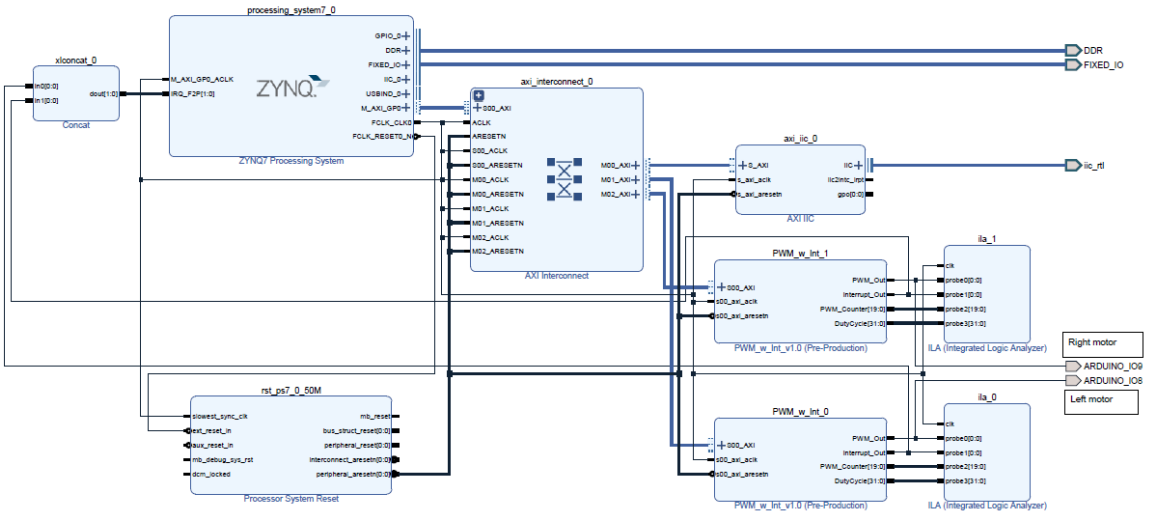


Fonte: dos autores

3.3.1 Desenvolvimento em FPGA

Utilizando o ambiente Vivado (Figura 26), foi construído o diagrama de blocos da Figura 28, caracterizando a utilização da arquitetura FPGA.

Figura 28 – Diagrama de blocos para uso na MiniZed



Fonte: dos autores

Pode-se notar na Figura 28, o processador – bloco *ZYNQ7 Processing System* –, que deve ser sempre instanciado, fazendo a interação com a Interface Extensível Avançada (AXI - do inglês, *Advanced eXtensible Interface*) – bloco *AXI Interconnect* –, explanada na Seção 3.3.4. Este, por sua vez, realiza a comunicação com os blocos responsáveis pela geração do sinal de PWM dos motores – *PWM_w_Int* – e pela leitura dos sensores via protocolo de Circuito Inter-Integrado (I^2C - do inglês, *Inter-Integrated Circuit*) – *AXI IIC*. Os outros blocos vistos na Figura servem para garantia do correto funcionamento do sistema principal, lidando com interrupções, verificando a lógica interna dos blocos e concatenado os sinais de interrupção enviados ao processador.

Muito da estrutura FPGA utilizada vem pronta do catálogo de núcleos de propriedade intelectual (IP - do inglês,*Intellectual Property*). Assim, fez-se necessário apenas a construção, em linguagens Verilog e VHDL, dos blocos de PWM para acionamento dos ESCs e as devidas configurações dos núcleos de IP para correto funcionamento do sistema.

3.3.2 Acionamento dos motores

Os motores utilizados – Figura 20 – são trifásicos e sem escovas, utilizando ímãs permanentes e enrolamentos para gerar movimento. Os ímãs do rotor alinham-se com o campo magnético gerado nos enrolamentos, permitindo a rotação em velocidades distintas com base na corrente que atravessa os enrolamentos.

Devido a seu princípio de funcionamento, faz-se necessário o uso de um ESC para a ativação e controle de velocidade de cada motor. O ESC atua medindo a força contraeletromotriz gerada nos enrolamentos para deduzir a posição do rotor e fazer o ajuste da velocidade (DEJAN, 2019b).

Com as altas velocidades e o empuxo gerado pelos propulsores, o ESC necessita de alta potência para seu correto funcionamento, como é o caso dos ESCs utilizados na bancada, classificados para uso com corrente de 30 A e tensão nominal no intervalo de $v_{in,min} = 5.6\,V$ a $V_{in,max} = 16.8\,V$, sendo alimentados por uma tensão de $V_{Source} = 12\,V$ fornecida pela fonte de tensão utilizada.

Para o acionamento dos controladores de velocidade, é necessário um sinal modulado por largura de pulso, isto é, um sinal PWM. Com o uso na plataforma MiniZed, é enviado um sinal digital de tensão de pico $V_P = 3.3\,V$ e frequência $f_{PWM} = 50\,Hz$, ou seja, de período $T_{PWM} = 20\,ms$, a cada um dos ESCs. Alterando-se a duração dos pulsos, faz-se a alteração da velocidade dos motores pelo ESC, sendo um pulso com duração $l_{min} = 1\,ms$ responsável por desligar o motor ligado ao ESC e de duração $l_{max} = 2\,ms$ responsável por ativar o motor em sua velocidade máxima (FLEMING, 2020). A relação entre a duração do pulso e o período do sinal caracteriza o *duty cycle* sendo utilizado. Assim, tem-se que $d_{min} = \frac{l_{min}}{T_{PWM}} = 5\%$ representa o *duty cycle* que desliga os motores e $d_{max} = \frac{l_{max}}{T_{PWM}} = 10\%$ o *duty cycle* que ativa os motores em velocidade máxima.

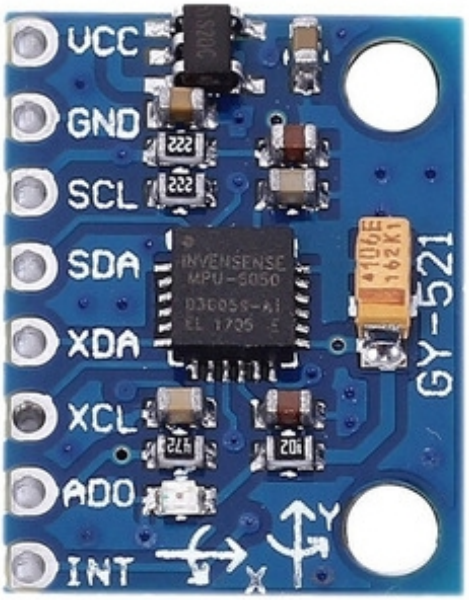
Como visto na Figura 28, existem, na parte FPGA, dois blocos distintos para envio dos sinais de PWM, sendo um para cada ESC – e cada ESC conectado a um único motor. Isso permite que os motores movimentem-se em velocidades distintas, necessário para o controle da gangorra.

O diretório denominado “PWM_w_Int” do repositório (vide anexo A) traz o código desenvolvido para ativação dos motores.

3.3.3 Leitura dos sensores

Para a obtenção da posição angular da gangorra, foram utilizados o acelerômetro e giroscópios disponibilizados pela IMU GY-521, um componente simples, porém confiável. O módulo pode ser visto na Figura 29.

Figura 29 – Módulo de IMU utilizado



Fonte: Adaptado de (EPÓRIO DO ARDUINO, 2022)

Para a leitura dos dados, é utilizado o protocolo I^2C, que fornece uma maneira eficaz de transporte de dados simultaneamente de diversos dispositivos (NXP SEMICONDUCTORS, 2021).

A leitura dos dados seguiu a especificação padrão do protocolo e do *datasheet* do aparelho, implementando a leitura dos dados sequenciais e em fila.

Para obter dados mais confiáveis, foi utilizado o filtro de Kalman com os dados do giroscópio e acelerômetro, realizando a fusão dos sensores.

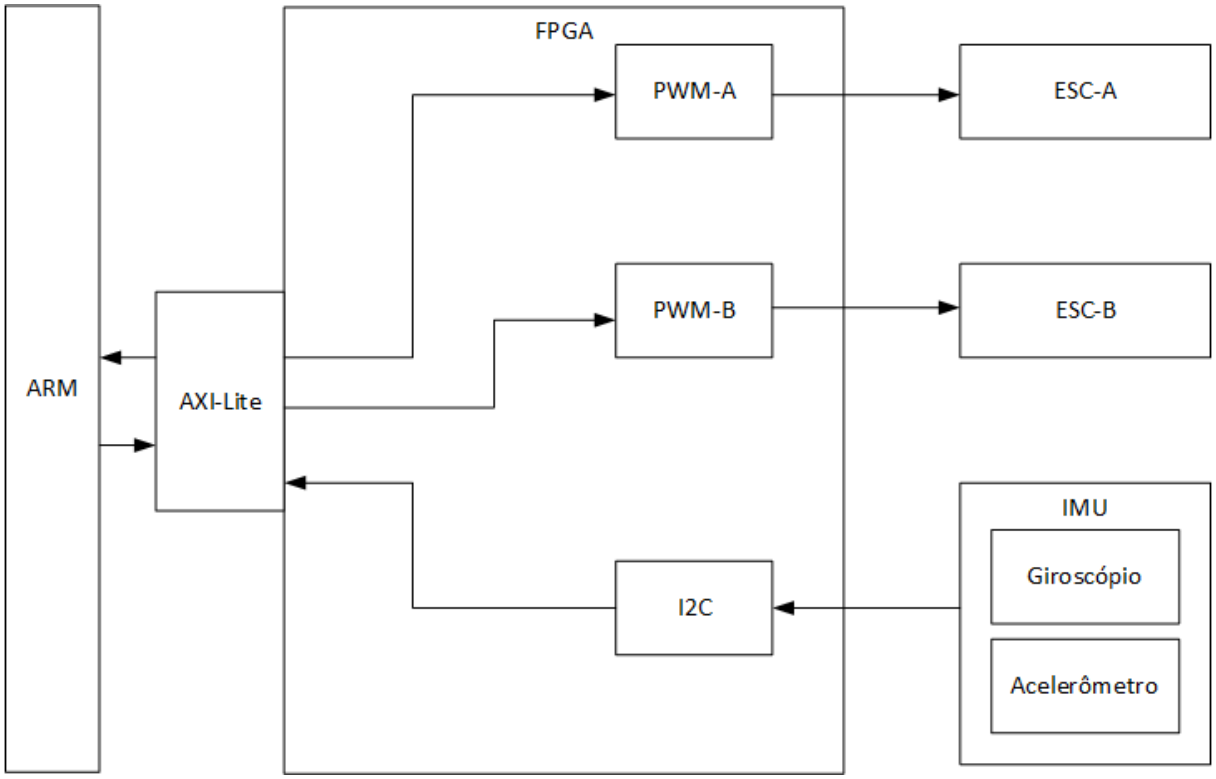
O código para leitura e fusão dos sensores foi contribuição de um colega (DUMONT; BRITO VIEIRA, 2022). O diretório com o código relativo aos sensores pode ser encontrado na pasta denominada “IIC_MPU” no repositório *online* (vide anexo A).

3.3.4 Interface AXI-Lite

A MiniZed, assim como outras placas de desenvolvimento da Avnet, utiliza a interface AXI-Lite para permitir a comunicação entre o processador ARM e a arquitetura FPGA (XILINX, 2017).

De forma simplificada, a Figura 30 mostra a relação entre as arquiteturas ARM e FPGA e como a interface AXI se situa entre ambas.

Figura 30 – Detalhe da interface AXI entre arquiteturas



Fonte: dos autores

A interface AXI permite a divisão entre dispositivos “mestres” e “escravos”, sendo os primeiros responsáveis por realizar a leitura e escrita de dados e os segundos por fornecer dados para leitura e onde são escritos.

A plataforma Vivado gerencia os blocos que utilizam AXI, dividindo-os na memória em endereços distintos e únicos para cada componente. Tais endereços são disponibilizados ao processador, permitindo chamadas aos componentes em cada endereço de memória e garantindo a interação entre arquiteturas. A fim de ilustração, os blocos *PWM_w_Int_0* e *PWM_w_Int_1* da Figura 28 ocupam, respectivamente, endereços de memória de 0x43C00000 a 0x43C0FFFF e de 0x43C10000 a 0x43C1FFFF, passíveis de serem alterados manualmente conforme fizer-se necessário.

3.3.5 *Firmware*

O *firmware* do projeto pode ser dividido entre a leitura dos sensores, feita via *hardware* em FPGA pelos blocos de leitura do protocolo I^2C, e o acionamento dos motores, também em FPGA pelos blocos de sinais PWM.

A leitura dos sensores é feita pela leitura dos dados em fila. O sensor envia dados à MiniZed, que os enfileira em uma estrutura de dados por ordem de chegada. O código verifica a existência de dados suficientes em fila para, então, acessá-los e removê-los da fila, atualizando-a.

Já o acionamento dos motores acontece de forma mais simples. O programa deve apenas enviar a quantidade de instantes em que o sinal deve permanecer no estado lógico alto levando em consideração o *clock* do processador. Com isso, o bloco para geração do sinal PWM realiza a divisão do *clock* em um frequência menor já estabelecida e ativa o sinal de saída a quantidade de vezes definida. Dessa maneira, o sinal atinge as porcentagens em que permanece ativo conforme necessário para ativação dos motores pelos ESCs.

3.4 Caracterização da Bancada

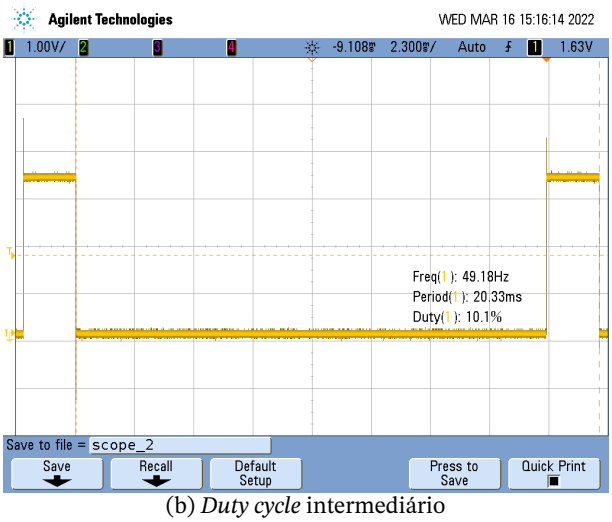
Para estudo de sistemas de instrumentação e controle, é necessário caracterizar os elementos da bancada. A seguir estão as caracterizações de diversos dos equipamentos utilizados.

3.4.1 Caracterização do sinal de PWM

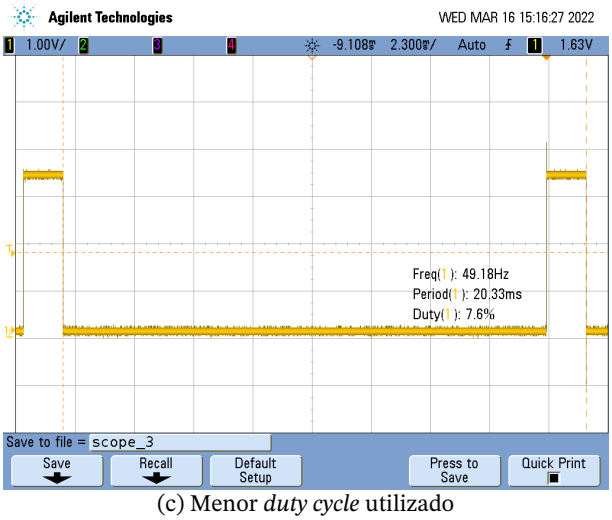
Como discutido anteriormente, os ESCs, que controlam a velocidade dos motores, utilizam um sinal de PWM como entrada para escolha da velocidade dos motores. De forma a obter a forma do sinal obtido através da MiniZed, foram realizados testes com um osciloscópio capturando a forma de onda com o circuito aberto, isto é, apenas capturando o sinal nos pinos de saída da placa de desenvolvimento. A Figura 31 mostra as capturas feitas pelo osciloscópio.

Figura 31 – Sinais de saída da MiniZed

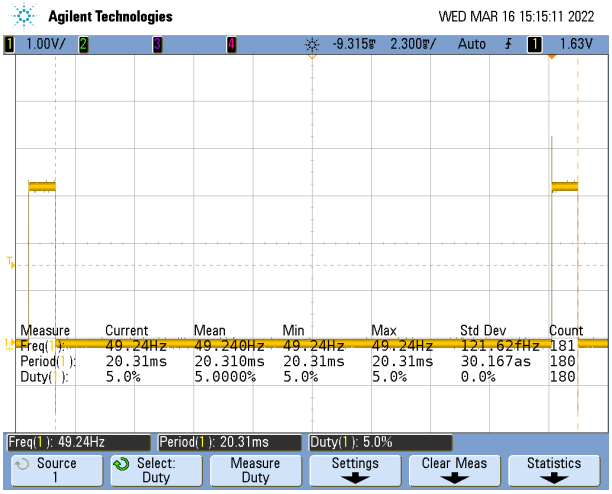
(a) Maior *duty cycle* utilizado



(b) *Duty cycle* intermediário



(c) Menor *duty cycle* utilizado



Fonte: dos autores

Com o objetivo de obter maior acurácia nos valores de *duty cycle*, foram utilizados os cursores do osciloscópio em conjunto com a função de *zoom*, permitindo o uso de valores

mais próximos da realidade para a calibração do conjunto propulsor e obtenção da função de transferência do sistema. Logo, a faixa de valores de *duty cycle* utilizada, que, teoricamente, estaria no intervalo $[5\%, 10\%]$ foi ajustada para o intervalo $[5.07\%, 10.11\%]$, sem alteração significativa no funcionamento dos motores.

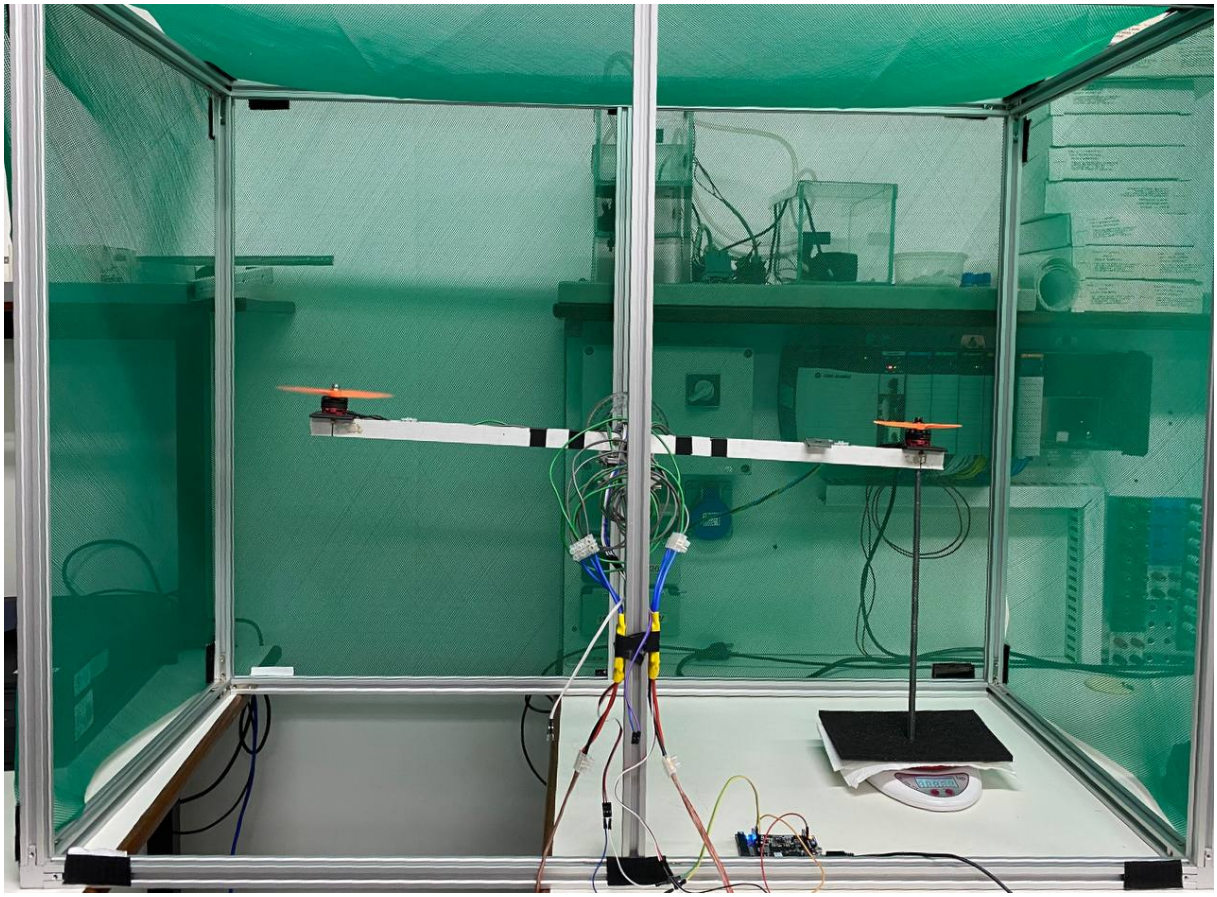
Nota-se pelas Figuras 31a, 31b e 31c que o sinal de PWM não atingiu a frequência de 50 Hz, mas sim de 49.2 Hz. Isso ocorre devido à forma como define-se o *clock* do sistema: o processador, bloco "*ZYNQ7 Processing System*" na figura 28, possui saltos discretos em que seu *clock* pode ser definido. Esse valor rege todo o sistema e é usado por todos os componentes. Para gerar o PWM, esse sinal – com frequência original de 51.6129 MHz – é dividido em uma frequência menor, também em saltos discretos e, dessa forma, não consegue atingir a frequência exata de 50 Hz. Entretanto, essa alteração na frequência não modificou a resposta dos ESCs, mantendo o comportamento esperado.

3.4.2 Caracterização do conjunto propulsor

Com o objetivo de caracterizar a força produzida pelos propulsores, montou-se um experimento para medição do empuxo gerado por um dos motores. Com o auxílio de uma balança de cozinha, que possui resolução de 1 g, foi realizado o experimento como na Figura 32, onde o motor direito permaneceu desativado enquanto os sinais foram enviados ao motor esquerdo. A fim de reduzir o efeito do fluxo de ar interagindo com a superfície onde a bancada se encontrava, foi deixado um vão proposital abaixo do motor a ser ativado.

Com o experimento montado, utilizou-se um programa para enviar distintos e arbitrários sinais de *duty cycle* ao ESC, realizando, assim, medidas tanto para sinais crescentes e decrescentes em magnitude, buscando notar se há efeito considerável de histerese, que não foi encontrado.

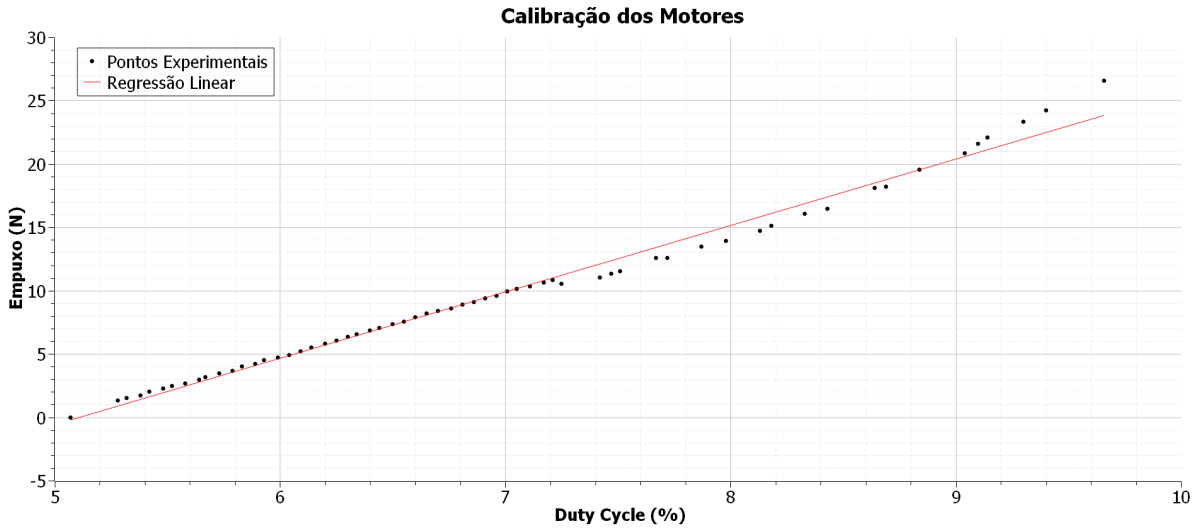
Figura 32 – Experimento para aquisição da força de empuxo



Fonte: dos autores

A partir das medições, em gramas-força, da força de empuxo gerada pelo conjunto de motor e hélice, foi realizada a conversão da força para unidades do Sistema Internacional de Medidas (Newton) com uso da aceleração da gravidade na Universidade de Brasília, onde o experimento foi realizado: $g = 9.78088 m/s^2$ (MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÕES, 2014). Com 4 medições em cada ponto, foi calculada a média aritmética dos dados e, com o auxílio de programa computacional SciDAVis, foi feita a regressão linear da curva que passa pelos pontos encontrados. Dessa forma, obteve-se a curva da Figura 33, que segue a Equação 3.1 – onde $F_e(d)$ é a força de empuxo gerada pelo motor e d é o *duty cycle*.

Figura 33 – Curva linearizada do empuxo do motor



Fonte: dos autores

$$F_e(d) = 5.246d - 26.826 \tag{3.1}$$

Nota-se que em torno de 7.3% de *duty cycle*, o comportamento do ESC foi alterado. Isso pode ser ocasionado devido a alguma tentativa de linearização da saída, feita pelo próprio ESC.

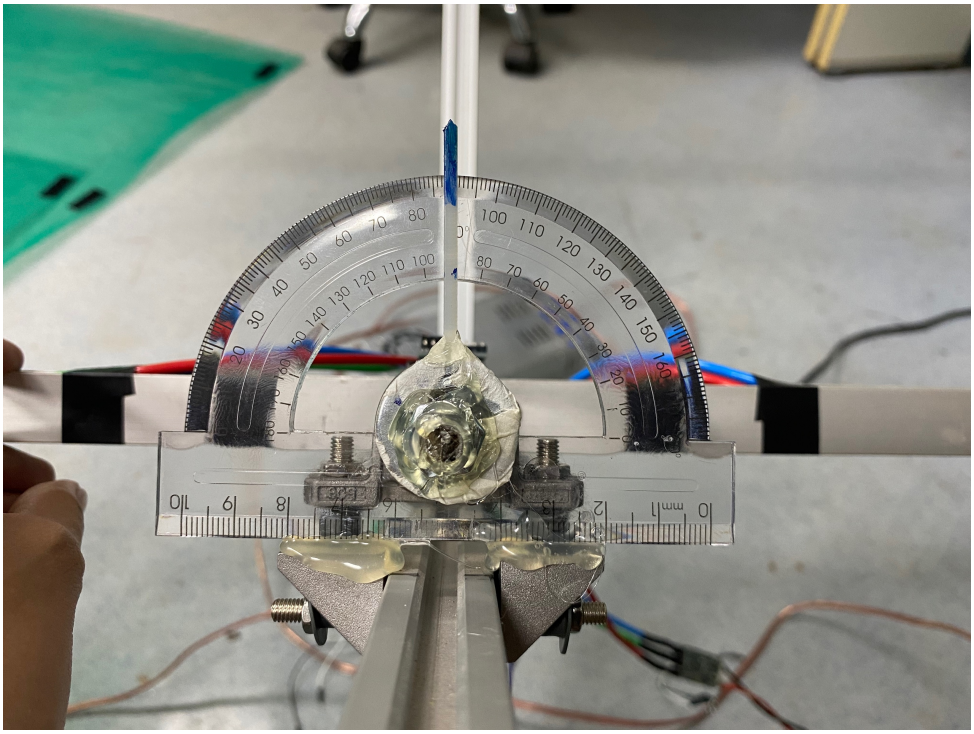
Como os motores permanecem estacionários ao utilizar valores de *duty cycle* inferiores a 5.3%, isto é, o controle de velocidade realizado pelos ESCs não é suficiente para mover o rotor, o experimento utilizou apenas valores superiores a esse limite. Além disso, de forma a evitar danos aos componentes, como os ESCs ou os próprios motores, evitou-se o uso de valores de *duty cycle* superiores a 9.66%, pois geravam velocidades perigosamente altas.

3.4.3 Caracterização dos sensores

Utilizando o programa desenvolvido por um colega (DUMONT; BRITO VIEIRA, 2022), foi feita a calibração dos sensores de acelerômetro e giroscópio disponíveis na IMU, bem como a fusão dos sensores via filtro de Kalman.

De forma a obter medidas de posição angular inerciais, foram acoplados um transferidor e um ponteiro ao eixo de rotação. Assim, à medida que a bancada gira, é possível obter a posição angular da bancada como mostram a Figura 34 com base em um referencial inercial.

Figura 34 – Transferidor acoplado à bancada



Fonte: dos autores

A calibração toma a posição da IMU durante o processo como o ponto de referência. Assim, pode-se definir qualquer posição angular da gangorra como o ponto inicial. Por simplicidade, definiu-se a posição horizontal – isto é, com o ponteiro marcando o ângulo de 90°– como referência.

A partir de uma sequência de calibrações, registraram-se os valores de viés dos sensores e utilizou-se a média simples desses valores para a calibração do sensor. Dado que os valores são específicos para cada sensor e montagem, esses precisam ser alterados para diferentes bancadas, ou seja, recalibrados.

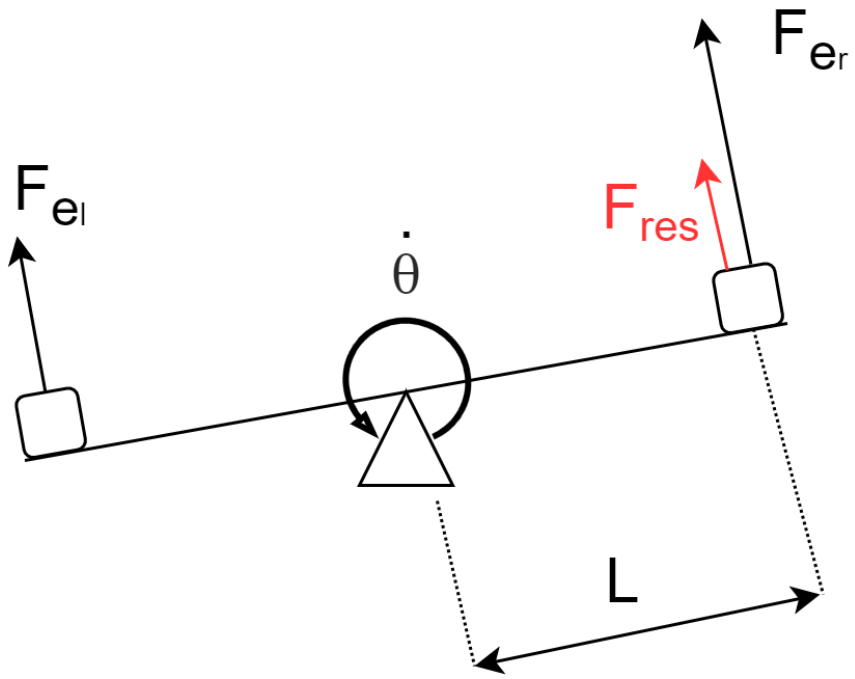
3.4.4 Função de transferência do sistema

Considerando que o empuxo gerado pelo conjunto propulsor é sempre perpendicular ao braço de alavanca da gangorra, a segunda lei de Newton para rotação nos dá a relação da Equação 3.2, onde $F_{res}(d)$ é a força resultante dada pela diferença entre os empuxos gerados por cada propulsor em função do *duty cycle* enviado aos motores; L é o comprimento do braço de alavanca – considerando que a gangorra está centralizada e ambos os motores encontram-se à mesma distância do centro de rotação –, isto é, a gangorra tem comprimento como um todo de $2L$; J é o momento de inércia da gangorra; $\dot{\theta}$ é sua velocidade angular; $\ddot{\theta}$ é sua aceleração angular; e B é a componente do atrito viscoso do sistema.

$$F_{res}(d)L = J\ddot{\theta}(t) + B\dot{\theta}(t) \tag{3.2}$$

A Figura 35 representa o diagrama de corpo livre da planta com as forças atuantes no sistema, as dimensões relevantes e o sentido de rotação considerado positivo.

Figura 35 – Diagrama de corpo livre da gangorra



Fonte: dos autores

A partir da Equação 3.1 para cada um dos propulsores, é possível obter a relação da Equação 3.3.

$$\begin{aligned} F_{res}(d) &= F_{e_r}(d_r) - F_{e_l}(d_l) \\ &= 5.246(d_r - d_l) \cancel{-26.826 + 26.826} \\ &= 5.246\, d_{res} \end{aligned} \tag{3.3}$$

Tomando a transformada de Laplace da Equação 3.2 em conjunto com a Equação 3.3 e isolando os termos relevantes, obtém-se a relação da Equação 3.4, função de transferência relacionando o ângulo de saída ao *duty cycle* resultante – diferença entre o valor enviado ao motor direito e ao esquerdo.

$$
\begin{aligned}
F_{res}(s)L &= Js^2\Theta_{out}(s) + Bs\Theta_{out}(s) \\
\Rightarrow 5.246\, d_{res}(s)L &= (Js^2 + Bs)\Theta_{out}(s) \\
\Rightarrow \frac{\Theta_{out}(s)}{d_{res}(s)} &= \frac{5.246\, L}{Js^2 + Bs}
\end{aligned} \tag{3.4}
$$

Para calcular o momento de inércia total do sistema, J, serão considerados o eixo e os motores. De acordo com Young e Freedman (YOUNG; FREEDMAN, 2016), pode-se afirmar que J é a soma dos momentos de inércia do eixo e dos motores, sendo caracterizados a seguir.

Considerando o eixo que suporta os motores como uma barra delgada, seu momento de inércia é dado pela Equação 3.5 (YOUNG; FREEDMAN, 2016), onde $2L$ é o comprimento do eixo.

$$
J_{eixo} = \frac{m_{eixo}(2L)^2}{12} = \frac{m_{eixo}L^2}{3} \tag{3.5}
$$

Já para cada conjunto propulsor, considerando-os como massas pontuais, tem-se que seu momento de inércia é dado pela Equação 3.6 (YOUNG; FREEDMAN, 2016).

$$
J_{propulsor} = m_{propulsor}L^2 \tag{3.6}
$$

Assim, unindo as Equações 3.5 e 3.6, o momento de inércia da gangorra é dado pela Equação 3.7, pois há dois conjuntos propulsores, um em cada lado da gangorra.

$$
\begin{aligned}
J &= J_{eixo} + 2J_{propulsor} \\
&= \frac{m_{eixo}L^2}{3} + 2m_{propulsor}L^2 \\
&= \left(2m_{propulsor} + \frac{m_{eixo}}{3}\right)L^2
\end{aligned} \tag{3.7}
$$

Com o auxílio da balança utilizada em 3.4.2 mediram-se as massas do conjunto propulsor e do eixo e, com o uso de uma trena, mediu-se o comprimento do braço de alavanca. Os dados foram compilados na Tabela 1.

Tabela 1 – Medições referentes aos componentes da gangorra

Variável	Valor
$m_{propulsor}$	0.0435 kg
m_{eixo}	0.107 kg
L	0.316 m

Substituindo os dados da Tabela 1 na Equação 3.7, obtém-se o momento de inércia total da gangorra, dado pela Equação 3.8.

$$J = 0.01225\, kg \cdot m^2 \tag{3.8}$$

A partir das Equações 3.4 e 3.8, obtém-se o a seguinte função de transferência para o sistema, disposta na Equação 3.9. O valor de B será definido em sequência.

$$\frac{\Theta(s)}{d_{res}(s)} = \frac{1.6577}{0.01225s^2 + Bs} = \frac{135.325}{s^2 + \left(\frac{B}{0.01225}\right)s} \tag{3.9}$$

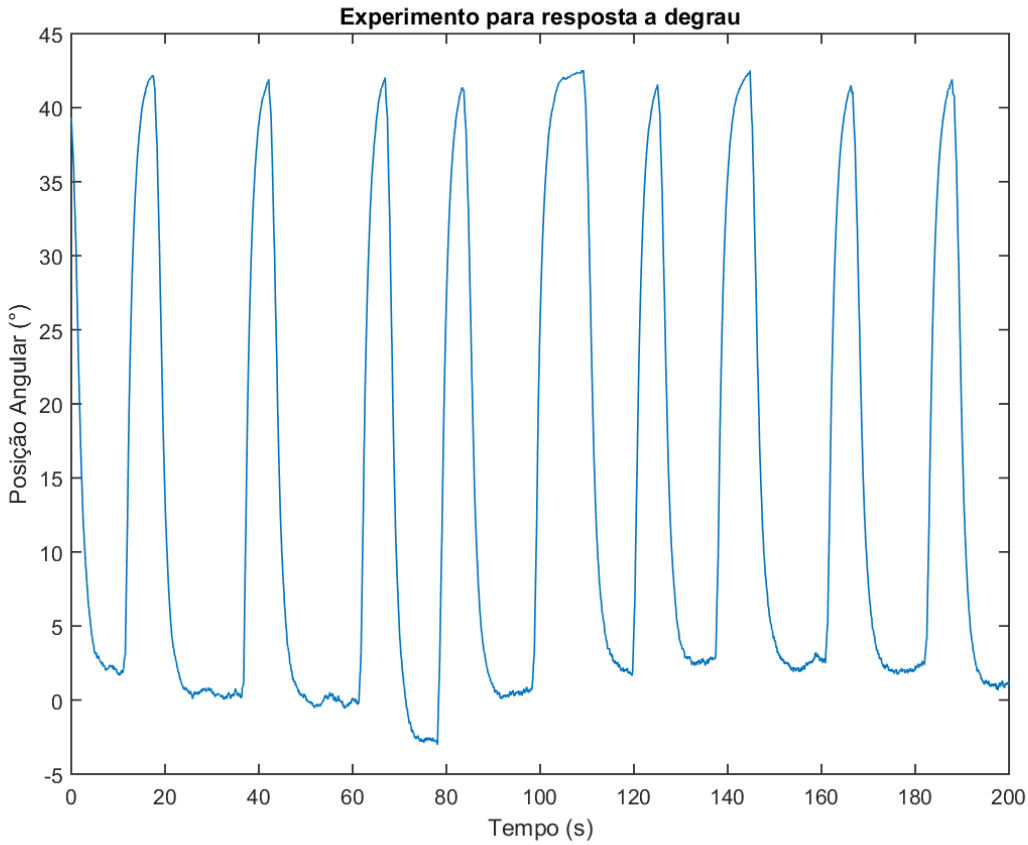
Para obter a função de transferência completa da planta, é necessário que seja estimado o valor do coeficiente B na Equação. Esse componente representa algumas perdas energéticas no sistema, sendo causadas por atrito, elasticidade dos cabos ou fluxo de ar, por exemplo. Desta forma, fez-se necessário que esse valor fosse estimado experimentalmente.

O procedimento foi realizado considerando o formato geral da Equação já encontrada em 3.9, e se caracterizou da seguinte maneira:

1. Com o sistema em malha aberta, enviou-se um sinal PWM para o motor esquerdo suficiente para que a gangorra se mantivesse na referência aproximada de 0°, mantendo o motor direito desligado;
2. Com a IMU ligada, a gangorra era inclinada ao seu fim de curso de aproximadamente -45°, e então liberada para que retornasse à referência inicial;
3. Realizaram-se repetidas vezes os passos anteriores a fim de obter mais pontos e amostras.

O resultado obtido foi representado graficamente pela Figura 36 a seguir.

Figura 36 – Representação dos dados obtidos para identificação do sistema

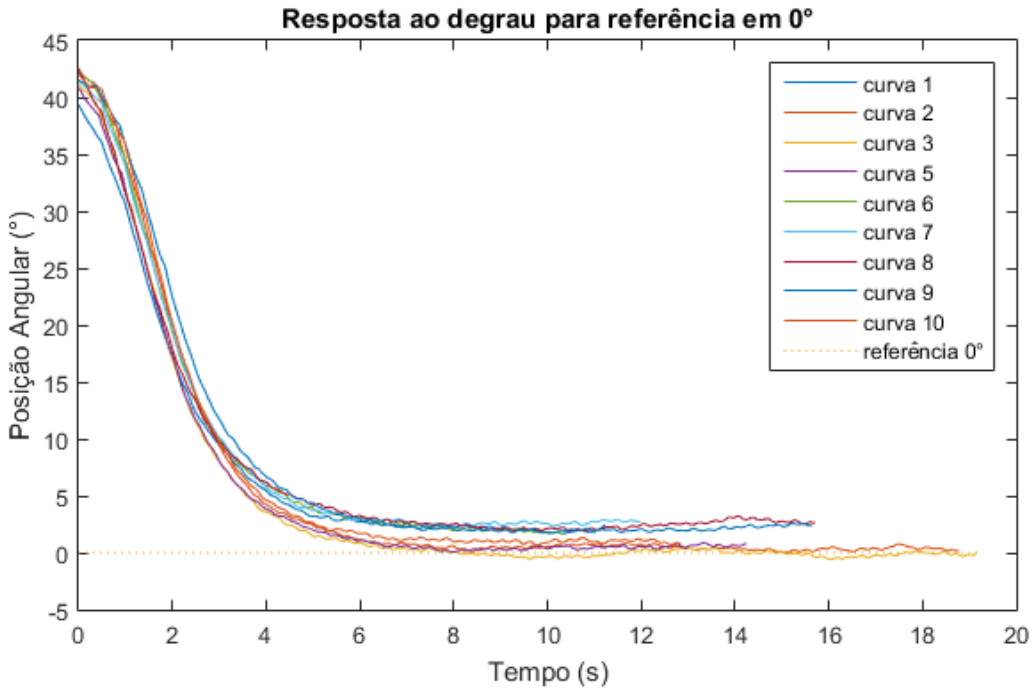


Fonte: dos autores

Com o auxílio da ferramenta computacional MATLAB (MATLAB, 2010), foi realizado o isolamento de cada amostra e a compensação no eixo x, para que fosse medido um valor médio de resposta ao degrau.

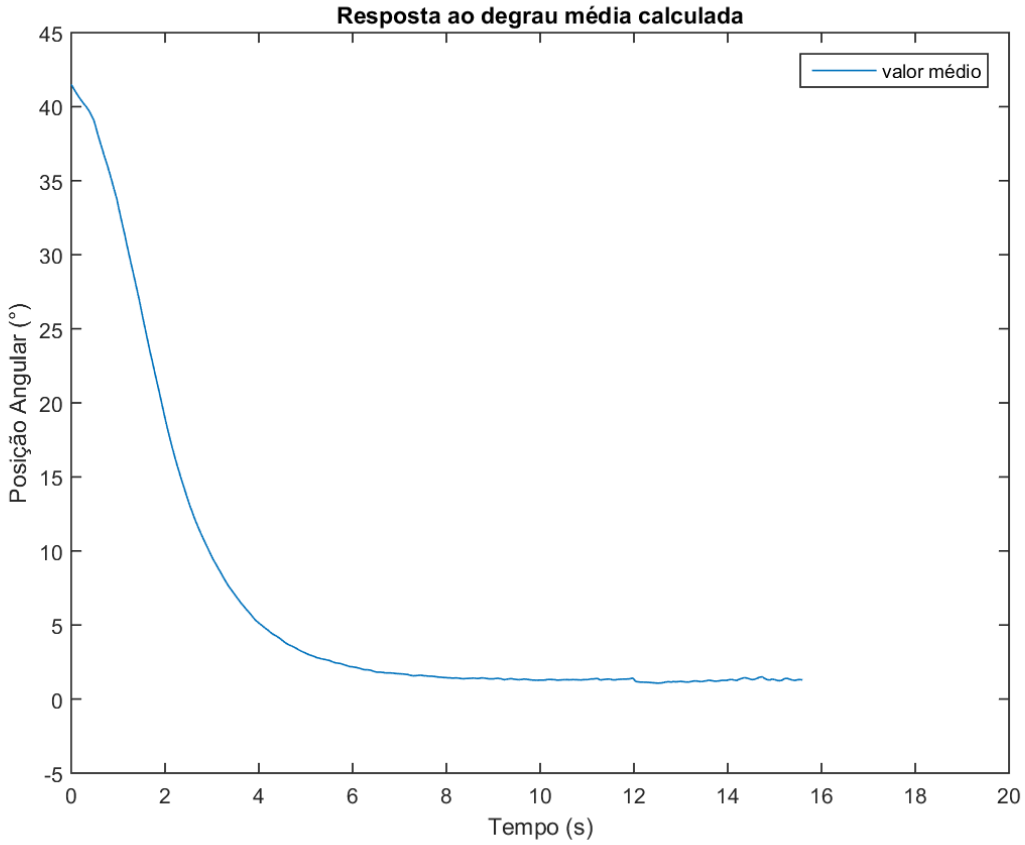
É importante ressaltar que, a fim de obter um valor mais confiável na média, foi implementado um algoritmo de interpolação linear para o cálculo dos pontos médios. Esse procedimento foi necessário visto que as amostras de tempo obtidas em cada ciclo de resposta não eram correspondentes. Nota-se também que a amostra 4 foi suprimida nessa análise, por apresentar valores discrepantes em relação às outras amostras.

Figura 37 – Medições isoladas e retificadas no tempo



Fonte: dos autores

Figura 38 – Curva obtida após interpolação



Fonte: dos autores

A partir dos valores obtidos para a resposta ao degrau representados na Figura 38 e com o auxílio da *toolbox* de identificação de sistemas do MATLAB, estimou-se uma função de transferência com formato baseado nas Equações encontradas em 3.9. Com isso, foi encontrada uma estimativa para B, no SI, representada na Equação 3.10 a seguir.

$$B = 0.025676 \tag{3.10}$$

Substituindo a Equação 3.10 em 3.9, obtemos a seguinte Equação de transferência para a bancada.

$$\frac{\Theta(s)}{d_{res}(s)} = \frac{135.325}{s^2 + (2.096)\,s} \tag{3.11}$$

3.5 Experimento de Controle

A fim de demonstrar a controlabilidade do sistema, foram realizados testes com a bancada aplicando algumas técnicas de controle distintas.

3.5.1 Sem controle

Apenas para ilustrar o comportamento do sistema sem controle em malha aberta sem controle aplicado, foram aplicados valores distintos de *duty cycle* a cada ESC. Como esperado, quando os valores de *duty cycle* são iguais ou muito próximos, a gangorra permanece estática, com leves oscilações devido ao fluxo de ar no ambiente ou vibrações na bancada. Já com valores distintos de *duty cycle*, o propulsor que está em maior velocidade empurra seu lado da gangorra para cima com velocidade angular dependente da diferença de velocidade entre os propulsores.

3.5.2 Controle PID

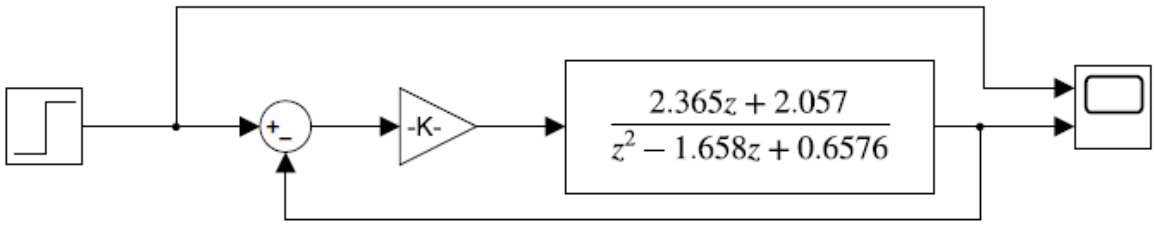
Utilizou-se o MATLAB para discretizar a função de transferência da Equação 3.11 com taxa de amostragem $T = 0.2\,s$ e discretização por segurador de ordem zero, obtendo o sistema da Equação 3.12.

$$G(z) = \frac{2.365z + 2.057}{z^2 - 1.658z + 0.6576} = \frac{2.3648(z + 0.8697)}{(z - 1)(z - 0.6576)} \tag{3.12}$$

É importante ressaltar que a taxa de amostragem, definida em 0.2 segundos, é considerada bastante elevada em se tratando de sistemas críticos como um *drone*. No entanto, esse valor foi escolhido para fins didáticos, a fim de ilustrar a implementação de um controlador digital com a MiniZed.

Baseado no método de Ziegler-Nichols de ganho limite (OGATA, 2010), com auxílio do Simulink para simular o comportamento da planta, obteve-se o valor do ganho e do período críticos do sistema em malha fechada. A Figura 39 mostra a simulação da planta.

Figura 39 – Diagrama de simulação no Simulink



Fonte: dos autores

A partir da simulação, obtiveram-se os valores de $K_{cr} = 0.1665$ e $T_{cr} = 1.419\,s$. Continuando o método, calcularam-se os valores das constantes para um controlador PID (OGATA, 2010), obtendo os resultados $K_p = 0.0999$, $K_i = 0.1408$ e $K_d = 0.0177$.

Aplicando o controlador PID com as constantes calculadas, o sistema não atingiu a estabilidade. Buscando melhorar a resposta, optou-se por utilizar um controlador do tipo proporcional.

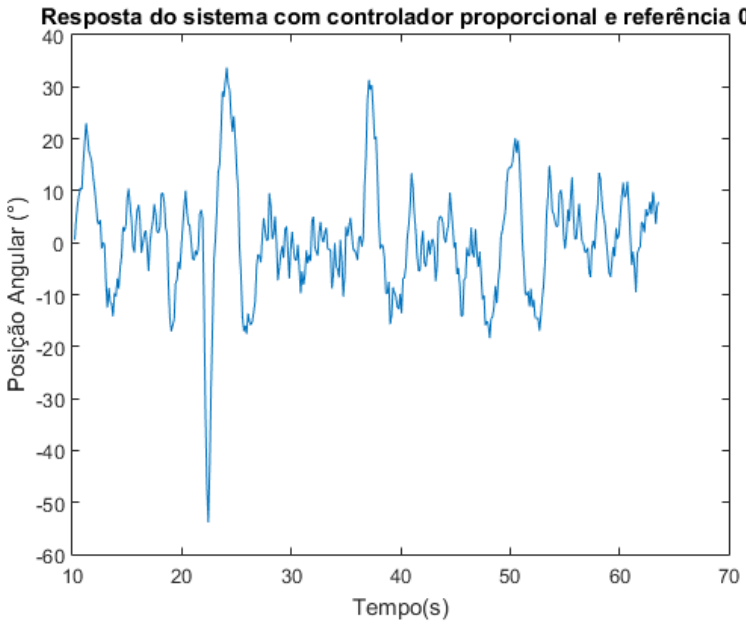
3.5.3 Controle Proporcional

Usando o ganho crítico obtido na Seção 3.5.2 como palpite inicial, foi realizado experimento empírico para obter um valor satisfatório para a constante de ganho K_p. Dessa forma, foram realizados diferentes ensaios com valores diversos para a constante, atingindo o valor considerado ótimo de $K_p = 0.0000625$.

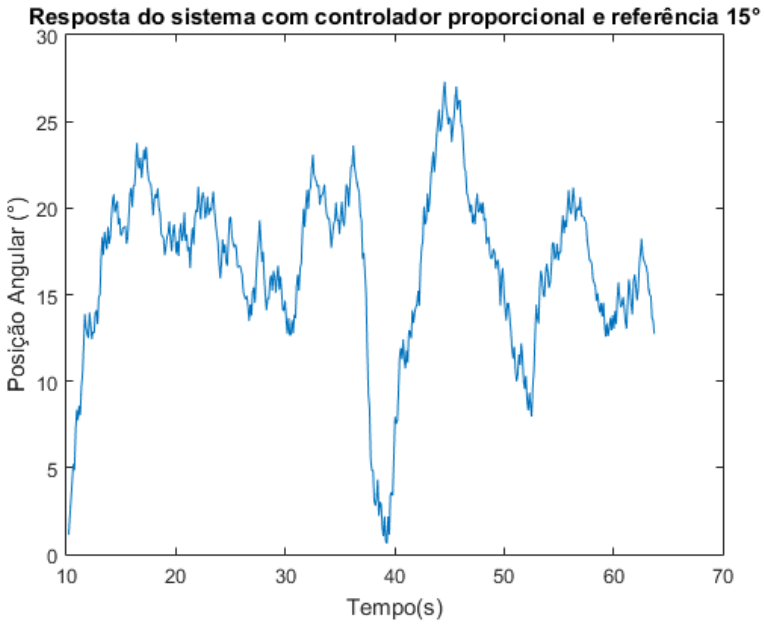
Os gráficos da Figura 40 mostram a posição angular da gangorra para o controlador proporcional para dois ângulos de referência. Durante a captura dos dados, foram inseridas perturbações no sistema para avaliar o comportamento. As perturbações se deram na forma de alteração da posição da gangorra com uma vareta. Vídeos do funcionamento da bancada podem ser encontrados em `https://youtube.com/playlist?list=PLW6HhZyspW_rZ1BpashWYxj97Xy2m1Mfr`.

Figura 40 – Resultados obtidos com controlador proporcional

(a) Resposta obtida com referência em 0°



(b) Resposta obtida com referência em 15°



Fonte: dos autores

Apesar da aparência ruidosa dos dados dos sensores, a balança atingiu o equilíbrio sem grandes oscilações, como visto nos vídeos de seu funcionamento.

4 Conclusões

Este capítulo traz as conclusões após a finalização da construção da bancada de ensaios, dos testes realizados e da análise dos resultados obtidos, além das perspectivas para trabalhos futuros baseados neste

4.1 Considerações Gerais

Pode-se afirmar que, por este trabalho possuir como proposta a construção inicial da bancada de testes com ARM e FPGA, há diversas possibilidades de aprimoramentos que podem ser realizadas a partir do estado atual da bancada. Foram observadas algumas características durante este projeto que prejudicaram o seu desenvolvimento. Essas características foram identificadas nos componentes do circuito eletrônico montado, na configuração e programação da FPGA e do processador da MiniZed e nos experimentos de controle.

Durante os testes de identificação de sistema e de controle, houve a desconfiguração ou perda total de 6 ESCs – todos da marca inicial proposta. Esses eventos causaram atraso no desenvolvimento do projeto, pois o trajeto desde a obtenção de um novo componente até a troca e o teste do equipamento era demorado. Essas falhas podem ter ocorrido pela ausência de um circuito eletrônico de proteção tanto para o sinal PWM a partir da MiniZed quanto para a fonte de alimentação de 12V. Outra possível causa é a própria escolha do componente, que pode não ser de boa qualidade.

Os experimentos de identificação e de controle da planta também revelaram outro fator limitante, sendo esse a interrupção do *loop* de controle na MiniZed. Durante a execução do programa, houve alguns bloqueios na leitura da IMU, que bloqueavam o fluxo, causando a parada total no sistema de controle. Esse distúrbio causava a necessidade de reinicio total da placa, visto que apenas selecionar o seu botão de reinicio não era suficiente para que voltasse a funcionar.

Outra característica importante a ser pontuada é o denominado efeito solo. O empuxo gerado pela rotação das hélices é bastante influenciado pelo ambiente ao qual pertence. A presença de paredes ou barreiras ao redor ou do próprio solo perto é suficiente para mudar o comportamento e aumentar ou diminuir essa força. Isso foi bastante observado durante este projeto, visto que a bancada sempre esteve com a rede de proteção aos lados e sobre uma mesa – similar ao solo – com exceção do experimento de identificação do motor. Percebeu-se que abrir ou fechar a rede de proteção já era suficiente para que o comportamento do sistema propulsor fosse alterado.

4.2 Perspectivas Futuras

Para trabalhos futuros, sugerem-se algumas melhorias baseadas nas dificuldades encontradas durante o desenvolvimento ou em observações com base no comportamento da bancada.

Assim, propõe-se que, para uma caracterização mais acurada da bancada, sejam realizados testes com medição de temperatura, pressão e umidade relativa do ar e o efeito de cada um desses componentes no funcionamento do conjunto propulsor. Além disso, aconselha-se um estudo mais aprofundado sobre a propulsão em si e como as hélices afetam o funcionamento do sistema, sendo possível analisar o efeito de diferentes hélices nos ensaios.

Uma possível melhoria, dadas as perdas de ESCs durante a execução de testes, é a criação de um circuito eletrônico de proteção dos equipamentos. Com isso, busca-se reduzir a quantidade de componentes danificados e de gastos com o projeto.

Sugere-se também que sejam aplicadas outras formas de identificação do sistema. Seria possível levar em consideração, por exemplo, os efeitos das medidas referentes ao ar. É aconselhável também aprimorar o cálculo do momento de inércia dos componentes da bancada, reduzindo as simplificações realizadas, buscando encontrar uma função de transferência mais próxima do sistema real.

Dado que apenas uma malha de controle foi efetivamente implementada, propõe-se que outras técnicas de controle sejam implementadas para uso na bancada. Controladores do tipo PID, por avanço de fase, por atraso de fase ou até mesmo liga-desliga podem ter seu funcionamento avaliado.

Como discutido durante o trabalho, o uso de baterias de *drone* pode ser aplicado para simular um sistema mais próximo da realidade. Seu uso possibilita verificar o efeito que a descarga da bateria teria em uma aeronave por exemplo.

Por fim, de forma a utilizar mais do potencial da arquitetura FPGA, é sugerido o estudo da implementação do controlador utilizado em *hardware*. Possibilitando análises das respostas obtidas de acordo com as diversas combinações de implementações do controlador.

Referências

ALLAIN, R. **The Physics of How Drones Fly | WIRED**. Mai. 2017. Disponível em: <https://www.wired.com/2017/05/the-physics-of-drones/>. Acesso em: 5 mai. 2022. Citado na p. 27.

ARDUINO. **Arduino Documentation**. 2015. Disponível em: <https://docs.arduino.cc/>. Acesso em: 15 abr. 2022. Citado na p. 35.

AVNET. **Minized**. 2016. Disponível em: <https://www.avnet.com/wps/portal/us/products/avnet-boards/avnet-board-families/minized/>. Acesso em: 16 jun. 2021. Citado nas pp. 18, 19.

BARBOSA, F. d. S. 4DOF Quadcopter: Development, Modeling and Control, 6 set. 2017. Disponível em: <https://teses.usp.br/teses/disponiveis/3/3139/tde-23102017-144556/pt-br.php>. Citado nas pp. 38, 39, 43.

BAÚ DA ELETRÔNICA. **ESC 30A Brushless com Bec Interno 2A/5V**. Disponível em: <https://www.baudaeletronica.com.br/esc-30a-brushless-com-bec-interno-2a-5v-aeromodelos.html>. Acesso em: 5 mai. 2022. Citado na p. 45.

BUCCO-LECHAT, C. **File:WMCH Drone.jpg**. Dez. 2016. Disponível em: <https://commons.wikimedia.org/wiki/File:WMCH_Drone.jpg>. Acesso em: 5 mai. 2022. Citado na p. 23.

CAMPOS, V. A. d.; HERNANDES, A. C. Educational testbed for aerial angular control: Project and study case, nov. 2018. Disponível em: <https://www.researchgate.net/publication/329958988_Educational_Testbed_for_Aerial_Angular_Control_Project_and_Study_Case>. Acesso em: 15 mar. 2021. Citado nas pp. 40, 41, 44.

CIOBANU, E. How Long Does a Drone Battery Last? What You Need to Know!, mar. 2021. Disponível em: <https://www.droneblog.com/2021/03/29/how-long-does-a-drone-battery-last-what-you-need-to-know/>. Acesso em: 22 jun. 2021. Citado na p. 35.

CONNER-SIMONS, A. Design your own custom drone, dez. 2016. Disponível em: <https://news.mit.edu/2016/design-your-own-custom-drone-1205>. Acesso em: 7 abr. 2022. Citado na p. 25.

DEJAN. **Arduino Brushless Motor Control Tutorial**. Fev. 2019a. Disponível em: <https://howtomechatronics.com/tutorials/arduino/arduino-brushless-motor-control-tutorial-esc-bldc/>. Acesso em: 5 mai. 2022. Citado na p. 34.

DEJAN. **How Brushless Motor and ESC Work**. 2019b. Disponível em: <https://howtomechatronics.com/how-it-works/how-brushless-motor-and-esc-work/>. Acesso em: 30 mar. 2022. Citado na p. 54.

DEVEZAS, T.; LEITÃO, J.; SARYGULOV, A. Industry 4.0: Entrepreneurship and Structural Change in the New Digital Landscape. **Studies on Entrepreneurship, Structural Change and Industrial Dynamics**, Springer International Publishing, 2017. DOI: 10.1007/978-3-319-49604-7. Citado na p. 20.

DJI. **Manifold 2 - Specification**. 2019. Disponível em: <https://www.dji.com/manifold-2/specs>. Acesso em: 16 jun. 2021. Citado na p. 23.

DUMONT, J. E. F.; BRITO VIEIRA, M. R. de. Estimação em tempo real de posição e orientação de câmeras inteligentes, mai. 2022. Não publicado. Citado nas pp. 56, 62.

EHRHARDT, T. **Drone Thermal Imaging Hexacopter**. Jul. 2018. Disponível em: <https://pixabay.com/photos/drone-thermal-imaging-hexacopter-3565438/>. Acesso em: 5 mai. 2022. Citado na p. 24.

ELECTRONOOBS. **PID brushless motor control tutorial**. Mai. 2017. Disponível em: <https://youtu.be/AN3yxIBAxTA>. Acesso em: 7 jun. 2021. Citado na p. 42.

ELECTROYA. **LittleBee 30A Spring BLHeli_S ESC DSHOT**. Mai. 2017. Disponível em: <https://www.electroya.com/en/producto/littlebee-30a-spring-blheli_s-esc-dshot/>. Acesso em: 5 mai. 2022. Citado na p. 44.

EPÓRIO DO ARDUINO. **Acelerômetro Giroscópio Mpu-6050 3 Eixo 6 Dof Gy-521**. Disponível em: <https://produto.mercadolivre.com.br/MLB-1851615426-acelermetro-giroscopio-mpu-6050-3-eixo-6-dof-gy-521-_JM>. Acesso em: 5 mai. 2022. Citado na p. 56.

EUREKA DYNAMICS. **FFT Gyro 600 PRO**. 2018. Disponível em: <https://eurekadynamics.com/fft-gyro-600/>. Acesso em: 13 abr. 2022. Citado na p. 37.

FLEMING, E. **What is PWM in ESC?** Fev. 2020. Disponível em: <https://www.sidmartinbio.org/what-is-pwm-in-esc/>. Acesso em: 30 mar. 2022. Citado na p. 55.

FLYPRO. **Anatomia de um Drone**. Abr. 2020. Disponível em: <https://www.flypro.com.br/pagina/anatomia-de-um-drone.html>. Acesso em: 5 mai. 2022. Citado na p. 28.

GETFPV. All About Multirotor Drone Batteries, fev. 2018a. Disponível em: <https://www.getfpv.com/learn/new-to-fpv/all-about-multirotor-fpv-drone-battery/>. Acesso em: 22 jun. 2021. Citado na p. 35.

GETFPV. All about Multirotor Drone FPV Propellers, fev. 2018b. Disponível em: <https://www.getfpv.com/learn/new-to-fpv/all-about-multirotor-fpv-drone-propellers/>. Acesso em: 7 abr. 2022. Citado na p. 46.

GLOBAL BRANDS. **Top 10 Drone Companies in the world - 2020**. 2020. Disponível em: <https://www.globalbrandsmagazine.com/top-10-drone-companies-in-the-world-2020/>. Acesso em: 16 jun. 2021. Citado na p. 23.

GOVBR. ***Drones***. 2016. Disponível em: <https://www.gov.br/anac/pt-br/assuntos/drones>. Acesso em: 23 jun. 2021. Citado na p. 17.

HERRMANN, S.; MARKERT, F. **Drone regulations worldwide**. Out. 2020. Disponível em: <https://drone-traveller.com/drone-regulations-worldwide/>. Citado na p. 16.

IFORCE2D. **DIY thrust test stand intro**. Mai. 2016. Disponível em: <https://www.youtube.com/watch?v=V3kxl4dHAwE>. Acesso em: 15 mar. 2022. Citado na p. 36.

JRVZ. **File:Yaw Axis Corrected.svg**. Fev. 2010. Disponível em: <https://commons.wikimedia.org/wiki/File:Yaw_Axis_Corrected.svg>. Acesso em: 5 mai. 2022. Citado na p. 26.

KUKSOV, I. Alerta aéreo: 8 incidentes perigosos envolvendo drones, nov. 2019. Disponível em: <https://www.kaspersky.com.br/blog/drone-incidents/12492/>. Acesso em: 7 abr. 2022. Citado na p. 18.

LANDELL-MILLS, N. Introdução à Fusão de Sensores - Parte 1 - Embarcados, mar. 2022. Disponível em: <https://www.embarcados.com.br/introducao-fusao-de-sensores-parte-1/>. Acesso em: 18 abr. 2022. Citado na p. 32.

LUTKEVICH, B. What is a Drone?, dez. 2021. Disponível em: <https://www.techtarget.com/iotagenda/definition/drone>. Acesso em: 22 abr. 2022. Citado na p. 16.

MADRUGA, S. P. Projeto de Sistema de Controle Embarcado para Controle de Voo de Quadricópteros, p. 25–35, jul. 2018. Disponível em: <https://repositorio.ufpb.br/jspui/bitstream/123456789/13346/1/Arquivototal.pdf>. Acesso em: 17 jun. 2021. Citado nas pp. 39, 40.

MATLAB. **version 7.10.0 (R2010a)**. Natick, Massachusetts: The MathWorks Inc., 2010. Citado na p. 67.

MILLETT, P. Brushless Vs Brushed DC Motors: When and Why to Choose One Over the Other, 2020. Disponível em: <https://www.monolithicpower.com/en/brushless-vs-brushed-dc-motors>. Acesso em: 7 abr. 2022. Citado na p. 46.

MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÕES. 2014. Disponível em: <https://gravimetria.webnode.com/>. Acesso em: 18 mar. 2022. Citado na p. 61.

NEVES, F. Introdução à Fusão de Sensores - Parte 1 - Embarcados, mar. 2017. Disponível em: <https://www.embarcados.com.br/introducao-fusao-de-sensores-parte-1/>. Acesso em: 16 abr. 2022. Citado na p. 31.

NXP SEMICONDUCTORS. **I²C-bus specification and user manual**. Out. 2021. Disponível em: <https://www.nxp.com/docs/en/user-guide/UM10204.pdf>. Acesso em: 21 abr. 2022. Citado na p. 56.

OGATA, K. **Modern Control Engineering**. 5. ed.: Prentice Hal, 2010. P. 568–577. Citado na p. 70.

OLIVEIRA, R. S. de. **Fundamentos dos Sistemas de Tempo Real**. 1. ed. Florianópolis: Amazon, out. 2018. Citado na p. 17.

PAULA LIMA, F. de. Implementação de Controle de Rolagem e Arfagem com Realimentação Visual Aplicado a um Quadrirrotor Comercial, abr. 2013. Disponível em: <https://bdm.unb.br/handle/10483/15749>. Acesso em: 13 abr. 2022. Citado na p. 25.

PLOEG, A. van der. **Why use an FPGA instead of a CPU or GPU?** The (dis)advantages of Field Programmable Gate Arrays. Ago. 2018. Disponível em: <https://blog.esciencecenter.nl/why-use-an-fpga-instead-of-a-cpu-or-gpu-b234cd4f309c>. Acesso em: 26 nov. 2021. Citado na p. 20.

RANKY, R.; SIVAK, M.; LEWIS, J.; GADE, V.; DEUTSCH, J.; MAVROIDIS, C. Modular mechatronic system for stationary bicycles interfaced with virtual environment for rehabilitation. **Journal of neuroengineering and rehabilitation**, v. 11, p. 93, jun. 2014. DOI: 10.1186/1743-0003-11-93. Disponível em: <https://www.researchgate.net/publication/262932137_Modular_mechatronic_system_for_stationary_bicycles_interfaced_with_virtual_environment_for_rehabilitation>. Acesso em: 15 mar. 2022. Citado na p. 36.

ROGERS, S. These Window-Cleaning Drones Make Cleaning Skyscrapers Look Easy, jan. 2019. Disponível em: <https://interestingengineering.com/video/these-window-cleaning-drones-make-cleaning-skyscrapers-look-easy>. Acesso em: 18 mar. 2021. Citado na p. 16.

SAXENA, S. Advantages and Disadvantages of ARM processor, jul. 2020. Disponível em: <https://www.geeksforgeeks.org/advantages-and-disadvantages-of-arm-processor/>. Acesso em: 26 nov. 2021. Citado na p. 20.

SCIENCE LEARNING HUB. Wings and Lift, set. 2011. Disponível em: <https://www.sciencelearn.org.nz/resources/300-wings-and-lift>. Citado na p. 26.

SEARA DA CIÊNCIA. Força de Coriolis, mar. 2019. Disponível em: <https://seara.ufc.br/pt/tintim-por-tintim/fisica/a-forca-de-coriolis/>. Acesso em: 16 abr. 2022. Citado na p. 29.

SLOSS, A.; SYMES, D.; WRIGHT, C. **ARM System Developer's Guide: Designing and Optimizing System Software**. 1. ed. Oxford, Reino Unido: Morgan Kaufmann, 2004. P. 3–6. (The Morgan Kaufmann Series in Computer Architecture and Design). Citado na p. 19.

STONE, K. **Octocopter**. Jul. 2017. Disponível em: <https://www.flickr.com/photos/nanopixl/32580071556/>. Acesso em: 5 mai. 2022. Citado na p. 24.

U.S. DEPARTMENT OF TRANSPORTATION. **Flight Training Handbook**. 1965. http://avstop.com/ac/f Acesso em: 5 mai. 2022. Citado na p. 32.

VIEIRA, L. **File:3D Gyroscope.png**. Out. 2006. Disponível em: <https://commons.wikimedia.org/wiki/File:3D_Gyroscope.png>. Acesso em: 5 mai. 2022. Citado na p. 30.

WATELECTRONICS. **Gyroscope Sensor: Working Principle, Types & Its Applications**. Mai. 2020. Disponível em: <https://www.watelectronics.com/what-is-a-gyroscope-sensor-working-its-applications/>. Acesso em: 16 abr. 2022. Citado na p. 29.

WATSON, J. **MEMS Gyroscope Provides Precision Inertial Sensing in Harsh, High Temperature Environments**. 2016. Disponível em: <https://www.analog.com/en/technical-articles/mems-gyroscope-provides-precision-inertial-sensing.html>. Acesso em: 8 mai. 2022. Citado na p. 30.

WIKIPEDIA. **Motor constants**. Abr. 2022. Disponível em: <https://en.wikipedia.org/wiki/Motor_constants>. Acesso em: 18 abr. 2022. Citado na p. 33.

WINKLER, C. *How Many Sensors are in a Drone, And What do they Do?*, jul. 2016. Disponível em: <https://www.fierceelectronics.com/components/how-many-sensors-are-a-drone-and-what-do-they-do>. Acesso em: 22 jun. 2021. Citado na p. 28.

WOODFORD, C. Accelerometers, fev. 2022. Disponível em: <https://www.explainthatstuff.com/accelerometers.html>. Acesso em: 31 mar. 2022. Citado na p. 29.

WOODS, R.; MCALLISTER, J.; TURNER, R.; YI, Y.; LIGHTBODY, G. **FPGA-based Implementation of Signal Processing Systems**. 1. ed. West Sussex, Reino Unido: Wiley, 2008. P. 1–3. Citado na p. 20.

XILINX. **Vivado Design Suite – AXI Reference Guide**. Jul. 2017. Disponível em: <https://docs.xilinx.com/v/u/en-US/ug1037-vivado-axi-reference-guide>. Acesso em: 28 mar. 2022. Citado na p. 56.

XILINX. **Zynq-7000 SoC**. Jun. 2011. Disponível em: <https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html>. Acesso em: 16 jun. 2021. Citado nas pp. 18, 34.

YOUNG, H. D.; FREEDMAN, R. A. **Física I**. 14. ed. São Paulo, SP: Pearson, 2016. ISBN 978-85-4301-813-3. Citado na p. 65.

YÜZGEÇ, U.; İRFAN ÖKTEN, H. Ü.; GÜN, A. R.; TÜRKYILMAZ, T.; KESLER, M.; KARAKUZU, C.; UÇAR, G. Development of the Test Platform for Rotary Wing Unmanned Air Vehicle, 2016. Disponível em: <https://dergipark.org.tr/tr/download/article-file/368163>. Acesso em: 13 abr. 2022. Citado na p. 38.

Anexos

ANEXO A – Descrição do conteúdo do repositório *online*

Os arquivos do projeto, disponível em <https://github.com/embedded-computing/drone_testbed>, são divididos seguindo a estrutura a seguir, com os diretórios e arquivos relevantes:

```
/
├── Bancada_Drone/
│   ├── Bancada_Drone.hw/
│   ├── Bancada_Drone.ip_user_files/
│   ├── Bancada_Drone.runs/
│   ├── Bancada_Drone.sdk/
│   │   ├── ARM_Clock/
│   │   ├── IIC_MPU/
│   │   ├── IIC_Teste_wrapper_hw_platform_0/
│   │   ├── PID_Control/
│   │   ├── PWM_w_Int/
│   │   ├── Step_Response/
│   │   ├── standalone_bsp_0/
│   │   └── IIC_Teste_wrapper.hdf
│   ├── Bancada_Drone.srcs/
│   └── Bancada_Drone.xpr
├── ip_repo/
│   └── PWM_w_Int_1.0/
├── MiniZed_Constraints_Rev1_170613.xdc
└── README.md
```

Os diretórios Bancada_Drone.hw, Bancada_Drone.ip_user_files, Bancada_Drone.runs e Bancada_Drone.srcs contêm os arquivos gerados pelo *software* Vivado, incluindo os utilizados para a criação do diagrama de blocos para configuração da programação em FPGA. Mais especificamente, o arquivo Bancada_Drone.xpr contém o projeto do Vivado, que utiliza os diretórios mencionados, e o arquivo Bancada_Drone.srcs/sources_1/bd/IIC_Teste/IIC_Teste.bd contém o diagrama de blocos utilizado.

O diretório Bancada_Drone.sdk contém os arquivos para uso no Xilinx SDK, necessário para uso na arquitetura ARM. Os diversos projetos contidos tem diferentes utilidades. O arquivo IIC_Teste_wrapper.hdf e os projetos standalone_bsp_0 e IIC_Teste_wrapper_hw_platform_0 contém as configurações utilizadas na placa de desenvolvimento para integração

entre ARM e FPGA. O projeto ARM_Clock apresenta um teste simples do uso do relógio do processador. O projeto IIC_MPU foi desenvolvido por Júlio Eduardo e apresenta os módulos para leitura dos sensores via protocolo I^2C (IIC). O projeto PID_Control contém o projeto com o com o controlador da bancada, que, apesar do nome, teve seus elementos integrador e derivativo suprimidos. O projeto PWM_w_Int contém o básico para uso dos motores, programa utilizado para teste do envio do sinal de PWM e para a calibração dos conjuntos propulsores. Por fim, o projeto Step_Response possui os arquivos para os ensaios de resposta ao degrau da bancada, cujos resultados foram utilizados na identificação do sistema.

Para facilitar a visualização dos blocos de *hardware* utilizados, os códigos A.1, A.2 e A.3 mostram os arquivos para este propósito.

Código A.1 – Código VHDL de definição de pinos do bloco

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity PWM_w_Int_v1_0 is
6    generic (
7      -- Users to add parameters here
8              PWM_PERIOD : integer :=20;
9      -- User parameters ends
10     -- Do not modify the parameters beyond this line
11
12
13     -- Parameters of Axi Slave Bus Interface S00_AXI
14     C_S00_AXI_DATA_WIDTH  : integer := 32;
15     C_S00_AXI_ADDR_WIDTH  : integer := 4
16   );
17   port (
18     -- Users to add ports here
19         PWM_Out : out std_logic;
20         Interrupt_Out : out std_logic;
21         PWM_Counter : out std_logic_vector(PWM_PERIOD-1 downto 0);
22         DutyCycle : out std_Logic_vector(31 downto 0);
23     -- User ports ends
24     -- Do not modify the ports beyond this line
25
26
27     -- Ports of Axi Slave Bus Interface S00_AXI
28     s00_axi_aclk  : in std_logic;
29     s00_axi_aresetn : in std_logic;
30     s00_axi_awaddr  : in std_logic_vector(C_S00_AXI_ADDR_WIDTH-1
         downto 0);
31     s00_axi_awprot  : in std_logic_vector(2 downto 0);
32     s00_axi_awvalid : in std_logic;
33     s00_axi_awready : out std_logic;
34     s00_axi_wdata : in std_logic_vector(C_S00_AXI_DATA_WIDTH-1
         downto 0);
```

```
35      s00_axi_wstrb : in std_logic_vector((C_S00_AXI_DATA_WIDTH/8)-1
           downto 0);
36      s00_axi_wvalid  : in std_logic;
37      s00_axi_wready  : out std_logic;
38      s00_axi_bresp : out std_logic_vector(1 downto 0);
39      s00_axi_bvalid  : out std_logic;
40      s00_axi_bready  : in std_logic;
41      s00_axi_araddr  : in std_logic_vector(C_S00_AXI_ADDR_WIDTH-1
           downto 0);
42      s00_axi_arprot  : in std_logic_vector(2 downto 0);
43      s00_axi_arvalid : in std_logic;
44      s00_axi_arready : out std_logic;
45      s00_axi_rdata : out std_logic_vector(C_S00_AXI_DATA_WIDTH-1
           downto 0);
46      s00_axi_rresp : out std_logic_vector(1 downto 0);
47      s00_axi_rvalid  : out std_logic;
48      s00_axi_rready  : in std_logic
49    );
50  end PWM_w_Int_v1_0;
51
52  architecture arch_imp of PWM_w_Int_v1_0 is
53
54    -- component declaration
55    component PWM_Controller_Int is
56          generic (
57              period : integer := 20
58             );
59            port (
60              Clk : in std_logic;
61              DutyCycle : in std_logic_vector(31 downto 0);
62              Reset : in std_logic;
63              PWM_out : out std_logic_vector(0 downto 0);
64              Interrupt : out std_logic;
65              count : out std_logic_vector(period-1 downto 0)
66             );
67             end component PWM_Controller_Int;
68
69    component PWM_w_Int_v1_0_S00_AXI is
70      generic (
71      C_S_AXI_DATA_WIDTH  : integer := 32;
72      C_S_AXI_ADDR_WIDTH  : integer := 4
73      );
74      port (
75      slave_reg0: out std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto
           0);
76      S_AXI_ACLK  : in std_logic;
77      S_AXI_ARESETN : in std_logic;
78      S_AXI_AWADDR  : in std_logic_vector(C_S_AXI_ADDR_WIDTH-1
           downto 0);
79      S_AXI_AWPROT  : in std_logic_vector(2 downto 0);
80      S_AXI_AWVALID : in std_logic;
81      S_AXI_AWREADY : out std_logic;
```

```
82      S_AXI_WDATA : in std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto
            0);
83      S_AXI_WSTRB : in std_logic_vector((C_S_AXI_DATA_WIDTH/8)-1
            downto 0);
84      S_AXI_WVALID  : in std_logic;
85      S_AXI_WREADY  : out std_logic;
86      S_AXI_BRESP : out std_logic_vector(1 downto 0);
87      S_AXI_BVALID  : out std_logic;
88      S_AXI_BREADY  : in std_logic;
89      S_AXI_ARADDR  : in std_logic_vector(C_S_AXI_ADDR_WIDTH-1
            downto 0);
90      S_AXI_ARPROT  : in std_logic_vector(2 downto 0);
91      S_AXI_ARVALID : in std_logic;
92      S_AXI_ARREADY : out std_logic;
93      S_AXI_RDATA : out std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto
            0);
94      S_AXI_RRESP : out std_logic_vector(1 downto 0);
95      S_AXI_RVALID  : out std_logic;
96      S_AXI_RREADY  : in std_logic
97      );
98    end component PWM_w_Int_v1_0_S00_AXI;
99
100 signal DutyCycle_int : std_logic_vector(31 downto 0);
101
102
103 begin
104
105 DutyCycle <= DutyCycle_int;
106
107 -- Instantiation of Axi Bus Interface S00_AXI
108 PWM_w_Int_v1_0_S00_AXI_inst : PWM_w_Int_v1_0_S00_AXI
109   generic map (
110     C_S_AXI_DATA_WIDTH  => C_S00_AXI_DATA_WIDTH,
111     C_S_AXI_ADDR_WIDTH  => C_S00_AXI_ADDR_WIDTH
112   )
113   port map (
114       slave_reg0 => DutyCycle_int,
115     S_AXI_ACLK  => s00_axi_aclk,
116     S_AXI_ARESETN => s00_axi_aresetn,
117     S_AXI_AWADDR  => s00_axi_awaddr,
118     S_AXI_AWPROT  => s00_axi_awprot,
119     S_AXI_AWVALID => s00_axi_awvalid,
120     S_AXI_AWREADY => s00_axi_awready,
121     S_AXI_WDATA => s00_axi_wdata,
122     S_AXI_WSTRB => s00_axi_wstrb,
123     S_AXI_WVALID  => s00_axi_wvalid,
124     S_AXI_WREADY  => s00_axi_wready,
125     S_AXI_BRESP => s00_axi_bresp,
126     S_AXI_BVALID  => s00_axi_bvalid,
127     S_AXI_BREADY  => s00_axi_bready,
128     S_AXI_ARADDR  => s00_axi_araddr,
129     S_AXI_ARPROT  => s00_axi_arprot,
```

```
130     S_AXI_ARVALID => s00_axi_arvalid,
131     S_AXI_ARREADY => s00_axi_arready,
132     S_AXI_RDATA => s00_axi_rdata,
133     S_AXI_RRESP => s00_axi_rresp,
134     S_AXI_RVALID  => s00_axi_rvalid,
135     S_AXI_RREADY  => s00_axi_rready
136   );
137
138   -- Add user logic here
139 PWM_Controller_Int_Inst : PWM_Controller_Int
140                      generic map(
141                          period => PWM_PERIOD
142                      )
143                     port map (
144                     Clk =>s00_axi_aclk,
145                     DutyCycle =>DutyCycle_int,
146                     Reset=>s00_axi_aresetn,
147                     PWM_out(0) =>PWM_Out,
148                     Interrupt =>Interrupt_Out,
149                     count =>PWM_Counter
150                     );
151
152   -- User logic ends
153
154 end arch_imp;
```

Código A.2 – Código VHDL para definição da estrutura AXI

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity PWM_w_Int_v1_0_S00_AXI is
6    generic (
7      -- Users to add parameters here
8
9      -- User parameters ends
10     -- Do not modify the parameters beyond this line
11
12     -- Width of S_AXI data bus
13     C_S_AXI_DATA_WIDTH  : integer := 32;
14     -- Width of S_AXI address bus
15     C_S_AXI_ADDR_WIDTH  : integer := 4
16   );
17   port (
18     -- Users to add ports here
19           slave_reg0: out std_logic_vector(C_S_AXI_DATA_WIDTH-1
                downto 0);
20     -- User ports ends
21     -- Do not modify the ports beyond this line
22
23     -- Global Clock Signal
```

```
24      S_AXI_ACLK   : in std_logic;
25      -- Global Reset Signal. This Signal is Active LOW
26      S_AXI_ARESETN : in std_logic;
27      -- Write address (issued by master, acceped by Slave)
28      S_AXI_AWADDR   : in std_logic_vector(C_S_AXI_ADDR_WIDTH-1
           downto 0);
29      -- Write channel Protection type. This signal indicates the
30          -- privilege and security level of the transaction, and
               whether
31          -- the transaction is a data access or an instruction
               access.
32      S_AXI_AWPROT   : in std_logic_vector(2 downto 0);
33      -- Write address valid. This signal indicates that the master
           signaling
34          -- valid write address and control information.
35      S_AXI_AWVALID : in std_logic;
36      -- Write address ready. This signal indicates that the slave
           is ready
37          -- to accept an address and associated control signals.
38      S_AXI_AWREADY : out std_logic;
39      -- Write data (issued by master, acceped by Slave)
40      S_AXI_WDATA : in std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto
           0);
41      -- Write strobes. This signal indicates which byte lanes hold
42          -- valid data. There is one write strobe bit for each eight
43          -- bits of the write data bus.
44      S_AXI_WSTRB : in std_logic_vector((C_S_AXI_DATA_WIDTH/8)-1
           downto 0);
45      -- Write valid. This signal indicates that valid write
46          -- data and strobes are available.
47      S_AXI_WVALID   : in std_logic;
48      -- Write ready. This signal indicates that the slave
49          -- can accept the write data.
50      S_AXI_WREADY   : out std_logic;
51      -- Write response. This signal indicates the status
52          -- of the write transaction.
53      S_AXI_BRESP : out std_logic_vector(1 downto 0);
54      -- Write response valid. This signal indicates that the channel
55          -- is signaling a valid write response.
56      S_AXI_BVALID   : out std_logic;
57      -- Response ready. This signal indicates that the master
58          -- can accept a write response.
59      S_AXI_BREADY   : in std_logic;
60      -- Read address (issued by master, acceped by Slave)
61      S_AXI_ARADDR   : in std_logic_vector(C_S_AXI_ADDR_WIDTH-1
           downto 0);
62      -- Protection type. This signal indicates the privilege
63          -- and security level of the transaction, and whether the
64          -- transaction is a data access or an instruction access.
65      S_AXI_ARPROT   : in std_logic_vector(2 downto 0);
66      -- Read address valid. This signal indicates that the channel
67          -- is signaling valid read address and control information.
```

```
68      S_AXI_ARVALID : in std_logic;
69      -- Read address ready. This signal indicates that the slave is
70          -- ready to accept an address and associated control
               signals.
71      S_AXI_ARREADY : out std_logic;
72      -- Read data (issued by slave)
73      S_AXI_RDATA : out std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto
           0);
74      -- Read response. This signal indicates the status of the
75          -- read transfer.
76      S_AXI_RRESP : out std_logic_vector(1 downto 0);
77      -- Read valid. This signal indicates that the channel is
78          -- signaling the required read data.
79      S_AXI_RVALID  : out std_logic;
80      -- Read ready. This signal indicates that the master can
81          -- accept the read data and response information.
82      S_AXI_RREADY  : in std_logic
83    );
84  end PWM_w_Int_v1_0_S00_AXI;
85
86  architecture arch_imp of PWM_w_Int_v1_0_S00_AXI is
87
88    -- AXI4LITE signals
89    signal axi_awaddr : std_logic_vector(C_S_AXI_ADDR_WIDTH-1 downto
         0);
90    signal axi_awready  : std_logic;
91    signal axi_wready : std_logic;
92    signal axi_bresp  : std_logic_vector(1 downto 0);
93    signal axi_bvalid : std_logic;
94    signal axi_araddr : std_logic_vector(C_S_AXI_ADDR_WIDTH-1 downto
         0);
95    signal axi_arready  : std_logic;
96    signal axi_rdata  : std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto
         0);
97    signal axi_rresp  : std_logic_vector(1 downto 0);
98    signal axi_rvalid : std_logic;
99
100   -- Example-specific design signals
101   -- local parameter for addressing 32 bit / 64 bit
         C_S_AXI_DATA_WIDTH
102   -- ADDR_LSB is used for addressing 32/64 bit registers/memories
103   -- ADDR_LSB = 2 for 32 bits (n downto 2)
104   -- ADDR_LSB = 3 for 64 bits (n downto 3)
105   constant ADDR_LSB  : integer := (C_S_AXI_DATA_WIDTH/32)+ 1;
106   constant OPT_MEM_ADDR_BITS : integer := 1;
107   ------------------------------------------------
108   ---- Signals for user logic register space example
109   ------------------------------------------------
110   ---- Number of Slave Registers 4
111   signal slv_reg0 :std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
112   signal slv_reg1 :std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
113   signal slv_reg2 :std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
```

```
114   signal slv_reg3 :std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
115   signal slv_reg_rden : std_logic;
116   signal slv_reg_wren : std_logic;
117   signal reg_data_out :std_logic_vector(C_S_AXI_DATA_WIDTH-1
        downto 0);
118   signal byte_index : integer;
119   signal aw_en  : std_logic;
120
121 begin
122   -- I/O Connections assignments
123     slave_reg0 <= slv_reg0;
124   S_AXI_AWREADY <= axi_awready;
125   S_AXI_WREADY  <= axi_wready;
126   S_AXI_BRESP <= axi_bresp;
127   S_AXI_BVALID  <= axi_bvalid;
128   S_AXI_ARREADY <= axi_arready;
129   S_AXI_RDATA <= axi_rdata;
130   S_AXI_RRESP <= axi_rresp;
131   S_AXI_RVALID  <= axi_rvalid;
132   -- Implement axi_awready generation
133   -- axi_awready is asserted for one S_AXI_ACLK clock cycle when
        both
134   -- S_AXI_AWVALID and S_AXI_WVALID are asserted. axi_awready is
135   -- de-asserted when reset is low.
136
137   process (S_AXI_ACLK)
138   begin
139     if rising_edge(S_AXI_ACLK) then
140       if S_AXI_ARESETN = '0' then
141         axi_awready <= '0';
142         aw_en <= '1';
143       else
144         if (axi_awready = '0' and S_AXI_AWVALID = '1' and
              S_AXI_WVALID = '1' and aw_en = '1') then
145           -- slave is ready to accept write address when
146           -- there is a valid write address and write data
147           -- on the write address and data bus. This design
148           -- expects no outstanding transactions.
149              axi_awready <= '1';
150              aw_en <= '0';
151           elsif (S_AXI_BREADY = '1' and axi_bvalid = '1') then
152              aw_en <= '1';
153              axi_awready <= '0';
154         else
155           axi_awready <= '0';
156         end if;
157       end if;
158     end if;
159   end process;
160
161   -- Implement axi_awaddr latching
162   -- This process is used to latch the address when both
```

```
163   -- S_AXI_AWVALID and S_AXI_WVALID are valid.
164
165   process (S_AXI_ACLK)
166   begin
167     if rising_edge(S_AXI_ACLK) then
168       if S_AXI_ARESETN = '0' then
169         axi_awaddr <= (others => '0');
170       else
171         if (axi_awready = '0' and S_AXI_AWVALID = '1' and
                S_AXI_WVALID = '1' and aw_en = '1') then
172           -- Write Address latching
173           axi_awaddr <= S_AXI_AWADDR;
174         end if;
175       end if;
176     end if;
177   end process;
178
179   -- Implement axi_wready generation
180   -- axi_wready is asserted for one S_AXI_ACLK clock cycle when
        both
181   -- S_AXI_AWVALID and S_AXI_WVALID are asserted. axi_wready is
182   -- de-asserted when reset is low.
183
184   process (S_AXI_ACLK)
185   begin
186     if rising_edge(S_AXI_ACLK) then
187       if S_AXI_ARESETN = '0' then
188         axi_wready <= '0';
189       else
190         if (axi_wready = '0' and S_AXI_WVALID = '1' and
                S_AXI_AWVALID = '1' and aw_en = '1') then
191           -- slave is ready to accept write data when
192           -- there is a valid write address and write data
193           -- on the write address and data bus. This design
194           -- expects no outstanding transactions.
195           axi_wready <= '1';
196         else
197           axi_wready <= '0';
198         end if;
199       end if;
200     end if;
201   end process;
202
203   -- Implement memory mapped register select and write logic
        generation
204   -- The write data is accepted and written to memory mapped
        registers when
205   -- axi_awready, S_AXI_WVALID, axi_wready and S_AXI_WVALID are
        asserted. Write strobes are used to
206   -- select byte enables of slave registers while writing.
207   -- These registers are cleared when reset (active low) is
        applied.
```

```
208   -- Slave register write enable is asserted when valid address
          and data are available
209   -- and the slave is ready to accept the write address and write
          data.
210   slv_reg_wren <= axi_wready and S_AXI_WVALID and axi_awready and
          S_AXI_AWVALID ;
211
212   process (S_AXI_ACLK)
213   variable loc_addr :std_logic_vector(OPT_MEM_ADDR_BITS downto 0);
214   begin
215     if rising_edge(S_AXI_ACLK) then
216       if S_AXI_ARESETN = '0' then
217         slv_reg0 <= (others => '0');
218         slv_reg1 <= (others => '0');
219         slv_reg2 <= (others => '0');
220         slv_reg3 <= (others => '0');
221       else
222         loc_addr := axi_awaddr(ADDR_LSB + OPT_MEM_ADDR_BITS downto
              ADDR_LSB);
223         if (slv_reg_wren = '1') then
224           case loc_addr is
225             when b"00" =>
226               for byte_index in 0 to (C_S_AXI_DATA_WIDTH/8-1) loop
227                 if ( S_AXI_WSTRB(byte_index) = '1' ) then
228                   -- Respective byte enables are asserted as per
                          write strobes
229                   -- slave registor 0
230                   slv_reg0(byte_index*8+7 downto byte_index*8) <=
                          S_AXI_WDATA(byte_index*8+7 downto
                          byte_index*8);
231                 end if;
232               end loop;
233             when b"01" =>
234               for byte_index in 0 to (C_S_AXI_DATA_WIDTH/8-1) loop
235                 if ( S_AXI_WSTRB(byte_index) = '1' ) then
236                   -- Respective byte enables are asserted as per
                          write strobes
237                   -- slave registor 1
238                   slv_reg1(byte_index*8+7 downto byte_index*8) <=
                          S_AXI_WDATA(byte_index*8+7 downto
                          byte_index*8);
239                 end if;
240               end loop;
241             when b"10" =>
242               for byte_index in 0 to (C_S_AXI_DATA_WIDTH/8-1) loop
243                 if ( S_AXI_WSTRB(byte_index) = '1' ) then
244                   -- Respective byte enables are asserted as per
                          write strobes
245                   -- slave registor 2
246                   slv_reg2(byte_index*8+7 downto byte_index*8) <=
                          S_AXI_WDATA(byte_index*8+7 downto
                          byte_index*8);
```

```
247                 end if;
248               end loop;
249             when b"11" =>
250               for byte_index in 0 to (C_S_AXI_DATA_WIDTH/8-1) loop
251                 if ( S_AXI_WSTRB(byte_index) = '1' ) then
252                   -- Respective byte enables are asserted as per
                         write strobes
253                   -- slave registor 3
254                   slv_reg3(byte_index*8+7 downto byte_index*8) <=
                         S_AXI_WDATA(byte_index*8+7 downto
                         byte_index*8);
255                 end if;
256               end loop;
257             when others =>
258               slv_reg0 <= slv_reg0;
259               slv_reg1 <= slv_reg1;
260               slv_reg2 <= slv_reg2;
261               slv_reg3 <= slv_reg3;
262           end case;
263         end if;
264       end if;
265     end if;
266   end process;
267
268   -- Implement write response logic generation
269   -- The write response and response valid signals are asserted by
       the slave
270   -- when axi_wready, S_AXI_WVALID, axi_wready and S_AXI_WVALID
       are asserted.
271   -- This marks the acceptance of address and indicates the status
       of
272   -- write transaction.
273
274   process (S_AXI_ACLK)
275   begin
276     if rising_edge(S_AXI_ACLK) then
277       if S_AXI_ARESETN = '0' then
278         axi_bvalid  <= '0';
279         axi_bresp   <= "00"; --need to work more on the responses
280       else
281         if (axi_awready = '1' and S_AXI_AWVALID = '1' and
              axi_wready = '1' and S_AXI_WVALID = '1' and axi_bvalid
              = '0'  ) then
282           axi_bvalid <= '1';
283           axi_bresp  <= "00";
284         elsif (S_AXI_BREADY = '1' and axi_bvalid = '1') then
              --check if bready is asserted while bvalid is high)
285           axi_bvalid <= '0';                                 --
                (there is a possibility that bready is always
                asserted high)
286         end if;
287       end if;
```

```
288     end if;
289   end process;
290 
291   -- Implement axi_arready generation
292   -- axi_arready is asserted for one S_AXI_ACLK clock cycle when
293   -- S_AXI_ARVALID is asserted. axi_awready is
294   -- de-asserted when reset (active low) is asserted.
295   -- The read address is also latched when S_AXI_ARVALID is
296   -- asserted. axi_araddr is reset to zero on reset assertion.
297 
298   process (S_AXI_ACLK)
299   begin
300     if rising_edge(S_AXI_ACLK) then
301       if S_AXI_ARESETN = '0' then
302         axi_arready <= '0';
303         axi_araddr  <= (others => '1');
304       else
305         if (axi_arready = '0' and S_AXI_ARVALID = '1') then
306           -- indicates that the slave has acceped the valid read
                address
307           axi_arready <= '1';
308           -- Read Address latching
309           axi_araddr  <= S_AXI_ARADDR;
310         else
311           axi_arready <= '0';
312         end if;
313       end if;
314     end if;
315   end process;
316 
317   -- Implement axi_arvalid generation
318   -- axi_rvalid is asserted for one S_AXI_ACLK clock cycle when
      both
319   -- S_AXI_ARVALID and axi_arready are asserted. The slave
      registers
320   -- data are available on the axi_rdata bus at this instance. The
321   -- assertion of axi_rvalid marks the validity of read data on the
322   -- bus and axi_rresp indicates the status of read
      transaction.axi_rvalid
323   -- is deasserted on reset (active low). axi_rresp and axi_rdata
      are
324   -- cleared to zero on reset (active low).
325   process (S_AXI_ACLK)
326   begin
327     if rising_edge(S_AXI_ACLK) then
328       if S_AXI_ARESETN = '0' then
329         axi_rvalid <= '0';
330         axi_rresp  <= "00";
331       else
332         if (axi_arready = '1' and S_AXI_ARVALID = '1' and
              axi_rvalid = '0') then
333           -- Valid read data is available at the read data bus
```

```
334           axi_rvalid <= '1';
335           axi_rresp  <= "00"; -- 'OKAY' response
336         elsif (axi_rvalid = '1' and S_AXI_RREADY = '1') then
337           -- Read data is accepted by the master
338           axi_rvalid <= '0';
339         end if;
340       end if;
341     end if;
342   end process;
343
344   -- Implement memory mapped register select and read logic
      generation
345   -- Slave register read enable is asserted when valid address is
      available
346   -- and the slave is ready to accept the read address.
347   slv_reg_rden <= axi_arready and S_AXI_ARVALID and (not
      axi_rvalid) ;
348
349   process (slv_reg0, slv_reg1, slv_reg2, slv_reg3, axi_araddr,
      S_AXI_ARESETN, slv_reg_rden)
350   variable loc_addr :std_logic_vector(OPT_MEM_ADDR_BITS downto 0);
351   begin
352       -- Address decoding for reading registers
353       loc_addr := axi_araddr(ADDR_LSB + OPT_MEM_ADDR_BITS downto
          ADDR_LSB);
354       case loc_addr is
355         when b"00" =>
356           reg_data_out <= slv_reg0;
357         when b"01" =>
358           reg_data_out <= slv_reg1;
359         when b"10" =>
360           reg_data_out <= slv_reg2;
361         when b"11" =>
362           reg_data_out <= slv_reg3;
363         when others =>
364           reg_data_out  <= (others => '0');
365       end case;
366   end process;
367
368   -- Output register or memory read data
369   process( S_AXI_ACLK ) is
370   begin
371     if (rising_edge (S_AXI_ACLK)) then
372       if ( S_AXI_ARESETN = '0' ) then
373         axi_rdata  <= (others => '0');
374       else
375         if (slv_reg_rden = '1') then
376           -- When there is a valid read address (S_AXI_ARVALID)
              with
377           -- acceptance of read address by the slave (axi_arready),
378           -- output the read dada
379           -- Read address mux
```

```
380             axi_rdata <= reg_data_out;      -- register read data
381           end if;
382         end if;
383       end if;
384   end process;
385
386
387   -- Add user logic here
388
389   -- User logic ends
390
391 end arch_imp;
```

Código A.3 – Código Verilog de definição da lógica do bloco

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////
3  // Company: Avnet
4  // Engineer: JLB
5  //
6  // Create Date: 09/10/2012 03:32:02 PM
7  // Design Name:
8  // Module Name: PWM_Controller_Int
9  // Project Name: PWM Controller_Int
10 // Target Devices: Any Xilinx FPGA
11 // Tool Versions: Created in Vivado 2013.3
12 // Description: PWM Controller with Interrupt output to PS
13 // Generates Interrupt when invalid PWM range is written into
      block.
14 //
15 // Dependencies:
16 //
17 // Revision:
18 // Revision 0.01 - File Created
19 // Additional Comments:
20 //
21 //////////////////////////////////////////////
22
23
24 module PWM_Controller_Int #(
25     parameter integer period = 20)
26     (
27     input Clk,
28     input [31:0] DutyCycle,
29     input Reset,
30     output reg [0:0] PWM_out,
31   output reg Interrupt,
32   output reg [period-1:0] count
33     );
34
35     // Sets PWM Period.  Must be calculated vs. input clk period.
```

```
36      // For example, setting this to 20 will divide the input clock
            by 2^20, or 1 Million.
37      // So a 50 MHz input clock will be divided by 1e6, thus this
            will have a period of 1/50
38      // reg [period-1:0] count;
39      // reg [0:0] PWM_out;
40
41      always @(posedge Clk)
42       if (!Reset)
43          count <= 0;
44       else
45          count <= count + 1;
46
47       always @(posedge Clk)
48         if (count < DutyCycle)
49             PWM_out <= 1;
50         else
51             PWM_out <= 0;
52
53     always @(posedge Clk)
54         if (!Reset)
55             Interrupt <= 0;
56         else if (DutyCycle > 990000)
57             Interrupt <= 1;
58         else
59             Interrupt <= 0;
60
61  endmodule
```

Já para o código de controle, que roda diretamente no processador sem sistema operacional, isto é, em *bare metal*, o programa principal pode ser visto no código A.4.

Código A.4 – Código principal para controle do movimento da gangorra

```
1  //@Author: Leonardo Teixeira Alves, Rosana Santos Ribeiro
2  //@Date: 2022-04-18
3  //@Vers: 1.0
4
5  #include <stdio.h>
6
7  #include "xparameters.h"
8
9  #include "MPU6050.h"
10 #include "MPU6050_PSHI.h"
11
12 #include "kfilter.h"
13 #include "cfilter.h"
14 #include "xtime_l.h"
15 #include "xil_io.h"
16 #include "xstatus.h"
17 #include "xscugic.h"
18 #include "xil_exception.h"
```

```
19
20 /************************** Constant Definitions
   *****************************/
21 /* The following constant maps to the name of the hardware
   instances that
22  * were created in the Vivado system design. */
23 #define PWM_LEFT_BASE_ADDRESS
   XPAR_PWM_W_INT_0_S00_AXI_BASEADDR
24
25 #define PWM_RIGHT_BASE_ADDRESS
   XPAR_PWM_W_INT_1_S00_AXI_BASEADDR
26
27 /* The following definitions are related to handling interrupts
   from the
28  * PWM controller. */
29 #define INTC_PWM_INTERRUPT_ID
   XPAR_FABRIC_PWM_W_INT_0_INTERRUPT_OUT_INTR
30 #define INTC                        XScuGic
31 #define INTC_HANDLER                  XScuGic_InterruptHandler
32 #define INTC_DEVICE_ID                  XPAR_PS7_SCUGIC_0_DEVICE_ID
33
34 #define INTC_PWM_INTERRUPT_ID1
   XPAR_FABRIC_PWM_W_INT_1_INTERRUPT_OUT_INTR
35
36 #define PWM_IN_MIN_VALUE          0.05
37 #define PWM_IN_MAX_VALUE          0.1
38 #define PWM_OUT_MIN_VALUE         53200
39 #define PWM_OUT_MAX_VALUE         106000
40
41 #define SPEED_SETPOINT            0.06
42
43 /************************** Variable Definitions
   *****************************/
44
45 /*
46  * The following are declared globally so they are zeroed and so
     they are
47  * easily accessible from a debugger
48  */
49
50 /* motor signals are now global to make them visible to the ISR. */
51 volatile u32 signal_left_motor = 0, signal_right_motor = 0;
52 /* The Instance of the Interrupt Controller Driver */
53 static INTC Intc;
54
55 //Bias values taken from average of 6 measurements
56 float gyroBias[3] = {-3.72068633, 2.06643767, -0.29631067};
57 float accelBias[3] = {0.0400585, 0.00805233, 0.0602545};
58
59 /*
60  * Interrupt service routine for the PWM signal
61  *
```

```
62  * @param InstancePtr is a pointer to the XScuGic instance
63  *
64  * @returns None.
65  * */
66 void PWMIsr(void *InstancePtr)
67 {
68   /* Inform the user that an invalid value was detected by the PWM
69     * controller. */
70   print("PWM Value exceeded. PWM signals reset to zero. Please
        enter new value:\r\n");
71
72   /* Set the PWM value to a safe value and write it to the
73     * PWM controller in order to clear the pending interrupt. */
74   signal_left_motor = 0;
75   signal_right_motor = 0;
76   Xil_Out32(PWM_LEFT_BASE_ADDRESS, signal_left_motor);
77   Xil_Out32(PWM_RIGHT_BASE_ADDRESS, signal_right_motor);
78 }
79
80 /*****************************************************************
81 /**
82 * This function sets up the interrupt system for the PWM dimmer
     controller.
83 * The processing contained in this function assumes the hardware
     system was
84 * built with an interrupt controller.
85 *
86 * @param  None.
87 *
88 * @return A status indicating XST_SUCCESS or a value that is
     contained in
89 *   xstatus.h.
90 *
91 * @note   None.
92 *
93 *****************************************************************
94 int SetupInterruptSystem()
95 {
96   int result;
97   INTC *IntcInstancePtr = &Intc;
98
99   XScuGic_Config *IntcConfig;
100
101   /* Initialize the interrupt controller driver so that it is
        ready to
102    * use. */
103   IntcConfig = XScuGic_LookupConfig(INTC_DEVICE_ID);
104   if (IntcConfig == NULL) {
105     return XST_FAILURE;
106   }
107
108   /* Initialize the SCU and GIC to enable the desired interrupt
```

```
109     * configuration. */
110   result = XScuGic_CfgInitialize(IntcInstancePtr, IntcConfig,
111           IntcConfig->CpuBaseAddress);
112   if (result != XST_SUCCESS) {
113     return XST_FAILURE;
114   }
115
116   XScuGic_SetPriorityTriggerType(IntcInstancePtr,
        INTC_PWM_INTERRUPT_ID,
117           0xA0, 0x3);
118
119   XScuGic_SetPriorityTriggerType(IntcInstancePtr,
        INTC_PWM_INTERRUPT_ID1,
120             0xA0, 0x3);
121
122   /* Connect the interrupt handler that will be called when an
123    * interrupt occurs for the device. */
124   result = XScuGic_Connect(IntcInstancePtr, INTC_PWM_INTERRUPT_ID,
125         (Xil_ExceptionHandler) PWMIsr, 0);
126   if (result != XST_SUCCESS) {
127     return result;
128   }
129
130   result = XScuGic_Connect(IntcInstancePtr, INTC_PWM_INTERRUPT_ID1,
131           (Xil_ExceptionHandler) PWMIsr, 0);
132   if (result != XST_SUCCESS) {
133     return result;
134   }
135
136   /* Enable the interrupt for the PWM controller device. */
137   XScuGic_Enable(IntcInstancePtr, INTC_PWM_INTERRUPT_ID);
138
139   XScuGic_Enable(IntcInstancePtr, INTC_PWM_INTERRUPT_ID1);
140
141   /* Initialize the exception table and register the interrupt
        controller
142    * handler with the exception table. */
143   Xil_ExceptionInit();
144   Xil_ExceptionRegisterHandler(XIL_EXCEPTION_ID_INT,
        (Xil_ExceptionHandler)INTC_HANDLER, IntcInstancePtr);
145
146   /* Enable non-critical exceptions */
147   Xil_ExceptionEnable();
148
149   return XST_SUCCESS;
150 }
151
152 /**
153  * This function maps an input value from an
154  * input range to an output range
155  *
156  * @param x input value to map
```

```
157  *
158  * @param in_min input range minimum value
159  *
160  * @param in_max input range maximum value
161  *
162  * @param out_min output range minimum value
163  *
164  * @param out_max output range maximum value
165  *
166  * @returns mapped value
167  *
168  * @note This function does not check if the input value is out of
         bounds
169  *
170  * */
171 float map(float x, float in_min, float in_max, float out_min,
        float out_max) {
172   return ((x - in_min) * (out_max - out_min) / (in_max - in_min) +
          out_min);
173 }
174
175 void startupMotors (float wait_time) {
176
177   xil_printf("Starting up motors...\r\n");
178
179   /* Map the duration of the pulse to
180   * the actual values used for the PWM */
181   signal_right_motor = map(PWM_IN_MIN_VALUE, PWM_IN_MIN_VALUE,
          PWM_IN_MAX_VALUE, PWM_OUT_MIN_VALUE, PWM_OUT_MAX_VALUE);
182   signal_left_motor = map(PWM_IN_MIN_VALUE, PWM_IN_MIN_VALUE,
          PWM_IN_MAX_VALUE, PWM_OUT_MIN_VALUE, PWM_OUT_MAX_VALUE);
183
184   /* Send PWM signal to the motors*/
185   Xil_Out32(PWM_LEFT_BASE_ADDRESS, signal_left_motor);
186   Xil_Out32(PWM_RIGHT_BASE_ADDRESS, signal_right_motor);
187
188   delayMS(wait_time * 1000);
189
190   xil_printf("Motors started successfully!\r\n");
191 }
192
193 float getAngle(kalmanFilter filter[3],  matrix inputData[3],
        matrix Xo1, matrix Po1, matrix Yo1)
194 {
195   float angle = NAN, data[12];
196   u16 remainingAmount = 0;
197   //    mpu6050_calibrate(gyroBias, accelBias);
198
199   if(mpu6050_samplesInFIFO() >= 12){
200     mpu6050_readFifoData(data, 0x0, &remainingAmount);
201
202     /*Converting accelerometer data and adding it to input matrix*/
```

```
203     for(int index = 0; index < 3; index++)
204       data[index] -= accelBias[index];
205
206     inputData[1].v[1][0] = rad_to_deg(atanf(-1.0 * data[0] /
           fsqrt(data[1]*data[1] + data[2]* data[2])));
207     inputData[0].v[1][0] = rad_to_deg(atanf(data[1] /
           fsqrt(data[0] * data[0] + data[2] * data[2])));
208     inputData[2].v[1][0] = 0.0;
209
210     /*Adding gyroscope data to input matrix*/
211     for(int index = 3; index < 6; index++)
212       inputData[index-3].v[0][0] = data[index];
213
214     //printf("%f %f %f \r\n", data[3], data[4], data[5]);
215     //printf("%f %f %f \r\n", gyroData[0].v[0][0],
           gyroData[1].v[0][0], gyroData[2].v[0][0]);
216     kfilter(&filter[0], inputData[0], A1, At1, B1, H1, Ht1, R1,
           Q1);
217     kfilter(&filter[1], inputData[1], A1, At1, B1, H1, Ht1, R1,
           Q1);
218     kfilter(&filter[2], inputData[2], A2, At2, B2, H1, Ht1, R1,
           Q1);
219     //printf("%f %f %f \r\n", filter[0].Xkp.v[0][0],
           filter[1].Xkp.v[0][0], filter[2].Xkp.v[0][0]);
220
221     /* Negate signal so ccw rotation means positive angle*/
222     angle = -filter[0].Xkp.v[0][0];
223   }
224
225   return angle;
226 }
227
228 int main()
229 {
230   int comStatus, PWMStatus = XST_SUCCESS, i = 10000;
231
232   float elapsed_time, pwm_left, pwm_right, error, previous_error =
         0;
233   float PID = 0, pid_p=0, pid_i=0, pid_d=0;
234   float angle, desired_angle = 0;
235
236   /*
237    * From Ziegler-Nichols
238    *
239    * Kcr = 0.085
240    * Tcr = 1.42
241    *
242    * Kp = 0.6*Kcr = 0.051
243    * Ti = 0.5*Tcr = 0.71
244    * Td = 0.125*Tcr = 0.1775
245    * Ki = Kp/Ti = 0.07183
246    * Kd = Kp*Td = 0.0090525
```

```
247     * */
248
249   double kp = 0.0000625;
250   double ki = 0.00000225;
251   double kd = 0.00000625;
252
253   char output_string[100];
254
255   XTime time_measurement_previous, time_measurement;
256
257   init_platform();
258
259   /* Initialize the PWM controller to a safe PWM value. */
260   Xil_Out32(PWM_LEFT_BASE_ADDRESS, 0);
261   Xil_Out32(PWM_RIGHT_BASE_ADDRESS, 0);
262
263   /* Setup the interrupts such that interrupt processing can
        occur.  If an
264     * error occurs while setting up interrupts, then exit the
          application. */
265   PWMStatus = SetupInterruptSystem();
266   if (PWMStatus != XST_SUCCESS) {
267       return XST_FAILURE;
268   }
269
270   /*
271     * Initialize connection with motors here, so that the ESCs
          recognize the signal
272     * Wait for connection, at least 5 seconds recommended
273     * */
274   startupMotors(10);
275
276   //Initialize MPU --- Default initialization method, but not the
        only one.
277   mpu6050_init();
278
279   //Disable sleep mode to make sure MPU is running
280   comStatus = mpu6050_controlSleepMode(FALSE);
281   if(comStatus != XST_SUCCESS)
282       return XST_FAILURE;
283     // ----- Code here ----
284
285 //  printf("accelBias = [%f, %f, %f]\r\ngyroBias = [%f, %f,
      %f]\r\n", accelBias[0], accelBias[1], accelBias[2],
      gyroBias[0], gyroBias[1], gyroBias[2]);
286
287   kalmanFilter filter[3];
288   matrix inputData[3];
289   matrix Xo1 = {.size = {2, 1}, .v = {{0}, {0}}};
290   matrix Po1 = {.size = {2, 2}, .v = {{0.005,0},{0,0.005*5}}};
291   matrix Yo1 = {.size = {1, 1}, .v = {{0.5}}};
292
```

```
293   inputData[0] = nullMatrix(2,1);
294   inputData[1] = nullMatrix(2,1);
295   inputData[2] = nullMatrix(2,1);
296   Xo1.v[1][0] = gyroBias[0];
297   kfilter_init(&filter[0], A1, Q1, R1, H1, &Ht1, &At1, Xo1, Po1,
          Yo1);
298   Xo1.v[1][0] = gyroBias[1];
299   kfilter_init(&filter[1], A1, Q1, R1, H1, &Ht1, &At1, Xo1, Po1,
          Yo1);
300   Xo1.v[1][0] = gyroBias[2];
301   kfilter_init(&filter[2], A2, Q1, R1, H1, &Ht1, &At2, Xo1, Po1,
          Yo1);
302
303   XTime_GetTime(&time_measurement);
304
305   while(i--){
306     time_measurement_previous = time_measurement;
307     XTime_GetTime(&time_measurement);
308
309     elapsed_time = ((float) time_measurement - (float)
            time_measurement_previous)/COUNTS_PER_SECOND;
310
311     angle = getAngle(filter, inputData, Xo1, Po1, Yo1);
312
313     if(isnanf(angle)) {
314       i++;
315       continue;
316     }
317
318     /*Calculating current error*/
319     error = desired_angle - angle;
320
321     /*Proportional part of controller*/
322     pid_p = kp * error;
323
324     /* Integral part of controller */
325     if(-8 < error && error < 8)
326       pid_i += (ki * error);
327
328
329     /*Derivative part of controller*/
330     if (elapsed_time > 0)
331       pid_d = kd * ((error - previous_error)/elapsed_time);
332     else
333       pid_d = 0;
334
335     PID = (pid_p /*+ pid_i + pid_d*/);
336
337     pwm_right = SPEED_SETPOINT + PID/2;
338     pwm_left = SPEED_SETPOINT - PID/2;
339
340     /*if(pwm_right >= 0.0723) {
```

```
341       pwm_right += 0.003;
342     }
343
344     if(pwm_left >= 0.0723) {
345       pwm_left += 0.003;
346     }*/
347
348     /* Right */
349     if(pwm_right < PWM_IN_MIN_VALUE)
350       pwm_right = PWM_IN_MIN_VALUE;
351     else if(pwm_right > 0.85 * PWM_IN_MAX_VALUE)
352       pwm_right = 0.85 * PWM_IN_MAX_VALUE;
353
354     /* Left */
355     if(pwm_left < PWM_IN_MIN_VALUE)
356       pwm_left = PWM_IN_MIN_VALUE;
357     else if(pwm_left > 0.85 * PWM_IN_MAX_VALUE)
358       pwm_left = 0.85 * PWM_IN_MAX_VALUE;
359
360   /* Map the duration of the pulse to
361     * the actual values used for the PWM */
362     signal_right_motor = map(pwm_right, PWM_IN_MIN_VALUE,
          PWM_IN_MAX_VALUE, PWM_OUT_MIN_VALUE, PWM_OUT_MAX_VALUE);
363     signal_left_motor = map(pwm_left, PWM_IN_MIN_VALUE,
          PWM_IN_MAX_VALUE, PWM_OUT_MIN_VALUE, PWM_OUT_MAX_VALUE);
364
365     sprintf(output_string, "ts:%.5f\tAngle = %.4f\tDuty left =
          %.4f%%\tDuty right = %.4f%%\r\n",
          (float)time_measurement/COUNTS_PER_SECOND, angle,
          pwm_left*100, pwm_right*100);
366     xil_printf("i = %04d: %s", i, output_string);
367
368     /* Send PWM signal to the motors*/
369     Xil_Out32(PWM_LEFT_BASE_ADDRESS, signal_left_motor);
370     Xil_Out32(PWM_RIGHT_BASE_ADDRESS, signal_right_motor);
371
372     previous_error = error;
373
374
375   delayMS(195);
376
377   getAngle(filter, inputData, Xo1, Po1, Yo1);
378   getAngle(filter, inputData, Xo1, Po1, Yo1);
379   getAngle(filter, inputData, Xo1, Po1, Yo1);
380
381
382   delayMS(75);
383
384   }
385
386   xil_printf("Goodbye!\r\n");
387
```

```
388     // ---------------------
389     cleanup_platform();
390     return 0;
391 }
```

ANEXO B – Programas utilizados

A seguinte lista compila os programas computacionais utilizados durante o desenvolvimento do projeto e do relatório:

- Vivado 2019.1
- Xilinx SDK 2019.1
- SciDAVis 2.7
- diagrams.net
- Microsoft Excel
- MATLAB
- Overleaf