UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE TECNOLOGIA E GEOCIÊNCIAS
DEPARTAMENTO DE ENGENHARIA MECÂNICA
CURSO DE GRADUAÇÃO EM ENGENHARIA MECÂNICA

ÍTALO JOSÉ DO NASCIMENTO SILVA ARAÚJO DIAS

CONSTRUÇÃO DE PLATAFORMA DE TESTE PARA ALGORITMOS DE CONTROLE DE DRONE

Recife
2021

ÍTALO JOSÉ DO NASCIMENTO SILVA ARAÚJO DIAS

CONSTRUÇÃO DE PLATAFORMA DE TESTE PARA ALGORITMOS DE CONTROLE DE DRONE

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia Mecânica da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Bacharel em Engenharia Mecânica.

Orientador: Pedro Manuel Gonzalez Del Foyo.

Coorientador: Victor Gomes Cardoso.

Recife

2021

Catalogação na fonte:
Bibliotecária Sandra Maria Neri Santiago, CRB4-1267

Folha reservada para ficha catalográfica que deve ser elaborada após a defesa e alterações sugeridas pela banca examinadora.

Para solicitar a ficha catalográfica do trabalho, o usuário deve entrar em contato com a Biblioteca Setorial do Centro Acadêmico ao qual o Programa de Pós-graduação está vinculado.

ÍTALO JOSÉ DO NASCIMENTO SILVA ARAÚJO DIAS

CONSTRUÇÃO DE PLATAFORMA DE TESTE PARA ALGORITMOS DE CONTROLE DE DRONE

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia Mecânica da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Bacharel em Engenharia Mecânica.

Aprovado em: 03/12/2021.

BANCA EXAMINADORA

_____

Prof. Dr. Pedro Manuel Gonzalez Del Foyo (Orientador)
Universidade Federal de Pernambuco

_____

Prof. João Paulo Cerquinho Cajueiro (Examinador Interno)
Universidade Federal de Pernambuco

_____

Prof. Adrien Durand-Petiteville (Examinador Interno)
Universidade Federal de Pernambuco

AGRADECIMENTOS

Gostaria de agradecer primeiramente à Deus por me conceder sabedoria e saúde para chegar até aqui. Segundamente, aos meus pais: Gilson Dias e Vera Luciene. Obrigado por acreditar em mim, me incentivar, me apoiar e participar ativamente na construção dessa plataforma, com todo amor do mundo. Foi com ajuda de vocês que consegui correr em busca dos meus sonhos. Em seguida deixo meu agradecimento ao meu orientador Pedro Manuel Gonzalez Del Foyo que me ajudou com ideias para desenvolver esse trabalho e transmitiu bastante conhecimento. Agradeço também ao meu coorientador Victor Gomes Cardoso que debateu comigo algumas ideias para o desenvolvimento desse trabalho, revisões deste documento e sempre se mostrou disponível para ajudar sempre que precisei. Agradeço também a minha namorada Laís Moura pelo suporte, carinho e pela revisão de algumas partes deste documento. Obrigado ao meu irmão Pedro Lucas, pelos abraços acolhedores. Não menos importante, agradeço aos amigos que fiz durante minha jornada acadêmica, em especial a Igor Melo, Marina Dias e Rodrigo Luiz que compartilharam as conquistas e frustrações de perto e sempre estiveram ao meu lado. Por fim, agradeço a todos aqueles que contribuíram direta e indiretamente para minha evolução. A todos vocês meu muito obrigado!

RESUMO

Este trabalho apresenta o desenvolvimento de uma plataforma estática para testes de algoritmos de controle em drones, com o objetivo de facilitar o ensino da engenharia de controle por meio de uma bancada didática e segura. A plataforma foi desenvolvida com sensores e atuadores que permitem mensurar o comportamento do drone e atuar de maneira análoga à realidade. A solução implementada foi baseada na utilização de placas de Arduino, associado a uma aplicação de computador, desenvolvida em Python, para prover os recursos necessários para os testes da plataforma, como a comunicação entre os dispositivos, a integração entre os componentes eletrônicos e a geração de uma interface gráfica para monitorar o comportamento do drone em tempo real. Com base na arquitetura implementada, foi possível testar algumas estratégias de controle de altitude e do ângulo de inclinação do drone. Os resultados obtidos mostram que a arquitetura proposta de monitoramento e atuação para avaliação dos algoritmos foi um sucesso. No entanto, embora o controle da altitude tenha funcionado adequadamente, controlar o ângulo de forma precisa foi desafiador, estando aberto a melhorias em trabalhos futuros.

Palavras-chave: Drone, *Hardware-In-the-Loop*, Arduino e Python.

ABSTRACT

This work presents the development of a static platform to test control algorithms on drones. The main goal is to ease the teaching of control engineering through a didactic and safe bench. The platform was developed with sensors and actuators, allowing the measurement of the drones' behavior and act in a way analogous to reality. The solution implemented was based on the use of Arduino boards, associated to a computer application, developed in Python, to give the resources needed for the platform tests, like the communication between the devices, the integration of electronic components and the creation of a graphical interface to monitor the behavior of the drone in real time. Based on the architecture implemented, it was possible to test some of the drone's altitude and lean angle control strategies. The results obtained show that the proposed architecture of monitoring and acting for the evaluation of the algorithm was a success. However, while the altitude control worked properly, to control the angle accurately was challenging. Therefore, it is open to future improvements.

Keywords: Drone, Hardware-In-the-Loop, Arduino and Python.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CC	Célula de Carga
DIY	*Do It Yourself*
DR	Drone
VANT	Veículo Aéreo Não Tripulado
RF	Rádio Frequência
m	Massa
NA	Não se aplica
a_z	Aceleração Resultante na Vertical
K_{cc}	Constante para converter o valor da célula de carga em massa
T_{cc}	Tara da célula de carga ou Valor retornado da célula de carga quando os motores estão desligados
V_{cc}	Valor retornado pela célula de carga naquele instante
G	Gravidade
z_i	Altitude do Drone
v_i	Velocidade vertical do Drone
T_s	Tempo de amostragem
PWM	*Pulse Width Module*
ESC	*Electronic Speed Controller*

SUMÁRIO

1 INTRODUÇÃO

Os drones são veículos aéreos não tripulados (VANTs) e podem exercer funções autônomas ou controladas por operadores humanos. Os drones, segundo Nonami et al. (2010), podem ser aplicados em diversas áreas, exercendo o encargo de: inspeção, transporte, georreferenciamento e monitoramento. No âmbito da inspeção, tem-se o exemplo de inspecionar torres eólicas, possibilitando a visualização de difíceis pontos de acesso. Já no campo do transporte, os drones podem ser utilizados para a entrega de mercadorias e suprimentos, agilizando a forma de transporte e facilitando o acesso a determinados lugares. No georreferenciamento, os VANTs são utilizados para mapear o relevo dos locais, que são úteis tanto na agricultura, quanto na área da construção civil. No caso do monitoramento, é possível utilizá-los para fazer contagem dos animais, combater incêndios, dentre outras aplicações. Assim, vê-se que essa tecnologia é aplicada em diversos âmbitos e pode ajudar no desenvolvimento dessas áreas.

Apesar de demonstrar ser útil em diversas aplicações, os drones não são uma tecnologia de baixo custo e seu desenvolvimento pode ser perigoso. Pois, como é um sistema instável pode ocasionar acidentes, principalmente quando suas hélices giram em alta velocidade, havendo possibilidade de ferir os desenvolvedores, modificar o local do desenvolvimento e auto destruir-se, em caso de queda.

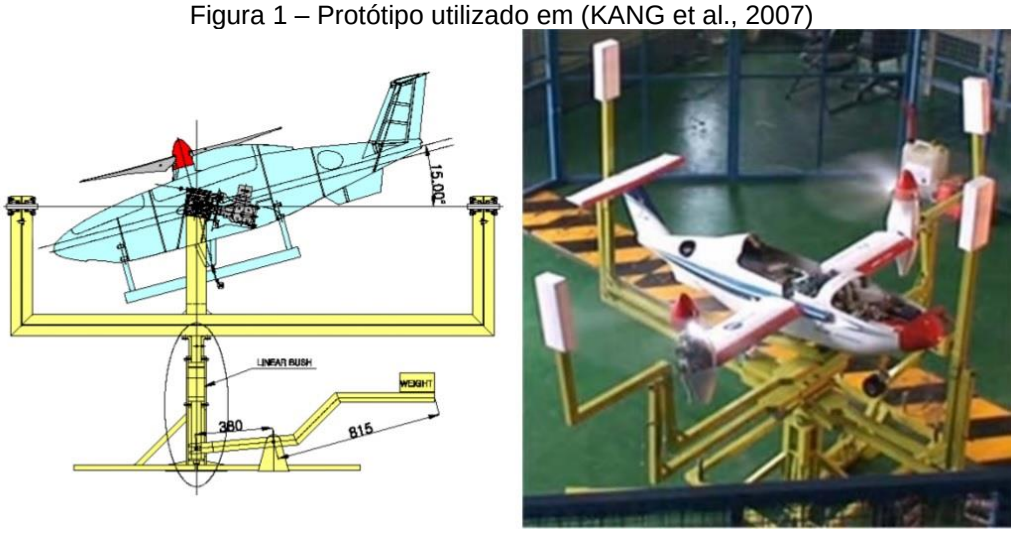
Segundo Silva e Libardi (2016), muitos centros de pesquisa pelo mundo desenvolvem tecnologias para drones, mas percebe-se que no Brasil ainda é pouco popular. Com essa diversidade de aplicações, baixa popularidade no Brasil e baixa segurança nos testes de drones, esse trabalho de conclusão de curso tem o intuito de desenvolver uma plataforma para testes de algoritmos de controle de drone, onde possa testar algoritmos de controle nessa plataforma, sem colocar em risco a integridade física do drone e das pessoas ao redor. Essa plataforma ficará disponível para a UFPE usar como base para estudos futuros e ser utilizada para o ensino de engenharia de controle, aumentando o número de recursos disponíveis para o ensino de conteúdos neste campo do conhecimento.

A estrutura desta plataforma consiste em um drone que pode se movimentar num plano. O drone poderá fazer movimentos de rotação em torno de um eixo, porém, limitando-se a não executar os movimentos de translação. A inspiração para esta plataforma veio do trabalho de V.LARA et al. (2018), que possui um protótipo físico em que, tem-se a simulação de uma parte e a outra parte é real. A inovação da plataforma apresentada neste trabalho consiste na medição da força de arrasto, produzida pelas hélices, por meio do uso de uma célula de carga presente no sistema.

Esse tipo de teste em que o VANT é avaliado em uma plataforma sem estar livre para voo é chamado de *Ground Test*, e vem sendo utilizado para testar alguns objetos voadores, como pode ser visto em KANG et al. (2007) e YOO et al. (2010). As Figuras 1 e 2 mostram as plataformas que foram utilizadas para realizar esses testes com aeronaves. Como uma das etapas do *Ground Test* tem-se o *Hardware In the Loop* (HIL), que é uma etapa de um projeto que consiste em testar o software de controle junto com o hardware que vai ser utilizado no produto final, mas sem ter o produto final ainda. O teste de HIL é muito utilizado na indústria automotiva, pois é possível simular várias condições do carro em um simulador e analisar como o sistema se comportaria com a estratégia de controle que está sendo utilizada no momento.

Além da vantagem de poder testar os recursos de software e hardware num protótipo muito similar ao produto final, esse tipo de metodologia proporciona mais segurança ao processo de desenvolvimento e testes.

Figura 1 – Protótipo utilizado em (KANG et al., 2007)



Fonte: (KANG et al., 2007)

Figura 2 – Protótipo utilizado em (YOO et al., 2010)

Fonte: (YOO et al., 2010)

Neste trabalho, o problema do controle de um drone foi simplificado para um drone de duas hélices que se move num plano com o intuito de facilitar o estudo e o ensino da engenharia de controle. Esta simplificação foi planejada de modo a não tornar muito trabalhoso a modificação deste formato para um drone com mais graus de liberdade. Visto que para fazer isto, seria necessário adicionar mais eixos de rotação e replicar a estratégia utilizada para o único eixo aos demais. O hardware utilizado no VANT simplificado seria praticamente o mesmo utilizado na versão tridimensional, com exceção de que a nova versão contaria com um número maior de motores. Dessa forma, o trabalho realizado no drone simplificado pode ser reaproveitado para construção de um VANT tridimensional.

1.1 OBJETIVOS

Os objetivos deste trabalho se dividem em objetivo geral e objetivos específicos, sendo estes últimos imprescindíveis para alcançar do objetivo geral.

1.1.1 Objetivo geral

Este trabalho tem por objetivo projetar e construir uma plataforma que permita avaliar o comportamento dinâmico de um drone de forma simplificada e segura.

1.1.2 Objetivos específicos

I. Realizar um estudo do estado da arte do funcionamento de drones;
II. Projetar e construir a plataforma de teste, integrando os componentes do drone a serem testados;
III. Projetar o sistema de aquisição que permite avaliar o funcionamento do drone e sua interface com o computador;
IV. Desenvolver os algoritmos que permitem coletar a informação de altitude estimada e inclinação, assim como os acionamentos que garantem a operação do drone;
V. Realizar os testes de integração da plataforma e avaliar o funcionamento da mesma.

1.2 ESTRUTURA DO TRABALHO

O presente trabalho está dividido em 5 capítulos. O primeiro introduz o contexto atual dos drones, mostra sua importância, e além disso mostra a justificativa deste trabalho e seus objetivos gerais e específicos.

No segundo capítulo, tem-se a fundamentação teórica que é necessária para o entendimento do funcionamento dos drones e construção da bancada didática.

O terceiro capítulo apresenta a metodologia que foi utilizada para construir a bancada, fazer os algoritmos que foram embarcados no drone e no computador, integrar todo o conjunto e mostrar como todos os módulos trocam informações entre si.

No quarto capítulo, apresenta-se os resultados obtidos para o controle de altitude do drone, o controle do ângulo e os dois controles funcionando em conjunto.

O quinto capítulo descreve as principais conclusões obtidas pelo trabalho, mostrando a importância do que foi construído, o que pode ser melhorado e sugestões de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo aborda alguns dos principais conteúdos para prover uma melhor compreensão acerca dos temas explorados no trabalho.

2.1 Veículo Aéreo Não Tripulado (VANT)

Segundo Behnck (2014), veículos aéreos não tripulados, também conhecidos como drones, são definidos como aeronaves que não carregam um operador e podem ser controladas remotamente ou de forma autônoma. Os VANTs podem ser classificados como asa fixa ou asa rotativa, como podem ser vistos nas Figuras 3 e 4.

Figura 3 – VANT de asa fixa

Fonte: Delair® (2021, imagem digital)

Figura 4 – VANT de asa rotativa

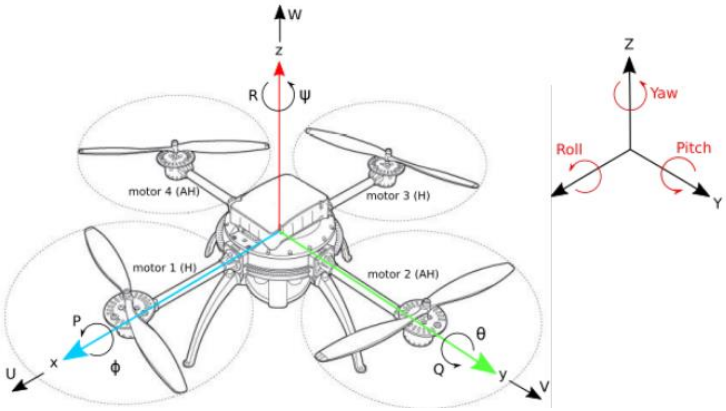
Fonte: DJI® (2021, imagem digital)

Como pode ser visto na Figura 4, cada extremidade do drone tem um motor com uma hélice, seus eixos de rotação são paralelos, fixos e verticais. Com essa configuração, os únicos atuadores relevantes para controlar o

movimento é a velocidade de rotação das hélices. Vale salientar que se deve ter um conjunto de hélices girando no sentido horário e outro no sentido anti-horário, para que exista conservação do momento angular e não exista um torque resultante no eixo vertical. Dessa forma evitando um rotor traseiro, como é o caso do helicóptero.

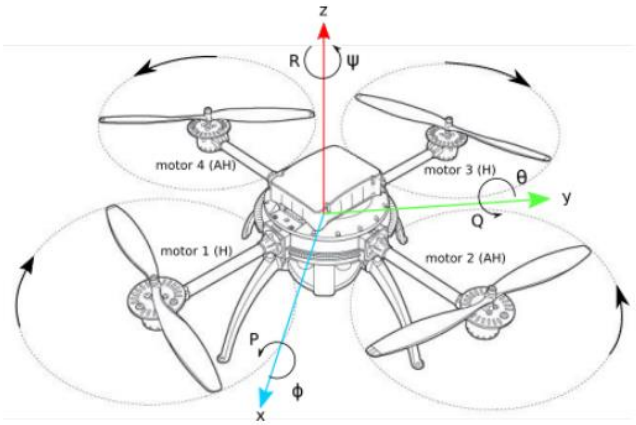
Os eixos cartesianos podem ser posicionados no esquema em “+” e em “x” como mostram as Figuras 5 e 6, para o presente trabalho, a configuração adotada será o esquema em “+”. O sistema possui 6 graus de liberdade, mas no caso do quadricóptero, que é o drone que este trabalho está se baseando, só tem 4 atuadores, fazendo com que o sistema seja subatuado. Das variáveis disponíveis, as selecionadas para melhor controlar o sistema são: *Throttle* ("quantidade de aceleração"), *Roll* ("balanço"), *Pitch* ("arfagem") e *Yaw* ("guinada").

Figura 5 – Esquema de eixos em “+”



Fonte: (MADRUGA, 2018)

Figura 6 – Esquema de eixos em “x”

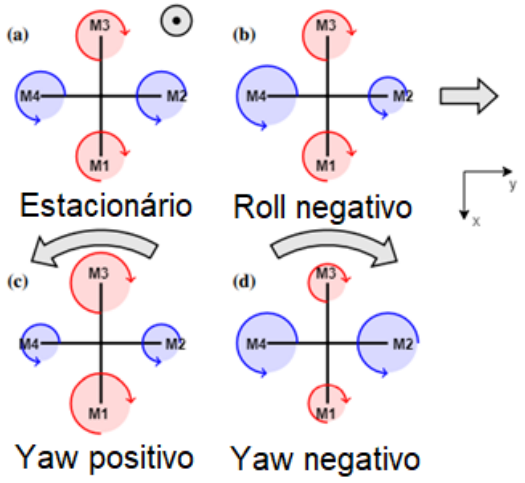


Fonte: (MADRUGA, 2018)

Explicando mais detalhadamente cada variável selecionada, temos que:

- *Throttle* é responsável pelo movimento translacional na direção z, onde os eixos cartesianos estão fixos em relação ao drone, esse movimento é causado pelo aumento ou diminuição das velocidades dos quatros rotores na mesma quantidade. Na Figura 7(a), acontece o aumento do *Throttle*, dessa forma causando uma força no sentido saindo da folha.
- *Roll* é responsável pelo movimento rotacional em torno do eixo x, para ter esse movimento deve-se aumentar a velocidade do motor 4 e diminuir a do motor 2 proporcionalmente, ou vice-versa, essa proporcionalidade se faz necessária para manter a altitude. Com esse movimento de *Roll,* o empuxo resultante terá componente horizontal, dessa forma irá gerar uma força para a direita como se pode ver na Figura 7(b).
- *Pitch* é responsável pelo movimento rotacional em torno do eixo y, e ele funciona de uma forma bem similar ao *Roll*.
- *Yaw* é responsável pelo movimento rotacional em torno do eixo z, para ter esse movimento é necessário aumentar a velocidade dos motores que giram no sentido horário e diminuir a velocidade dos motores que giram no sentido anti-horário, dessa forma cria-se um torque resultante no sistema fazendo com que o drone gire no sentido anti-horário em torno do eixo z, como pode-se ver na Figura 7(c). Na Figura 7(d) pode-se ver a mesma situação invertida.

Figura 7 – Esquema simplificado da movimentação do drone



Fonte: (MONTEIRO, 2015) Adaptada

2.2 Controlador PID

O controlador PID é uma das estratégias mais empregadas nos controles industriais. Em um sistema a ser controlado, tem-se a variável manipulada que é a ação de entrada para o atuador, a variável controlada que é a variável que se deseja ter num valor determinado, em que esse valor determinado é o *setpoint*. Em cada iteração do controle, a variável controlada é medida e comparada com o *setpoint*. Com essa comparação pode ser calculada três tipos de ações que no final do processo são somadas para ser enviada para a variável manipulada e controlar o processo. Os três tipos de ações são: Proporcional, Integrativa e Derivativa.

2.2.1 Ação Proporcional

A ação proporcional, como o nome sugere, é uma ação que é proporcional ao erro. Essa ação pode ser calculada da seguinte maneira:

$$Ação Proporcional = K_P \cdot (Medida(t) - Setpoint)$$

Onde a Medida(t) é o valor atual da variável controlada, e o K_p é uma constante proporcional que deve ser ajustada para atender aos requisitos de projeto. A ação proporcional atua na resposta transitória do sistema. Quanto maior for o K_p, mais rápida é a resposta do sistema. Em alguns casos, apenas a ação proporcional não é suficiente para controlar o sistema, fazendo com que seja necessário adicionar ações de outras naturezas para obter melhores respostas.

2.2.2 Ação Integrativa

A ação integrativa, é uma ação que é proporcional à integral do erro. Essa ação pode ser calculada da seguinte maneira:

$$Ação Integrativa = K_i \cdot \int (Medida(t) - Setpoint)\, dt$$

Onde K_i é uma constante integrativa que deve ser ajustada para atender aos requisitos de projeto. A ação integrativa tem como principal vantagem a

redução do erro em regime permanente. No entanto, ela pode impactar negativamente o comportamento do sistema no regime transitório.

Muitas plantas possuem limitações físicas que fazem com que a integração do erro deixe de afetar a saída do sistema, a este comportamento se dá o nome de *Windup* (BOHN; ATHERTON; 1995). Para evitar que a integração assuma valores elevados, e com isso gere um retardo na resposta, pode-se implementar alguma técnica de *anti-Windup*. Essas técnicas consistem, basicamente, na saturação da integral na ação integrativa.

2.2.3 Ação Derivativa

A ação derivativa é uma ação proporcional à derivada do erro no tempo, ou seja, é proporcional à velocidade que o erro varia. Ela pode ser calculada da seguinte maneira:

$$Ação Derivativa = K_d \cdot \frac{d(Medida(t) - Setpoint)}{dt}$$

Onde K_d é uma constante derivativa que deve ser ajustava para atender aos requisitos de projeto. A ação derivativa é principalmente utilizada para melhorar a resposta do sistema no regime transitório. Devido às suas características de funcionamento, esta ação induz o sistema a acompanhar a tendência do erro, colaborando para a correção do mesmo de forma preditiva.

Para cada ação citada anteriormente, é necessário encontrar os valores das constantes K_p, K_i e K_d para que o projeto atenda aos critérios especificados, como sobrepasso e tempo de assentamento. Encontrar essas constantes pode ser uma tarefa árdua, mas existem algumas técnicas que podem ser utilizadas para encontrar essas constantes de uma maneira mais fácil, como por exemplo usar Algoritmo Genético. A sintonização do controle PID usando Algoritmo Genético pode ser visto em Khuwaja et al. (2018), nesse trabalho pode ser visto que a performance do controle da altitude do drone melhorou bastante quando comparado ao controle sem a utilização do algoritmo genético para a sintonização dos parâmetros do PID.

2.3 Arduino

Segundo MCROBERTS (2010), Arduino é uma plataforma de computação física ou embarcada que pode interagir com o ambiente externo por hardware e software. As placas de Arduino tem portas de entrada e saída que permitem a comunicação com atuadores e sensores. Essas plataformas de desenvolvimento permitem controlar componentes físicos através de programação e assim implementar algoritmos de controle. A programação do Arduino é baseada na linguagem C++. Por meio dela, é possível estabelecer uma comunicação serial com outros dispositivos digitais e trocar informações entre múltiplas plataformas. Deste modo, é possível desenvolver aplicações no computador e fazer a comunicação com o Arduino por meio da porta Serial.

2.3.1 Linguagem de programação C++

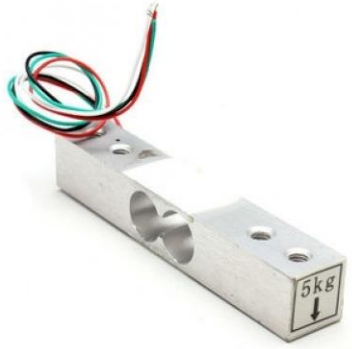
A linguagem de programação C++ é compilada, de multi-paradigmas e ela é baseada na linguagem C. Através dessa linguagem é possível programar o Arduino para executar comandos. Para fazer essa programação existe uma *Interface Development Enviroment* que possibilita digitar todo o código do Arduino, compilar e transferir o código-fonte para o micro controlador.

2.4 Célula de Carga

Célula de carga é um transdutor de força, nesse dispositivo tem um sensor chamado *strain gauge*, que é um resistor de folha fina. O strain gauge é um sensor que varia sua resistência de acordo com a tensão mecânica que é aplicado sobre ele. Esse fenômeno foi descoberto por Lord Kelvin em 1856. (MULLER et al., 2010)

A célula de carga tem uma geometria específica para que o *strain gauge* fique numa posição satisfatória para captar adequadamente a tensão mecânica. Na Figura 8, o *strain gauge* está localizado na parte branca, local de menor área transversal da peça, fazendo com que se tenha uma maior tensão superficial naquela localização.

Figura 8 – Célula de Carga de 5kg



Fonte: VIDA DE SILÍCIO® (2021, imagem digital)

Os *strain gauges* são dispostos no circuito com um arranjo de Ponte de Wheatstone. Isso implica que é necessário colocar uma tensão de excitação no circuito e, de acordo com a medida que pode ser mensurada no desequilíbrio da ponte, pode-se obter uma relação com a força aplicada na célula de carga. Para descobrir o valor da força é necessário realizar uma calibração para encontrar a relação entre a tensão elétrica e a força aplica na célula de carga. Geralmente, esse desequilíbrio na ponte possui um valor muito pequeno e se faz necessário utilizar amplificadores de sinais.

2.5 Acelerômetro e giroscópio

Acelerômetro é um sensor capaz de identificar a aceleração de um sistema, e isso pode acontecer de diversas maneiras, pode ser por indução magnética, piezoeletricidade ou até mesmo sensores ópticos. Com essa aceleração é possível conhecer a inclinação de um determinado sistema, mas esse sensor é muito sensível a vibração.

Giroscópio é um sensor capaz de mensurar a rotação relativa de um sistema, mas não é preciso para dizer a inclinação absoluta. Então, para se utilizar o melhor dos dois sensores e obter uma inclinação de um sistema mais precisa, pode-se utilizar o filtro de Kalman.

O filtro de Kalman é responsável por fazer a fusão de sensores, ele faz a estimativa do próximo estado do sistema e dependendo da calibração do sistema pode-se tirar algumas incertezas e ruídos da medida.

2.6 Python

Python é uma linguagem de programação muito poderosa e de alto nível, divulgada em 1991 pelo programador e matemático holandês Guido Van Rossum. Segundo uma pesquisa do *Stack Overflow* em 2021, Python é a terceira linguagem de programação mais usada no mundo e a mais querida pelo seu quinto ano consecutivo. Uma das características dela é que ela é interpretada e não compilada, no qual faz com que não precise perder tempo para compilar e depois rodar o script, essa vantagem dá uma liberdade para testar comandos direto na linha de comando. Outra vantagem é que o Python tem uma sintaxe muito próxima do inglês, no qual faz com que a leitura do código seja muito mais intuitiva. Nessa linguagem de programação existem diversas bibliotecas para aplicações nas mais diversas áreas de atuação como visão computacional, inteligência artificial, automação de processos, jogos, entre outras. Como destaque, pode-se citar a biblioteca serial, onde é possível fazer uma interface de comunicação via porta serial entre o Arduino e o computador. Outro destaque é a biblioteca *PyGame* que possibilita a criação de interfaces para jogos, mas também pode ser utilizada para mostrar em tempo real a posição simulada de um drone e capturar teclas do teclado para controlar o setpoint da posição do drone, por exemplo.

2.7 Motor Brushless DC e ESC

O motor brushless DC funciona com corrente contínua e não tem escovas. O motor tem o rotor e o estator, onde o rotor é composto pelos ímãs permanentes e o estator pelas bobinas que são energizadas. Essas bobinas quando energizadas criam uma força de atração ou repulsão, dependendo do sentido no qual a corrente flui, para rotacionar o rotor. A energização das bobinas é feita através de um ESC (*Eletronic Speed Controller*). Logo, o controle de velocidade do motor é feito pela frequência na qual as bobinas são energizadas pelo ESC. O ESC é controlado através de um sinal PWM com o mesmo padrão no qual os servos motores para Arduino são controlados, então para facilitar o acionamento do ESC pode-se usar a biblioteca “Servo.h”, onde já se tem funções prontas para

enviar esse PWM. O comando para acionar os servos motores através da função *servo.write* pode variar de 0 a 180, mas para acionar o conjunto ESC mais motor brushless deve ser enviado um valor de 17 a 169. Para qualquer valor que não esteja dentro do intervalo citado anteriormente, o ESC entra num modo de configuração.

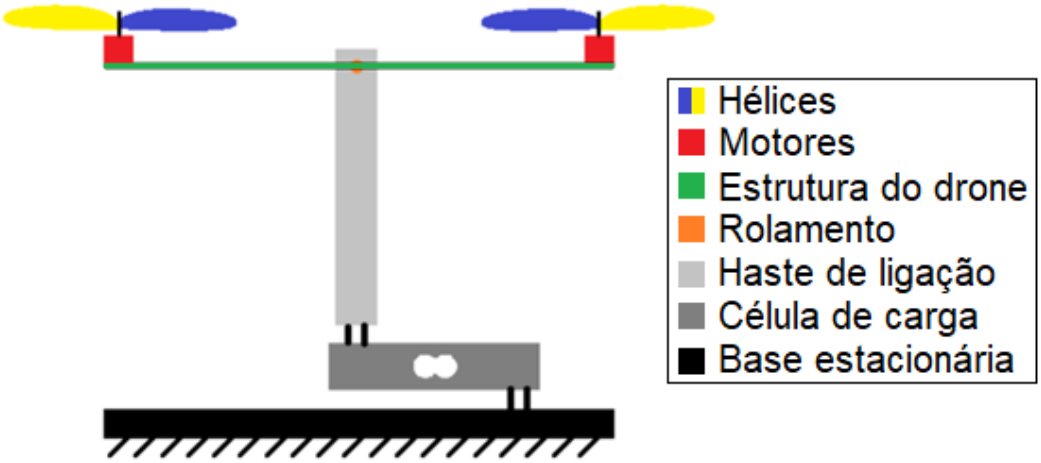
3 METODOLOGIA

Neste capítulo, serão abordados detalhes da metodologia aplicada nesse trabalho.

Normalmente, os testes dos algoritmos de controle para drones são feitos através da avaliação da capacidade dos mesmos de executar trajetórias pré-determinadas (de forma autônoma ou informada através de dispositivos de RF). Para fazer isso, um protótipo foi desenvolvido para emular a dinâmica de um drone de maneira estática. Por critérios de simplificação, foi definido que o protótipo permitirá a avaliação das trajetórias, por ele percorridas de maneira estimada, ao longo do plano XZ. Essa abordagem foi pensada desta maneira para ser utilizada, com praticidade, em Laboratórios didáticos no ensino de teoria de controle. Isto foi proposto visto que os algoritmos de controle aplicados para um drone que se movimenta num plano são similares aos algoritmos de um drone que se movimenta no espaço. Para o drone que se movimenta no plano, é preciso utilizar apenas duas variáveis de controle que são o *throttle* e o *pitch*, já para o espaço é necessário controlar mais duas variáveis que são o *roll* e o *yaw*, o algoritmo para controlar o *roll* funciona de forma muito similar ao *pitch*, mas o *yaw* funciona de uma forma diferente.

Para realizar esta tarefa, foi projetada uma plataforma como está mostrada de forma simplificada na Figura 9.

Figura 9 – Esquema da montagem mecânica da plataforma



Fonte: Produzida pelo autor

Uma vez que o algoritmo de controle a ser avaliado esteja carregado nos microcontroladores, a trajetória definida será executada. Os sensores de altitude (célula de carga) e inclinação (acelerômetro) capturam a execução dessa trajetória de forma que se possa comparar a trajetória pré-definida com a executada, o que permite avaliar a capacidade do algoritmo de controle executar, de forma precisa, a trajetória planejada.

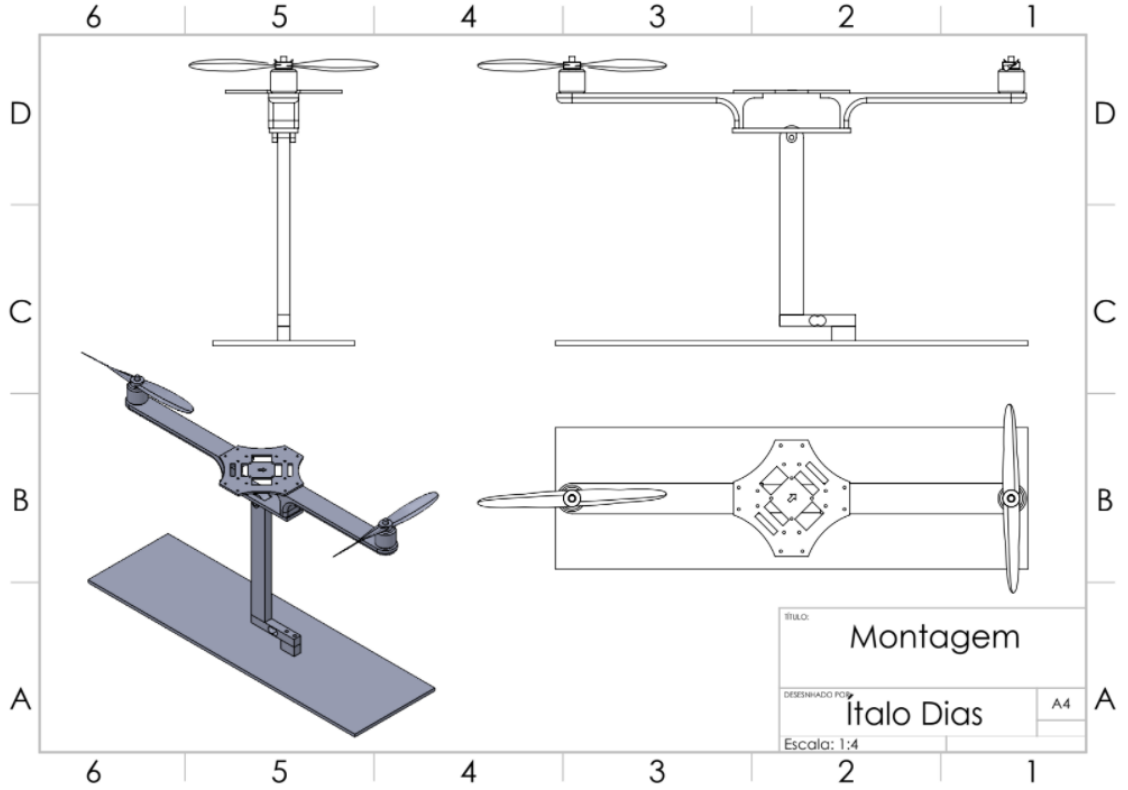
3.1 Projeto da plataforma de suporte para drone 2D

A construção da plataforma teve como base a ideia da Figura 9. Com essa idealização, foi adquirida uma estrutura em fibra de nylon e em fibra de vidro que é empregada na montagem de drones DIY, e essa parte pode ser vista na Figura 10. Com a chegada dos equipamentos, foi feita a modelagem num software CAD de uma plataforma para apoiar a estrutura da Figura 10 numa base fixa. Essa modelagem foi feita de maneira que permitisse a rotação do drone em torno de um eixo e tivesse um apoio para célula de carga, o resultado da modelagem pode ser visto na Figura 11.

Figura 10 – Estrutura do drone comprada

Fonte: Produzida pelo autor

Figura 11 – Resultado da modelagem que foi enviada para fabricação



Fonte: Produzida pelo autor

Na fabricação, foi utilizada uma chapa de aço na base para ter bastante peso e deixar a estrutura estável. Na chapa de aço foi feito um degrau para poder fixar a célula de carga. Foi adquirido um tubo de alumínio retangular e uma cantoneira de alumínio para poder fazer a fixação do tubo na célula de carga. Posteriormente, a estrutura foi levada para um torneiro fazer o mecanismo de rotação da estrutura, para isso foi utilizado um tubo circular, um rolamento, um eixo e chapas de alumínio. Com isso, toda a estrutura mecânica ficou pronta e pode ser vista na Figura 12.

Figura 12 – Estrutura mecânica da plataforma

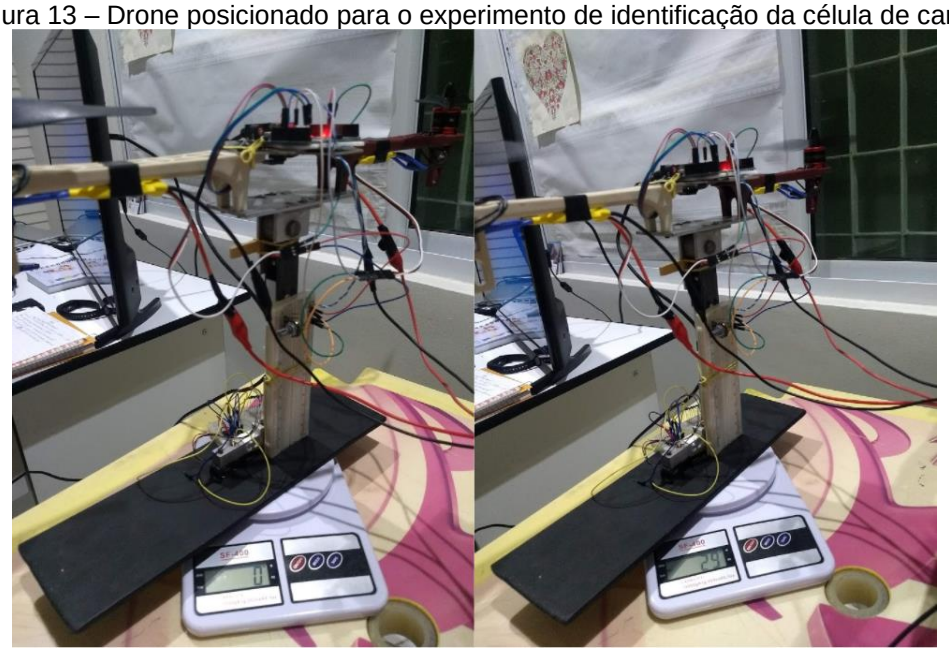
Fonte: Produzida pelo autor

3.2 Algoritmo de estimação da altitude do drone

Para estimar a altitude do drone, a célula de carga do conjunto foi utilizada para mensurar as forças de empuxo geradas pelos motores. Inicialmente foi feita uma identificação da célula de carga, que consiste em tratar o sinal analógico que é lido pelo Arduino, discretizado em valores no intervalo de 0 a 1023, e convertê-lo para um valor de força. Para realizar a identificação, foi feito um

experimento onde a rotação do drone em torno do eixo *y* foi bloqueada, dessa forma mantendo o drone na horizontal, e foi colocada toda a estrutura em cima de uma balança digital. Esse experimento consiste em variar a potência dos motores (consequentemente variando o empuxo gerado), capturar a leitura da célula de carga no Arduino e a leitura, em gramas, na balança. Esse experimento foi realizado 3 vezes e na Figura 13 é possível ver a plataforma posicionada para a realização do experimento.

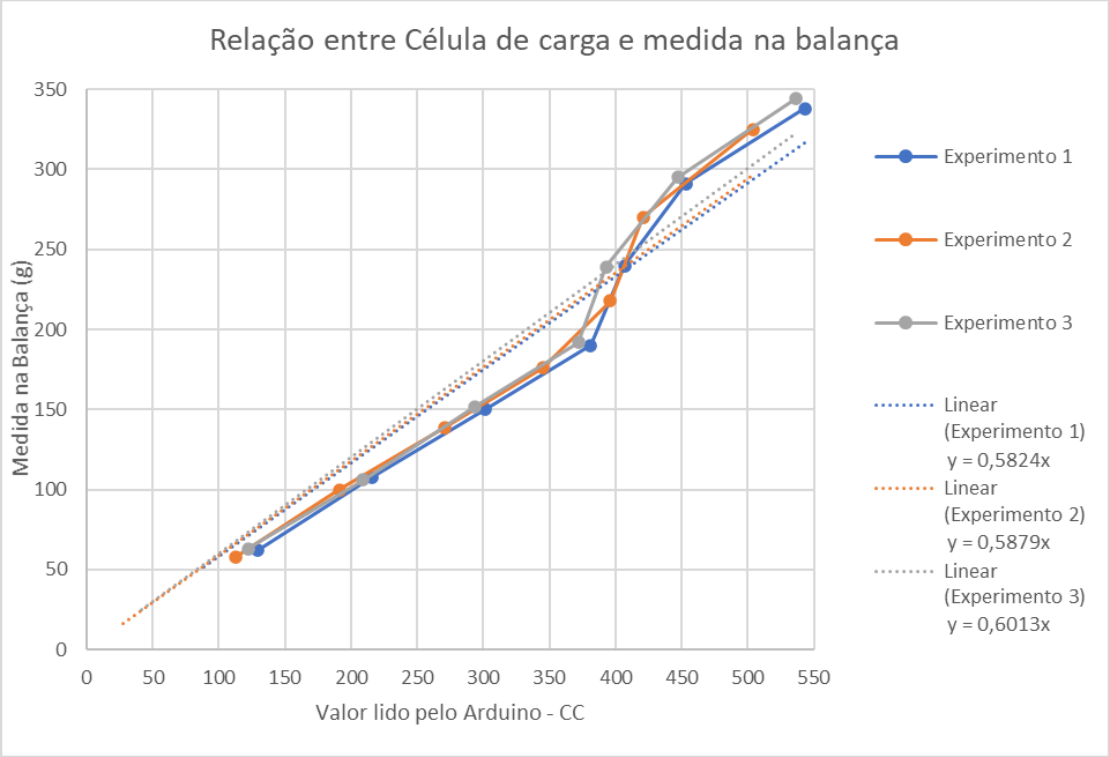
Figura 13 – Drone posicionado para o experimento de identificação da célula de carga



Fonte: Produzida pelo autor

A partir do tratamento dos dados obtidos experimentalmente constatou-se uma relação linear entre a leitura da tensão lida pelo Arduino e a leitura da força gerada na balança, como pode ser visto na Figura 14. Uma regressão linear foi aplicada para cada experimento feito e no final foi feita a média entre os 3 coeficientes angulares para obter a constante que relaciona o valor lido da célula de carga e a força de empuxo. Foi adotada uma regressão linear para facilitar a implementação, pois dessa forma só precisaria multiplicar o coeficiente angular da reta pela leitura da célula de carga para obter a força de empuxo.

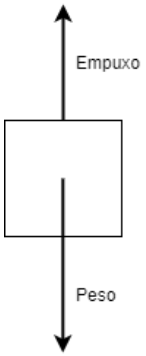
Figura 14 – Gráfico da relação entre o valor lido pela célula de carga e medida na balança



Fonte: Produzida pelo autor

Após a obtenção da equação de identificação da célula de carga, foi feita uma modelagem do sistema para poder estimar a altitude. Para essa modelagem, primeiramente é necessário encontrar a aceleração resultante na vertical. O diagrama de forças que atua no drone na vertical pode ser visto na Figura 15. É considerado que a célula de carga captura apenas as forças na vertical, mesmo que o drone não esteja na horizontal. Com esse diagrama de força é possível calcular a aceleração resultante vertical através da seguinte equação.

Figura 15 – Diagrama de forças atuantes no Drone na vertical



Fonte: Produzida pelo autor

$$Força_{Resultante} = Empuxo - Peso$$
$$m \cdot a_z = K_{CC} \cdot (T_{CC} - V_{CC}) \cdot g - m \cdot g$$
$$a_z = \frac{K_{CC} \cdot (T_{CC} - V_{CC}) \cdot g - m \cdot g}{m}$$

Após o cálculo da aceleração vertical resultante, o próximo passo foi inseri-la na equação horária da posição, dessa forma a altitude do drone pode ser calculada para cada iteração. A equação horária da posição e a equação horária da velocidade podem ser vistas nas respectivas equações seguintes. Para poder usar essas equações, é considerado que a aceleração é constante entre uma iteração e outra.

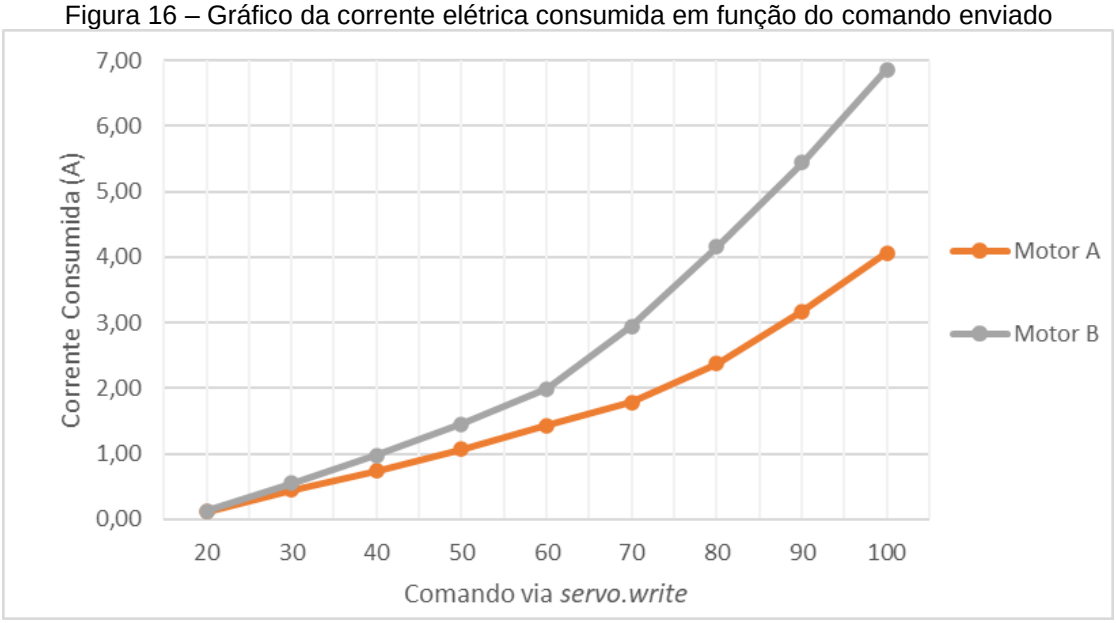
$$z_{i+1} = z_i + v_i \cdot T_s + \frac{a_z \cdot {T_s}^2}{2}$$
$$v_{i+1} = v_i + a_i \cdot T_s$$

3.3 Controle de malha aberta dos motores brushless

Ao realizar o acionamento simultâneo dos motores, notou-se que os motores não geravam um mesmo empuxo e não consumiam a mesma corrente elétrica quando era enviado um mesmo comando via *servo.write* para os dois motores. Então, foi realizada uma identificação/caracterização dos dois motores, correlacionando o sinal de comando de entrada com a corrente consumida para cada um deles. O intuito deste procedimento é ter a capacidade de implementar o controle em malha aberta da corrente elétrica.

Para realizar a identificação, um motor foi analisado por vez, o método consistiu em enviar um comando via *servo.write* a cada 10 segundos, onde o comando começa no valor de 20, é incrementado de 10 em 10 e tem o valor final de 100. O valor 100 foi adotado como limite porque a corrente já estava ficando muito alta e a intenção era ter uma corrente máxima de 4A. A cada comando enviado, a corrente elétrica consumida pelos motores foi medida através do visor da fonte de alimentação. Esse experimento foi realizado 3 vezes para cada motor. O gráfico da corrente elétrica em função do comando enviado pode ser visto na Figura 16.

Figura 16 – Gráfico da corrente elétrica consumida em função do comando enviado

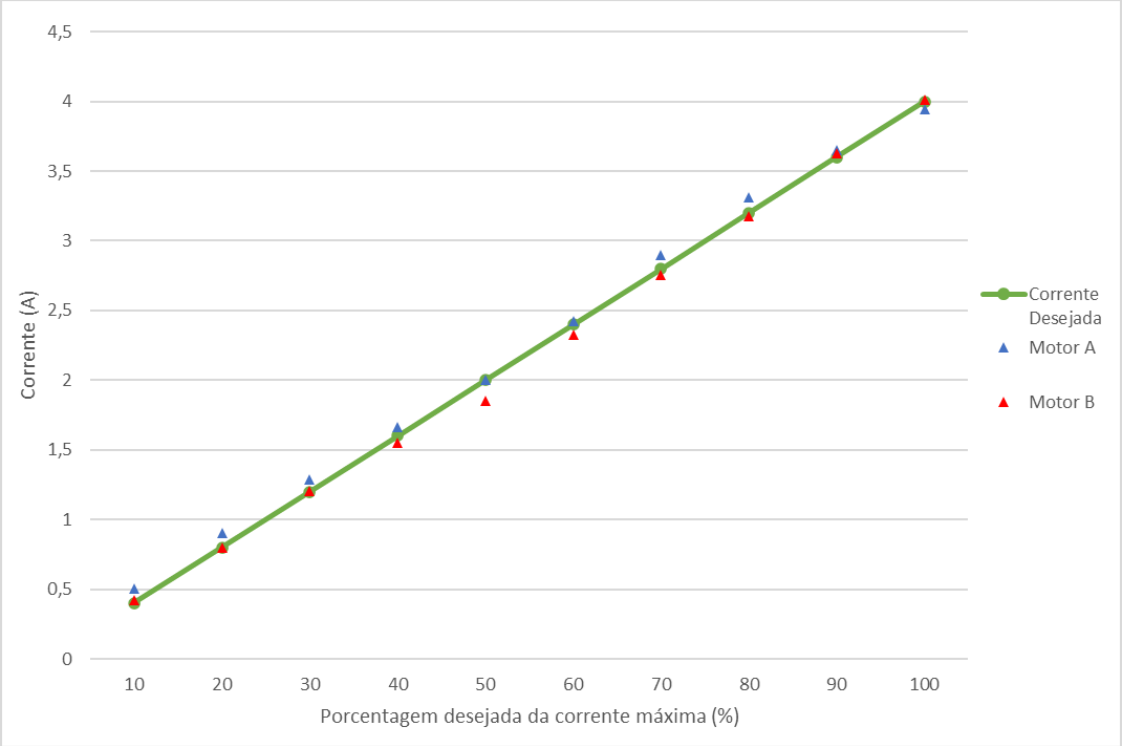


Fonte: Produzida pelo autor

A partir da análise do gráfico, constatou-se que os motores se comportavam de maneira diferente. Foi feito uma regressão linear para encontrar uma equação para cada motor, no qual a entrada dessa equação contém a corrente desejada que o motor deve consumir e a saída é comando que deve ser enviado para o motor.

Após a determinação da equação que rege cada motor, foi definido que a corrente máxima que cada motor deve consumir é 4A, pois com essa corrente os motores já geram um empuxo bastante considerável e já é suficiente para controlar o drone. Em seguida um experimento foi realizado para validar a equação encontrada. Nesse experimento, foi definida a corrente desejada e foi mensurado se a corrente consumida pelos motores correspondia com a corrente desejada, e o resultado do experimento pode ser visto na Figura 17. É notório que as correntes ficaram bem próximas das correntes desejadas. Dessa forma, à medida que os controles de ângulo e altitude do drone calculam a porcentagem da corrente que o controle deve enviar para os motores, a equação obtida será responsável por calcular o comando a ser enviado para os motores.

Figura 17 – Validação do modelo motores

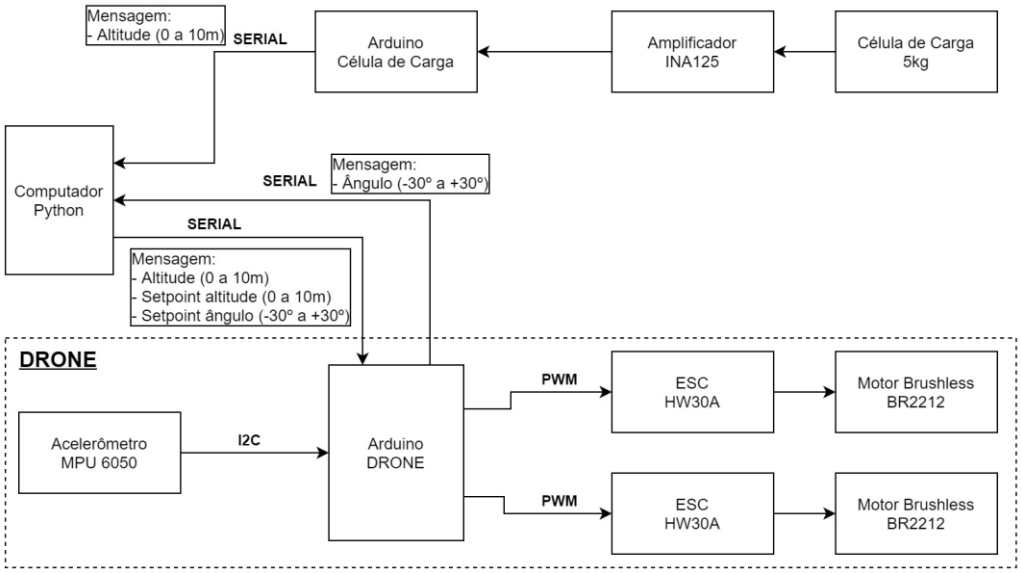


Fonte: Produzida pelo autor

3.4 Arquitetura do sistema

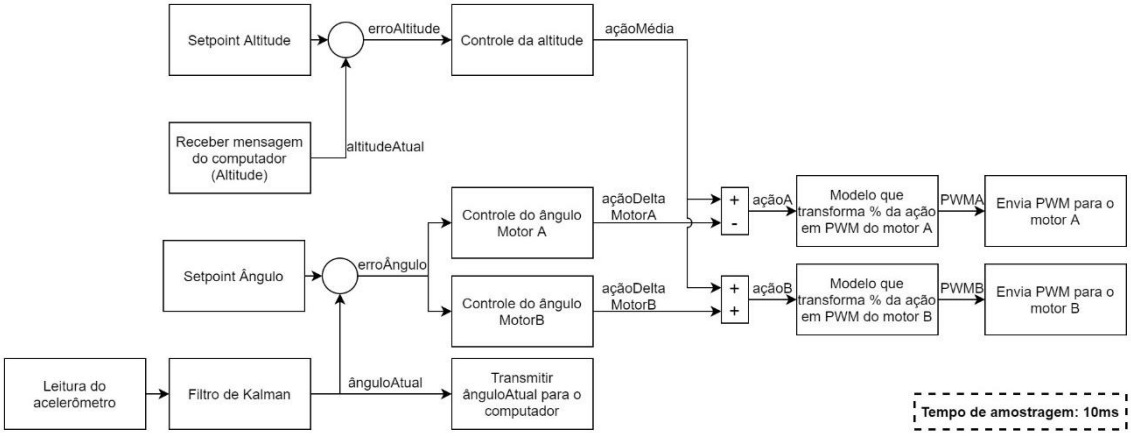
Para executar as rotinas de controle, a leitura dos sensores e o acionamento dos motores, foram utilizados dois arduinos. Um computador também foi utilizado com o objetivo de exibir os resultados e servir de interface de comunicação entre os dois arduinos. O Arduino - CC é responsável apenas por ler o sinal da célula de carga, filtrar o sinal, estimar a altitude do drone e enviar a altitude para o computador. O Arduino - DR é o microcontrolador que seria embarcado no drone. Esse Arduino é responsável por ler os sinais do acelerômetro e passar esses dados por um filtro de Kalman com o objetivo de obter um ângulo mais confiável. Além disso, ele recebe do computador uma mensagem contendo a altitude, o setpoint da altitude e o setpoint do ângulo. Outra função desempenhada é executar os algoritmos de controle do ângulo e da altitude, enviar os sinais de ação para os motores e transmitir uma mensagem para o computador com o ângulo atual do drone. A arquitetura proposta pode ser vista na Figura 18 e o fluxograma do algoritmo que ficou embarcado no Arduino do drone pode ser visto na Figura 19.

Figura 18 – Arquitetura com todos os periféricos



Fonte: Produzida pelo autor

Figura 19 – Fluxograma do algoritmo utilizado no Arduino do drone



Fonte: Produzida pelo autor

3.5 Testes de integração da plataforma

Inicialmente foi implementado o filtro de Kalman no Arduino - DR para fundir as medidas do acelerômetro e giroscópio, e estimar a inclinação do drone. Essa implementação foi baseada na biblioteca disponível por LAUSZUS (2012). Nessa biblioteca já tem preparada toda a parte de comunicação I2C entre o acelerômetro e o Arduino, e toda a parte de fusão dos sensores. Todo o resto do código fonte do Arduino – DR foi implementado em cima do exemplo dessa biblioteca.

Depois da implementação do filtro de Kalman, foi feita uma validação do ângulo para checar se o ângulo filtrado estava condizente com o valor físico. A

forma encontrada para validar foi aplicar impulsos manuais na plataforma e analisar visualmente se o ângulo respondia adequadamente e como eles se comportavam quando o drone atingia os fins de curso. Foi notado que o ângulo com o filtro de Kalman representou bem a posição do drone quando o sistema foi submetido a impulsos manuais, assim que a velocidade do ângulo trocava de sentido, era possível notar essa troca de sentido no ângulo filtrado. Quando o drone tocou no fim de curso, o ângulo estava muito próximo da medida do ângulo obtida com o transferidor.

Depois que a rotina de mensurar o ângulo estava pronta e validada, foi feita uma rotina para acionar os motores, nesse ponto se notou que quando era enviado um mesmo comando através da função *servo.write* para os motores, um motor gerava mais empuxo que o outro. Logo, foi feita uma identificação para poder controlar os motores em malha aberta, como pode ser visto na seção 3.3. Com o modelo de cada motor validado, notou-se que a respostas dos motores melhorou quando foi enviada uma mesma porcentagem de corrente para os motores.

Foi desenvolvida uma rotina de leitura na célula de carga no Arduino – CC e feita a comunicação com o computador. Decidiu-se utilizar a linguagem de programação Python para fazer a comunicação entre o computador e os Arduinos. Essa escolha foi motivada pela facilidade de implementar, diversidade de tutoriais disponíveis e a diversidade de bibliotecas que possibilitam várias soluções.

Após implementar algoritmos para detectar possíveis erros na comunicação e validar a comunicação entre os dispositivos, realizou-se a identificação da célula de carga, como pode ser vista na seção 3.2. Posteriormente, testes foram realizados para avaliar o desempenho do algoritmo de estimativa de altitude, a partir dos dados enviados do Arduino – CC para o Arduino – DR.

Com toda a comunicação pronta e as variáveis necessárias para encontrar a posição do drone sendo calculadas, uma interface gráfica foi incrementada ao script em Python. Essa interface gráfica tem como objetivo mostrar de forma visual para o usuário a posição estimada do drone e qual seria a posição setpoint. Como pode ser visto na Figura 20, a posição atual do drone

seria o quadrado vermelho e o seu setpoint seria o quadrado verde. Por convenção, foi adotado que o eixo vertical da interface simularia a altitude e o eixo horizontal simularia o ângulo e não a posição horizontal do drone.

Figura 20 – Interface gráfica mostrando a posição do drone em tempo real

Fonte: Produzida pelo autor

Com os resultados já obtidos nesse teste de integração mostram que o objetivo principal do trabalho já foi alcançado. Mas adicionalmente foi testado alguns algoritmos de controle para controlar a altitude e o ângulo do drone.

Com a interface gráfica pronta, buscou-se calibrar um controlador PID para o ângulo do protótipo, desconsiderando as implicações relativas ao controle de altitude. Foi fixada uma ação média e foi implementado o controle do ângulo baseado na Figura 19. Após muitas tentativas, chegou-se a um conjunto de Kp, Ki e Kd para cada motor que deixou o drone em torno do seu setpoint.

Em seguida, foi feita a sintonização do controle da altitude, para isso foi fixado o drone na horizontal para evitar qualquer rotação e as ações que seriam enviadas para os dois motores seriam a ação média. Após algumas tentativas, convergiu-se para as constantes que estabilizavam o drone na altitude desejada.

Com todos os controles sintonizados, foi o momento de liberar a rotação do drone e notar como o controle de altitude se comporta em conjunto com o controle do ângulo. Os resultados dos controles para o ângulo isolado, depois a altitude isolada e por fim o controle do ângulo funcionando juntamente com o controle da altitude podem ser vistos no próximo capítulo. Os códigos-fonte utilizados para esses controles podem ser vistos nos apêndices A, B e C.

4 RESULTADOS

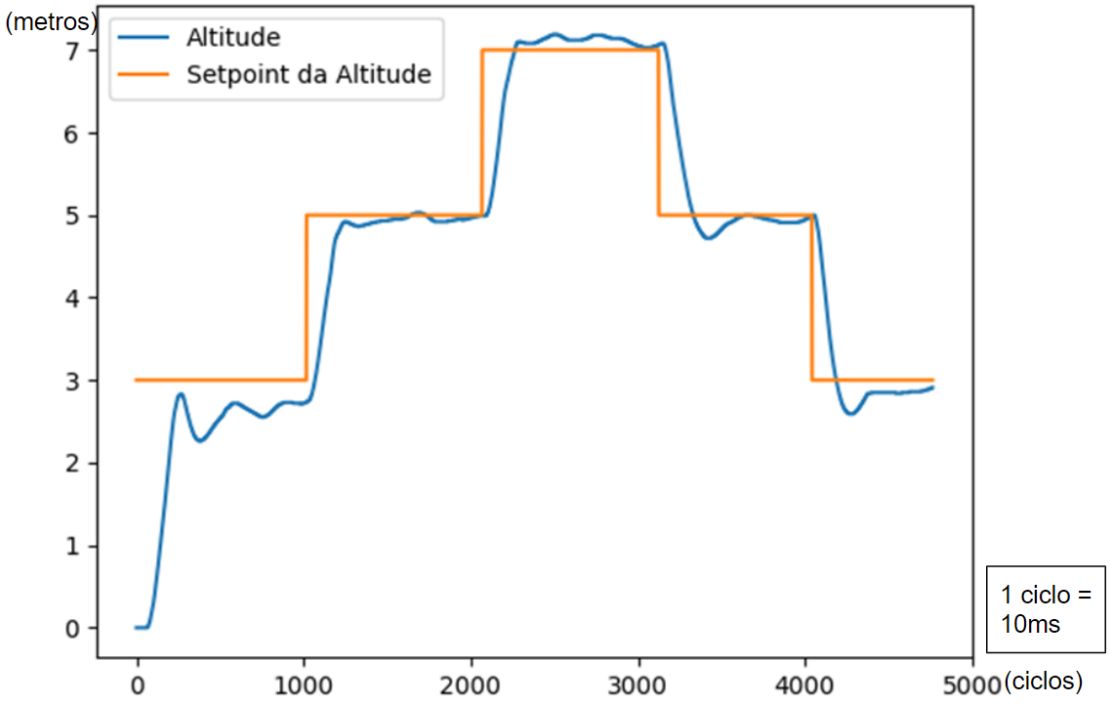
Neste capítulo, serão abordados os resultados obtidos neste trabalho. Como forma de validar o sistema, foram realizados testes para controlar o ângulo de inclinação do drone e sua altitude simulada.

4.1 Controle da altitude

Para conduzir os experimentos objetivando estabelecer apenas um controle de altitude para o drone, o mesmo foi fixado mecanicamente na horizontal, impedindo a rotação em torno do eixo. Em seguida, foram realizados alguns testes para sintonizar os ganhos do controlador PID por tentativa e erro. Após vários testes, $K_p = 0,1$, $K_i = 0,008$ e $K_d = 0,1$ foram os melhores valores encontrados para controlar a altitude. Durante a sintonização, notou-se que o drone decolava e logo em seguida começava a cair porque a ação integrativa não estava grande o suficiente para manter o drone estável na altitude desejada. Então decidiu-se iniciar a ação integrativa com o valor de 30, dessa forma o drone decolava e convergia para a altitude desejada.

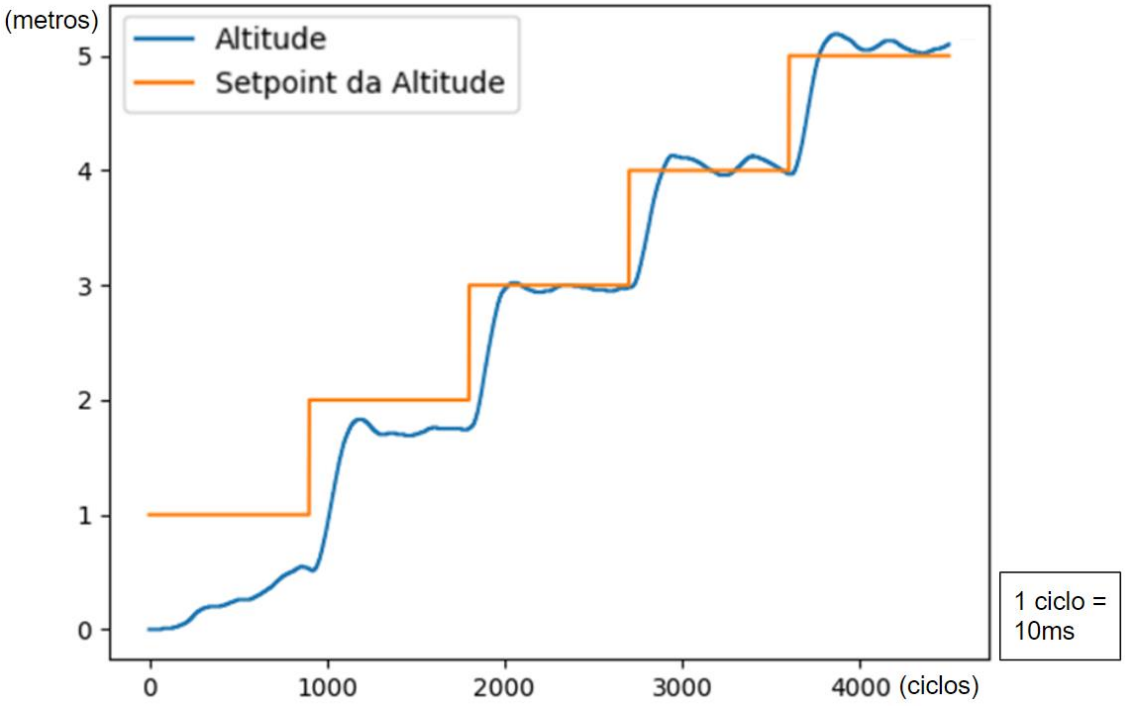
Com o controlador sintonizado, foi definida uma trajetória na qual o drone deveria seguir e essa trajetória seria enviada do computador para o drone. Alguns resultados podem ser vistos nas figuras seguintes e no Vídeo 1 do Apêndice E.

Figura 21 – Drone executando a trajetória 1 definida



Fonte: Produzida pelo autor

Figura 22 – Drone executando a trajetória 2 definida



Fonte: Produzida pelo autor

Analisando a Figura 21, nota-se que o drone fica muito próximo do setpoint nas altitudes de 5 e 7 metros, mas na altitude de 3 metros existe um erro no regime permanente. Na Figura 22, a partir do setpoint de 3 metros, a altitude

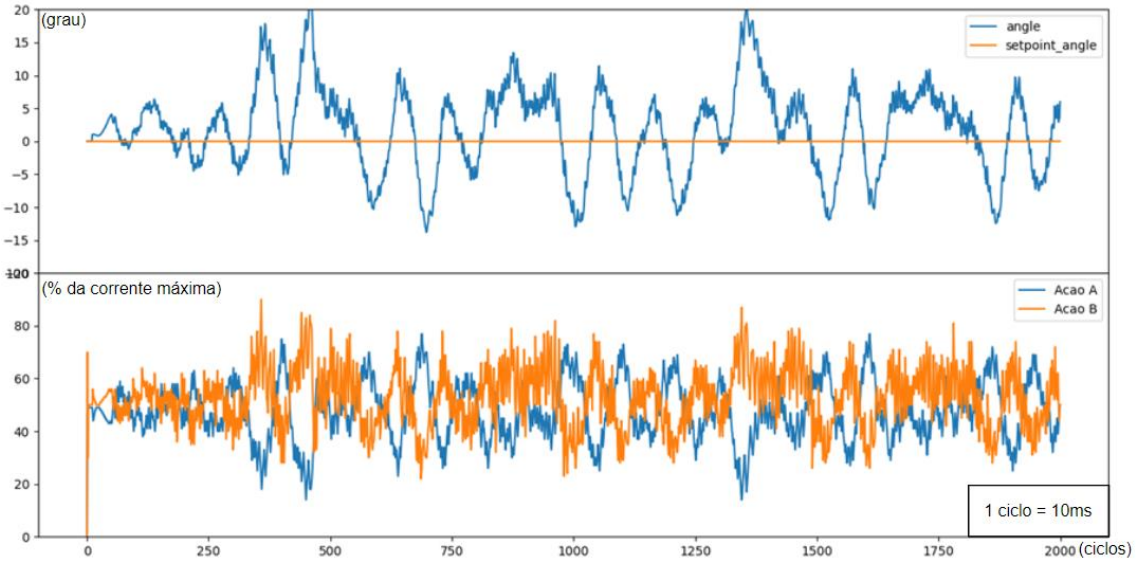
estimada está próxima do setpoint, mas para as altitudes de 1 e 2 metros existe um erro significativo no regime permanente. Isso mostra que o controle PID precisa ser sintonizado para diferentes setpoints se for preciso trabalhar em diferentes altitudes. Logo poderia fazer uma lógica fuzzy de forma híbrida com o PID para que dependendo da altitude do drone, o controlador PID tenha diferentes ganhos.

4.2 Controle do ângulo

Para controlar o ângulo foi desconsiderado o controle da altitude. Dessa forma, pode-se sintonizar o PID para o ângulo separadamente. Inicialmente, a ideia era manter um controlador PID que calculasse uma ação delta que seria somada na ação de um motor e subtraída na ação do outro motor. Mas foi percebido que mesmo com o controle de malha aberta dos motores, os motores precisavam de um tratamento diferente para as diversas situações do controle. Com um controlador simples, não foi possível manter o drone equilibrado, então outra estratégia de controle foi implementada.

A segunda estratégia utilizada foi um controle PID individual para cada motor, onde cada motor teria suas constantes do controlador. Para sintonizar o controle foi mantido um controlador com uma resposta mais lenta e as constantes do outro motor foram sendo variadas, fazendo com que esse segundo motor ficasse com um controlador que respondesse mais rápido. Depois de ter obtido constantes que o drone não ficava totalmente descontrolado, o controlador que estava com a resposta mais lenta foi sendo ajustado para ajudar no controle do sistema. As constantes encontradas foram $K_p = 0,69$, $K_i = 0,08$ e $K_d = 0,25$ para o motor A e $K_p = 0,69$, $K_i = 0,90$ e $K_d = 0,20$ para o motor B e o resultado dessa estratégia pode ser visto na Figura 23 e no Vídeo 2 do Apêndice E.

Figura 23 – Resposta do drone ao controle do ângulo



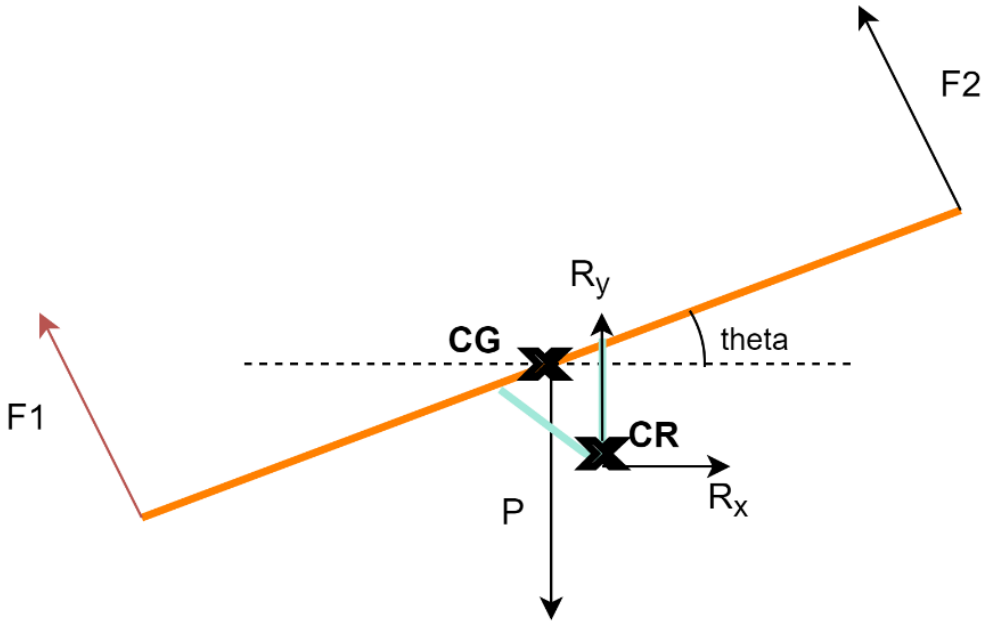
Fonte: Produzida pelo autor

Nota-se que o drone apresenta um comportamento com variações significativas em torno do setpoint, não conseguindo manter-se no ângulo de referência por muito tempo. Este foi o melhor resultado que foi obtido para controlar o ângulo do drone sem tocar nos fins de curso.

Um dos motivos que o controle do ângulo não tenha ficado bom é porque a estimativa do ângulo foi muito ruidosa. Mesmo quando o drone está travado na horizontal, o ângulo estimado pelo acelerômetro apresenta uma variação que pode ter impactado significativamente o algoritmo de controle.

Um segundo motivo é que o centro de rotação dessa plataforma é fixo e diferente do centro de gravidade, e eles estão situados de uma maneira que a força peso sempre está gerando uma força que desequilibra o drone, como pode ser visto na Figura 24. Isso faz com que o drone opere de maneira similar a um pêndulo invertido, no qual fica mais difícil de controlar. Esse problema não acontece num drone real, pois o centro de rotação (CR) não é fixo e pode variar de acordo com as forças de empuxo geradas pelos motores.

Figura 24 – Posição do Centro de Gravidade e do Centro de Rotação do Drone



Fonte: Produzida pelo autor

Um outro motivo que explique o porquê do comportamento agressivo do sistema se deve ao fato do controlador PID funcionar bem para uma determinada faixa linear de operação. Depois que o drone sai dessa faixa, o controlador não atua de forma satisfatória aos novos erros, logo faz-se necessário que os ganhos dos controladores variem para diferentes intervalos de erro. Para resolver esse problema, futuramente um controlador PID pode ser implementado de forma híbrida com a lógica fuzzy.

4.3 Controle do ângulo com a altitude

A implementação do controle simultâneo da altitude e do ângulo foi aplicada ao drone. No entanto, não foi possível obter um comportamento satisfatório do sistema. Esse resultado era esperado devido à instabilidade do controle do ângulo do drone. O resultado pode ser visto no Vídeo 3 do Apêndice E.

5 CONCLUSÃO

O avanço da tecnologia na área de VANT é notável em todo mundo, dessa forma o estudo e a pesquisa nessa área é extremamente importante dentro da universidade. Partindo dessa ideia, o presente trabalho teve por objetivo iniciar o estudo dessa tecnologia dentro da UFPE. Por se tratar de um projeto piloto, foi optado por simplificar a implementação para entender melhor os algoritmos de controle e poder aplicá-los em outros projetos.

A sintonização do controle PID foi uma tarefa árdua e de muito aprendizado. Quando esse tipo de controle foi aplicado no algoritmo da altitude, nota-se que ele controla bem para certa altitude, mas precisa ser recalibrado para diferentes altitudes. Já quando foi aplicado para controlar o ângulo, o PID conseguiu controlar dentro de um determinado intervalo, mas não conseguiu deixar o drone muito estável.

Mesmo não atingindo uma performance muito estável na execução do controle, conclui-se que o objetivo desse trabalho foi atingido totalmente, pois o objetivo era construir uma plataforma para testar algoritmos de controle de drone. Hoje o Laboratório de Automação Industrial tem uma plataforma para testar algoritmos de controle de drones, de forma segura.

Por fim, percebeu-se durante a execução desse projeto, que algumas melhorias podem ser feitas em trabalhos futuros:

- Acoplar um sensor ao eixo do drone para poder validar o ângulo estimado pelo acelerômetro e calibrar o filtro de Kalman;
- Fazer uma função de compensação dos motores de forma similar à função da seção 3.3, mas usando a velocidade de rotação dos motores no lugar da corrente consumida;
- Testar o algoritmo Fuzzy para controlar o ângulo do drone;
- Desenvolver um mecanismo para liberar a rotação do drone apenas quando o drone tenha decolado;
- Implementar algoritmo para autocalibrar o modelo dos motores ou fazer o controle dos motores em malha fechada.

REFERÊNCIAS

BEHNCK, L. **Controle de Missão de Voo de Veículo Aéreo Não-Tripulado**. Porto Alegre: [s.n.], 2014

Bohn, C. e Atherton D. P., **"An Analysis Package Comparing PID Anti-Windup Strategies**", IEEE Control Systems Magazine, pp. 34-40, 1995.

DELAIR. **Commercial Drones: discover Delair's professional UAV models**. Disponível em: < https://delair.aero/delair-commercial-drones/#dt26 > . Acesso em: 19 abr. 2021.

Dos Santos, Wellington & Barbosa, Valter & de Souza, Ricardo & Ribeiro, Reiga & Feitosa, Allan & Silva, Victor & Ribeiro, David & Covello de Freitas, Rafaela & Lima, Manoela & Soares, Natália & Valença, Rodrigo & Ogava, Rodrigo & Dias, Ítalo. (2018). **Image Reconstruction of Electrical Impedance Tomography Using Fish School Search and Differential Evolution**. 10.4018/978-1-5225-5134-8.ch012.

DJI. **Phantom 4 Advanced – DJI**. Disponível em: < https://www.dji.com/br/phantom-4-adv >. Acesso em: 19 abr. 2021.

LAUSZUS, Kristian. (2012). **GitHub - TKJElectronics/KalmanFilter: This is a Kalman filter used to calculate the angle, rate and bias from from the input of an accelerometer/magnetometer and a gyroscope.** Disponível em: < https://github.com/TKJElectronics/KalmanFilter >. Acesso em: 28 dez. 2021.

KANG, Y. S. et al. **Ground test results of rotor governor and rate SAS for small tilt rotor UAV**. ICCAS 2007 - International Conference on Control, Automation and Systems, p. 830–835, 2007.

KANG, Y. S. et al. **Ground test results of flight control system for the Smart UAV.** ICCAS 2010 - International Conference on Control, Automation and Systems, p. 2533–2536, 2010.

Khuwaja, Komal & Tarca, Ioan & Lighari, Noor-u-Zaman & Tarca, Radu. (2018). **PID Controller Tuning Optimization with Genetic Algorithms for a Quadcopter.** Recent Innovations in Mechatronics. 5. 10.17667/riim.2018.1/11.

Lambora, A., Gupta, K., & Chopra, K. (2019). **Genetic Algorithm- A Literature Review. 2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)**. doi:10.1109/comitcon.2019.8862255

MADRUGA, Sarah Pontes (2018). **Projeto de sistema de controle embarcado para controle de voo de quadricópteros**. Disponível em: < https://repositorio.ufpb.br/jspui/handle/123456789/13346 > Dissertação de Mestrado (Mestrado em Informática) - Universidade Federal da Paraíba, João Pessoa, Paraíba, Brasil, 2018.

MCROBERTS, Michael. **Beginning Arduino**. Apress. Nova Iorque: 2010.

MICHALEWICZ, Z. **Genetic Algorithms + Data Structures = Evolution Programs**. 3rd edition, Springer,1996.

MONTEIRO, João (2015). **Modelagem e Controle de um Veículo Quadrirrotor**. Trabalho de Conclusão de Curso (Bacharelado em Engenharia de Controle e Automação) – Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2015.

MULLER, I. et al. **Load cells in force sensing analysis – theory and a novel application**. IEEE Instrumentation Measurement Magazine, v. 13, n. 1, p. 15-19, February 2010. ISSN 1094-6969.

NONAMI, K. et al. **Autonomous Flying Robots**. Springer Japan, 2010. Disponível em: https://doi.org/10.1007/978-4-431-53856-1.

OGATA, K. **Engenharia de Controle Moderno**. 4 ed. São Paulo. Pearson, 2003.

SILVA, B. C. da; LIBARDI, P. O. **Sistema de Controle de um VANT**. 2016. Monografia (Bacharel em Engenharia Elétrica), UnB (Universidade de Brasília), Brasília, Brasil. Disponível em: http://bdm.unb.br/bitstream/10483/15956/1/2016_BrunoCastrodaSilva_PriscilaOrtolanLibardi_tcc.pdf.

Stack Overflow Developer Survey 2021. Disponível em <https://insights.stackoverflow.com/survey/2021>. Acesso em: 18/09/2021

VIDA DE SÍLICIO. **Sensor de Peso / Célula de Carga 5kg**. Disponível em: < https://www.vidadesilicio.com.br/sensor-de-peso-celula-de-carga-5kg >. Acesso em: 20 abr. 2021.

V.LARA, A. et al. **Hardware-in-the-loop simulation environment for testing of tilt-rotor UAV's control strategies**. In: Proceedings XXII Congresso Brasileiro de Automática. SBA Sociedade Brasileira de Automática, 2018. Disponível em: https://doi.org/10.20906/cps/cba2018-0758.

YOO, C.; KANG, Y.; PARK, B. **Hardware-in-the-loop simulation test for actuator control system of smart uav**. In: ICCAS 2010 - International Conference on Control, Automation and Systems. [S.l.: s.n.], 2010. p. 1729–1732.

APÊNDICE A – Código-fonte utilizado no Arduino – CC (ArduinoCelulaCarga.ino)

```
// --- Constantes Timers Interrupcao ---
const uint16_t T1_init = 0;
const uint16_t T1_comp = 625; //625 é a constante que equivale a uma
interrupção de 10ms

#define SOP 255
#define SecondsT 0.01
#define valorInicial 700 //Valor inicial para iniciar o filtro de
butterworth
#define python 1

#define Kcc 0.000590 //Constante encontrada de forma experimental
int valorEquilibrio = 620; //Valor inicial, mas na função setup esse
valor é recalculado
#define massa 0.200
#define gravidade 9.8

uint32_t Tant;

float x[4]={valorInicial,valorInicial,valorInicial,valorInicial};
float y[4]={valorInicial,valorInicial,valorInicial,valorInicial};

int celCargaFiltrada = 700;

float a_y = 0;
int a_y_int = 0; //a_y_int= a_y * 100
float veloc_y = 0;
float pos_y = 0;
float delta_pos_y =0;
int pos_y_cm = 0;
bool decolou = 0;


int led1 = 13; // Porta onde o led será inserido
int leitura;
int valorRealSensor = 0;

void setup(){
  Serial.begin(115200);
  pinMode(led1, OUTPUT); // Porta onde o led será inserido,
configurado como saida
  int i = 0;
  valorEquilibrio = 0;
  Tant = millis();
  while(!RX()) //Travando o script para rodar o loop apenas quando o
primeiro bit for recebido
  { //Setup para descobrir o valor da tara da balança
    if(millis()-Tant> 250){
      Tant = millis();
      valorEquilibrio += analogRead(0);
      i++;
    }
  }
  valorEquilibrio = valorEquilibrio/i;
  x[0]=valorEquilibrio;
```

```
  x[1]=valorEquilibrio;
  x[2]=valorEquilibrio;
  x[3]=valorEquilibrio;

  //Configurando Interrupcao
  //Modo de Comparação
  TCCR1A = 0;
  //Prescaler 1:256
  TCCR1B |=  (1 << CS12);
  TCCR1B &= ~(1 << CS11);
  TCCR1B &= ~(1 << CS10);
  //Inicializa Registradores
  TCNT1 = T1_init;
  OCR1A = T1_comp;
  //Habilita Interrupção do Timer1
  TIMSK1 = (1 << OCIE1A);
}
void loop(){
  //Não faz nada / Toda lógica está dentro da função de interrupçãp
}

bool RX(){
  if(Serial.available()>0){
    leitura = Serial.read(); // Variavel que receberá os valores
enviados pelo programa em python
    return 1; // Se o arduino receber algum byte via serial
  }
  return 0; // Se o arduino NÃO receber nenhum byte via serial
}

void TX(){
  byte vetor[5];
  //Preenchendo o buffer da mensagem para enviar pela serial
  vetor[0]= (byte)SOP;
  vetor[1]= (byte)((((uint16_t)pos_y_cm) >> 8) & 0x00FF);
  vetor[2]= (byte)(((uint16_t)pos_y_cm) & 0x00FF);
  vetor[3]= (byte)((((uint16_t)a_y_int) >> 8) & 0x00FF);
  vetor[4]= (byte)(((uint16_t)a_y_int) & 0x00FF);
  #if python
  Serial.write((uint8_t*)vetor,5);
  #else
  Serial.print(pos_y);Serial.print("\t");
  Serial.print(veloc_y);Serial.print("\t");
  Serial.print(a_y);Serial.print("\t");
  Serial.println(celCargaFiltrada);
  #endif
}

void gettingPosition(){
  //Convertendo o valor analógico da célula de carga em aceleração
  a_y = ((Kcc*(valorEquilibrio - celCargaFiltrada)) - massa) *
gravidade /massa;
  if(a_y > 0){ //Detectando se o drone já teria decolado
    decolou = 1;
  }
  if(decolou == 0){ //Posição só começa a variar quando a aceleração
resultante é positiva
    a_y = 0;
  } else {
```

```
    delta_pos_y = (veloc_y*SecondsT) + (a_y*SecondsT*SecondsT/2);
//Equação horária da posição
    pos_y = pos_y + delta_pos_y;
    veloc_y = veloc_y + a_y*SecondsT; //Equação horária da velocidade
  }
  a_y_int = ((int)(a_y*100));
  pos_y_cm = ((int)(pos_y*100));
}

//
================================================================================
// --- Interrupção ---
ISR(TIMER1_COMPA_vect)
{
  TCNT1 = T1_init;      //reinicializa TIMER1
  valorRealSensor = analogRead(0); //Lendo o valor atual da célula de
carga

  //Filtrando a célula de carga usando um Butterworth com as seguintes
características
  //Butterworth fc = 5Hz  fs= 100Hz ordem 3
  // a = 1.0000   -2.3741    1.9294   -0.5321
  // b = 0.0029    0.0087    0.0087    0.0029
  x[0]=x[1];
  x[1]=x[2];
  x[2]=x[3];
  x[3]=valorRealSensor;
  y[0]=y[1];
  y[1]=y[2];
  y[2]=y[3];
  y[3]= (0.0029*x[3])+(0.0087*x[2])+(0.0087*x[1])+(0.0029*x[0]);
  y[3]= y[3]+(2.3741*y[2])-(1.9294*y[1])+(0.5321*y[0]);

  celCargaFiltrada = y[3]; //butterworth
  gettingPosition(); //Atualizando a variável com o valor da posição
atual

  TX();
  RX();
  if(leitura == 49){
    digitalWrite(led1, HIGH); // Liga a porta 13 se o valor recebido
for 1
  }
  else if(leitura == 50){
    digitalWrite(led1, LOW); // Desliga a porta 13 se o valor recebido
for 2
  }
} //end ISR
```

APÊNDICE B.1 – Código-fonte utilizado no Arduino – DR (ArduinoDrone.ino)

```
// --- Constantes Timers Interrupcao ---
const uint16_t T2_init = 0;
const uint16_t T2_comp = 156; //156 é a constante que equivale a uma
interrupção de 10ms

#define SOP 255
#define Ts 10000
#define Ts_sec 0.01
#define python 1
#define controleAngulo 1 //Flag para ativar o controle do ângulo
#define controleAltitude 0 //Flag para ativar o controle da altitude

bool flagInterrupcao = 0;
uint32_t Tant;
int acaoA = 0;
int acaoB = 0;
float correnteA = 0;
float correnteB = 0;
int pwmA = 17;
int pwmB = 17;
int acao_altitude;
float acao_angle, acaoInst;
float setpoint_angle= 0;
float erro_angle[3]={0,0,0};
float angle[3] = {0,0,0};
float aceleracao, delta_Kp, delta_Ki, delta_Kd;
float Kp_angle, Ki_angle, Kd_angle, Ti_angle, Td_angle;
float acaoP_angle, acaoI_angle, acaoD_angle;
int acao_max_angle = 30;
int acaoI_max_angle = 15;
int acaoD_max_angle = 15;
float q0, q1, q2;

float acao_angle_A, acao_angle_B;
float Kp_angle_A, Ki_angle_A, Kd_angle_A;
float acaoP_angle_A, acaoI_angle_A, acaoD_angle_A;
float acaoD_angle_A_prev;
int acao_max_angle_A = 30;
int acaoI_max_angle_A = 15;
int acaoD_max_angle_A = 15;
int delta_acaoD_max_angle_A = 10;
float Kp_angle_B, Ki_angle_B, Kd_angle_B;
float acaoP_angle_B, acaoI_angle_B, acaoD_angle_B;
int acao_max_angle_B = 30;
int acaoI_max_angle_B = 15;
int acaoD_max_angle_B = 15;

int setpoint_altitude;
int erro_altitude[3]={0,0,0};
float Kp_altitude, Ki_altitude, Kd_altitude;
float acaoP_altitude, acaoI_altitude, acaoD_altitude;
int acao_max_altitude = 80;
int acaoI_max_altitude = 40;

int pos_y_cm = 0;
```

```
#include <Wire.h>
#include <Kalman.h> // Source:
https://github.com/TKJElectronics/KalmanFilter
#include <Servo.h>

//Variáveis para controlar os motores
Servo myservoA;
Servo myservoB;
//#define RESTRICT_PITCH // Comment out to restrict roll to ±90deg
instead - please read:
http://www.freescale.com/files/sensors/doc/app_note/AN3461.pdf

Kalman kalmanX; // Create the Kalman instances
Kalman kalmanY;

/* IMU Data */
double accX, accY, accZ;
double gyroX, gyroY, gyroZ;
int16_t tempRaw;

double gyroXangle, gyroYangle; // Angle calculate using the gyro only
//double compAngleX, compAngleY; // Calculated angle using a
complementary filter
double kalAngleX, kalAngleY; // Calculated angle using a Kalman filter

uint32_t timer;
uint8_t i2cData[14]; // Buffer for I2C data

// TODO: Make calibration routine

void setup() {
  pinMode(6,OUTPUT);
  pinMode(4,OUTPUT);
  pinMode(7,OUTPUT);
  pinMode(13,OUTPUT);
  digitalWrite(13,LOW);
  Serial.begin(115200);
  myservoA.attach(9);
  myservoB.attach(10);
  myservoA.write(17);
  myservoB.write(17);
  initializingPID();
  #if python
  while(!RX()); //Travando o script para rodar o loop apenas quando o
primeiro bit for recebido
  #else
  while(millis()<5000);
  #endif
   //Configurando Interrupcao
   //Modo de Comparação
   TCCR2A = 0;
   //Prescaler 1:1024
   TCCR2B |= (1 << CS22);
   TCCR2B |= (1 << CS21);
   TCCR2B |= (1 << CS20);
   //Inicializa Registradores
   TCNT2 = T2_init;
   OCR2A = T2_comp;
   //Habilita Interrupção do Timer2
   TIMSK2 = (1 << OCIE2A);
```

```
  digitalWrite(13,HIGH);

  Wire.begin();
#if ARDUINO >= 157
  Wire.setClock(400000UL); // Set I2C frequency to 400kHz
#else
  TWBR = ((F_CPU / 400000UL) - 16) / 2; // Set I2C frequency to 400kHz
#endif

  i2cData[0] = 7; // Set the sample rate to 1000Hz - 8kHz/(7+1) =
1000Hz
  i2cData[1] = 0x00; // Disable FSYNC and set 260 Hz Acc filtering,
256 Hz Gyro filtering, 8 KHz sampling
  i2cData[2] = 0x00; // Set Gyro Full Scale Range to ±250deg/s
  i2cData[3] = 0x00; // Set Accelerometer Full Scale Range to ±2g
  while (i2cWrite(0x19, i2cData, 4, false)); // Write to all four
registers at once
  while (i2cWrite(0x6B, 0x01, true)); // PLL with X axis gyroscope
reference and disable sleep mode

  while (i2cRead(0x75, i2cData, 1));
  if (i2cData[0] != 0x68) { // Read "WHO_AM_I" register
    //Serial.print(F("Error reading sensor"));
    while (1);
  }

  delay(100); // Wait for sensor to stabilize

  /* Set kalman and gyro starting angle */
  while (i2cRead(0x3B, i2cData, 6));
  accX = (int16_t)((i2cData[0] << 8) | i2cData[1]);
  accY = (int16_t)((i2cData[2] << 8) | i2cData[3]);
  accZ = (int16_t)((i2cData[4] << 8) | i2cData[5]);

  // Source:
http://www.freescale.com/files/sensors/doc/app_note/AN3461.pdf eq. 25
and eq. 26
  // atan2 outputs the value of -π to π (radians) - see
http://en.wikipedia.org/wiki/Atan2
  // It is then converted from radians to degrees
#ifdef RESTRICT_PITCH // Eq. 25 and 26
  double roll  = atan2(accY, accZ) * RAD_TO_DEG;
  double pitch = atan(-accX / sqrt(accY * accY + accZ * accZ)) *
RAD_TO_DEG;
#else // Eq. 28 and 29
  double roll  = atan(accY / sqrt(accX * accX + accZ * accZ)) *
RAD_TO_DEG;
  double pitch = atan2(-accX, accZ) * RAD_TO_DEG;
#endif

  kalmanX.setAngle(roll); // Set starting angle
  kalmanY.setAngle(pitch);
  gyroXangle = roll;
  gyroYangle = pitch;
  //compAngleX = roll;
  //compAngleY = pitch;

  timer = micros();
  Tant = micros();
}
```

```
void loop() {
  /* Update all the values */
  if(flagInterrupcao){ //Só atualiza o filtro de kalman depois de uma
interrupção
    flagInterrupcao = 0;
    digitalWrite(4,HIGH);
    while (i2cRead(0x3B, i2cData, 14));
    accX = (int16_t)((i2cData[0] << 8) | i2cData[1]);
    accY = (int16_t)((i2cData[2] << 8) | i2cData[3]);
    accZ = (int16_t)((i2cData[4] << 8) | i2cData[5]);
    tempRaw = (int16_t)((i2cData[6] << 8) | i2cData[7]);
    gyroX = (int16_t)((i2cData[8] << 8) | i2cData[9]);
    gyroY = (int16_t)((i2cData[10] << 8) | i2cData[11]);
    gyroZ = (int16_t)((i2cData[12] << 8) | i2cData[13]);;

    double dt = (double)(micros() - timer) / 1000000; // Calculate
delta time
    timer = micros();
    // Source:
http://www.freescale.com/files/sensors/doc/app_note/AN3461.pdf eq. 25
and eq. 26
    // atan2 outputs the value of -π to π (radians) - see
http://en.wikipedia.org/wiki/Atan2
    // It is then converted from radians to degrees
  #ifdef RESTRICT_PITCH // Eq. 25 and 26
    double roll  = atan2(accY, accZ) * RAD_TO_DEG;
    double pitch = atan(-accX / sqrt(accY * accY + accZ * accZ)) *
RAD_TO_DEG;
  #else // Eq. 28 and 29
    double roll  = atan(accY / sqrt(accX * accX + accZ * accZ)) *
RAD_TO_DEG;
    double pitch = atan2(-accX, accZ) * RAD_TO_DEG;
  #endif

    double gyroXrate = gyroX / 131.0; // Convert to deg/s
    double gyroYrate = gyroY / 131.0; // Convert to deg/s

  #ifdef RESTRICT_PITCH
    // This fixes the transition problem when the accelerometer angle
jumps between -180 and 180 degrees
    if ((roll < -90 && kalAngleX > 90) || (roll > 90 && kalAngleX < -
90)) {
      kalmanX.setAngle(roll);
      //compAngleX = roll;
      kalAngleX = roll;
      gyroXangle = roll;
    } else
      kalAngleX = kalmanX.getAngle(roll, gyroXrate, dt); // Calculate
the angle using a Kalman filter

    if (abs(kalAngleX) > 90)
      gyroYrate = -gyroYrate; // Invert rate, so it fits the restriced
accelerometer reading
    kalAngleY = kalmanY.getAngle(pitch, gyroYrate, dt);
  #else
    // This fixes the transition problem when the accelerometer angle
jumps between -180 and 180 degrees
    if ((pitch < -90 && kalAngleY > 90) || (pitch > 90 && kalAngleY <
-90)) {
```

```
      kalmanY.setAngle(pitch);
      //compAngleY = pitch;
      kalAngleY = pitch;
      gyroYangle = pitch;
    } else
      kalAngleY = kalmanY.getAngle(pitch, gyroYrate, dt); // Calculate
the angle using a Kalman filter

    if (abs(kalAngleY) > 90)
      gyroXrate = -gyroXrate; // Invert rate, so it fits the restriced
accelerometer reading
    kalAngleX = kalmanX.getAngle(roll, gyroXrate, dt); // Calculate
the angle using a Kalman filter
  #endif

    gyroXangle += gyroXrate * dt; // Calculate gyro angle without any
filter
    gyroYangle += gyroYrate * dt;

    // Reset the gyro angle when it has drifted too much
    if (gyroXangle < -180 || gyroXangle > 180)
      gyroXangle = kalAngleX;
    if (gyroYangle < -180 || gyroYangle > 180)
      gyroYangle = kalAngleY;
    //delay(2);
    digitalWrite(4,LOW);
  }
  setMotor(); //Função escreve o último PWM calculado nos motores
}

void controlAltitude(){
  //input = altitude
  //output = porcentagem média da corrente para os motores
  erro_altitude[2] = erro_altitude[1];
  erro_altitude[1] = erro_altitude[0];
  erro_altitude[0] = setpoint_altitude - pos_y_cm; //Calculando o erro

  acaoP_altitude = Kp_altitude * erro_altitude[0];
  acaoI_altitude += Ki_altitude * erro_altitude[0];
  acaoD_altitude = Kd_altitude * (erro_altitude[0] -
erro_altitude[1]);

  if (acaoI_altitude > acaoI_max_altitude){ //Saturando a ação
integrativa
    acaoI_altitude = acaoI_max_altitude;
  } else if (acaoI_altitude < ((-1)*acaoI_max_altitude)){
    acaoI_altitude = ((-1)*acaoI_max_altitude);
  }

  acao_altitude = acaoP_altitude + acaoI_altitude + acaoD_altitude;

  if (acao_altitude > acao_max_altitude){ //Saturando a ação da
altitude
    acao_altitude = acao_max_altitude;
  } else if (acao_altitude < 0){
    acao_altitude = 0;
  }
}

void controlAngle(){
```

```
  //input = ângulo
  //output = porcentagem da corrente para os motores
  angle[2] = angle[1];
  angle[1] = angle[0];
  angle[0] = 0.700*angle[1] + 0.300*getKalmanAngle(); //Filtro passa
baixa
  erro_angle[2] = erro_angle[1];
  erro_angle[1] = erro_angle[0];
  erro_angle[0] = setpoint_angle - angle[0]; //Calculando o erro

  acaoP_angle_A =  Kp_angle_A * erro_angle[0];
  acaoI_angle_A += Ki_angle_A *erro_angle[0];
  acaoD_angle_A_prev = acaoD_angle_A;
  acaoD_angle_A =  Kd_angle_A *(erro_angle[0]-erro_angle[1]);
  if( (acaoD_angle_A - acaoD_angle_A_prev) < ((-
1)*delta_acaoD_max_angle_A)){
    acaoD_angle_A = acaoD_angle_A_prev - delta_acaoD_max_angle_A;
  } else if ( (acaoD_angle_A - acaoD_angle_A_prev) >
delta_acaoD_max_angle_A){
    acaoD_angle_A = acaoD_angle_A_prev + delta_acaoD_max_angle_A;
  }
  acaoP_angle_B =  Kp_angle_B * erro_angle[0];
  acaoI_angle_B += Ki_angle_B *erro_angle[0];
  acaoD_angle_B =  Kd_angle_B *(erro_angle[0]-erro_angle[1]);
  if (acaoI_angle_A > acaoI_max_angle_A){
    acaoI_angle_A = acaoI_max_angle_A;
  } else if (acaoI_angle_A < ((-1)*acaoI_max_angle_A)){
    acaoI_angle_A = ((-1)*acaoI_max_angle_A);
  }
  if (acaoD_angle_A > acaoD_max_angle_A){
    acaoD_angle_A = acaoD_max_angle_A;
  } else if (acaoD_angle_A < ((-1)*acaoD_max_angle_A)){
    acaoD_angle_A = ((-1)*acaoD_max_angle_A);
  }
  if (acaoI_angle_B > acaoI_max_angle_B){
    acaoI_angle_B = acaoI_max_angle_B;
  } else if (acaoI_angle_B < ((-1)*acaoI_max_angle_B)){
    acaoI_angle_B = ((-1)*acaoI_max_angle_B);
  }
  if (acaoD_angle_B > acaoD_max_angle_B){
    acaoD_angle_B = acaoD_max_angle_B;
  } else if (acaoD_angle_B < ((-1)*acaoD_max_angle_B)){
    acaoD_angle_B = ((-1)*acaoD_max_angle_B);
  }
  if (erro_angle[0]*erro_angle[1]<0){ //Anti-windup / Troca de sinal
do erro
    acaoI_angle_A = 0;
    acaoI_angle_B = 0;
  }
  acao_angle_A = acaoP_angle_A + acaoI_angle_A + acaoD_angle_A;
  acao_angle_B = acaoP_angle_B + acaoI_angle_B + acaoD_angle_B;
  if (acao_angle_A > acao_max_angle_A){
    acao_angle_A = acao_max_angle_A;
  } else if (acao_angle_A < ((-1)*acao_max_angle_A)){
    acao_angle_A = ((-1)*acao_max_angle_A);
  }
  if (acao_angle_B > acao_max_angle_B){
    acao_angle_B = acao_max_angle_B;
  } else if (acao_angle_B < ((-1)*acao_max_angle_B)){
    acao_angle_B = ((-1)*acao_max_angle_B);
```

```
  }
}

void initializingPID(){
  //Constantes PID para o controle do ângulo
  Kp_angle_A = 0.69;
  Ki_angle_A = 0.08;
  Kd_angle_A = 0.25;
  Kp_angle_B = 0.69;
  Ki_angle_B = 0.9;
  Kd_angle_B = 0.20;
  acaoI_angle_A = 0;
  acao_max_angle_A = 40;
  acaoI_max_angle_A = 15;
  acaoD_max_angle_A = 25;
  delta_acaoD_max_angle_A = 3;
  acaoI_angle_B = 0;
  acao_max_angle_B = 40;
  acaoI_max_angle_B = 15;
  acaoD_max_angle_B = 40;

  Ki_angle_A = Ki_angle_A*Ts_sec;
  Kd_angle_A = Kd_angle_A/Ts_sec;
  Ki_angle_B = Ki_angle_B*Ts_sec;
  Kd_angle_B = Kd_angle_B/Ts_sec;

  //Constantes PID para o controle da altitude
  Kp_altitude = 0.1;
  Ki_altitude = 0.008;
  Kd_altitude = 0.10;
  acao_max_altitude = 80;
  acaoI_max_altitude = 70;
  acaoI_altitude = 30;

  Ki_altitude = Ki_altitude*Ts_sec;
  Kd_altitude = Kd_altitude/Ts_sec;
}

inline double getKalmanAngle()  {return kalAngleX;} //Função para
retornar o ângulo filtrado

void setMotor(){
  // input = porcentagem da corrente
  // output = pwm para os motores
  //Convertendo a porcentagem da ação para cada motor em corrente
  correnteA = (acaoA+4) *4.0/100.0;
  correnteB = (acaoB-2) *4.0/100.0;

  //Modelo para calcular o PWM necessário para cada motor dependendo
da corrente desejada
  pwmA = (0.2996*pow(correnteA,3)) -(5.8052*pow(correnteA,2))
+(39.6417*correnteA) + 14.3045;
  pwmB = (0.2977*pow(correnteB,3)) -(4.5052*pow(correnteB,2))
+(29.1644*correnteB) + 16.0212;

  //Saturando os valores de PWM
  if(pwmA < 17){
    pwmA = 17;
  } else if (pwmA >100){
    pwmA = 100;
```

```
  }
  if(pwmB < 17){
    pwmB = 17;
  } else if (pwmB > 100){
    pwmB = 100;
  }
  myservoA.write(pwmA);
  myservoB.write(pwmB);
}

bool RX(){
  if(Serial.available()>0){
    #if python
    if(Serial.read()==SOP){ //Primeiro byte é o SOP esperado
      pos_y_cm = Serial.read();
      pos_y_cm = (pos_y_cm << 8) | Serial.read(); //pos_y em cm
      Serial.read();  //Apenas para tirar da Serial a a_y
      Serial.read();  //Apenas para tirar da Serial a a_y
      setpoint_altitude = Serial.read();
      setpoint_altitude = setpoint_altitude*100;
      setpoint_angle = Serial.read();
    }
    #else
    setpoint_angle = 0;
    #endif
    return 1;
  }
  return 0;
}

void TX(){
  byte vetor[7];
  vetor[0]= (byte)SOP;
  vetor[1]= (byte)((((int)(getKalmanAngle()*100)) >> 8) & 0x00FF);
  vetor[2]= (byte)(((int)(getKalmanAngle()*100)) & 0x00FF);
  #if controleAngulo
  vetor[3]= (byte)(((int)(acaoA)) & 0x00FF);
  vetor[4]= (byte)(((int)(acaoB)) & 0x00FF);
  vetor[5]= (byte)(((int)(acaoD_angle)) & 0x00FF);
  vetor[6]= (byte)(((int)(acao_angle)) & 0x00FF);
  #endif
  #if controleAltitude
  vetor[3]= (byte)(((int)(acaoP_altitude)) & 0x00FF);
  vetor[4]= (byte)(((int)(acaoI_altitude)) & 0x00FF);
  vetor[5]= (byte)(((int)(acaoD_altitude)) & 0x00FF);
  vetor[6]= (byte)(((int)(acao_altitude)) & 0x00FF);
  #endif

  #if python
  Serial.write((uint8_t*)vetor,7);
  #else
  Serial.print(erro_angle[0]); Serial.print('\t');
  Serial.print(angle[0]); Serial.print('\t');
  Serial.print(acaoA); Serial.print('\t');
  Serial.print(acaoP_angle_A); Serial.print('\t');
  Serial.print(acaoI_angle_A); Serial.print('\t');
  Serial.print(acaoD_angle_A); Serial.print('\n');
  #endif
}
```

```
// ============================================================
// --- Interrupção ---
ISR(TIMER2_COMPA_vect)
{
  digitalWrite(7,HIGH);
  flagInterrupcao = 1;
  TCNT2 = T2_init;      //reinicializa TIMER1

  RX();
  TX();

  #if controleAltitude
  controlAltitude();
  #else
  acao_altitude = 50;
  #endif

  #if controleAngulo
  controlAngle();
  #else
  acao_angle = 0;
  #endif

  acaoA = acao_altitude + acao_angle_A;
  acaoB = acao_altitude - acao_angle_B;

  if(acaoA > 100){
    acaoA = 100;
  } else if(acaoA<0){
    acaoA = 0;
  }

  if(acaoB > 100){
    acaoB = 100;
  } else if(acaoB<0) {
    acaoB = 0;
  }
  digitalWrite(7,LOW);
} //end ISR
```

APÊNDICE B.2 – Código-fonte utilizado no Arduino – DR (I2C.ino)

```
/* Copyright (C) 2012 Kristian Lauszus, TKJ Electronics. All rights
reserved.

 This software may be distributed and modified under the terms of the
GNU
 General Public License version 2 (GPL2) as published by the Free
Software
 Foundation and appearing in the file GPL2.TXT included in the
packaging of
 this file. Please note that GPL2 Section 2[b] requires that all works
based
 on this software must also be made publicly available under the terms
of
 the GPL2 ("Copyleft").

 Contact information
 -------------------

 Kristian Lauszus, TKJ Electronics
 Web      :  http://www.tkjelectronics.com
 e-mail   :  kristianl@tkjelectronics.com
 */

const uint8_t IMUAddress = 0x68; // AD0 is logic low on the PCB
const uint16_t I2C_TIMEOUT = 1000; // Used to check for errors in I2C
communication

uint8_t i2cWrite(uint8_t registerAddress, uint8_t data, bool sendStop)
{
  return i2cWrite(registerAddress, &data, 1, sendStop); // Returns 0
on success
}

uint8_t i2cWrite(uint8_t registerAddress, uint8_t *data, uint8_t
length, bool sendStop) {
  Wire.beginTransmission(IMUAddress);
  Wire.write(registerAddress);
  Wire.write(data, length);
  uint8_t rcode = Wire.endTransmission(sendStop); // Returns 0 on
success
  if (rcode) {
    Serial.print(F("i2cWrite failed: "));
    Serial.println(rcode);
  }
  return rcode; // See:
http://arduino.cc/en/Reference/WireEndTransmission
}

uint8_t i2cRead(uint8_t registerAddress, uint8_t *data, uint8_t
nbytes) {
  uint32_t timeOutTimer;
  Wire.beginTransmission(IMUAddress);
  Wire.write(registerAddress);
  uint8_t rcode = Wire.endTransmission(false); // Don't release the
bus
  if (rcode) {
    Serial.print(F("i2cRead failed: "));
    Serial.println(rcode);
```

```
    return rcode; // See:
http://arduino.cc/en/Reference/WireEndTransmission
  }
  Wire.requestFrom(IMUAddress, nbytes, (uint8_t)true); // Send a
repeated start and then release the bus after reading
  for (uint8_t i = 0; i < nbytes; i++) {
    if (Wire.available())
      data[i] = Wire.read();
    else {
      timeOutTimer = micros();
      while (((micros() - timeOutTimer) < I2C_TIMEOUT) &&
!Wire.available());
      if (Wire.available())
        data[i] = Wire.read();
      else {
        Serial.println(F("i2cRead timeout"));
        return 5; // This error value is not already taken by
endTransmission
      }
    }
  }
  return 0; // Success
}
```

APÊNDICE C – Código-fonte utilizado no Computador (SimuladorGamer.py)

```
import pygame
# To install the pygame, we need: pip install pygame
import serial
# To install the serial, we need: pip install serial
from random import randint
import time
from datetime import datetime
import matplotlib.pyplot as plt

#Constantes paras Cores
white = (255,255,255)
black = (0,0,0)
red = (255,0,0)
green = (0,255,0)
blue = (0,0,255)

pygame.init()

#Constantes Tamanho de tela
largura = 300
altura = 300
tamanho = 20

imagemDrone = pygame.image.load('imagens/drone.png')
imagemDrone = pygame.transform.scale(imagemDrone,(tamanho,tamanho))

#Definindo um relógio para fazer o fps
relogio = pygame.time.Clock()
ts = 0.02
fs = 1/ts

fundo = pygame.display.set_mode(size=(largura,altura))
pygame.display.set_caption("Drone")
font = pygame.font.SysFont(None,15)

def texto(msg, cor,x,y):
    texto1 = font.render(msg,True,cor)
    fundo.blit(texto1,[x,y])

def draw_drone(pos_x,pos_y): #Desenha a posição atual do drone em
vermelho
    pygame.draw.rect(fundo, red, [pos_x,pos_y,tamanho,tamanho])
    fundo.blit(imagemDrone,[pos_x,pos_y])

def draw_setpoint(pos_x,pos_y): #Desenha a posição do setpoint em
verde
    pygame.draw.rect(fundo, green, [pos_x,pos_y,tamanho,tamanho])
    fundo.blit(imagemDrone,[pos_x,pos_y])

def m2px(metro): #Transformando uma medida de metro para pixel
    #Altura: 300px -> 9
    #Altura: 10m seja em altura-30
    #270px é 9m, logo 30px é 1m
    px = (altura - 30) - metro*30
    return px
```

```
def ang2px(ang):  #Transformando uma medida de ângulo para pixel
    px = (largura/2) + (ang*2)
    return px

def RX(celCarga,drone,nBytesFromCelCarga,nBytesFromDrone):
    dataByteCelCarga = celCarga.read(nBytesFromCelCarga)
    celCarga.reset_input_buffer() #limpando o buffer depois de toda
leitura para evitar formação de pilha
    dataByteDrone = drone.read(nBytesFromDrone)
    drone.reset_input_buffer() #limpando o buffer depois de toda
leitura para evitar formação de pilha
    dataIntCelCarga = int.from_bytes(dataByteCelCarga, byteorder=
'big')
    dataIntDrone = int.from_bytes(dataByteDrone, byteorder= 'big')
    #If byteorder is "big", the most significant byte is at the
beginning of the byte array.
    #If byteorder is "little", the most significant byte is at the end
of the byte array.
    if dataByteCelCarga != b'' and dataByteDrone != b'' : #Se alguma
mensagem for vazia, significa que alguma mensagem não foi transmitida
corretamente
        MM_failing = 0
    else:
        MM_failing = 1
    #Se o SOP for diferente de 255, siginifica que a comunicação não
está sincronizada
    if gettingSignal(dataIntCelCarga, nBytesFromCelCarga, 0, 1) == 255
and gettingSignal(dataIntDrone, nBytesFromDrone, 0, 1) == 255:
        SOP_failing = 0
    else:
        SOP_failing = 1

    return
[dataByteCelCarga,dataByteDrone,dataIntCelCarga,dataIntDrone,MM_failin
g,SOP_failing]

def TX(drone,dataByteCelCarga,setpoint_altitude,setpoint_angle):
    drone.reset_output_buffer() #Limpando o buffer de output antes da
escrita para evitar a formação de pilha
    drone.write(dataByteCelCarga)
    drone.write(bytes([setpoint_altitude]))
    drone.write(bytes([setpoint_angle]))

def unsignedToSigned(value,sizeOfBytes): #Função para transformar um
sinal unsigned em signed
    range = 2**(sizeOfBytes*8)
    if(value > ((range/2)-1)):
        value = value - range
    else:
        value = value
    return value

def gettingSignal(msg, nBytesOfMsg, firstByteOfSignal,
sizeByteOfSignal): #Função para facilitar ler o buffer da mensagem
    #SOP is allocated on position 0
    mask = (256**sizeByteOfSignal)-1
    msg = msg >> (8*(nBytesOfMsg-firstByteOfSignal-sizeByteOfSignal))
    signal = msg & mask
    return signal
```

```
teste = 0 #Flag para facilitar alguns testes

def jogo(): #Função que cria a janela da interface
    sair = True
    fimdejogo = False
    angulo = []
    altitude = []
    angle = []
    tempo = []
    setpoint_altitude = 0
    setpoint_angle = 0
    data_setpoint_altitude = []
    data_setpoint_angle = []
    dataA = []
    dataB = []
    dataD = []
    dataAcao = []
    MM_counter = 0
    SOP_counter = 0
    if teste == False:
        # Inicializando as portas seriais
        celCarga = serial.Serial('COM3', 115200)
        nBytesFromCelCarga = 5
        drone = serial.Serial('COM4', 115200)
        nBytesFromDrone = 7
        celCarga.timeout = 0.010
        drone.timeout = 0.010

        time.sleep(5) #Delay para dar tempo de configurar a porta
serial
        #Resetando os buffers
        celCarga.reset_input_buffer()
        celCarga.reset_output_buffer()
        drone.reset_input_buffer()
        drone.reset_output_buffer()
        tic = time.time()
        #Enviando a primeira mensagem para os arduinos para começar a
comunicação
        celCarga.write(b"2")
        drone.write(bytes([255])) #enviando SOP
        file =
open("Resultados\\ControleAngulo\\Controle2motores\\Data.txt","w")

    #posicao inicial do drone
    [pos_x,pos_y] = [ang2px(0), m2px(0)]

    i = 0
    while(sair):
        while fimdejogo: #Se o diagnóstico de MM e/ou SOP acusar
falha, o usuário pode escolher se repete a simulação ou sai da
aplicação
            if teste == False:
                celCarga.close()
                drone.close()

            fundo.fill(white)
            texto("Fim de jogo, para continuar tecle C ou S para
sair", red,largura/10,altura/2) #Mensagem de falha
            pygame.display.update()
            for event in pygame.event.get():
```

```
                if event.type == pygame.QUIT:
                    sair = False
                    fimdejogo = False
                if event.type == pygame.KEYDOWN:
                    if event.key == pygame.K_c:
                        jogo()
                    if event.key == pygame.K_s:
                        sair = False
                        fimdejogo = False

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                sair = False
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_LEFT:
                    setpoint_angle -= 5
                if event.key == pygame.K_RIGHT:
                    setpoint_angle += 5
                if event.key == pygame.K_UP:
                    setpoint_altitude += 2
                if event.key == pygame.K_DOWN:
                    setpoint_altitude -= 2
        #Salvando os dados
        data_setpoint_angle.append(setpoint_angle)
        data_setpoint_altitude.append(setpoint_altitude)

        if teste == False:
            ####Recebendo a mensagem####

[dataByteCelCarga,dataByteDrone,dataIntCelCarga,dataIntDrone,MM_failin
g,SOP_failing] = RX(celCarga,drone,nBytesFromCelCarga,nBytesFromDrone)
            if MM_failing == False:
                if MM_counter > 0:
                    MM_counter -= 1
                if SOP_failing == False:
                    if SOP_counter > 0:
                        SOP_counter -= 1
                    #Só atualiza as variáveis se o SOP e o MM não
estivere falhando
                    altitudeHardware = gettingSignal(dataIntCelCarga,
nBytesFromCelCarga, 1, 2)
                    altitudeEngineering =
unsignedToSigned(altitudeHardware, 2)/100
                    altitudeEngineering = setpoint_altitude
                    altitude.append(altitudeEngineering)
                    aceleracaoHardware =
gettingSignal(dataIntCelCarga, nBytesFromCelCarga, 3, 2)
                    aceleracaoHardware =
unsignedToSigned(aceleracaoHardware, 2)
                    aceleracaoEngineering = aceleracaoHardware/100

                    anguloHardware = gettingSignal(dataIntDrone,
nBytesFromDrone, 1, 2)
                    anguloEngineering =
unsignedToSigned(anguloHardware, 2)/100
                    angle.append(anguloEngineering)
                    acaoA = gettingSignal(dataIntDrone,
nBytesFromDrone, 3, 1)
                    acaoB = gettingSignal(dataIntDrone,
nBytesFromDrone, 4, 1)
```

```
                    dataA.append(acaoA)
                    dataB.append(acaoB)
                    ##########TX##########
                    if len(altitude) > 1:
                        if (abs(altitude[-1] - altitude[-2]) < 1) and
(abs(aceleracaoEngineering) < 20): # Só passa a msg para o drone se a
diferença de altitude for menor que 1m, isso implica em 100m/s e
aceleração menor que 20m/s² ou 2g

TX(drone,dataByteCelCarga,setpoint_altitude,setpoint_angle)
                        else: # Falha: altitude variou muito de uma
iteração para outra ou aceleração muito alta
                            altitude[-1] = altitude[-2]
                    elif abs(aceleracaoEngineering) < 20:

TX(drone,dataByteCelCarga,setpoint_altitude,setpoint_angle)
                else:
                    SOP_counter += 3
                    altitude.append(altitude[-1])
            else:
                MM_counter += 3
                altitude.append(altitude[-1])
        else:
            altitude.append(0)
            angle.append(0)

        if MM_counter>=15:
            print("MM FAILED")
            fimdejogo = True
        if SOP_counter>=15:
            print("SOP FAILED")
            fimdejogo = True

        fundo.fill(white) #Desenhando o fundo

        [setpoint_x,setpoint_y] = [ang2px(setpoint_angle),
m2px(setpoint_altitude)] #Calculando a posição em pixels do setpoint
        draw_setpoint(setpoint_x,setpoint_y) #Desenhando o sepoint

        [pos_x,pos_y] = [ang2px(angle[-1]), m2px(altitude[-1])]
#Calculando a posição em pixels do drone
        pos_x = pos_x%largura #Se passar das bordas laterais, o drone
aparece do outro lado
        pos_y = pos_y%altura #Se passar das bordas superior ou
inferior, o drone aparece do outro lado
        draw_drone(pos_x, pos_y) #Desenhando o drone
        #Imprimindo os valores atuais
        texto("Altura Real: " +str(int(altitude[-1]*100)/100) +"
m",red, 10,10)
        texto("Angulo Real: " +str(angle[-1]) + " °",red, 10,20)
        texto("Setpoint Altura: " +str(setpoint_altitude) +" m",green,
10,30)
        texto("Setpoint Angulo: " +str(setpoint_angle) + " °",green,
10,40)

        pygame.display.update() #O tempo de atualização depende do
tempo do recebimento da mensagem
        i = i+1
        toc = time.time()
        tempo.append(toc-tic) #Salvando o tempo passado desde o início
```

```
    #Quando fechar a tela, executa as seguintes instruções:
    if teste == False:
        drone.write(bytes([17])) #Desliga os motores
        drone.write(bytes([17])) #Desliga os motores
        celCarga.close() #Fecha a comunicação serial
        drone.close() #Fecha a comunicação serial
        plt.figure() #Plota alguns gráficos
        plt.plot(altitude, label = "Altitude")
        plt.plot(data_setpoint_altitude, label = "Setpoint da
Altitude")
        plt.legend()
        plt.show()
        file.write("altitude = " + str(altitude)) #Salva os dados num
arquivo txr
        file.write("\nsetpoint_altitude = " + str(setpoint_altitude))
        file.write("\nangle = " + str(angle))
        file.write("\nsetpoint_angle = " + str(setpoint_angle))
        file.write("\ndataA = " + str(dataA))
        file.write("\ndataB = " + str(dataB))
        file.write("\ntempo = " + str(tempo))
        file.close()

jogo()


pygame.quit()
```

APÊNDICE D – Tabela de custos do projeto

Tabela 1 – Custo estimado dos principais componentes do projeto

Item	Quantidade	Custo Unitário	Custo
Célula de Carga ELDOER de 5kg	1	9,43	9,43
Amplificador de Instrumentação INA 101	1	23,00	23,00
ESC Brushless Motor XXD HW30A	2	27,62	55,24
Brushless Motor BR2212 980KV	2	30,80	61,60
8045 Carbon Nylon CW/CCW Propeller	1	9,65	9,65
PCB Quadcopter Frame Kit 450mm	1	80,72	80,72
Acelerômetro MPU-6050 GY-521	1	12,90	12,90
		Total	252,54

Fonte: Produzido pelo autor

APÊNDICE E – Vídeos dos resultados

Vídeo 1 - Controle da altitude:
https://drive.google.com/file/d/1YQIFiSFx9qv0mEPL7DbYhvBagaQ6n0zp/view

Vídeo 2 - Controle do ângulo:
https://drive.google.com/file/d/1aMudT8AdMdw1vAy4nOfp9O8LKo2wQb8h/view

Vídeo 3 - Controle do ângulo + altitude:
https://drive.google.com/file/d/1x4kKeqtWaCw5Qj1H6BE1zpKGQFL0Xsc_/view