UNIVERSIDADE DE SÃO PAULO
ESCOLA POLITÉCNICA

REGINALDO CARDOSO

Planejamento de Trajetória e Controle para Veículo Aéreo Não tripulado para Inspeção de Gasoduto

São Paulo
2024

REGINALDO CARDOSO

Planejamento de Trajetória e Controle para Veículo Aéreo Não tripulado para Inspeção de Gasoduto

Versão Corrigida

Tese apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do título de Doutor em Ciências.

São Paulo
2024

REGINALDO CARDOSO

Planejamento de Trajetória e Controle para Veículo Aéreo Não tripulado para Inspeção de Gasoduto

Versão Corrigida

Tese apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do título de Doutor em Ciências.

Área de Concentração:
Engenharia de Controle e Automação Mecânica

Orientador:
Prof. Dr. Décio Crisol Donha

São Paulo
2024

Este exemplar foi revisado e corrigido em relação à versão original, sob responsabilidade única do autor e com a anuência de seu orientador.

São Paulo, _____ de _____ de _____

Assinatura do autor: _____

Assinatura do orientador: _____

Catalogação-na-publicação

Cardoso, Reginaldo
Planejamento de Trajetória e Controle para Veículo Aéreo Não tripulado para Inspeção de Gasoduto / R. Cardoso -- versão corr. -- São Paulo, 2024.
170 p.

Tese (Doutorado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia Mecânica.

1.Veículos autônomos 2.Detecção de vazamentos 3.Gasoduto 4.Controle não-linear I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia Mecânica II.t.

CARDOSO, R. **Planejamento de Trajetória e Controle para Veículo Aéreo Não tripulado para Inspeção de Gasoduto.** 2024. 166 f. Tese (Doutorado em Ciências em Engenharia de Controle e Automação Mecânica) – Escola Politécnica, Universidade de São Paulo, São Paulo, 2024.

Aprovado em:

Banca Examinadora

Prof. Dr. _____

Instituição: _____

Julgamento: _____

Prof. Dr. _____

Instituição: _____

Julgamento: _____

Prof. Dr. _____

Instituição: _____

Julgamento: _____

Prof. Dr. _____

Instituição: _____

Julgamento: _____

Prof. Dr. _____

Instituição: _____

Julgamento: _____

Este trabalho é dedicado a todas as pessoas que de alguma maneira me apoiaram e me ajudaram a chegar até este momento, em especial a minha esposa Priscila Akiti que sempre esteve ao meu lado.

Resumo

CARDOSO, R. **Planejamento de Trajetória e Controle para Veículo Aéreo Não tripulado para Inspeção de Gasoduto.** 2024. 166 f. Tese (Doutorado em Ciências em Engenharia de Controle e Automação Mecânica) – Escola Politécnica, Universidade de São Paulo, São Paulo, 2024.

Este projeto concentra-se na modelagem, controle e construção de um veículo aéreo pequeno não tripulado (UAV - *Unmanned aerial vehicle*), do tipo quadricóptero, para identificar vazamentos em gasodutos visíveis (não enterrados). Todas as atividades do veículo devem ser de forma autônoma, assim, o UAV deve realizar a decolagem, o voo de cruzeiro seguindo o gasoduto e o pouso de maneira autônoma. Para garantir a estabilidade e o desempenho desejado, técnicas de controle robustas (controle de modos deslizantes) e não lineares (*backstepping*) serão utilizadas. O UAV será equipado com os sensores necessários para o voo (IMU -*inertial measurement unit*, barômetro e GPS -*global positioning system*), acrescido de uma câmera de 8 "megapixels", para identificar e gerar a trajetória rastreio do gasoduto e outra câmera do tipo WIR - *Wavelength Infrared*, capaz de identificar e distinguir hidrocarbonetos como CH_4 e/ou CO_2.

Palavras chaves: Veículos autônomos, Detecção de vazamentos, Gasoduto, Controle não-linear.

Abstract

CARDOSO, R. **Planejamento de Trajetória e Controle para Veículo Aéreo Não tripulado para Inspeção de Gasoduto.** 2024. 166 f. Tese (Doutorado em Ciências em Engenharia de Controle e Automação Mecânica) – Escola Politécnica, Universidade de São Paulo, São Paulo, 2024.

This project focuses on the modeling, control and construction of a quadricopter-type small unmanned aerial vehicle (UAV - Unmanned aerial vehicle) to identify leaks in visible (overground) pipelines. All vehicle activities must be autonomous, thus, the UAV must carry out the takeoff, cruise flight following the pipeline and landing autonomously. To ensure stability and the desired performance, robust (sliding mode control) and non-linear (backstepping) control techniques will be used. The UAV will be equipped with the sensors for the flight (IMU - inertial measurement unit, barometer and GPS - global positioning system), added to an 8 megapixel camera, to identify and generate the trajectory tracking of the pipeline and another WIR-type camera - Wavelength Infrared, capable of identifying and distinguishing hydrocarbons such as CH_4 and/or CO_2.

Key-words: Unmanned aerial vehicles, Leak detection, Pipeline, Nonlinear control.

Sumário

1 Introdução

O número de aplicações de veículos aéreos não tripulados (VANT), também conhecidos pela sigla em inglês UAV (*Unmanned Aerial Vehicles*), tem aumentado nos últimos anos devido à sua eficiência em realizar diversas atividades e ao baixo custo de construção (LABBADI; CHERKAOUI, 2019).

Além disso, a tecnologia dos VANT tem evoluído rapidamente, permitindo a incorporação de recursos como câmeras de alta resolução, sensores de detecção e equipamentos de comunicação, o que aumenta ainda mais suas capacidades e possibilidades de uso em áreas como agricultura, monitoramento ambiental, segurança pública, entre outras (KIM et al., 2019).

Os VANT podem ser classificados em dois grupos: veículos de asa fixa, conforme Figura 1a e veículos multirotores (que usam rotores como sustentação), conforme Figura 1b (NOURMOHAMMADI; JAFARI; ZANDER, 2018).

Figura 1 – Exemplos de VANT.

(a) VANT de asa fixa, Skywalker.

(b) VANT multirotores, AKS Raven X8.

Fonte: Boon, Drijfhout e Tesfamichael (2017)

É importante ressaltar que cada tipo de VANT possui características distintas, o que os torna mais adequados para diferentes aplicações. Os veículos multirotores, como quadricópteros, possuem alta capacidade de manobra, decolagem e pouso vertical (VTOL-sigla em inglês: *Vertical Take-off and Landing*) o que permite realizar atividades em ambientes fechados, pequenos e com superfície irregular. Por estas vantagens este tipo de veículos são mais atrativos que os demais veículos (WANG et al., 2019).

Os VANT podem ser operados de duas maneiras distintas: por meio de controle remoto realizado por um operador, ou por meio de operação autônoma, utilizando uma combinação de processadores, sensores e técnicas de controle para executar a atividade desejada (RENSHAW; WIGGINS, 2017).

Cabe ressaltar que a escolha da forma de operação deve ser feita com base nas

necessidades específicas de cada aplicação. Na operação por controle remoto, o operador assume a responsabilidade de controlar o VANT, o que pode reduzir o custo com sensores embarcados, sendo mais adequada para atividades que exigem maior flexibilidade e adaptação às condições do ambiente. Já na operação autônoma, o VANT é capaz de executar a tarefa sem a intervenção humana direta, o que pode ser útil em operações de maior complexidade ou atividades repetitivas.

Os VANT são sistemas dinâmicos com múltiplas entradas e múltiplas saídas (MIMO - sigla em inglês: *Multiple-Input Multiple-Output*) altamente não lineares, incluindo incertezas paramétricas, perturbações e dinâmicas não modeladas. Além disso, esses sistemas são subatuados, o que significa que possuem seis graus de liberdade, mas apenas quatro entradas de controle, no caso de um quadricóptero. Isso pode levar a um forte acoplamento dinâmico, tornando o controle desses sistemas ainda mais complexo (LABBADI; CHERKAOUI, 2019).

As contribuições presentes neste trabalho incluem o desenvolvimento da combinação das técnicas de controle *backstepping* com o controle de modos deslizantes, a utilização do algoritmo de detecção de bordas para identificar o gasoduto e o emprego do algoritmo de detecção de pluma de gás em uma imagem. Além disso, apresenta-se o desenvolvimento de um novo modo de voo na plataforma ArduPilot, permitindo a comutação do controlador, tanto na posição quanto na atitude, do quadricóptero.

1.1 Motivação

O gás natural (GN) é composto por uma mistura de gases, com destaque para o metano (CH_4), que representa mais de 95% do total. Sua combustão emite menos gases de efeito estufa do que outras fontes de combustível fóssil, como o carvão e os derivados do petróleo (BARKANOV, 2018).

Em 2015, aconteceu em Paris, a vigésima primeira conferência das partes (COP 21 - sigla em inglês: *21st Conference of the Parties*) sobre mudanças climáticas. Um dos acordos foi a redução de carbono na atmosfera, a fim de diminuir o aquecimento global (SUTTER; BERLINGER, 2015).

O GN é considerado um grande aliado na luta contra o aquecimento global, não apenas por sua combustão emitir menos gases de efeito estufa, mas também por ser empregado em diversas indústrias. No entanto, é importante lembrar que o metano, componente majoritário do GN, também é um gás de efeito estufa. Apesar disso, muitos questionam os benefícios da substituição do carvão e dos derivados do petróleo pelo GN, que pode levar a uma significativa redução das emissões de gases do efeito estufa em curto prazo. É importante destacar que a concentração global de metano na atmosfera vem aumentando, e as causas desse fenômeno ainda não são totalmente compreendidas

(BRANDT et al., 2014).

O transporte do GN é feito principalmente por meio de gasodutos (GD), que podem atravessar áreas remotas e altamente povoadas. É crucial monitorar esses dutos durante o transporte para evitar vazamentos. No entanto, de acordo com a Agência de Proteção Ambiental dos EUA, $1,5\%$ de todo GN já produzido pelos EUA foi perdido devido a vazamentos (BRANDT et al., 2014).

As técnicas tradicionais de detecção de vazamentos em tubulações, como a variação de pressão e sondas internas, não são sensíveis o suficiente para detectar pequenos vazamentos, como $0,1$ litro por minuto ou 1% do fluxo total da tubulação (GRAVEL et al., 2016). O principal problema é a ocorrência de alarmes falsos, que diminuem a confiabilidade do sistema de prevenção e exigem trabalho adicional, como o deslocamento de uma equipe de manutenção para verificar a situação (RAMADEVI; JAIGANESH; KRISHNAMOORTHY, 2019).

Outro método de inspeção é o externo, que utiliza sensores para detectar a presença de gás na parte externa do gasoduto. O sensor pode ser posicionado em um helicóptero, carro ou, mais recentemente, em VANT. A inspeção e o monitoramento de dutos com helicópteros são realizados periodicamente ou quando algum sensor detecta um possível vazamento. Segundo Birchbauer, Wakolbinger e Hornacek (2017), os VANT oferecem vantagens, pois permitem uma reação mais rápida e inspeções mais frequentes. O uso de VANT para inspeção de uma rede de GD pode diminuir o tempo e o custo da inspeção em $22,07\%$ e $66,48\%$, respectivamente, em comparação com uma equipe de 18 trabalhadores, como demonstrado por Yan et al. (2018).

1.2 Revisão Bibliográfica

Esta seção do trabalho foi divida da seguinte maneira: na subseção 1.2.1, apresenta-se a revisão bibliográfica do controle *backstepping* em conjunto com o de modos deslizantes. Na subseção 1.2.2 apresenta-se uma revisão sobre os métodos de identificação de vazamentos e por fim na subseção 1.2.3 apresenta-se um revisão bibliográfica de trabalhos com a mesma temática desta tese.

1.2.1 Revisão bibliográfica sobre o método de controle

Apesar de os VANT serem sistemas altamente não lineares e subatuados, muitos trabalhos têm sido desenvolvidos com técnicas de controle lineares, incluindo o controle PID (BOUABDALLAH; NOTH; SIEGWART, 2004) (LIU; GAO, 2017), que envolve a linearização prévia do sistema. Para lidar com a subatuação, a força vertical é decomposta em função dos ângulos de rotação em torno dos eixos x e y, gerando assim duas novas variáveis de controle que tornam o sistema não subatuado (VOOS, 2006).

Na linearização, alguns aspectos considerados não importantes do modelo podem ser negligenciados. Por essa razão o desempenho do veículo como precisão de rastreamento e robustez contra as perturbações externas (vento por exemplo), pode ser ruim, até mesmo não aceitável (TANG; LI, 2015).

Com o aumento do uso de VANT, tem sido aplicado um número crescente de técnicas de controle não lineares. No entanto, em muitas aplicações, ainda é utilizada a estratégia de controle linear para resolver o problema da subatuação. Para isso, o modelo do veículo é dividido em dois sistemas independentes: posição (externo) e atitude (interno). Isso se deve ao fato de que o sistema de velocidades angulares não depende das velocidades lineares do outro sistema (BOUABDALLAH; SIEGWART; CAPRARI, 2006).

Assim, é possível aplicar uma técnica de controle em cada sistema. Destaca-se o uso das técnicas de modos deslizantes para controlar a atitude e *backstepping* para controlar a posição do veículo (ALMAKHLES, 2020). Vale ressaltar que o controle de atitude é mais importante, uma vez que a posição desejada só pode ser alcançada se a atitude desejada for alcançada primeiro (VOOS, 2006) (ALMAKHLES, 2020).

O controle *backstepping* é amplamente utilizado por ser uma técnica de controle não linear capaz de controlar e garantir a estabilidade do sistema simultaneamente. Isso ocorre porque a cada etapa do desenvolvimento do controle há uma realimentação, o que significa que o controle é projetado em etapas sucessivas para obter o desempenho desejado do sistema (KRSTIC; KANELLAKOPOULOS; KOKOTOVIC, 1995).

O controle *backstepping* é uma técnica de controle recursiva que utiliza realimentações para construir leis de controle, as quais são projetadas por meio de uma função de *Lyapunov*. Essa abordagem é semelhante à linearização por realimentação de estados (*feedback-linearization*), com a diferença de que, no caso do *backstepping*, apenas as parcelas instáveis são canceladas, mantendo-se as demais. As variáveis de realimentação do sistema são denominadas variáveis virtuais de entrada. Para que o controlador opere corretamente, é necessário possuir conhecimento prévio do modelo e dos parâmetros, os quais serão utilizados na realimentação dos estados (KOKOTOVIC; ARCAK, 2001).

Uma das limitações do uso da técnica de controle *backstepping* em quadricópteros é a presença de incertezas na obtenção dos parâmetros do modelo matemático. É fundamental que os parâmetros do sistema sejam corretamente identificados e atualizados para que os ganhos do controlador possam ser adequadamente ajustados.

Seguindo essa linha de raciocínio, alguns estudos têm abordado a inclusão de uma lei adaptativa no controle *backstepping* a fim de lidar com as incertezas paramétricas do modelo. Um exemplo é o trabalho realizado por Nguyen, Xuan-Mung e Hong (2019), no qual foi desenvolvido um controlador *backstepping* em conjunto com uma lei adaptativa para abordar as seguintes incertezas paramétricas constantes: momento de inércia, comprimento

do veículo e coeficiente de arrasto. A estimação desses parâmetros foi realizada utilizando um método de gradiente descendente do erro.

Uma abordagem alternativa na obtenção da lei adaptativa é a utilização do teorema de estabilidade de *Lyapunov*. Nessa abordagem, um limite de erro de estimação é estabelecido, e caso o erro exceda esse limite, a lei adaptativa é modificada para manter o erro dentro de valores aceitáveis, caso contrário, a lei permanece inalterada. Diversos estudos têm explorado essa teoria, como o trabalho de Bhatia et al. (2019), que propuseram uma lei adaptativa composta por componentes derivativos e integrativos do erro de estimação, e o trabalho de Xie et al. (2022), que propuseram uma lei adaptativa baseada em uma função exponencial do erro. Em ambas as abordagens, os trabalhos demonstraram, por meio de simulações, uma rápida convergência da estimação, alcançando resultados satisfatórios em menos de $2,5$ segundos.

No estudo conduzido por Bhatia et al. (2019), foi assumido que as incertezas são originadas de fatores externos, como variações no peso e energia de recuo durante o disparo de uma arma. Já no estudo realizado por Xie et al. (2022), foi assumido que apenas a massa do veículo é desconhecida, sendo conduzidos testes experimentais na presença de distúrbios externos, como o vento. Em ambos os casos, o veículo foi capaz de seguir a trajetória desejada sem apresentar grandes erros.

Na literatura há vários estudos que usam um termo integral do erro da variável virtual junto com o *backstepping* (DAVIDI; BERMAN; AROGETI, 2011), (FAN; CAO; LI, 2017), (ELIKER; ZHANG, 2020). A inclusão desse termo integrativo tem o propósito de garantir que o erro da variável convirja para zero no estado estacionário (AL-YOUNES; JARRAH; SUKKARIEH, 2009).

Da mesma maneira que a parcela integrativa é adicionada no controlador *backstepping*, alguns autores fazem a junção de outras técnicas de controle com o intuito de acrescentar robustez ao controlador, devido as incertezas paramétricas e aos distúrbios externos (HE; ZHAO; ZHAO, 2016). A combinação mais comum para quadricópteros é o integral *backstepping* com o controle de modos deslizantes (SMC - sigla em inglês: *sliding mode controller*) (JIANG; SONG; LIN, 2019), (LABBADI; CHERKAOUI, 2019).

O SMC é caracterizado pelo uso de uma lei de controle descontínua que realiza comutações (funções de chaveamento) entre um conjunto de funções das variáveis de estado do modelo, seguindo uma regra estabelecida. Essas funções de chaveamento são projetadas de forma a permitir que o sistema alcance e se mantenha em uma superfície denominada superfície de deslizamento. O SMC oferece vantagens significativas, como a capacidade de rejeição de perturbações não modeladas ou variações de parâmetros (OLIVEIRA, 2006). No entanto, uma desvantagem desse tipo de controle é o surgimento do fenômeno conhecido como *chattering*, que ocorre quando o sistema oscila em torno da superfície de deslizamento. Uma solução para esse problema é a introdução de uma

camada limite, que tem como objetivo tornar a superfície de deslizamento mais espessa (PERRUQUETTI; BARBOT, 2002).

Na literatura existe duas maneiras de combinação técnicas de controle *backstepping* e SMC, a mais comum é adicionando a superfície de deslizamento na função *Lyapunov* do controle *backstepping* (ZHANG; CHEN, 2019), (YI; WATANABE; NAGAI, 2021), (JIA et al., 2017). Alternativamente, é possível adicionar a superfície de deslizamento diretamente na lei de controle obtida pelo controle *backstepping* (ANTONIO-TOLEDO et al., 2018), (ALMAKHLES, 2020).

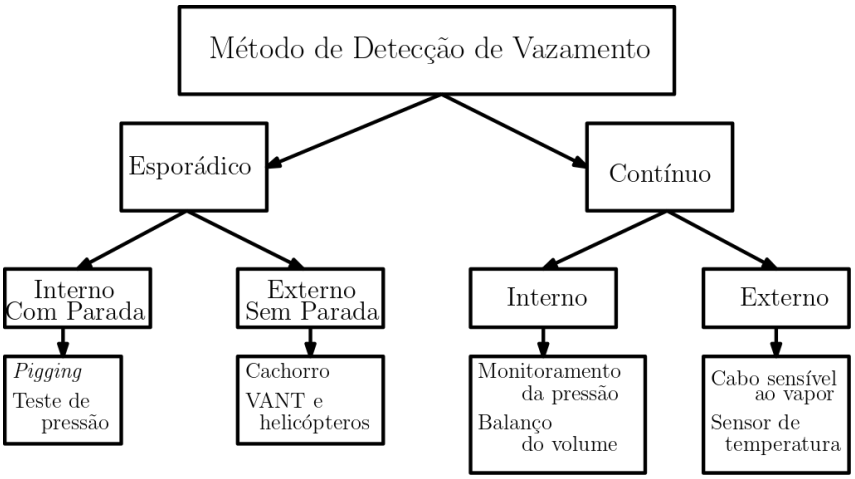
Este trabalho concentra no desenvolvimento de uma combinação do controle *backstepping* com o controle SMC. A abordagem apresentada é baseada no trabalho Cardoso et al. (2017), onde o controle descontínuo (superfície de deslizamento) é igualada ao erro de velocidade, do controle *backstepping*. Além disso, um estimador é usado para estimar os parâmetros do veículo.

1.2.2 Métodos de identificação de vazamentos

Nesta secção apresenta-se uma breve revisão dos métodos de identificação de vazamentos de gás em dutos.

Os métodos de detecção de vazamento podem ser divididos em duas categorias, os esporádicos e os contínuos. O método esporádico pode ser subdividido em outras duas subcategorias: interno (sem parada) e externo (com parada) (WALK, 2010). A mesma coisa com o método contínuo, que pode ser subdividido em outras duas subcategorias: interno e externo (BIRCHBAUER; WAKOLBINGER; HORNACEK, 2017), conforme pode-se observar na Figura 2.

Figura 2 – Métodos de detecção de vazamento em gasodutos.



Fonte: Adaptado de Birchbauer, Wakolbinger e Hornacek (2017) e Walk (2010).

O método esporádico interno com parada ocorre na parte interna do GD, por

exemplo *pigging*, onde um dispositivo com sensores desloca-se livremente dentro do GD realizando inspeções (GRUPA, 2016). Já o esporádico externo sem parada ocorre na parte externa do GD, por exemplo sensores embutidos em helicópteros, VANT ou cães treinados para farejar vazamento de gás (HOLLANDER; BOLLERMANN, 2007) (RAMADEVI; JAIGANESH; KRISHNAMOORTHY, 2019).

O método contínuo opera em paralelo ao transporte do gás. O contínuo interno baseia-se em algoritmos matemáticos que usam os dados de processo em tempo real para medir o fluxo, o balanço de massa ou volume e analisa a variação dos pontos de pressão. O contínuo externo utiliza sensores externos ao GD, como detectores de vapor e cabos de fibra óptica sensíveis a temperatura (WALK, 2010).

1.2.3 Trabalhos Similares

Alguns trabalhos apresentam a mesma temática desta tese, mas nenhum propõem o rastreamento do duto e a operação de maneira autônoma.

Os autores do trabalho Iwaszenko et al. (2021) utilizaram um sensor laser (Laser Methane mini SA3C321-BE) acoplado em um UAV (Matrice 600 Pro fabricado pela DJI Technology) para monitorar um gasoduto subterrâneo com $30m$ de comprimento. A parte de controle não é comentada no trabalho, somente informa que foi utilizada a navegação *waypoint*. O sensor possui uma comunicação via *Bluetooth* que transmite os dados para um *smartphone* que armazena os dados em conjunto com o sinal de GPS. Os dados são descarregados em um computador após o voo, onde é realizado o processamento dos dados. Primeiro, os autores calibraram a altura de voo do veículo com relação aos dados obtidos do sensor. Realizaram uma simulação de vazamento, proveniente de um cilindro com 95% de metano e 5% de nitrogênio posicionado no solo. O UAV realizou sobrevoos variando a altura ($2,5m$ até $25m$) e com velocidade constante de $1,4m/s$. O experimento com $3m$ de altura do vazamento apresentou o melhor resultado, mas os autores recomendam uma altura superior à $6m$, para que os rotores não atrapalhem a medição. Em outro experimento com condições reais, os autores Iwaszenko et al. (2021) sugeriram que a melhor altura para o veículo está entre $4m$ e $15m$ do solo.

Li et al. (2020) realizaram um estudo comparativo entre um UAV e um carro, para detectar vazamento de um gasoduto subterrâneo. O UAV modelo Matrice 600 fabricado pela DJI Technology, embarcava uma câmera de imagem óptica de gás (OGI - *Optical Gas Imaging*) e um *tunable diode laser absorption spectrometer* (TDLAS). No carro, um sensor laser (*laser methane copter*) fazia as medições e quando detectavam a presença de gás, estacionavam o carro e posicionavam uma câmera infravermelho (modelo GF320 da FLIR Systems) e filmavam o local. Realizaram 53 voos cobrindo $56km$ de dutos e com o carro foi possível percorrer somente $17km$. Com o UAV os autores encontraram 6 áreas com possíveis vazamentos, onde foi possível visualizar uma degradação da vegetação. Com o

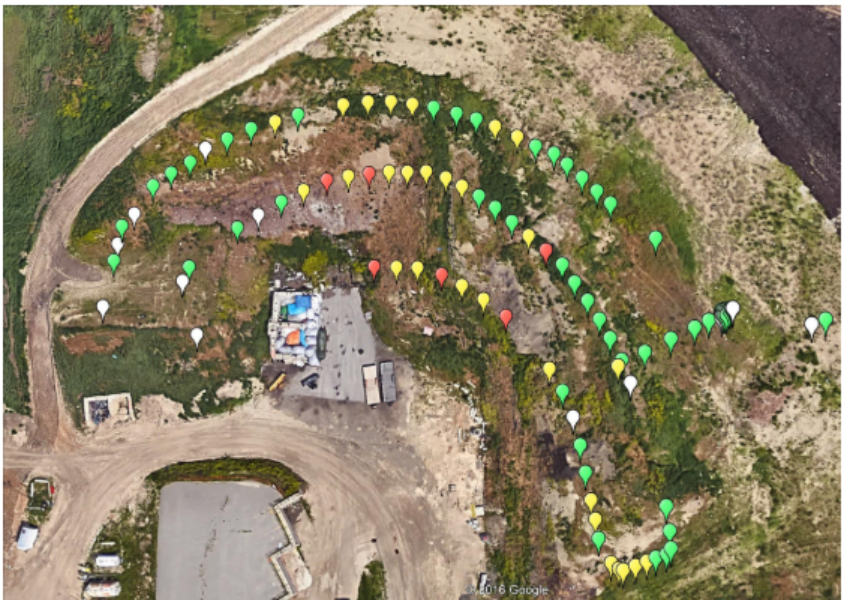
carro, encontraram dois vazamentos em uma válvula de plástico sobre o solo. Assim os autores chegaram à conclusão que o UAV acessa locais difíceis, é mais seguro em caso de vazamento mas possui limitações de operação como por exemplo vento constante e inferior a $2,5m/s$.

Nooralishahi, López e Maldague (2021) propuseram um algoritmo para estabilizar filmagens realizadas com câmera de infravermelho acoplada em um UAV, e assim visualizar a movimentação da pluma de gás. O primeiro quadro (*frame*) da filmagem é utilizado como referência, depois analisa-se a iluminação e define-se uma intensidade mínima para cada *pixel* , por fim a imagem é binarizada. Foram realizados 3 experimentos, no primeiro com a câmera modelo FLIR GF620 posicionada a um metro de uma garrafa com álcool e uma com gasolina. O segundo utilizou imagens de um banco de dados (da universidade do Colorado, *Methane Emissions Technology Evaluation Center*) captadas com a câmera modelo FLIR GF320 posicionada entre $4,6$ a $15,6m$ de um vazamento de metano. No terceiro experimento utilizou-se um UAV (DJI Mavic 2 Pro) com a câmera modelo FLIR GF309 em quatro cenários, o primeiro com o veículo sobrevoando o vazamento, o segundo segurando a câmera e simulando a movimentação do veículo, o terceiro com o veículo em constante movimentação e objetos se movendo no campo de visão da câmera e por último com o UAV em constante movimentação. Em todos os experimentos o algoritmo conseguiu ressaltar o fluxo de gás presente na imagem e detectar a região com maior concentração.

Outros trabalhos apresentam uma temática relacionada, como Emran, Tannant e Najjaran (2017), que apresentam o uso de um UAV (hexacóptero) com um sensor comercial à laser para detecção de metano. O feixe de laser é direcionado verticalmente para a superfície do solo. O algoritmo quantifica a concentração de metano em partes por milhão multiplicada pela distância entre o veículo e o solo ($ppm.m$). O sensor com os acessórios de montagem, pesa cerca de $0,6$ kg e opera a uma taxa de aproximadamente $10Hz$. O processamento não é realizado em tempo real. Os dados são armazenados em conjunto com o sinal do GPS, para o processamento. Os autores testaram o veículo sobre um aterro sanitário, com o objetivo de mapear os pontos de emissão de metano resultante da decomposição de resíduos. A resposta do teste pode ser observada na Figura 3, onde a cor branca significa concentração insignificante ($< 25ppm.m$), verde baixa concentração ($25 - 75ppm.m$), amarelo média ($75 - 150ppm.m$) e vermelho alta concentração ($> 150ppm.m$).

O trabalho Yang et al. (2018), também utiliza um sensor do tipo diodo laser com comprimento de onda variável (*Backscatter Tunable Diode Laser Absorption Spectroscopy*, TDLAS). Os autores desenvolveram um algoritmo capaz de identificar o vazamento e a sua origem. A detecção é feita em tempo real, mas os dados são transmitidos via *wireless* para uma estação no solo, onde o processamento é realizado. A aplicação foi idealizada para plantas industriais. O sensor quantifica a concentração de metano em partes por milhão

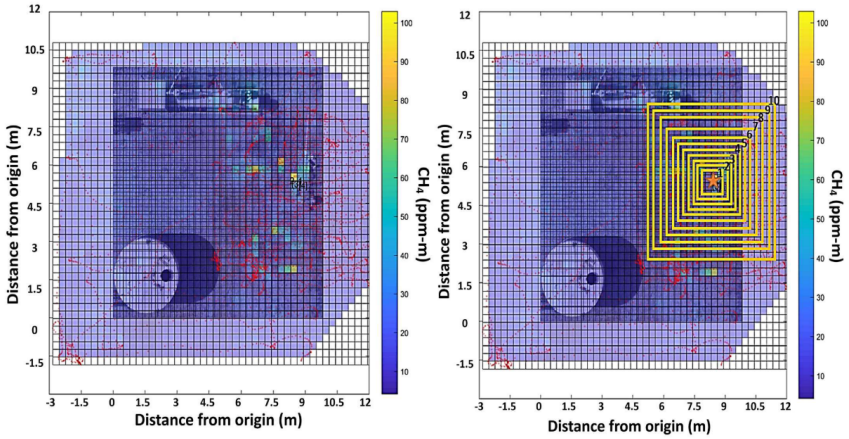
Figura 3 – Localizações dos pontos de concentração de metano em aterro.



Fonte: Emran, Tannant e Najjaran (2017)

multiplicada pela distância entre o veículo e o solo ($ppm.m$). Assim, o veículo precisa realizar uma varredura de toda a área da planta. O algoritmo depende da velocidade do vento, portanto utiliza-se um anemômetro. O algoritmo apresenta uma forte dependência na qualidade do sinal de GPS (ruído e erro) e das condições de vento (menor velocidade, pior o resultado). O resultado pode ser observado na Figura 4, a imagem à esquerda apresenta a vista aérea da área e a sua segmentação, já a imagem à direita, o resultado do algoritmo.

Figura 4 – Resultado do algoritmo desenvolvido para detectar a origem do vazamento.



Fonte: Yang et al. (2018)

Bretschneider e Shetti (2014) descrevem as vantagens e desvantagens de detectores baseados em imagens e laser na detecção de plumas de gás, proveniente de vazamento em gasoduto. A uma distância de 20 metros, com a câmera não foi possível detectar o gás. A

maior vantagem da câmera é a área de cobertura. A desvantagem é o custo e o peso, por exemplo o modelo FLIR Ultra 8000e com estabilizador (*gimbal*) pesa 13 kg. Já os sensores baseados em laser, fornecem medições quantitativas mais precisas e tamanho compacto. A desvantagem é a área de cobertura que é pontual. Após apresentarem as vantagens e desvantagens, os autores optaram pelo sensor à laser (*Laser Methane* mini G, LMmG, da Pergam), devido ao custo e peso. Os sensores foram ligados a um celular para transmitir os dados em tempo real, via 3G, para um computador no solo. Neste trabalho os autores não apresentaram nenhum pós-processamento dos dados.

Roldán et al. (2015) desenvolveram um UAV com sensores do tipo resistivo. O intuito do trabalho é a medição de temperatura, umidade, luminosidade e concentração de CO_2 no ambiente. O sensor utilizado precisa estar imerso na pluma de gás. Assim, os autores realizaram um estudo dinâmico computacional de fluido (CFD - *computational fluid dynamics*), para determinar a posição do sensor no UAV de modo que os rotores não afetassem a medida do sensor. Para ambos os sensores a melhor posição foi na parte superior no centro do UAV. Para validar os resultados, os autores fixaram o veículo com os rotores desligados e realizaram a medição, depois repetiram os experimento com os rotores ligados. O erro não ultrapassou os 4% para o sensor, já o sensor de CO_2 apresentou um erro de quase 15% a menos de um metro da fonte emissora, mas após $2m$ o erro foi para zero. Por fim, os autores realizaram o levantamento de um estufa de $106m$ por $47m$ com um erro de medição inferior a 4%.

1.3 Objetivos

Esta tese tem como objetivo a modelagem, controle e construção de um veículo aéreo pequeno não tripulado, do tipo quadricóptero, para identificar vazamento em dutos visíveis (não enterrados).

O desenvolvimento da combinação das técnicas de controle *backstepping* com o controle SMC, onde o controle descontínuo (superfície de deslizamento) é igualada ao erro de velocidade, do controle *backstepping*. Desenvolvimento e simulação do controle e do veículo na plataforma ArduPilot.

A construção de um algoritmo de detecção de bordas, método de *Canny*, em conjunto com a transformação de *Hough*, para identificar as bordas do duto. O desenvolvimento do controle servo visual baseado em posição.

Um algoritmo capaz de detectar a presença de uma plume de gás em uma imagem.

1.4 Organização da Tese

Esta tese é organizada da seguinte forma. O Capítulo 2 apresenta as hipóteses, forças e momentos levados em consideração no processo de modelagem do veículo. Em seguida apresenta-se o desenvolvimento da técnica de controle baseada na combinação do controle de modos deslizantes com o controle *backstepping*. Depois apresenta-se o desenvolvimento do algoritmo genético para a obtenção dos parâmetros do controlador. Por fim, apresenta-se a simulação (*model in the loop*) e uma breve discussão.

O Capítulo 3 apresenta do algoritmo de detecção de bordas desenvolvido para gerar a trajetória de referência do veículo. Em seguida, apresenta-se o controle servo visual baseado em posição (PBVS) e suas hipóteses. Por fim, a simulação dos controles (PBVS, *backstepping* e modos deslizantes) e uma breve discussão dos resultados.

O Capítulo 4 apresenta o algoritmo capaz de detectar a presença de gás na imagem por meio da técnica de binarização e uma explicação teórica do funcionamento de uma câmera térmica capaz de detectar plume de gás.

O Capítulo 5 apresenta a montagem do veículo e os principais componentes adquiridos, como placas de controle e controle de radio frequência. Em seguida foi adicionado o controle desenvolvido em um novo modo de voo criado no ArduPilot. Por fim, a simulação (*software in the loop*) e uma breve discussão.

Por fim, o Capítulo 6 apresenta os resultados alcançados e trabalhos futuros.

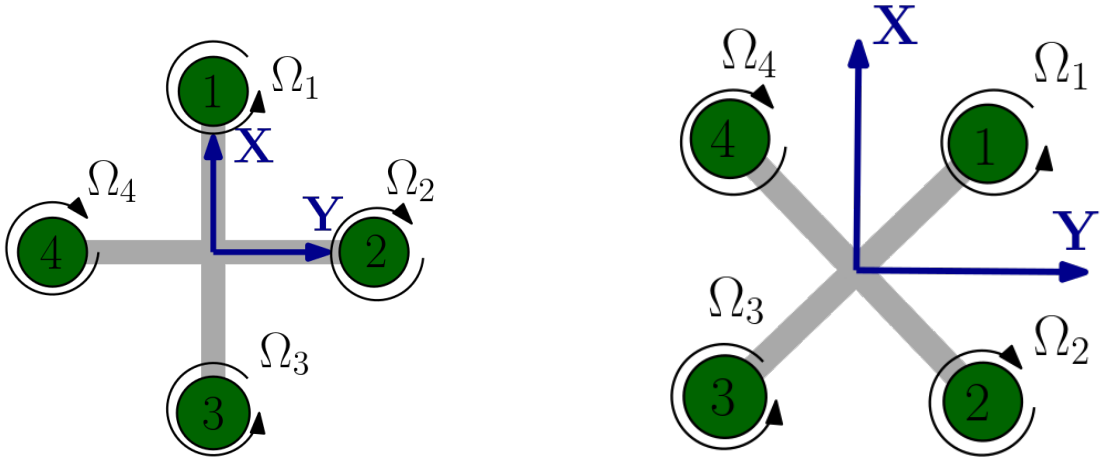
2 Modelagem e Controle do Veículo

Optou-se por um VANT do tipo multirotor, pois este tipo de veículo pode pairar sobre um determinado ponto e assim é possível fazer uma análise mais precisa do vazamento.

De acordo com Magnussen, Hovland e Ottestad (2014) a melhor performance dinâmica, considerando carga e tempo de voo, foi obtida na configuração com 4 rotores, assim optou-se por utilizar um veículo do tipo quadricóptero. Um outro motivo considerado foi o custo, que aumentaria com a compra de mais atuadores e seus respectivos controladores.

Um quadricóptero consiste em quatro rotores que podem ser montados em uma estrutura do tipo "X"ou "+"que é o tipo mais usado, conforme Figura 5.

Figura 5 – Configurações de montagem de um quadricópteros.

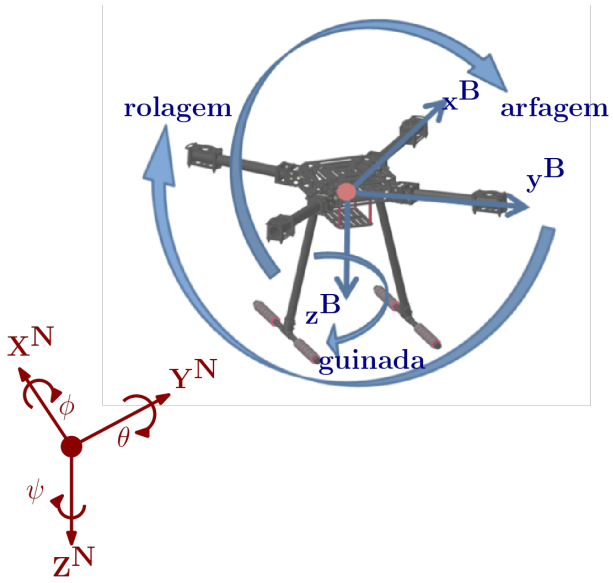


(a) Quadricóptero na configuração em "+". (b) Quadricóptero na configuração em "X".

Cada rotor consiste de uma hélice acoplada a um motor de corrente contínua do tipo *brushless* (sem escovas) e cada hélice gira em sentido contrário às vizinha. Conforme a Figura 5a, as hélices 1 e 3 giram no sentido anti-horário e as 2 e 4 no sentido horário.

Quadricópteros são veículos com seis graus de liberdade, onde 3 graus são referentes à posição (x, y e z), que medem o deslocamento do centro de massa do veículo ao longo do sistema referencial de navegação, e 3 para a atitude: ϕ ângulo de rolagem (*roll*), θ ângulo de arfagem (*pitch*) e ψ ângulo de guinada (*yaw*), como pode-se observar na Figura 6.

Figura 6 – Sistemas de referência de navegação (vermelho inercial) e do veículo (azul móvel).



De acordo com Pfeifer (2013), a força gerada em cada rotor é dada em função da velocidade angular de cada hélice e perpendicular ao plano da hélice. O momento gerado em cada rotor também pode ser escrito em função da velocidade angular da hélice e está relacionado com o empuxo da hélice. O sentido do empuxo em cada rotor é o contrário do sentido de rotação da hélice. Assim para que o veículo desloque-se, basta aumentar ou diminuir a velocidade angular dos motores.

2.1 Modelagem Dinâmica do Quadricóptero

Antes de começar o desenvolvimento do modelo matemático do quadricóptero, algumas considerações precisam ser feitas a respeito da estrutura física do veículo e da dinâmica dos atuadores, de modo a simplificar a modelagem.

* A estrutura do quadricóptero e as suas hélices são de corpos rígidos, ou seja não sofrem deformações, embora os materiais utilizados na construção do veículo não sejam rígidos, sua deformação pode ser considerada desprezível (SILVA, 2018).
* A estrutura do quadricóptero é simétrica, assim a matriz de momentos de inércia é diagonal e o momento de inércia ao redor do eixo x é igual ao do eixo y.
* Os quatros rotores, conjunto hélice e motor, são idênticos.

Para a modelagem de veículos móveis é conveniente a utilização de dois sistemas de referência: o sistema inercial de navegação (N) e o sistema móvel, solidário ao centro de gravidade do veículo (B), aos quais associam-se os sistemas de coordenadas ilustrados na Figura 6.

O sistema inercial de navegação (N) é localizado na Terra, com eixos $\boldsymbol{X}$ =

$\begin{bmatrix} X & Y & Z \end{bmatrix}^T$, onde X aponta para o Norte, Y para o Leste e Z aponta para o centro da Terra. Este sistema também é conhecido pelas iniciais NED que em inglês são *North, East e Down*.

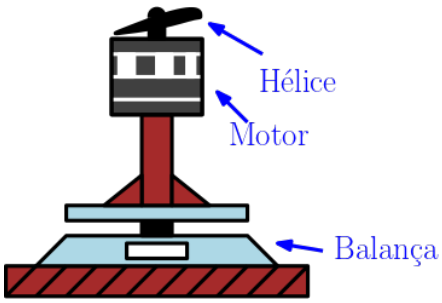
O sistema móvel (B) possui a sua origem coincidente com o centro de massa do quaricóptero e eixos $\boldsymbol{x} = \begin{bmatrix} x & y & z \end{bmatrix}^T$, onde x e y estão sobre as hastes que seguram os rotores 1 e 2, respectivamente. Por fim, z que está perpendicularmente ao plano das hastes, conforme a Figura 6.

2.1.1 Modelo dos Atuadores

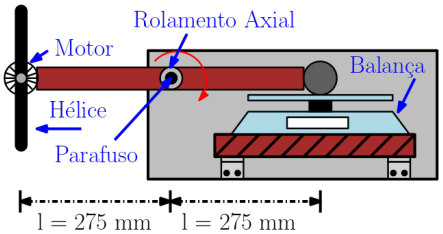
O controle do motor é realizado por meio do controle eletrônico de velocidade (ESC - *Electronic Speed Controller*), que regula a velocidade do motor, a qual está diretamente relacionada com a força e o momento gerados pela combinação da hélice e do motor. O ESC ajusta a velocidade do motor utilizando um sinal de modulação por largura de pulso (PWM - *Pulse-Width Modulation*).

Para a identificação da relação entre o conjunto motor e hélice, foi realizado um experimento utilizando uma configuração ilustrada nas Figura 7a e Figura 7b para medir a força e o momento, respectivamente. No experimento, foi variado o sinal PWM (σ), enquanto os valores de força (F) ou momento (M), velocidade (Ω), corrente de armadura (I_A), tensão de entrada (U_{in}) e tensão de saída (U_{out}) foram registrados utilizando uma balança, um multímetro e um wattímetro, capaz de fornecer todos os dados elétricos de entrada do ESC e motor.

Figura 7 – Esquemático dos experimentos para levantamento da curva do propulsor.



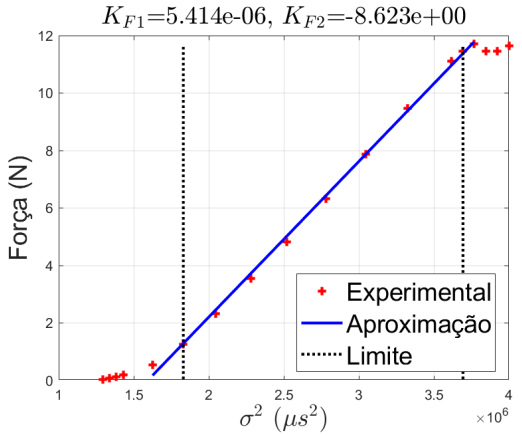
(a) Experimento para levantamento da força gerada pelo propulsor.



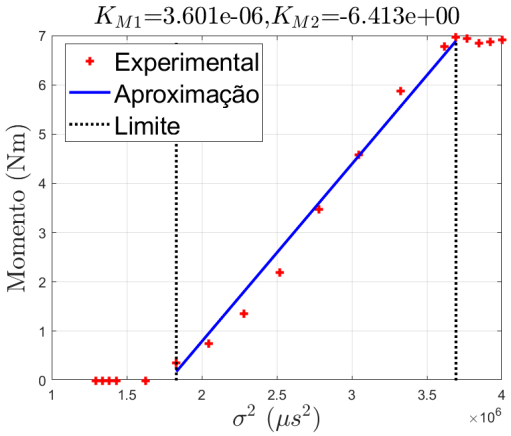
(b) Experimento para levantamento do momento gerada pelo propulsor. Comprimento do braço de torque igual ao veículo ($l = 275mm$)

Os dados e gráficos com mais detalhes podem ser visualizados no Anexo A. Com o intuito de evitar regiões de atuação extrema, como *dead-band* e a saturação, foram selecionados limites mínimo (15%) e máximo (90%) de atuação. Esses limites são representados pela linha pontilhada em preto na Figura 8.

Figura 8 – Aproximação da força e momento do propulsor.



(a) Comparação da força obtida (pontos em vermelho) e força aproximada (curva em azul).



(b) Comparação do momento obtido (pontos em vermelho) e momento aproximada (curva em azul).

As curvas em azul das Figura 8a e da Figura 8b, são representadas pelas seguintes equações,

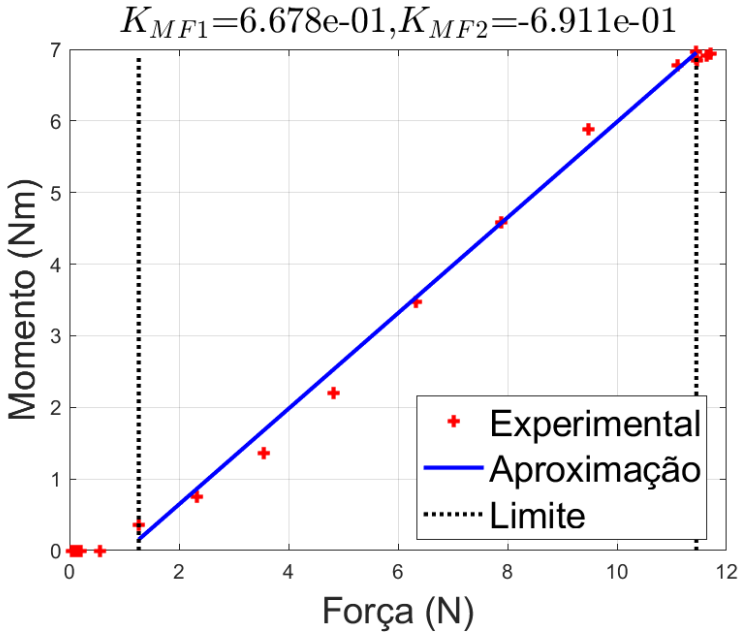
$$\begin{aligned} F_i &= K_{F1}\sigma^2 + K_{F2} \\ M_i &= K_{M1}\sigma^2 + K_{M2} \end{aligned} \tag{2.1}$$

onde K_{F1}, K_{F2}, K_{M1} e K_{M2} são constantes de ajuste da curva.

Para facilitar o desenvolvimento do controlador, na próxima seção o momento do motor foi identificado em função da força conforme a Figura 9, onde a curva em azul representa a seguinte equação,

$$M_i = F_i K_{MF1} + K_{MF2}. \tag{2.2}$$

Figura 9 – Identificação do momento do motor em função da força.



2.1.2 Modelo ESC, Motor e Hélice, potência consumida

Assumindo que o motor possui uma indutância muito baixa, podendo ser negligenciada, a tensão de armadura (U_A) do motor pode ser modelada pela Equação (2.3). A tensão foi medida experimentalmente para a obtenção da constante de força eletromotriz do motor, assim

$$\begin{aligned} U_A &= R_A I_A + K_E \Omega_i \\ K_E &= \frac{U_A - R_A I_A}{\Omega_i} \end{aligned} \tag{2.3}$$

onde I_A é a corrente de armadura, $R_A = 0,11$ ohms (resistência de armadura), Ω_i a sua velocidade de rotação e K_E constante de força eletromotriz. Assim, o valor da constante $K_E = 0,0087\ V/rad/s$.

O torque eletromecânico (M_i) é dado por:

$$M_i = K_T I_A \qquad \rightarrow \qquad K_T = \frac{M_i}{I_A} \tag{2.4}$$

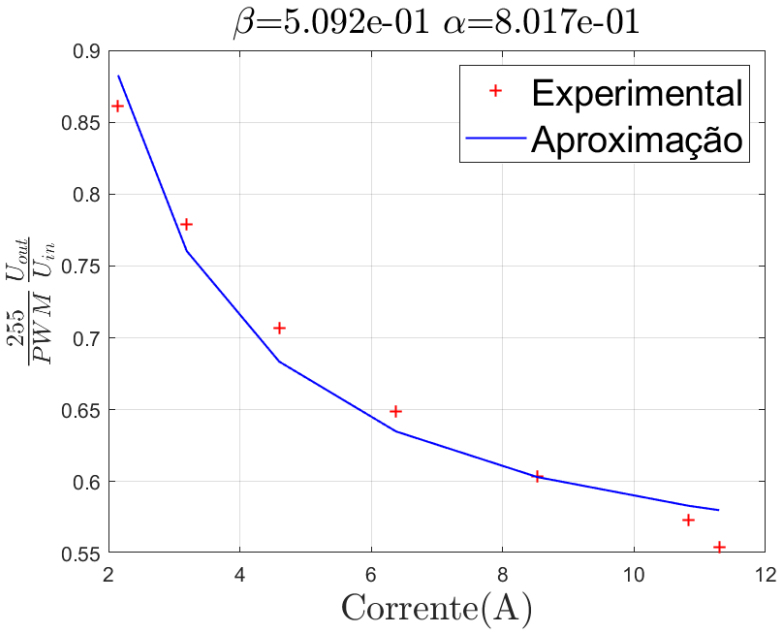
onde $K_T = 0,7403\ Nm/A$ é a constante de torque eletromotriz.

De acordo com Sendobry (2013), dispositivos ESC são baseados em circuitos do tipo MOSFET (*Metal Oxide Semiconductor Field Effect Transistor*) que dissipam muita energia. A identificação pode ser feita pela averiguação da tensão de saída (U_{out}) pela tensão de entrada (U_{in}) e medindo a corrente gerada (I), conforme Figura 10. Assim, pode-se obter a tensão de saída em função da corrente e da tensão de entrada,

$$U_{out} = U_{in} \frac{\sigma}{255} \left(\beta + \frac{\alpha}{I} \right) \tag{2.5}$$

onde o termo ($\frac{\sigma}{255}$) representa a variação da tensão de entrada no ESC, onde o valor $\sigma = 255 = 2^8$ (sinal de 8 bits) é a máxima tensão de entrada, α e β são constantes para ajustar uma curva nos postos da conforme Figura 10.

Figura 10 – Identificação do ESC, quociente da tensão de entrada pela tensão de saída.



Igualando a Equação (2.3) com a Equação (2.5) e isolando a corrente,

$$I = \frac{\beta}{2}\left(\frac{\sigma}{255}\frac{U_{in}}{R_A}\right) - \frac{K_T\Omega_i}{2R_A} + \sqrt{\left(\frac{\beta}{2}\left(\frac{\sigma}{255}\frac{U_{in}}{R_A}\right) - \frac{K_T\Omega_i}{2R_A}\right)^2 + \alpha\left(\frac{\sigma}{255}\frac{U_{in}}{R_A}\right)} \tag{2.6}$$

onde U_{in} é a tensão da bateria.

Aplicando a segunda lei de *Newton*, teorema do momento angular,

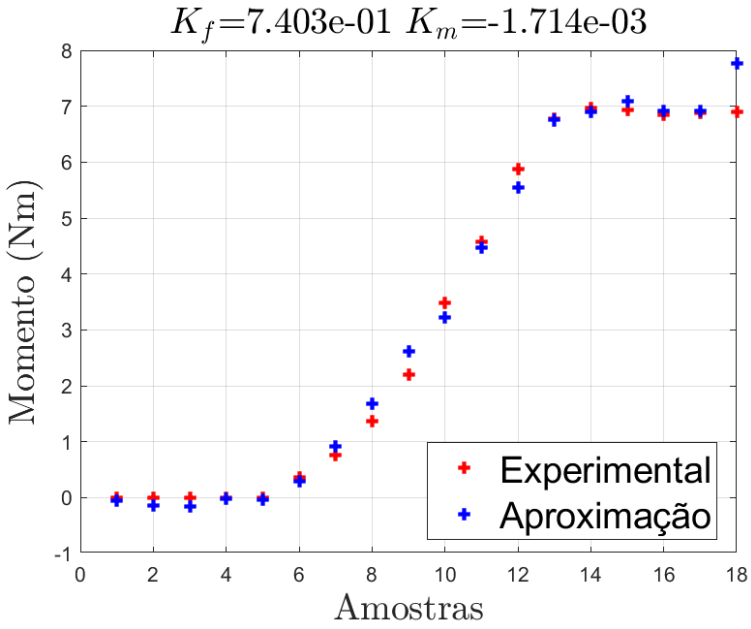
$$J_R\dot{\Omega}_i = M_i - M_l \tag{2.7}$$

onde J_R é o momento de inércia do conjunto motor mais hélice, M_l é o torque de reação da hélice que pode ser aproximado por:

$$M_l = K_f F_i + K_m\Omega_i \tag{2.8}$$

onde K_f e K_m são constantes que depende das características físicas da hélice e podem ser obtidas por meio do momento medido, pois a velocidade do motor varia pouco, podendo ser considerada como constante. A Figura 11 apresenta a curva do momento e os valores das constantes.

Figura 11 – Identificação das constantes da hélice K_f e K_m.



Assim a potência consumida em cada um dos conjuntos ESC e motor pode ser calculada,

$$P_i = U_A I_A. \tag{2.9}$$

2.1.3 Modelo Cinemático

Para a representação cinemática do veículo, foi adotada uma modelagem baseada nos ângulos de *Euler*. É importante ressaltar que os ângulos de *Euler* podem apresentar singularidades para certos valores, como $\pm 90°$. Portanto, movimentos que levem o veículo a esses valores devem ser evitados (OLIVEIRA, 2019).

Uma alternativa seria utilizar quatérnions, os quais não possuem um significado físico direto. No entanto, por questões de simplicidade, optou-se pela utilização dos ângulos de *Euler*.

A relação entre os dois sistemas de coordenadas, N (inercial de navegação) e B (corpo), é estabelecida por meio de uma matriz de rotação, a qual é obtida através de três rotações consecutivas ao longo dos eixos. Essas rotações são representadas pelos ângulos de *Euler* e a sequência de rotação pode ser realizada em qualquer ordem. Neste contexto, será considerada a sequência (X-Y-Z), na qual a primeira rotação ($\boldsymbol{R_\phi}$) ocorre em torno do eixo X e é representada pelo ângulo ϕ. Formalmente, tem-se:

$$\boldsymbol{R_\phi} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & \text{sen}(\phi) \\ 0 & -(\phi) & \cos(\phi) \end{bmatrix} \tag{2.10}$$

a segunda rotação ($\boldsymbol{R_\theta}$) entorno de Y, sendo representada pelo ângulo θ:

$$\boldsymbol{R_\theta} = \begin{bmatrix} \cos(\theta) & 0 & -\text{sen}(\theta) \\ 0 & 1 & 0 \\ \text{sen}(\theta) & 0 & \cos(\theta) \end{bmatrix} \tag{2.11}$$

e finalmente a terceira rotação ($\boldsymbol{R_\psi}$) entorno de Z é representada pelo ângulo ψ:

$$\boldsymbol{R_\psi} = \begin{bmatrix} \cos(\psi) & \text{sen}(\psi) & 0 \\ -\text{sen}(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.12}$$

Assim, o produto das três matrizes representa a matriz de rotação ($\boldsymbol{Rp_{N2B}} = \boldsymbol{R_\phi R_\theta R_\psi}$) que descreve a projeção do sistema inercial de navegação (N) no sistema móvel (B), ou seja

$$\boldsymbol{Rp_{N2B}} = \begin{bmatrix} \cos\theta\cos\psi & \cos\theta\text{sen}\psi & -\text{sen}\theta \\ \text{sen}\psi\text{sen}\theta\cos\psi - \cos\phi\text{sen}\psi & \cos\phi\cos\psi + \text{sen}\phi\text{sen}\theta\text{sen}\psi & \text{sen}\phi\cos\theta \\ \cos\phi\text{sen}\theta\cos\psi + \text{sen}\phi\text{sen}\psi & \text{sen}\theta\cos\phi\text{sen}\psi - \text{sen}\phi\cos\psi & \cos\theta\cos\phi \end{bmatrix}. \tag{2.13}$$

A matriz $\boldsymbol{Rp_{N2B}}$ é ortogonal, ou seja, $(\boldsymbol{Rp_{N2B}})^{-1} = (\boldsymbol{Rp_{N2B}})^T$. A transformada inversa (de B para N) pode ser escrita da seguinte maneira $\boldsymbol{Rp_{B2N}} = (\boldsymbol{Rp_{N2B}})^T$, assim:

$$\boldsymbol{Rp_{B2N}} = \begin{bmatrix} \cos\theta\cos\psi & \text{sen}\psi\text{sen}\theta\cos\psi - \cos\phi\text{sen}\psi & \cos\phi\text{sen}\theta\cos\psi + \text{sen}\phi\text{sen}\psi \\ \cos\theta\text{sen}\psi & \cos\phi\cos\psi + \text{sen}\phi\text{sen}\theta\text{sen}\psi & \text{sen}\theta\cos\phi\text{sen}\psi - \text{sen}\phi\cos\psi \\ -\text{sen}\theta & \text{sen}\phi\cos\theta & \cos\theta\cos\phi \end{bmatrix} \tag{2.14}$$

e portanto,

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = \boldsymbol{Rp_{B2N}} \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{2.15}$$

onde u, v e w são as velocidades lineares do veículo, ao longo dos eixos x, y e z respectivamente.

Pode-se também relacionar o vetor de velocidades angulares medido em B ($\boldsymbol{\nu_2^B} = \begin{bmatrix} p & q & r \end{bmatrix}^T$) com a taxa de variação no tempo dos ângulos de *Euler* ($\boldsymbol{\dot{p}_2^N} = \begin{bmatrix} \dot{\phi} & \dot{\theta} & \dot{\psi} \end{bmatrix}^T$). Assim, a taxa de variação dos ângulos de *Euler* pode ser projetada no sistema móvel (B). Neste caso é realizada uma transformação por vez que é propagada nas transformações seguintes, conforme segue:

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \boldsymbol{R_\phi R_\theta R_\psi} \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix} + \boldsymbol{R_\phi R_\theta} \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + \boldsymbol{R_\phi} \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} \tag{2.16}$$

colocando a Equação (2.16) na forma matricial, tem-se:

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} 1 & 0 & -\text{sen}\,(\theta) \\ 0 & \cos(\phi) & \text{sen}\,(\phi)\cos(\theta) \\ 0 & -\text{sen}\,(\phi) & \cos(\phi)\cos(\theta) \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix}. \tag{2.17}$$

Assim, a transformação inversa de B para N é dada por:

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & \text{tg}\,(\theta)\,\text{sen}\,(\phi) & \text{tg}\,(\theta)\,\text{sen}\,(\phi) \\ 0 & \cos(\phi) & -\text{sen}\,(\phi) \\ 0 & \dfrac{\text{sen}\,(\phi)}{\cos(\theta)} & \dfrac{\cos(\phi)}{\cos(\theta)} \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix}, \tag{2.18}$$

ou seja,

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \boldsymbol{Rv_{B2N}} \begin{bmatrix} p \\ q \\ r \end{bmatrix}. \tag{2.19}$$

2.1.4 Equações de Movimento

Como adotou-se o formalismo de *Newton-Euler* para descrever as equações de movimento do quadricóptero, assume-se que todas as forças e momentos externos são aplicados no centro de massa do veículo (BOUABDALLAH; SIEGWART, 2005) (CAVALLARO, 2019).

Pode-se escrever as equações de movimento translacional e rotacional do veículo no sistema móvel, representado pela letra B sobrescrito, $(\cdot)^{\boldsymbol{B}}$, por:

$$\begin{aligned} \boldsymbol{F^B} &= \boldsymbol{m\dot{\nu}_1^B} + \boldsymbol{\nu_2^B} \times \left(\boldsymbol{m\nu_1^B}\right) \\ \boldsymbol{M^B} &= \boldsymbol{I\dot{\nu}_2^B} + \boldsymbol{\nu_2^B} \times \left(\boldsymbol{I\nu_2^B}\right) \end{aligned} \tag{2.20}$$

onde $\boldsymbol{\nu_1^B} = \begin{bmatrix} u & v & w \end{bmatrix}^T$ é o vetor de velocidades lineares, $\boldsymbol{\nu_2^B} = \begin{bmatrix} p & q & r \end{bmatrix}^T$ é o vetor de velocidades angulares, $\boldsymbol{m}$ é a matriz diagonal de massa m, que é a massa total do quadricóptero, $\boldsymbol{I}$ é a matriz de inércia do veículo, que é diagonal com $I_{xx} = I_{yy}$, devido a simetria do veículo e configuração tipo +, e I_{zz} são os momentos de inércia do veículo ao longo dos eixos x, y e z respectivamente. O símbolo $\times$ representa o produto vetorial, por fim, $\boldsymbol{F^B}$ e $\boldsymbol{M^B}$ são os vetores de forças e momentos que atuam no veículo.

As forças que atuam no veículo são compostas pela força proveniente dos rotores, chamada de empuxo $\boldsymbol{F_E}$, combinação das forças individuais de cada rotor F_i (subseção 2.1.1), com as forças de arrasto ($\boldsymbol{F_D}$) do veículo, resultante da resistência do ar ao movimento do quadricóptero e da força gravitacional ($\boldsymbol{F_g}$).

O efeito giroscópio, será negligenciado pois de acordo com Oliveira (2019) este efeito é desprezível devido ao fato dos rotores adjacentes girarem em sentidos contrários.

A força gravitacional medida no sistema inercial de navegação é dada por:

$$[\boldsymbol{F_g}]^{\boldsymbol{N}} = m \begin{bmatrix} 0 & 0 & g \end{bmatrix}^T . \tag{2.21}$$

Transformando-a para o sistema móvel, tem-se:

$$[\boldsymbol{F_g}]^{\boldsymbol{B}} = m \left(\boldsymbol{Rp_{N2B}}\right) \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} = mg \begin{bmatrix} -\text{sen}\,(\theta) \\ \cos(\theta)\,\text{sen}\,(\phi) \\ \cos(\theta)\cos(\phi) \end{bmatrix} . \tag{2.22}$$

A força de arrasto pode ser escrita da seguinte maneira,

$$[\boldsymbol{F_D}]^{\boldsymbol{B}} = \frac{1}{2}\rho \boldsymbol{A}\left(\boldsymbol{C_{fD}}\right)\left(|\boldsymbol{\nu_1^B}|\boldsymbol{\nu_1^B}\right) \tag{2.23}$$

onde, ρ é a densidade do ar, $\boldsymbol{C_{fD}}$ é o coeficiente de arrasto e $\boldsymbol{A}$ é a matriz diagonal da área de referência. A área de referência é calculada associando um atrito entre o corpo do veículo com o ar. Assim tem-se três áreas constantes: A_{xz}, A_{yz} e A_{xy}, que representam a área ao longo dos eixos x, y e z, respectivamente. Devido á simetria assumida anteriormente, tem-se $A_{xz} = A_{yz}$.

Por fim, tem-se que a força de empuxo ($\boldsymbol{F_E}$) é a somatória das forças de cada um dos propulsores (F_i)

$$[\boldsymbol{F_E}]^{\boldsymbol{B}} = \begin{bmatrix} 0 \\ 0 \\ F_z \end{bmatrix} \tag{2.24}$$

onde $F_z = F_1 + F_2 + F_3 + F_4$, com F_1, F_2, F_3 e F_4 sendo as forças individuais de cada propulsor.

Os momentos atuantes no quadricóptero são os resultantes do empuxo, da reação de cada hélice ($\boldsymbol{M_E}$) e o de arrasto ($\boldsymbol{M_D}$).

O momento de arrasto é calculado da força de arrasto (Equação (2.23)), por:

$$[\boldsymbol{M_D}]^{\boldsymbol{B}} = \frac{1}{2}\rho \boldsymbol{A}\left(\boldsymbol{C_{mD}}\right)\left(|\boldsymbol{\nu_2^B}|\boldsymbol{\nu_2^B}\right) \tag{2.25}$$

onde $\boldsymbol{C_{mD}}$ é o coeficiente de arrasto.

Os momentos resultantes da força de empuxo e reação de cada hélice são dados por:

$$[\boldsymbol{M_E}]^{\boldsymbol{B}} = \begin{bmatrix} -lF_2 + lF_4 \\ lF_1 - lF_3 \\ -M_1 + M_2 - M_3 + M_4 \end{bmatrix} \tag{2.26}$$

onde l é a distância do centro de massa do veículo até o centro da hélice e M_1, M_2, M_3 e M_4 os momentos de cada rotor, dados pela Equação (2.2).

Agrupando $[\boldsymbol{F_E}]^{\boldsymbol{B}}$ e $[\boldsymbol{M_E}]^{\boldsymbol{B}}$ e substituindo a Equação (2.2), pode-se escrever a força de cada motor em função da força vertical e dos momentos de rotação como:

$$\begin{bmatrix} F_z \\ M_x \\ M_y \\ M_z \end{bmatrix} = \begin{bmatrix} F_z \\ [\boldsymbol{M_E}]^{\boldsymbol{B}} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & -l & 0 & l \\ l & 0 & -l & 0 \\ -K_{MF1} & K_{MF1} & -K_{MF1} & K_{MF1} \end{bmatrix} \begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \end{bmatrix} \tag{2.27}$$

$$\begin{aligned} F_1 &= \frac{F_z}{4} - \frac{M_z}{4K_{MF1}} + \frac{M_y}{2l} \\ F_2 &= \frac{F_z}{4} + \frac{M_z}{4K_{MF1}} - \frac{M_x}{2l} \\ F_3 &= \frac{F_z}{4} - \frac{M_z}{4K_{MF1}} - \frac{M_y}{2l} \\ F_4 &= \frac{F_z}{4} + \frac{M_z}{4K_{MF1}} + \frac{M_x}{2l}. \end{aligned} \tag{2.28}$$

Isolando os seguintes termos $\dot{\boldsymbol{\nu}}_{\mathbf{1}}^{\boldsymbol{B}}$ e $\dot{\boldsymbol{\nu}}_{\mathbf{2}}^{\boldsymbol{B}}$ da Equação (2.20) e juntando as equações cinemáticas Equação (2.15) e Equação (2.19) tem-se:

$$\dot{\boldsymbol{p}}_{\mathbf{1}}^{\boldsymbol{N}} = (\boldsymbol{Rv_{B2N}})\, \boldsymbol{\nu}_{\mathbf{1}}^{\boldsymbol{B}}, \tag{2.29a}$$

$$\dot{\boldsymbol{p}}_{\mathbf{2}}^{\boldsymbol{N}} = (\boldsymbol{Rp_{B2N}})\, \boldsymbol{\nu}_{\mathbf{2}}^{\boldsymbol{B}}, \tag{2.29b}$$

$$\dot{\boldsymbol{\nu}}_{\mathbf{1}}^{\boldsymbol{B}} = -\left(\frac{1}{m}\right)[\boldsymbol{F_E}]^{\boldsymbol{B}} + \boldsymbol{F_g} - \boldsymbol{F_D} - \boldsymbol{\nu}_{\mathbf{2}}^{\boldsymbol{B}} \times \boldsymbol{\nu}_{\mathbf{1}}^{\boldsymbol{B}}, \tag{2.29c}$$

$$\dot{\boldsymbol{\nu}}_{\mathbf{2}}^{\boldsymbol{B}} = \boldsymbol{I}^{-1}\left[[\boldsymbol{M_E}]^{\boldsymbol{B}} - \boldsymbol{M_D} - \boldsymbol{\nu}_{\mathbf{2}}^{\boldsymbol{B}} \times \boldsymbol{I}\boldsymbol{\nu}_{\mathbf{2}}^{\boldsymbol{B}}\right]. \tag{2.29d}$$

Para o desenvolvimento do controle, a Equação (2.29) será reescrita com todas as variáveis no sistema inercial ($\boldsymbol{N}$). Assim, chamando $\dot{\boldsymbol{p}}_{\mathbf{1}}^{\boldsymbol{N}} = \boldsymbol{\nu}_{\mathbf{1}}^{\boldsymbol{N}} = \begin{bmatrix} \dot{x} & \dot{y} & \dot{z} \end{bmatrix}^T$ e $\dot{\boldsymbol{p}}_{\mathbf{2}}^{\boldsymbol{N}} = \boldsymbol{\nu}_{\mathbf{2}}^{\boldsymbol{N}} = \begin{bmatrix} \dot{\phi} & \dot{\theta} & \dot{\psi} \end{bmatrix}^T$. Calculando a derivada da Equação (2.29a) e da Equação (2.29b), obtêm-se:

$$\begin{aligned} \ddot{\boldsymbol{p}}_{\mathbf{1}}^{\boldsymbol{N}} &= \left(\dot{\boldsymbol{R}}\boldsymbol{p_{B2N}}\right)\boldsymbol{\nu}_{\mathbf{1}}^{\boldsymbol{B}} + (\boldsymbol{Rp_{B2N}})\,\dot{\boldsymbol{\nu}}_{\mathbf{1}}^{\boldsymbol{B}}, \\ \ddot{\boldsymbol{p}}_{\mathbf{2}}^{\boldsymbol{N}} &= \left(\dot{\boldsymbol{R}}\boldsymbol{v_{B2N}}\right)\boldsymbol{\nu}_{\mathbf{2}}^{\boldsymbol{B}} + (\boldsymbol{Rv_{B2N}})\,\dot{\boldsymbol{\nu}}_{\mathbf{2}}^{\boldsymbol{B}}, \end{aligned} \tag{2.30}$$

substituindo a Equação (2.29c) e a Equação (2.29d) na Equação (2.30) e lembrando que:

$$\begin{aligned} \boldsymbol{\nu}_{\mathbf{1}}^{\boldsymbol{B}} &= (\boldsymbol{Rp_{N2B}})\,\dot{\boldsymbol{p}}_{\mathbf{1}}^{\boldsymbol{N}} = (\boldsymbol{Rp_{N2B}})\,\boldsymbol{\nu}_{\mathbf{1}}^{\boldsymbol{N}}, \\ \boldsymbol{\nu}_{\mathbf{2}}^{\boldsymbol{B}} &= (\boldsymbol{Rv_{N2B}})\,\dot{\boldsymbol{p}}_{\mathbf{2}}^{\boldsymbol{N}} = (\boldsymbol{Rv_{N2B}})\,\boldsymbol{\nu}_{\mathbf{2}}^{\boldsymbol{N}}, \end{aligned}$$

assim, chega-se as equações de movimento do veículo no sistema de navegação

$$\begin{aligned} \dot{\boldsymbol{p}}_{\mathbf{1}}^{\boldsymbol{N}} &= \boldsymbol{\nu}_{\mathbf{1}}^{\boldsymbol{N}}, \\ \dot{\boldsymbol{p}}_{\mathbf{2}}^{\boldsymbol{N}} &= \boldsymbol{\nu}_{\mathbf{2}}^{\boldsymbol{N}}, \\ \dot{\boldsymbol{\nu}}_{\mathbf{1}}^{\boldsymbol{N}} &= \boldsymbol{D_1}\boldsymbol{\nu}_{\mathbf{1}}^{\boldsymbol{N}} + \boldsymbol{C_1} - (\boldsymbol{Rp_{B2N}})\left(\left(\frac{1}{m}\right)[\boldsymbol{F_E}]^{\boldsymbol{B}} - \boldsymbol{F_g} + \boldsymbol{F_D}\right), \\ \dot{\boldsymbol{\nu}}_{\mathbf{2}}^{\boldsymbol{N}} &= \boldsymbol{D_2}\boldsymbol{\nu}_{\mathbf{2}}^{\boldsymbol{N}} - \boldsymbol{C_2} + (\boldsymbol{Rv_{B2N}})\,\boldsymbol{I}^{-1}\left([\boldsymbol{M_E}]^{\boldsymbol{B}} - \boldsymbol{M_D}\right), \end{aligned} \tag{2.31}$$

onde

$$\boldsymbol{D_1} = \left(\dot{\boldsymbol{R}}\boldsymbol{p_{B2N}}\right)(\boldsymbol{Rp_{N2B}}),$$
$$\boldsymbol{D_2} = \left(\dot{\boldsymbol{R}}\boldsymbol{v_{B2N}}\right)(\boldsymbol{Rv_{N2B}}),$$
$$\boldsymbol{C_1} = (\boldsymbol{Rp_{B2N}})\left[-(\boldsymbol{Rv_{N2B}})\,\boldsymbol{\nu_2^N} \times (\boldsymbol{Rp_{N2B}})\,\boldsymbol{\nu_1^N}\right] \text{ e}$$
$$\boldsymbol{C_2} = (\boldsymbol{Rv_{B2N}})\,\boldsymbol{I}^{-1}\left[(\boldsymbol{Rv_{N2B}})\,\boldsymbol{\nu_2^N} \times \boldsymbol{I}\,(\boldsymbol{Rp_{N2B}})\,\boldsymbol{\nu_1^N}\right].$$

2.2 Controle *backstepping* com controle de modos deslizantes

O controle *backstepping* é um controle recursivo que constrói leis de controle por meio de realimentações, projetadas por meio de uma função de *Lyapunov*. É similar à linearização por realimentação de estados (*feedback-linearization*), exceto que neste caso somente as linearidades instáveis são canceladas, as demais são mantidas (KOKOTOVIC; ARCAK, 2001). Para que o controlador funcione corretamente é necessário ter um conhecimento prévio do modelo e dos parâmetros, para que possam ser utilizados na realimentação dos estados. Por isso, decidiu-se juntar um controle de modos deslizantes, como uma maneira de acrescentar robustez ao controlador.

Dividiu-se o projeto do controlador em duas partes: controle de posição e controle de atitude. Esta abordagem é muito utilizada em veículos do tipo quadricóptero subatuados.

A ideia principal é fazer com que a força vertical (F_z) dos rotores sejam decompostas nas direções de subatuação.

Para facilitar o desenvolvimento do controle, o modelo matemático do veículo (Equação (2.31)), foi divido em dois subsistemas: subsistema de posição e subsistema de atitude. O subsistema de posição foi escrito da seguinte maneira:

$$\dot{\boldsymbol{p}}_{\boldsymbol{1}}^{\boldsymbol{N}} = \boldsymbol{\nu_1^N}, \tag{2.32a}$$
$$\dot{\boldsymbol{\nu}}_{\boldsymbol{1}}^{\boldsymbol{N}} = \boldsymbol{\Delta f} + \boldsymbol{F_f}, \tag{2.32b}$$

onde $\boldsymbol{p_1^N} = \begin{bmatrix} x & y & z \end{bmatrix}^T$, $\boldsymbol{\nu_1^N} = \begin{bmatrix} \dot{x} & \dot{y} & \dot{z} \end{bmatrix}^T$, $\boldsymbol{F_f} = -(\boldsymbol{Rp_{B2N}})\left(\frac{1}{m}\right)[\boldsymbol{F_E}]^{\boldsymbol{B}} = \begin{bmatrix} F_{vx} & F_{vy} & F_{vz} \end{bmatrix}^T$ e $\boldsymbol{\Delta f} = \left(\dot{\boldsymbol{R}}\boldsymbol{p_{B2N}}\right)(\boldsymbol{Rp_{N2B}})\,\boldsymbol{\nu_1^N} + (\boldsymbol{Rp_{B2N}})\left[\boldsymbol{F_g} - \boldsymbol{F_D} - (\boldsymbol{Rv_{N2B}})\,\boldsymbol{\nu_2^N} \times (\boldsymbol{Rp_{N2B}})\,\boldsymbol{\nu_1^N}\right]$.

O subsistema de atitude foi escrito da seguinte maneira,

$$\dot{\boldsymbol{p}}_{\boldsymbol{2}}^{\boldsymbol{N}} = \boldsymbol{\nu_2^N}, \tag{2.33a}$$
$$\dot{\boldsymbol{\nu}}_{\boldsymbol{2}}^{\boldsymbol{N}} = \boldsymbol{\Delta m} + \boldsymbol{BM_m}, \tag{2.33b}$$

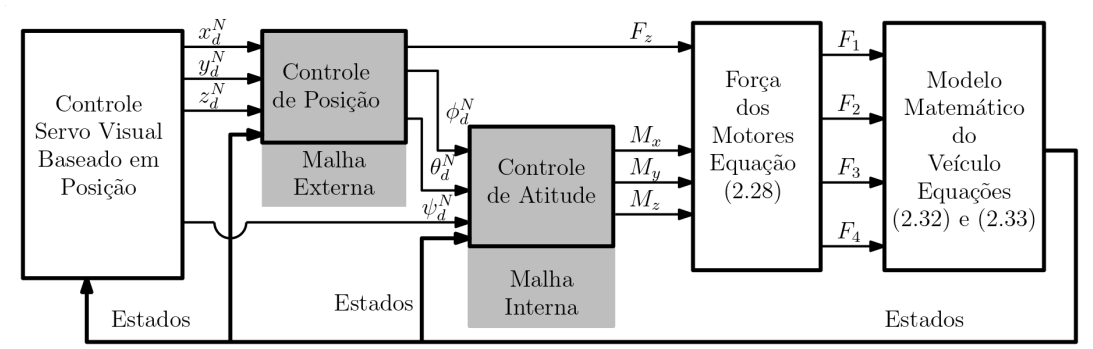
onde

$$\boldsymbol{p_2^N} = \begin{bmatrix} \phi & \theta & \psi \end{bmatrix}^T, \boldsymbol{\nu_2^N} = \begin{bmatrix} \dot{\phi} & \dot{\theta} & \dot{\psi} \end{bmatrix}^T,$$
$$\boldsymbol{BM_m} = (\boldsymbol{Rv_{B2N}})\,\boldsymbol{I}^{-1}\,[\boldsymbol{M_E}]^{\boldsymbol{B}} \text{ e}$$

$$\boldsymbol{\Delta m} = \left(\dot{\boldsymbol{Rv}}_{\boldsymbol{B2N}}\right)(\boldsymbol{Rv}_{\boldsymbol{N2B}})\,\boldsymbol{\nu}_{\boldsymbol{2}}^{\boldsymbol{N}} - (\boldsymbol{Rv}_{\boldsymbol{B2N}})\,\boldsymbol{I}^{-1}\left[\boldsymbol{M_D} + (\boldsymbol{Rv}_{\boldsymbol{N2B}})\,\boldsymbol{\nu}_{\boldsymbol{2}}^{\boldsymbol{N}} \times \boldsymbol{I}\,(\boldsymbol{Rp}_{\boldsymbol{N2B}})\,\boldsymbol{\nu}_{\boldsymbol{1}}^{\boldsymbol{N}}\right].$$

A interação entre os subsistemas e os controle de posição e de atitude, pode ser observada na Figura 12. Um diagrama de blocos somente sobre o controle desenvolvido, somente os blocos em cinza da Figura 12, apresenta-se no Apêndice C.

Figura 12 – Arquitetura do controle de posição (malha externa), controle de atitude (malha interna) e o controle servo visual baseado em posição.



2.2.1 Desenvolvimento da superfície de chaveamento

A superfície de chaveamento foi escolhida de acordo com a formulação desenvolvida por Slotine e Li (1991), que pode ser descrita como um ganho constante (λ_1, λ_2 e λ_3) multiplicado por um vetor de erro $\boldsymbol{Z_2}$,

$$\boldsymbol{s} = \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{bmatrix} \boldsymbol{Z_2} = \boldsymbol{C Z_2} \tag{2.34a}$$

$$\boldsymbol{Z_2} = \boldsymbol{\nu_1^N} - \boldsymbol{\alpha} \tag{2.34b}$$

onde $\boldsymbol{\alpha}$ é o vetor de controle virtual, definido na subseção 2.2.2 e as constantes $\lambda_{1,2,3}$ são obtidas por meio da seguinte relação:

$$\lambda_{1,2,3} < \frac{1}{5} T_s \tag{2.35}$$

onde T_s é a taxa de amostragem do controlador.

Adotou-se o mesmo modelo utilizado por Cardoso et al. (2017), para a parte descontínua do controle ($\boldsymbol{SMC}$), ou seja:

$$\boldsymbol{SMC} = -\frac{\boldsymbol{C}^T}{\epsilon + |\boldsymbol{C}^T \boldsymbol{s}|}\boldsymbol{s} = -\boldsymbol{G s} \tag{2.36}$$

onde ϵ é chamado de camada limite, utilizado para suavizar o chaveamento do controle (UTKIN; GULDNER; SHIJUN, 1999).

2.2.2 Controle de Posição

O primeiro passo é definir um erro de posição ($\boldsymbol{Z_1}$) com uma parcela integrativa, para forçar que o erro em estado estacionário convirja para zero. Formalmente:

$$\boldsymbol{Z_1} = \boldsymbol{e_1} + \boldsymbol{\gamma} \int_0^t (\boldsymbol{e_1})\, dt \tag{2.37}$$

onde $\boldsymbol{e_1} = \boldsymbol{p_1^N} - \boldsymbol{p_{1d}^N}$, com $\boldsymbol{p_{1d}^N}$ sendo a posição desejada, t é o tempo de simulação e $\boldsymbol{\gamma}$ é uma matriz diagonal, positiva e constante.

Definindo o erro de velocidade como $\boldsymbol{e_2} = \boldsymbol{\nu_1^N} - \boldsymbol{\nu_{1d}^N}$, com $\boldsymbol{\nu_{1d}^N}$ sendo a velocidade desejada, e combinando com a Equação (2.34b) (isolando $\boldsymbol{\nu_1^N}$), tem-se:

$$\boldsymbol{e_2} = \boldsymbol{Z_2} + \boldsymbol{\alpha} - \boldsymbol{\nu_{1d}^N}. \tag{2.38}$$

Calculando a derivada da Equação (2.37) e substituindo a Equação (2.38), tem-se:

$$\begin{aligned} \dot{\boldsymbol{Z}}_1 &= \boldsymbol{e_2} + \boldsymbol{\gamma e_1} \\ \dot{\boldsymbol{Z}}_1 &= \boldsymbol{Z_2} + \boldsymbol{\alpha} - \boldsymbol{\nu_{1d}^N} + \gamma \boldsymbol{e_1}. \end{aligned} \tag{2.39}$$

Definidos os vetores de erros, o próximo passo é definir uma função candidata de *Lyapunov*. A função escolhida foi a mesma utilizada por Cardoso et al. (2017), qual seja:

$$\boldsymbol{V_1} = \left(\frac{1}{2}\right) \boldsymbol{Z_1}^T \boldsymbol{Z_1}. \tag{2.40}$$

Para que a função escolhida seja uma função de *Lyapunov* do sistema é necessário garantir que a sua derivada seja negativa definida, assim:

$$\dot{\boldsymbol{V}}_1 = \boldsymbol{Z_1}^T \left(\boldsymbol{Z_2} + \boldsymbol{\alpha} - \boldsymbol{\nu_{1d}^N} + \boldsymbol{\gamma e_1}\right) \tag{2.41}$$

onde o controle virtual $\boldsymbol{\alpha}$ é responsável por levar o erro $\boldsymbol{Z_1}$ para zero e fazer com que a Equação (2.41) seja negativa definida. Escolhe-se $\boldsymbol{\alpha}$ como segue:

$$\boldsymbol{\alpha} = -\boldsymbol{K Z_1} + \boldsymbol{\nu_{1d}^N} - \boldsymbol{\gamma e_1} \tag{2.42}$$

onde $\boldsymbol{K}$ é uma matriz diagonal, positiva e constante.

A escolha de $\boldsymbol{\alpha}$ não foi suficiente para garantir que a derivada da função de *Lyapunov*

$$\dot{\boldsymbol{V}}_1 = -\boldsymbol{Z_1}^T \boldsymbol{K Z_1} + \boldsymbol{Z_1}^T \boldsymbol{Z_2} \tag{2.43}$$

fosse negativa definida, já que o primeiro termo da Equação (2.43) é negativo semi-definido, pois este termo é nulo na origem ($\boldsymbol{Z_1} = \boldsymbol{0}$). O segundo termo não pode ser determinado, assim este termo deve ser eliminado nos passos seguintes, para que Equação (2.43) torne-se, ao menos, negativa semi-definida.

Substituindo a Equação (2.42) na Equação (2.34b) e calculando sua derivada

$$\dot{\boldsymbol{Z_2}} = \dot{\boldsymbol{\nu}}_{\boldsymbol{1}}^{\boldsymbol{N}} + \boldsymbol{K}\dot{\boldsymbol{Z_1}} - \dot{\boldsymbol{\nu}}_{\boldsymbol{1d}}^{\boldsymbol{N}} + \boldsymbol{\gamma}\dot{\boldsymbol{e}}_{\boldsymbol{1}}. \tag{2.44}$$

Define-se então uma segunda função candidata de *Lyapunov*. A função escolhida foi adaptada do trabalho Qi et al. (2019), assim:

$$\boldsymbol{V_2} = \boldsymbol{V_1} + \left(\frac{1}{2}\right) \boldsymbol{Z_2}^T \boldsymbol{Z_2} + \left(\frac{1}{2}\right) \tilde{\boldsymbol{\Delta}}^T \boldsymbol{\Gamma}^{-1} \tilde{\boldsymbol{\Delta}} \tag{2.45}$$

onde $\boldsymbol{\Gamma}$ é uma matriz diagonal, positiva e constante e $\tilde{\boldsymbol{\Delta}} = \hat{\boldsymbol{\Delta}} - \boldsymbol{\Delta f}$ é a dinâmica do estimador de erro e $\hat{\boldsymbol{\Delta}}$ sendo o valor estimado de $\boldsymbol{\Delta f}$.

Como o veículo possui muitos parâmetros difíceis de serem obtidos, como os parâmetros aerodinâmicos referentes ao arrasto, optou-se por acrescentar esta parcela de estimação no controle para garantir robustez à presença de parâmetros desconhecidos ou não modelados.

A derivada da Equação (2.45), pode ser escrita como:

$$\dot{\boldsymbol{V_2}} = -\boldsymbol{Z_1}^T \boldsymbol{K} \boldsymbol{Z_1} + \boldsymbol{Z_1}^T \boldsymbol{Z_2} + \boldsymbol{Z_2}^T \dot{\boldsymbol{Z_2}} + \tilde{\boldsymbol{\Delta}}^T \boldsymbol{\Gamma}^{-1} \dot{\hat{\boldsymbol{\Delta}}} \tag{2.46}$$

supondo que a variação dos parâmetros em $\boldsymbol{\Delta}$ seja muito pequena, podendo ser considerados constantes assim: $\dot{\tilde{\boldsymbol{\Delta}}} = \dot{\hat{\boldsymbol{\Delta}}}$.

O primeiro termo da Equação (2.45) é negativo semi-definido, conforme visto anteriormente, assim é necessário garantir que o resto equação também seja negativo semi-definido.

A estratégia deste controle é assumir que a derivada do erro $\boldsymbol{Z_2}$ é igual ao controle descontínuo, que foi desenvolvido na subseção 2.2.1, assim:

$$\begin{aligned} \boldsymbol{SMC} &= \dot{\boldsymbol{Z_2}} \\ -\boldsymbol{Gs} &= \dot{\boldsymbol{\nu}}_{\boldsymbol{1}}^{\boldsymbol{N}} + \boldsymbol{K}\dot{\boldsymbol{Z_1}} - \dot{\boldsymbol{\nu}}_{\boldsymbol{1d}}^{\boldsymbol{N}} + \boldsymbol{\gamma}\dot{\boldsymbol{e}}_{\boldsymbol{1}}. \end{aligned} \tag{2.47}$$

Substituindo a Equação (2.32b) na Equação (2.47), renomeando a variável de controle $\boldsymbol{F_f}$ como $\hat{\boldsymbol{F}}_{\boldsymbol{1}}$ e isolando-a, tem-se:

$$\hat{\boldsymbol{F}}_{\boldsymbol{1}} = -\boldsymbol{Gs} - \hat{\boldsymbol{\Delta}} - \boldsymbol{K}\dot{\boldsymbol{Z_1}} + \dot{\boldsymbol{\nu}}_{\boldsymbol{1d}}^{\boldsymbol{N}} - \boldsymbol{\gamma}\dot{\boldsymbol{e}}_{\boldsymbol{1}}. \tag{2.48}$$

A variável de controle foi renomeada pois não representa mais o controle final, somente uma parcela do sinal de entrada final. Esta parcela satisfaz apenas a condição imposta pela Equação (2.47). Para obter a lei de controle é necessário completar o desenvolvimento do controle *backstepping* e portanto o próximo passo é garantir que a Equação (2.46) seja negativa semi-definida.

Assumindo que $\boldsymbol{F_f} = \hat{\boldsymbol{F}}_{\mathbf{1}} + \hat{\boldsymbol{F}}_{\mathbf{2}}$ e substituindo a Equação (2.32b), Equação (2.44) e a Equação (2.48) na Equação (2.44), vem:

$$\begin{array}{rcl} \dot{\boldsymbol{V}}_{\mathbf{2}} & = & -\boldsymbol{Z_1}^T \boldsymbol{K} \boldsymbol{Z_1} + \boldsymbol{Z_1}^T \boldsymbol{Z_2} + \boldsymbol{Z_2}^T \left[-\tilde{\boldsymbol{\Delta}} - \boldsymbol{Gs} + \hat{\boldsymbol{F}}_{\mathbf{2}} \right] + \ldots \\ \ldots & + & \tilde{\boldsymbol{\Delta}}^T \boldsymbol{\Gamma}^{-1} \dot{\hat{\boldsymbol{\Delta}}} \end{array} \tag{2.49}$$

onde

$$\dot{\hat{\boldsymbol{\Delta}}} = \boldsymbol{\Gamma} \boldsymbol{Z_2}. \tag{2.50}$$

Assim a Equação (2.49), fica:

$$\dot{\boldsymbol{V}}_{\mathbf{2}} = -\boldsymbol{Z_1}^T \boldsymbol{K} \boldsymbol{Z_1} + \boldsymbol{Z_1}^T \boldsymbol{Z_2} - \boldsymbol{Z_2}^T \boldsymbol{C} \boldsymbol{G} \boldsymbol{Z_2} + \boldsymbol{Z_2}^T \hat{\boldsymbol{F}}_{\mathbf{2}}. \tag{2.51}$$

Para que a Equação (2.51) seja negativa semi-definida é necessário escolher $\hat{\boldsymbol{F}}_{\mathbf{2}} = -\boldsymbol{Z_1}$, assim a Equação (2.51) pode ser escrita da seguinte maneira:

$$\dot{\boldsymbol{V}}_{\mathbf{2}} = -\boldsymbol{Z_1}^T \boldsymbol{K} \boldsymbol{Z_1} - \boldsymbol{Z_2}^T \boldsymbol{C} \boldsymbol{G} \boldsymbol{Z_2}, \tag{2.52}$$

onde as matrizes $\boldsymbol{G}$, $\boldsymbol{C}$ e $\boldsymbol{K}$ são constantes e positivas e portanto, a derivada da segunda função *Lyapunov* (Equação (2.52)) é negativa semi-definida.

A entrada de controle final, pode ser escrita da seguinte maneira:

$$\begin{array}{rcl} \boldsymbol{F_f} & = & -\boldsymbol{Gs} - \hat{\boldsymbol{\Delta}} - \boldsymbol{K} \dot{\boldsymbol{Z}}_{\mathbf{1}} + \dot{\boldsymbol{\nu}}_{\mathbf{1d}}^{\boldsymbol{N}} - \boldsymbol{\gamma} \dot{\boldsymbol{e}}_{\mathbf{1}} - \boldsymbol{Z}_{\mathbf{1}} \\ \boldsymbol{F_f} & = & \begin{bmatrix} F_{vx} & F_{vy} & F_{vz} \end{bmatrix}^T. \end{array} \tag{2.53}$$

De maneira simplificada podem-se escrever as três equações mais importantes, Equação (2.39), Equação (2.44) e Equação (2.50), matricialmente:

$$\begin{array}{rcl} \begin{bmatrix} \dot{\boldsymbol{Z}}_{\mathbf{1}} \\ \dot{\boldsymbol{Z}}_{\mathbf{2}} \end{bmatrix} & = & \begin{bmatrix} -\boldsymbol{K} & \mathbf{1} \\ -\mathbf{1} & -\boldsymbol{GC} \end{bmatrix} \begin{bmatrix} \boldsymbol{Z}_{\mathbf{1}} \\ \boldsymbol{Z}_{\mathbf{2}} \end{bmatrix} + \begin{bmatrix} \mathbf{0} \\ -\mathbf{1} \end{bmatrix} \tilde{\boldsymbol{\Delta}} \\ \dot{\hat{\boldsymbol{\Delta}}} & = & \begin{bmatrix} \mathbf{0} & \boldsymbol{\Gamma} \end{bmatrix} \begin{bmatrix} \boldsymbol{Z}_{\mathbf{1}} \\ \boldsymbol{Z}_{\mathbf{2}} \end{bmatrix}. \end{array} \tag{2.54}$$

O passo seguinte é calcular a força F_z e os ângulos de *Euler* desejados (ϕ_d, θ_d), mas primeiro analisando a força $\boldsymbol{F_f}$ encontrada com o modelo do subsistema de posição Equação (2.32b), tem-se:

$$\begin{array}{rcl} \boldsymbol{F_f} & = & -\left(\dfrac{1}{m} \right) \left(\boldsymbol{R} \boldsymbol{p}_{\boldsymbol{B2N}} \right) \left[\boldsymbol{F_E} \right]^{\boldsymbol{B}} \\ \begin{bmatrix} F_{vx} \\ F_{vy} \\ F_{vz} \end{bmatrix} & = & \begin{bmatrix} -\left(\operatorname{sen}\left(\phi\right) \operatorname{sen}\left(\psi\right) + \cos\left(\phi\right) \cos\left(\psi\right) \operatorname{sen}\left(\theta\right) \right) \\ \cos\left(\psi\right) \operatorname{sen}\left(\phi\right) - \cos\left(\phi\right) \operatorname{sen}\left(\psi\right) \operatorname{sen}\left(\theta\right) \\ -\cos\left(\phi\right) \cos\left(\theta\right) \end{bmatrix} \dfrac{F_z}{m} \\ \boldsymbol{F_f} & = & \boldsymbol{MV} \left(\dfrac{F_z}{m} \right). \end{array} \tag{2.55}$$

Calculando a pseudo-inversa de *Moore-Penrose*, $\boldsymbol{MV}^{+} = \boldsymbol{MV}^{T}\left(\boldsymbol{MVMV}^{T}\right)^{-1}$, obtém-se a força F_z da expressão acima:

$$F_z = \boldsymbol{MV}^{T}\left(\boldsymbol{MVMV}^{T}\right)^{-1}\boldsymbol{F_f}m. \tag{2.56}$$

Já os os ângulos de *Euler* desejados (ϕ_d, θ_d), são obtidos da Equação (2.55), isto é:

$$\begin{bmatrix} F_{vx} \\ F_{vy} \\ F_{vz} \end{bmatrix} = \begin{bmatrix} \text{sen}\,(\psi) & \cos(\psi) & 0 \\ -\cos(\psi) & \text{sen}\,(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -\text{sen}\,(\phi) \\ -\cos(\phi)\,\text{sen}\,(\theta) \\ -\cos(\phi)\cos(\theta) \end{bmatrix} \frac{F_z}{m} \tag{2.57}$$

multiplicando os dois lados da Equação (2.57) pela inversa da matriz central, tem-se:

$$\begin{aligned} F_{vx}\text{sen}\,(\psi) - F_{vy}\cos(\psi) &= -\frac{F_z}{m}\text{sen}\,(\phi) \\ F_{vx}\cos(\psi) + F_{vy}\text{sen}\,(\psi) &= -\frac{F_z}{m}\cos(\phi)\,sen\,(\theta) \\ F_{vz} &= -\frac{F_z}{m}\cos(\phi)\cos(\theta) \end{aligned} \tag{2.58}$$

por fim, calculam-se os ângulos desejados:

$$\begin{aligned} \phi_d &= arcsen\left(m\frac{F_{vy}\cos(\psi) - F_{vx}\text{sen}(\psi)}{F_z}\right) \\ \theta_d &= arctg\left(\frac{F_{vx}\cos(\psi) + F_{vy}\text{sen}(\psi)}{F_{vz}}\right). \end{aligned} \tag{2.59}$$

2.2.3 Controle de Atitude

Para o desenvolvimento do controle de atitude foram utilizados os mesmos passos do controle de posição. Os erros do controle de atitude podem ser descritos como: $\boldsymbol{e_3} = \boldsymbol{p_2^N} - \boldsymbol{p_{2d}^N}$ e $\boldsymbol{e_4} = \boldsymbol{\nu_2^N} - \boldsymbol{\nu_{2d}^N}$, onde $\boldsymbol{p_{2d}^N} = \begin{bmatrix}\phi_d & \theta_d & \psi_d\end{bmatrix}^T$ e $\boldsymbol{\nu_{2d}^N} = \begin{bmatrix}\dot{\phi}_d & \dot{\theta}_d & \dot{\psi}_d\end{bmatrix}^T$, que são as variáveis de estado desejadas. Portanto o controle de atitude pode ser sumarizado, por:

$$\begin{aligned} \begin{bmatrix} \dot{\boldsymbol{Z}}_{\boldsymbol{3}} \\ \dot{\boldsymbol{Z}}_{\boldsymbol{4}} \end{bmatrix} &= \begin{bmatrix} -\boldsymbol{K_A} & \boldsymbol{1} \\ \boldsymbol{-1} & -\boldsymbol{G_A C} \end{bmatrix} \begin{bmatrix} \boldsymbol{Z_3} \\ \boldsymbol{Z_4} \end{bmatrix} + \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{-1} \end{bmatrix} \tilde{\boldsymbol{\Delta}}_{\boldsymbol{A}} \\ \dot{\tilde{\boldsymbol{\Delta}}}_{\boldsymbol{A}} &= \begin{bmatrix} \boldsymbol{0} & \boldsymbol{\Gamma} \end{bmatrix} \begin{bmatrix} \boldsymbol{Z_3} \\ \boldsymbol{Z_4} \end{bmatrix} \\ \boldsymbol{M_m} &= -\boldsymbol{B}^{-1}\left[\boldsymbol{G_A s_A} + \hat{\boldsymbol{\Delta}}_{\boldsymbol{A}} + \boldsymbol{K_A}\dot{\boldsymbol{Z}}_{\boldsymbol{3}} - \dot{\boldsymbol{\nu}}_{\boldsymbol{2d}}^{\boldsymbol{N}} + \boldsymbol{\gamma_A}\dot{\boldsymbol{e}}_{\boldsymbol{3}} + \boldsymbol{Z_3}\right] \\ -\boldsymbol{G_A s_A} &= -\left(\frac{\boldsymbol{B}^T\boldsymbol{C}^T}{\epsilon + |\boldsymbol{B}^T\boldsymbol{C}^T\boldsymbol{s_A}|}\right)(\boldsymbol{CZ_4}). \end{aligned} \tag{2.60}$$

De acordo com Almakhles (2020), o controle de atitude é mais importante que o controle de posição, pois a atitude é responsável por realizar a decomposição da força vertical (F_z) nas direções em que o veículo não possui atuação. Portanto, o veículo deve convergir à atitude desejada antes, e assim, os ganhos do controle de atitude foram escolhidos para fazer com que este controle seja mais rápido que o controle de posição.

2.3 Obtenção dos parâmetros do controlador

Conforme descrito na seção anterior, seção 2.2, o controle proposto possui um grande número de parâmetros que precisam ser definidos e não possuem nenhuma regra específica para determiná-los, somente precisam ser positivos definidos para que o sistema seja estável. Os ganhos a serem definidos são 4 matrizes diagonais, positivas e constantes ($\boldsymbol{K}$, $\boldsymbol{K_A}$, $\boldsymbol{\gamma}$, $\boldsymbol{\gamma_A}$).

Para o ajuste dos parâmetros utilizou-se o algoritmo genético (GA - *Genetics Algorithms*), pois este tipo de algoritmo possui boa capacidade e adaptabilidade devido à sua natureza aleatória como método de otimização, e uma melhor relação desempenho versus velocidade (RODRÍGUEZ-ABREO et al., 2020).

O tamanho da população fornece diversidade genética, portanto, um grande número é desejado, mas se for muito grande, o custo computacional e o tempo de execução do algoritmo também aumentam (RODRÍGUEZ-ABREO et al., 2020). Os parâmetros utilizados no GA podem ser observados na Tabela 1.

Tabela 1 – Parâmetros do algoritmo genético.

Parâmetro	Valor
Tamanho da população	300
Número máximo de gerações	600
Probabilidade de mutação	20%
Intervalo inicial da população	[0;30]
Intervalo de valores para cada indivíduo	[0;50]

A função custo escolhida levou em consideração o erro da trajetória por meio das variáveis de posição, velocidade, atitude e velocidade angular do veículo e o esforço do controle, Fz, Mx, My e Mz.

O esforço de controle foi monitorado de maneira a evitar que apresente um chaveamento no sinal de saída, pois na presença de erro na trajetória do veículo, por exemplo erro de condição inicial, o controlador tenta corrigir esta diferença o mais rápido possível, fazendo com o que o sinal de saída controlador apresente chaveamento. A intensidade deste chaveamento está relacionado ao valor dos parâmetros do controlador, pois quanto maiores forem mais rápido será a convergência e maior será o chaveamento. O ideal é encontrar um valor que satisfaça ambos os casos.

Na simulação foi considerado um erro inicial de 1m entre o veículo e a trajetória desejada na direção do eixo x, conforme Equação (2.63). Um erro em x ou y afeta os demais sinais de controle e a attitude.

A função custo foi determinada como uma média quadrática multiplicada por uma ponderação que prioriza a attitude do veículo. No caso da saída do controlador, a média foi

feita entre o valor atual do empuxo ($X_{(i)}$) e o valor de cinco passos de integração anterior ($X_{(i-5PI)}$). Assim, a função custo pode ser observada na Equação (2.61),

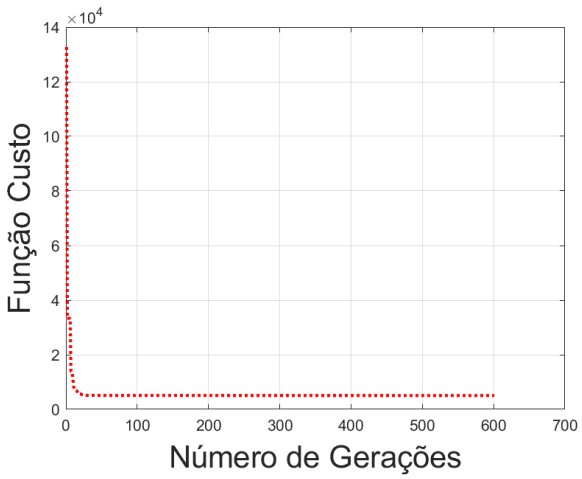
$$\begin{aligned} F_c = p_1\frac{\sum\left(Fz_{(i)} - Fz_{(i-5PI)}\right)^2}{len(Fz)} + p_2\frac{\sum\left(Mx_{(i)} - Mx_{(i-5PI)}\right)^2}{len(Mx)} + \dots \\ \dots + p_3\frac{\sum\left(My_{(i)} - My_{(i-5PI)}\right)^2}{len(My)} + p_4\frac{\sum\left(Mz_{(i)} - Mz_{(i-5PI)}\right)^2}{len(Mz)} + \dots \\ \dots + p_5\frac{\sum\left(x - x_d\right)^2}{len(x - x_d)} + p_5\frac{\sum\left(y - y_d\right)^2}{len(y - y_d)} + p_5\frac{\sum\left(z - z_d\right)^2}{len(z - z_d)} + \dots \\ \dots + p_5\frac{\sum\left(\dot{x} - \dot{x}_d\right)^2}{len(\dot{x} - \dot{x}_d)} + p_5\frac{\sum\left(\dot{y} - \dot{y}_d\right)^2}{len(\dot{y} - \dot{y}_d)} + p_5\frac{\sum\left(\dot{z} - \dot{z}_d\right)^2}{len(\dot{z} - \dot{z}_d)} + \dots \\ \dots + p_6\frac{\sum\left(\phi - \phi_d\right)^2}{len(\phi - \phi_d)} + p_6\frac{\sum\left(\theta - \theta_d\right)^2}{len(\theta - \theta_d)} + p_7\frac{\sum\left(\psi - \psi_d\right)^2}{len(\psi - \psi_d)} + \dots \\ \dots + p_6\frac{\sum\left(\dot{\phi} - \dot{\phi}_d\right)^2}{len(\dot{\phi} - \dot{\phi}_d)} + p_6\frac{\sum\left(\dot{\theta} - \dot{\theta}_d\right)^2}{len(\dot{\theta} - \dot{\theta}_d)} + p_7\frac{\sum\left(\dot{\psi} - \dot{\psi}_d\right)^2}{len(\dot{\psi} - \dot{\psi}_d)}. \end{aligned} \tag{2.61}$$

onde: $p_1 = p_5 = p_7 = 50$, $p_2 = p_3 = p_4 = 70$, $p_6 = 100$ e $len(X)$ é o tamanho de X.

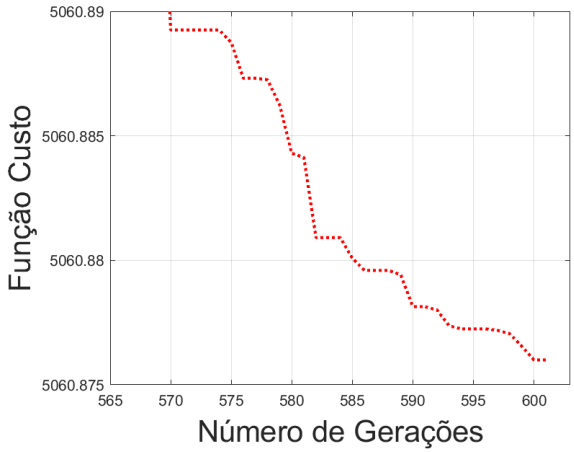
Os demais parâmetros utilizados para a simulação podem ser observados na seção 2.4. O tempo de simulação de cada amostra foi de 20 segundos, que é o instante onde surge o chaveamento e o tempo suficiente para o veículo convergir para a trajetória desejada.

O resultado final da função custo foi $F_c = 5060,876$ e a evolução da função custo é apresentada na Figura 13a. A função custo continua diminuindo, conforme Figura 13b, o que significa que pode-se aumentar o número máximo de gerações. No entanto, o tempo para esta simulação foi mais de 30 horas e a resposta da função custo quase não alterou, uma diferença de menos de 20, se comparado com o número máximo de gerações igual a 300.

Figura 13 – Evolução da função custo e o número de gerações.



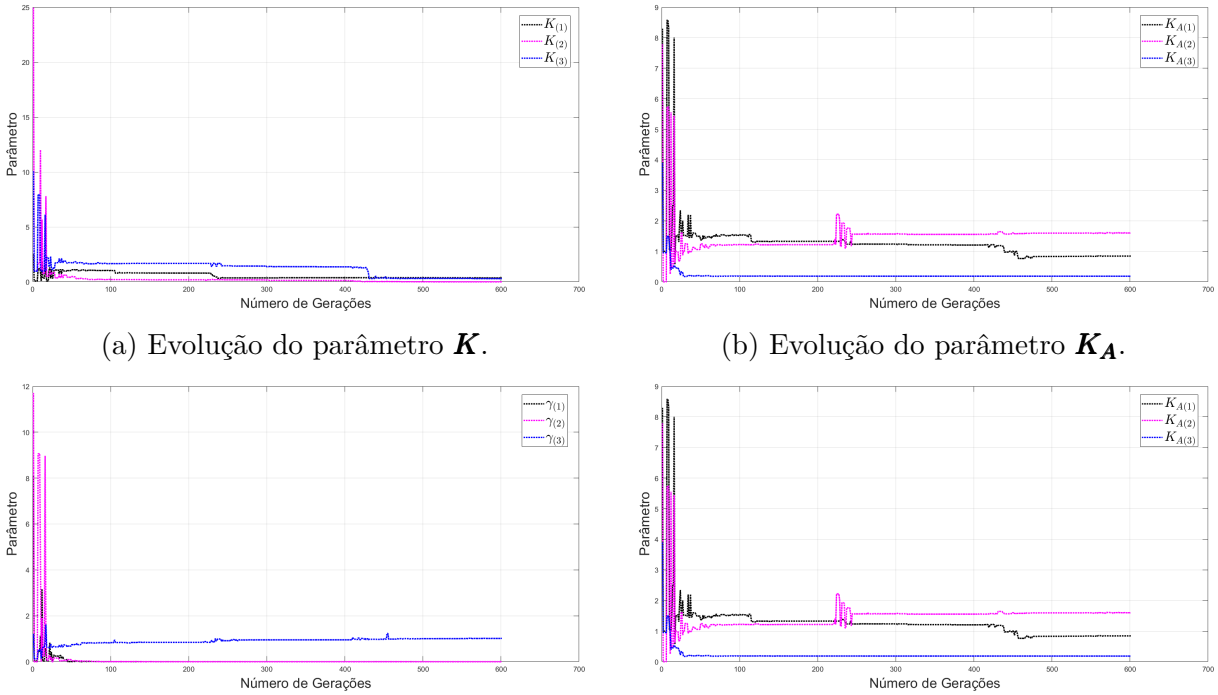
(a) Resultado da função custo.



(b) Ampliação da metade final do resultado da função custo.

Um outro indicativo é a variação dos parâmetros durante as simulações, na Figura 14 pode observar a variação de todos os parâmetros. Após a geração de número 500 a variação e quase nula.

Figura 14 – Evolução dos parâmetros e o número de gerações.



(a) Evolução do parâmetro $\boldsymbol{K}$.

(b) Evolução do parâmetro $\boldsymbol{K_A}$.

(c) Evolução do parâmetro $\boldsymbol{\gamma}$.

(d) Evolução do parâmetro $\boldsymbol{\gamma_A}$.

Os parâmetros do controlador obtidos com o algoritmo genético podem ser observados na Tabela 2.

Tabela 2 – Parâmetros do controlador obtidos com o algoritmo genético.

Parâmetros do Controlador com $F_c = 5060,876$
$\boldsymbol{K} = \begin{bmatrix} 2,83 & 0 & 0 \\ 0 & 4,6971 & 0 \\ 0 & 0 & 1,4559 \end{bmatrix}$
$\boldsymbol{K_A} = \begin{bmatrix} 1,9108 & 0 & 0 \\ 0 & 4,4171 & 0 \\ 0 & 0 & 8,3452 \end{bmatrix}$
$\boldsymbol{\gamma} = \begin{bmatrix} 6,2\text{x}10^{-3} & 0 & 0 \\ 0 & 16,1\text{x}10^{-3} & 0 \\ 0 & 0 & 1,1196 \end{bmatrix}$
$\boldsymbol{\gamma_A} = \begin{bmatrix} 1,8323 & 0 & 0 \\ 0 & 2,1454 & 0 \\ 0 & 0 & 2,6436 \end{bmatrix}$

O interessante é que o GA praticamente transforma o controlador nos eixos x e y, ou seja o controlador passa de um integral *backstepping* para um controle puramente *backstepping*, pois o parâmetro integrativo ($\gamma_{(1)}$ e $\gamma_{(2)}$) do controlar tem valor muito próximo a zero (na ordem de 10^{-3}), fazendo com que os termos integrativos (Equação (2.37)) nas direções dos eixos x e y sejam quase anulados.

2.4 Simulação

Para a simulação do modelo do veículo e do controlador, os parâmetros do veículo utilizados são apresentados na Tabela 3.

Tabela 3 – Parâmetros do veículo.

Parâmetro	Valor	Parâmetro	Valor
m	1,477 Kg	J_R	2,573×10^{-5} Kgm^2
g	9,81 m/s^2	C_{fD}	0,6152
ρ	1,225 Kg/m^3	C_{mD}	0,7382
I_{xx}	0,0146 Kgm^2	l	0,2750 m
I_{yy}	0,0146 Kgm^2	A_z	0,0328 m^2
I_{zz}	0,0266 Kgm^2	A_x	0,0224 m^2
Parâmetros do Controlador			
$\boldsymbol{\Gamma} = \begin{bmatrix} 0,01 & 0 & 0 \\ 0 & 0,01 & 0 \\ 0 & 0 & 0,01 \end{bmatrix}$		$\boldsymbol{C} = \begin{bmatrix} 80 & 0 & 0 \\ 0 & 80 & 0 \\ 0 & 0 & 80 \end{bmatrix}$	

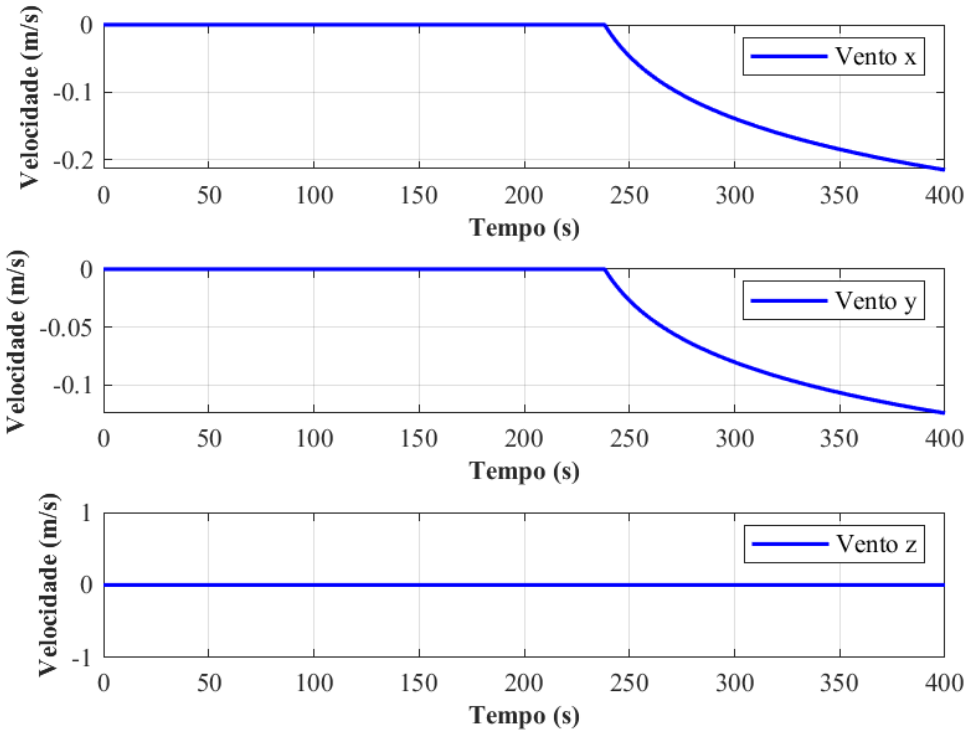
Para a simulação utilizou-se o passo de integração igual à $0,0025s$. Os parâmetros do controlador podem ser observados nas Tabela 2 e Tabela 3.

Com o objetivo de assegurar a robustez do controle na presença de distúrbios não modelados, adicionou-se uma componente de velocidade, que representa o vento, à velocidade do veículo nas direções dos eixos x e y. O modelo do vento varia de acordo coma altitude do veículo, conforme MathWorks (2023) e representado pela Equação (2.62),

$$\begin{aligned} U_w &= W_6 \frac{ln\left(\frac{h}{0,46}\right)}{2,5683}, \quad 0,9m < h < 304,8m \\ \boldsymbol{W_w} &= (\boldsymbol{Rp_{N2B}}) \begin{bmatrix} U_w \sin(d) \\ U_w \cos(d) \\ 0 \end{bmatrix} \end{aligned} \tag{2.62}$$

onde $W_6 = 0,25m/s$ que representa a velocidade do vento em 6 metros de altura, h é a altura do veículo e $d = 30°$ ângulo de incidência do vento no veículo. Supõe-se que o vento começou a atuar no veículo após o instante $230s$, conforme a Figura 15.

Figura 15 – Perfil do vento atuando no veículo veículo.



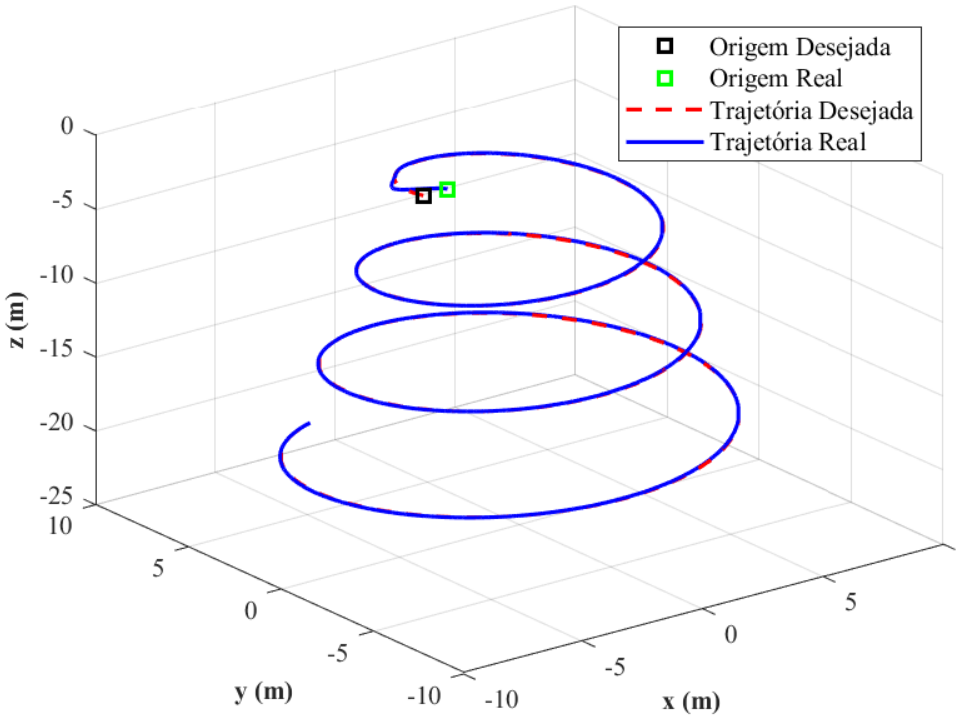
A trajetória desejada foi escolhida de maneira que o veículo realize movimentos nas direções dos eixos x, y e z, mas que não cause a saturação dos atuadores. Assim chegou-se à seguinte trajetória:

$$\boldsymbol{p_{1d}^N} = \begin{bmatrix} (1-0,01t)\cos(0,05t) \\ (4-0,01t)sen(0,05t) \\ -1-0,05t \end{bmatrix}. \tag{2.63}$$

Como condição inicial, escolheram-se todos os estados iguais à zero, exceto $z = -1m$.

A Figura 16 apresenta a trajetória tridimensional do veículo. Lembrando que o sistema de referência adotado é o NED (*North - East - Down*), assim o eixo z é invertido, portanto a trajetória em z começa em $-1m$ e termina em $-20m$.

Figura 16 – Trajetória do veículo.



A Figura 17 apresenta o deslocamento do veículo na direção dos eixos x, y e z. Pode-se notar nos instantes iniciais um erro que pode ser associado à diferença entre as condições inicias do veículo e da trajetória desejada.

A resposta do controlador apresentou erro com amplitude de aproximadamente $1m$ na direção do eixo x. E este erro é proveniente da condição inicial, conforme a Figura 18. Na direção do eixo y observa-se uma amplitude de $0,2m$, já na direção do eixo z o erro foi quase nulo. Estes erro podem ser corrigidos modificando-se os parâmetros do controlador.

Figura 17 – Deslocamento real (azul) e deslocamento desejado (vermelho).

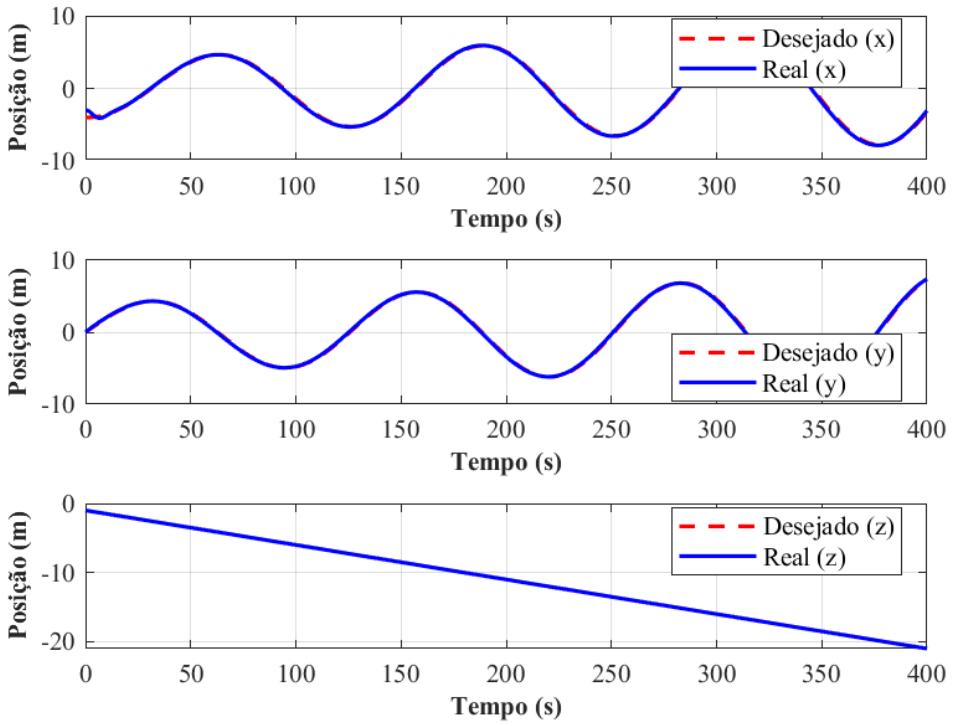
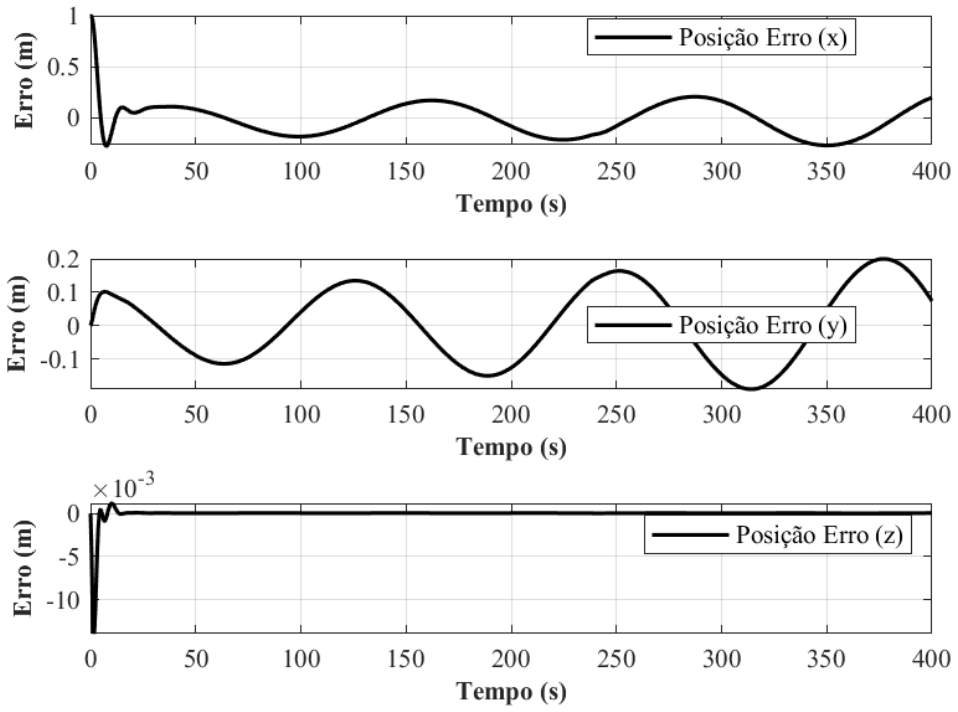


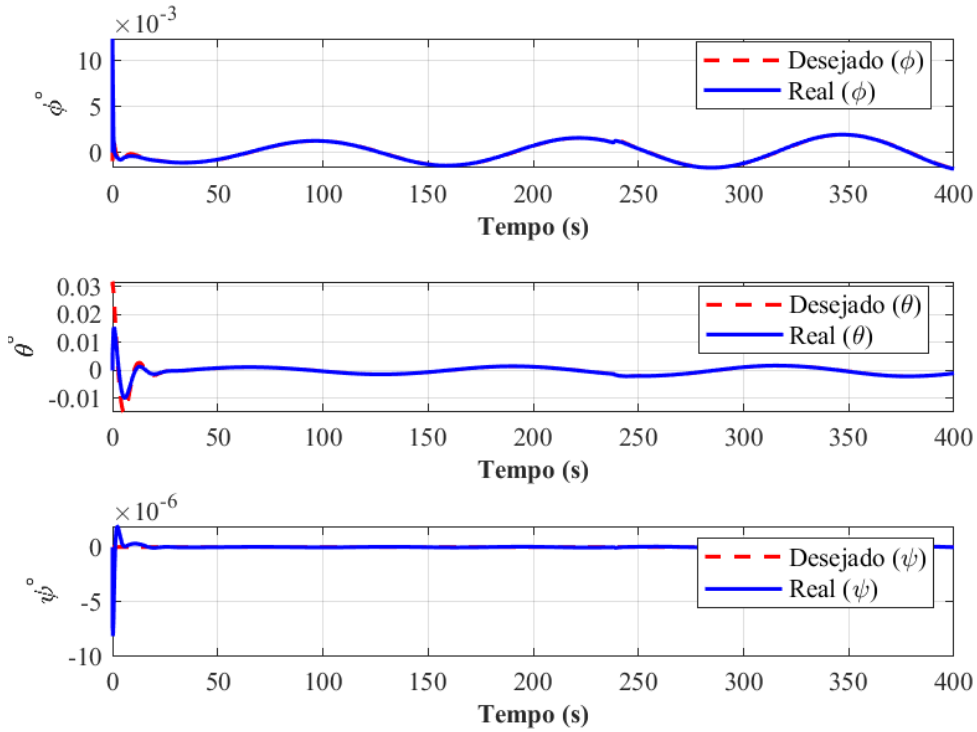
Figura 18 – Erro entre o deslocamento real e o desejado.



Nos primeiros instantes da simulação pode-se notar um pico que ocorreu devido ao erro na condição inicial em x. Como o veículo é subatuado, o controlador força o veículo a convergir primeiro para a trajetória desejada em x, que depende do ângulo θ. Como o veículo possui atuação direta em z o veículo converge rapidamente para o z desejado.

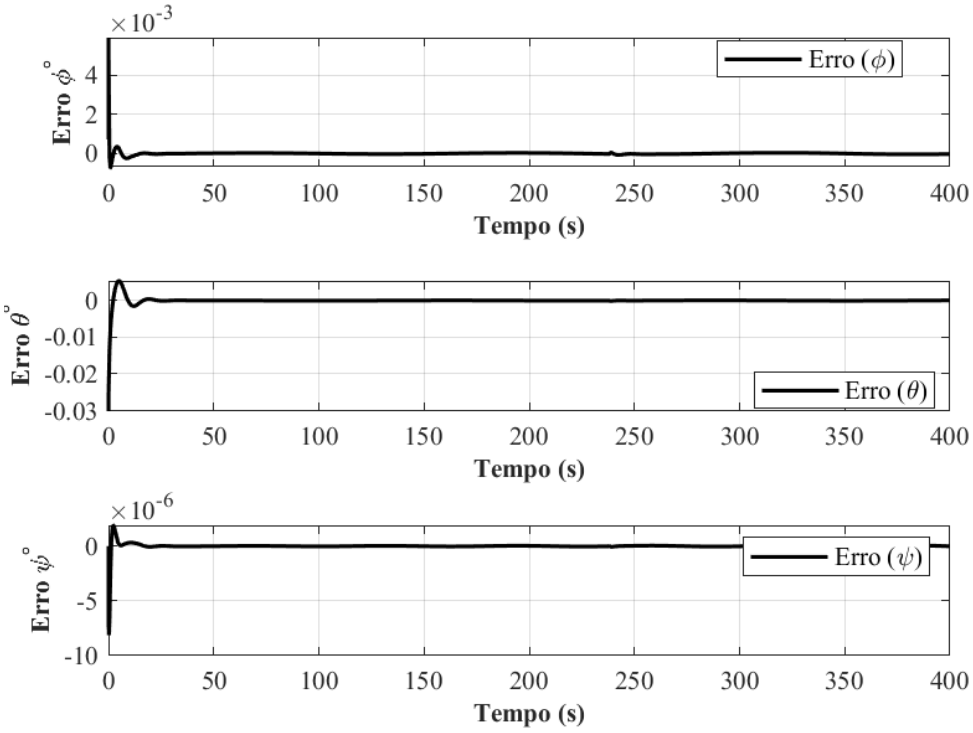
A Figura 19 apresenta a resposta desejada dos ângulos de *Euler*, curva vermelha, obtido por meio do controle de posição (ϕ e θ), responsável em decompor a forca vertical dos motores na direção dos eixos y e x. Um erro nestes ângulos ocasiona, consequentemente, erro de posição em y e x, e dependendo do erro pode afetar a posição em z também. Pode-se notar um erro mais expressivo nos instantes iniciais, pois é exatamente neste instante onde o veículo tenta corrigir o erro de condição inicial em x.

Figura 19 – Comportamento dos ângulos de *Euler*, ϕ, θ e ψ. Ângulo real (azul) e ângulo desejado (vermelho). ϕ e θ desejados são provenientes do controle de Posição.



A Figura 20 apresenta o erro dos ângulos *Euler*, ϕ, θ e ψ. Nos instantes iniciais ocorreu um pico no erro devido ao erro inicial em x. Pode se notar a semelhança nas oscilações do erro de posição em x com o erro θ e y com ϕ. Assim, o erro nos ângulos ϕ e θ está acarretando um erro nas posições y e x consequentemente.

Figura 20 – Erro dos ângulos de *Euler*, ϕ, θ e ψ.



A Figura 21 apresenta o empuxo requerido de cada um dos quatro motores para que o veículo realize a trajetória desejada. Na maior parte da simulação os empuxos mantiveram-se próximos a $3,6N$, o que representa um quarto do peso do veículo. Apenas nos instantes iniciais o controle apresenta um pico. A Figura 22 evidencia os esforços dos motores nos instantes iniciais. Apesar de não haver saturação a mudança foi muito abrupta, pois em menos de dois passo de integração o esforço teve uma amplitude de quase 60% (variação de 2N até 8N) do esforço máximo ($9,84$N).

Figura 21 – Esforço requerido dos quatro motores.

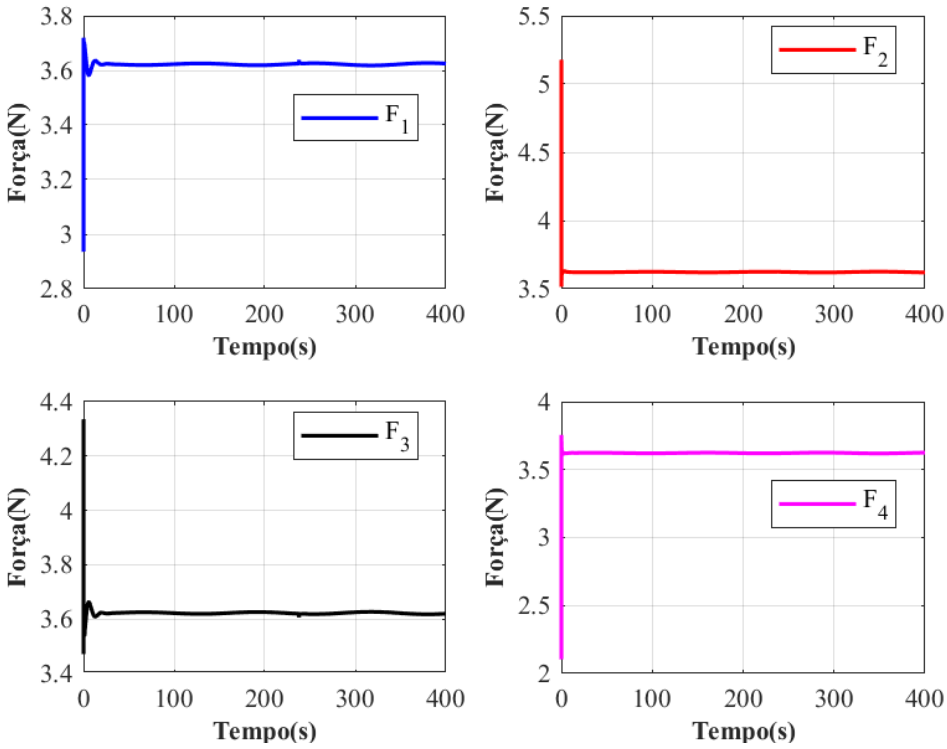
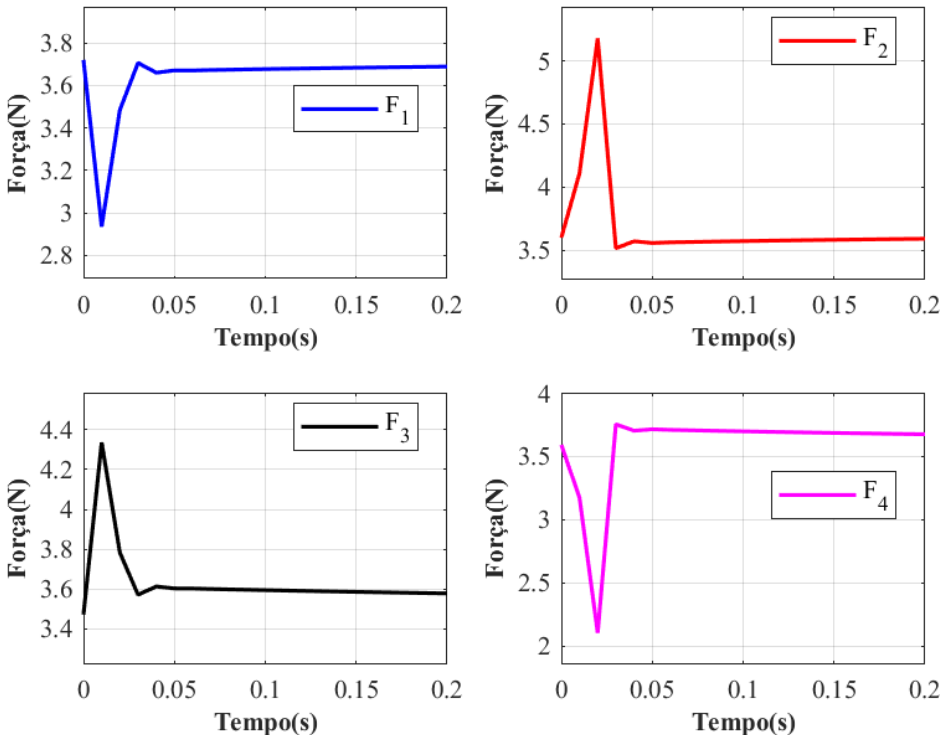


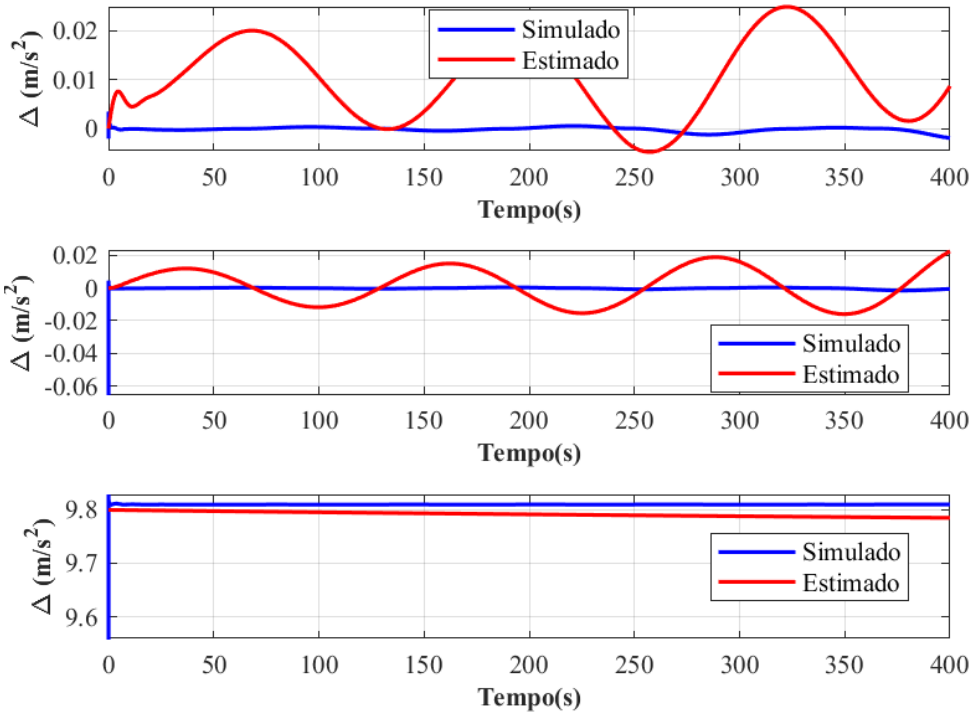
Figura 22 – Ampliação dos instantes iniciais dos esforços requeridos dos quatro motores.



O controlador do veículo foi construído considerando que a maior parte do modelo, incluindo os parâmetros, é desconhecida e foi estimada pelo próprio controle por meio das matrizes $\boldsymbol{\Delta}$ e $\boldsymbol{\Delta_A}$. As Figura 23 e Figura 24 apresentam as respostas obtidas pelo

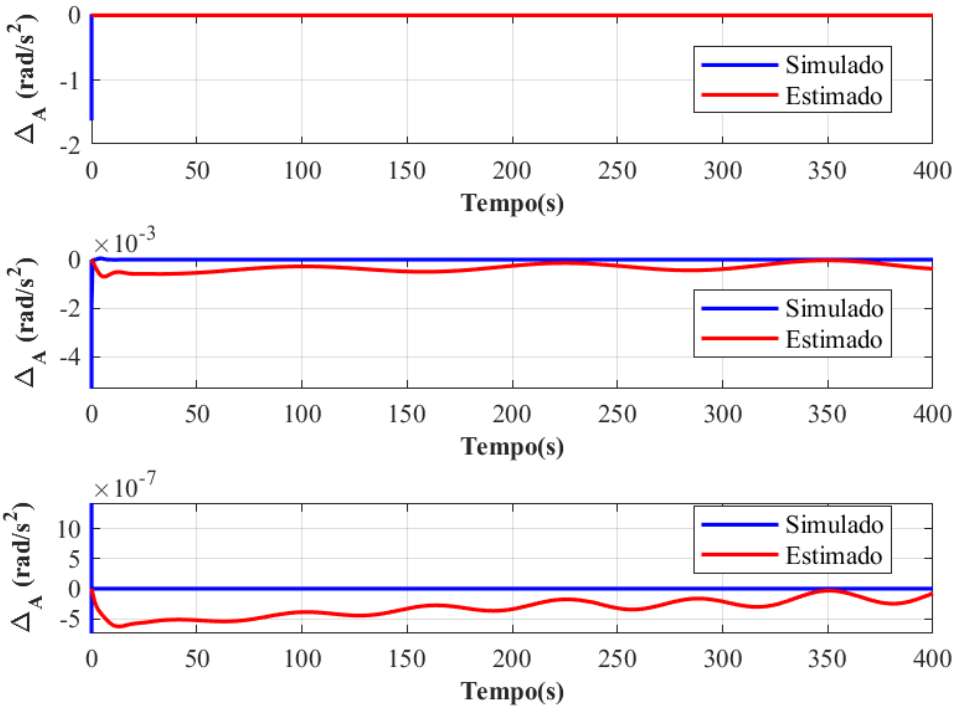
controlador. Pode-se notar que a condição inicial dos estimadores foi $\mathbf{\Delta}(0) = \begin{bmatrix} 0 & 0 & 9,8 \end{bmatrix}^T$, valores obtidos após uma simulação em malha aberta. Este valor é referente à aceleração da gravidade $(9,81 m/s^2)$ e o mesmo procedimento foi utilizado para o controle de atitude.

Figura 23 – Dinâmica do veículo ($\mathbf{\Delta}$) estimada pelo controlador de posição, curva em vermelho e a simulada, curva em azul.



A matriz $\mathbf{\Delta}$ apresentou uma variação da ordem de $10^{-3} m/s^2$ nos dois primeiros estados (referentes a aceleração em x e y) até a metade da simulação, que pode ser considerada baixa. Na metade final da simulação, depois dos 230 segundos onde o veículo está sujeito à ação do vento e o valor estimado começa a distanciar do simulado. O último estado apresentou um variação de aproximadamente $0,02 m/s^2$, referente à aceleração no estado z, conforme a Figura 23.

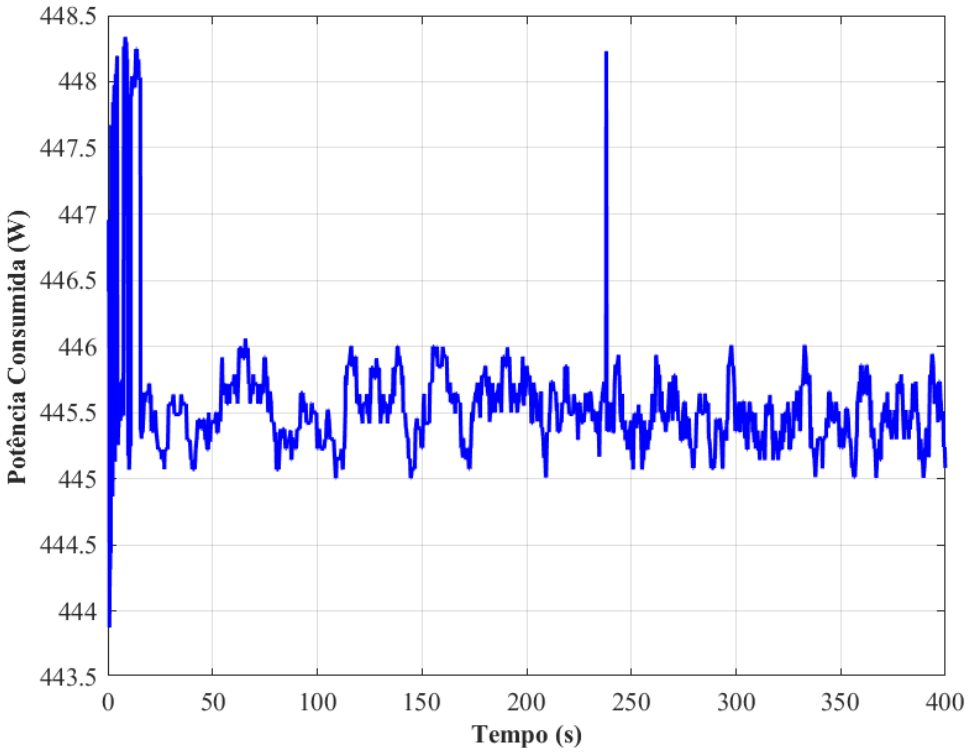
Figura 24 – Dinâmica do veículo ($\boldsymbol{\Delta_A}$) estimada pelo controlador de atitude, curva em vermelho e a simulada curva em azul.



A matriz $\boldsymbol{\Delta_A}$ apresentou uma variação próxima à zero nos dois últimos estados referente às acelerações angulares em θ e ψ, já no primeiro estado apresentou um variação igual a zero, referente à aceleração angular do estado ϕ, conforme a Figura 24.

A potência consumida pelo veículo para realizar esta trajetória pode ser observada na Figura 25. O consumo médio foi de aproximadamente $445,6$W, um pouco mais de 100W em cada um dos motores, considerando uma tensão de entrada constante (tensão da bateria) de $14,8$V.

Figura 25 – Consumo de potência do veículo em cada instante da simulação.

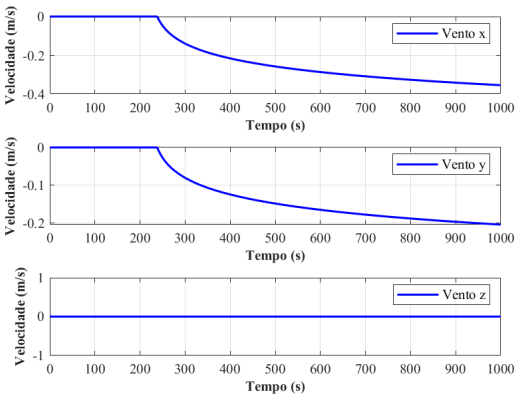


2.5 Discussão

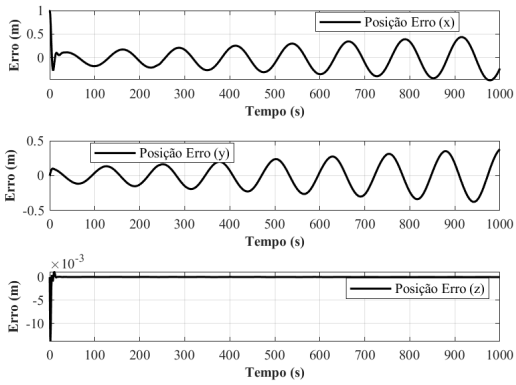
Apesar da demora, aproximadamente 20 segundos, para o veículo convergir para a trajetória desejada na direção do eixo x, onde existe um erro de condição inicial de 1 metro, o controle demonstrou ser robusto na presença de distúrbios externos não modelados. Na direção do eixo y, o erro antes do vento ficou entorno de 10 cm e após o vento subiu para 20 cm aproximadamente. Na direção do eixo z o erro foi muito baixo (na ordem de 10^{-3} m) nos instantes iniciais e depois permaneceu em zero. Estes erros e o tempo de convergência poderiam ser melhorados com o aumentando dos parâmetros do controlador mas isso causaria a saturação dos atuadores na presença do vento e este aumento levaria a um aumento no consumo da potência.

O movimento estimado do veículo antes do vento apresentou um erro baixo (na ordem de 10^{-2}), após o vento o erro cresce, principalmente em x e y, conforme a intensidade do vento aumenta. Mesmo assim este erro não foi grande o suficiente para impedir que o veículo siga a trajetória desejada. Aumentando o tempo de simulação para 1000 segundos, a velocidade do vento na direção de x chegou a $0,35\ m/s$ e em y a $0,21\ m/s$, conforme Figura 26a. Os erros em x e y aumentaram mas não passaram de 50 cm, conforme Figura 26b.

Figura 26 – Simulação com 1000 segundos. Perfil de vento e o erro de posição.



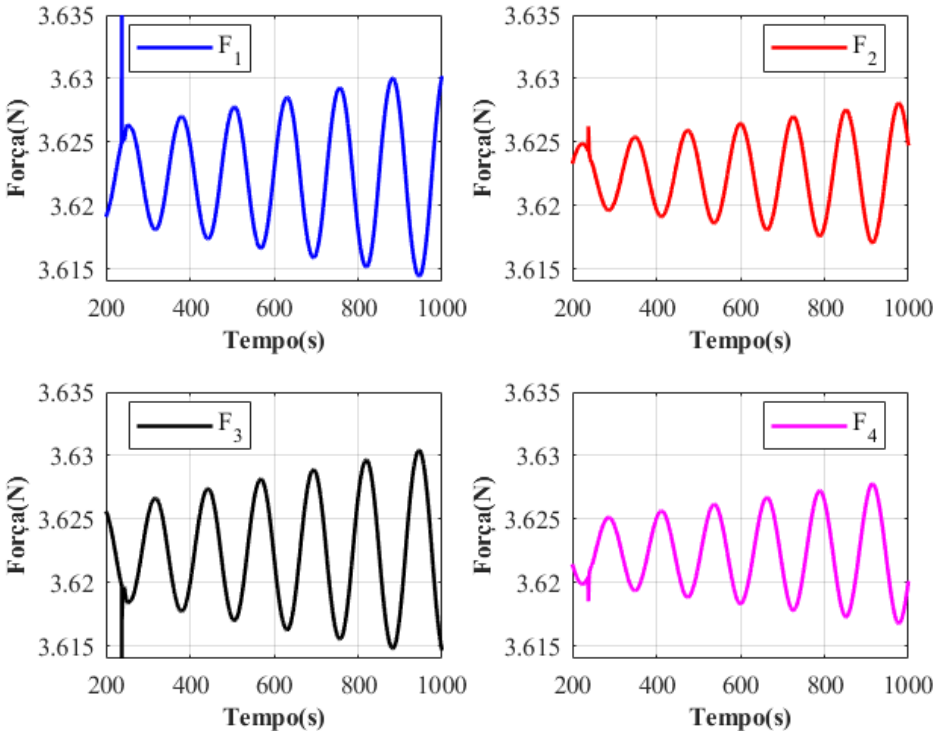
(a) Perfil do vento atuando no veículo veículo.



(b) Erro entre o deslocamento real e o desejado.

O esforço de controle para simulação de 1000 segundos, somente após o vento (de 200 s até 1000 s), pode ser observado na Figura 27. Pode-se notar que o esforço de cada um dos motores cresce, mas não chega a 50% do máximo. Indicando que seria possível aumentar os parâmetros do controlador para diminuir os erros.

Figura 27 – Esforço requerido dos quatro motores.



A aplicação da técnica de GA para obtenção dos parâmetros do controlador demonstrou ser eficaz pois foi possível obter os valores dos parâmetros todos de uma vez, sem a necessidade de realizar uma busca exaustiva para cada parâmetro. O GA também demonstrou ser capaz de ajustar o controlador, neste caso o GA praticamente transformou

um controle integral *backstepping* para um controlador puramente *backstepping*, na direção de x e praticamente desligou a ação do controlador na direção y, justificando o erro presente nestas duas direções.

3 Modelo de Rastreamento do Gasoduto

O veículo possui uma câmera fixa na sua parte de baixo de modo que ele consiga filmar tudo que esteja abaixo dele e dentro do campo de visão da câmera.

Para o rastreamento do gasoduto utiliza-se um algoritmo de detecção de bordas por meio do método de *Canny* (CANNY, 1986) em conjunto com a transformação de *Hough* (MUKHOPADHYAY; CHAUDHURI, 2015), gerando linhas em coordenadas polares que determinaram a trajetória do quadricóptero.

Dado que o veículo em questão manifesta subatuação nas direções x e y, faz-se necessário a execução de ajustes na atitude do quadricóptero, com o intuito de efetuar a decomposição da força vertical gerada pelos propulsores. Tal procedimento visa a criação deliberada de componentes de atuação nas direções x e y. Em alguns casos estes ajustes na atitude podem fazer com que o gasoduto saia do campo de visão da câmera. Uma estratégia para que isto não ocorra é limitar os ângulos do veículo (BENSALAH; M'SIRDI; NAAMANE, 2019) o que não foi necessário nesta simulação.

3.1 Detecção do Gasoduto

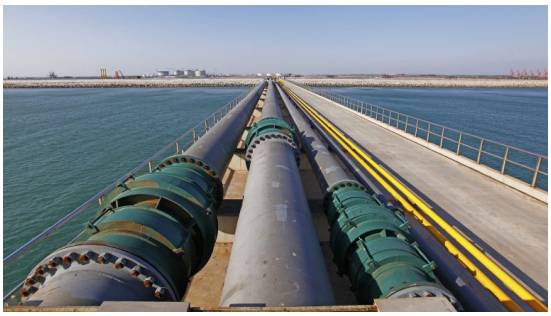
O método clássico de *Hough* para detecção de linhas é utilizado em conjunto com algum método de detecção de beiradas (*edge*) sendo o mais famoso o filtro de *Canny* (DAVIES, 2017).

A imagem pode ser considerada uma matriz com um número finito de linhas e colunas. A largura da imagem é representada pelo número de colunas na matriz e a altura da imagem, o número de linhas da matriz. A imagem é composta por vários *pixels*, representados pelos elementos da matriz. Cada *pixel* representa um aspecto da imagem, como brilho, escala e cor. Para cada um atribuí-se um valor que varia de 0 a 255 ou a combinação das seguintes cores vermelho, azul e verde (RGB-*red, green e blue*) (GOLLAPUDI, 2019).

Após busca na internet encontraram-se duas imagens aéreas de dutos que podem ser considerados gasodutos. Na Figura 28a pode-se observar três dutos. A imagem possui resolução de 768x432 *pixels* , já a Figura 28b, possui seis dutos e resolução de 400x530 *pixels* .

O primeiro passo é converter a imagem para a escala de cinzas, o que aumenta o contraste da imagem. A conversão é feita por uma média ponderada dos valores de RGB da imagem $(0,299 \cdot R + 0,587 \cdot G + 0,114 \cdot B)$, conforme a Figura 29.

Figura 28 – Fotos aéreas obtidas da internet para desenvolvimento do algoritmo de detecção de bordas do gasoduto.



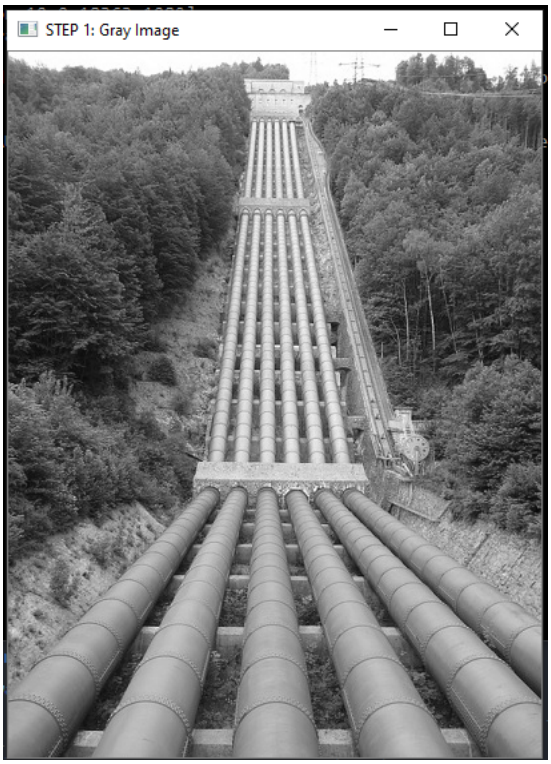
(a) Imagem com três dutos.

Fonte: CONSUMER ENERGY ALLIANCE (2017)

(b) Imagem com seis dutos.

Fonte: RW ENGENHARIA (2016)

Figura 29 – Imagem da Figura 28b, convertida em escala de cinza.



Quando algumas fotos ou filmagens são feitas podem haver áreas claras e escuras na imagem. Isto ocorre devido à sensibilidade da câmera à luz. Este efeito é considerado um ruído na imagem (MCHUGH, 2018). Portanto, precisa-se fazer uma suavização da imagem (*smoothing*) para remover o ruído e o excesso de detalhes, conhecido como *blur* que significa borrão em inglês, semelhante a uma fotografia desfocada (CHUNG, 2017).

A suavização da imagem é feita por meio de filtragem ou a convolução de duas imagens. Assim, por exemplo, o resultado da filtragem da Figura 29 pode ser observado na Figura 30.

Figura 30 – Remoção de ruídos.

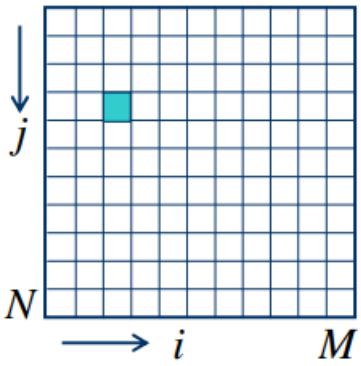
O passo seguinte consiste na detecção de bordas ou contornos de qualquer objeto na imagem. O computador não é capaz de entender do que se trata o objeto na imagem ele somente faz um escaneamento de cada *pixel* e seus vizinhos e detecta uma diferença de cor significativa em relação aos *pixels* próximos, assim pode-se concluir se este *pixel* pertence a algum contorno ou bordas de algum objeto na imagem (CHUNG, 2017).

De acordo com Vijayarani e Vinupriya (2007), o melhor método para detecção de bordas é o *Canny*, desenvolvido em 1986 por John Canny (CANNY, 1986), e pode ser dividido em 5 passos:

- Suavização da imagem: tornar a imagem borrada para remoção de ruídos;
- Encontrar gradientes: o gradiente de uma imagem é um vetor (magnitude, sentido e direção). O gradiente é calculado pelas derivadas parciais nas direções horizontal

(largura, $\boldsymbol{i}$) e vertical (altura, $\boldsymbol{j}$) da imagem. Por exemplo, considere uma imagem de dimensão NxM, conforme a Figura 31.

Figura 31 – Exemplo de uma imagem com dimensão NxM.



Usando a aproximação em série de *Taylor* para a derivada parcial de uma função discreta, calcula-se o gradiente da imagem (Equação (3.1)) e a orientação (Equação (3.2)).

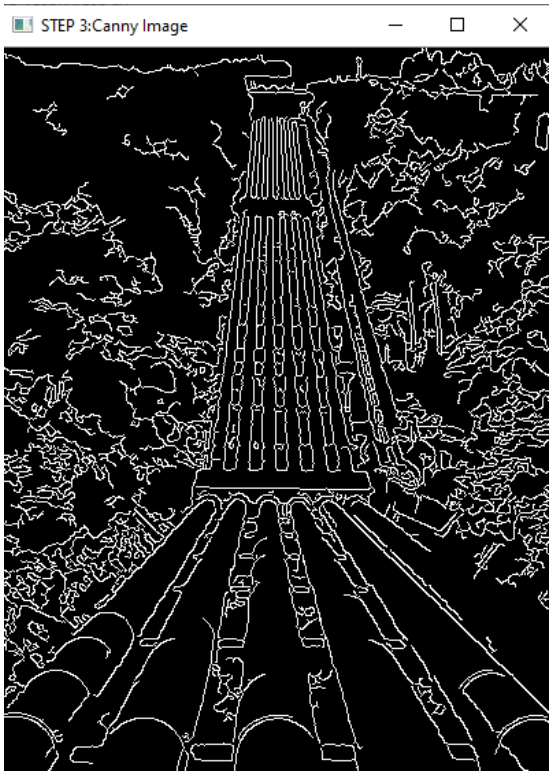
$$\begin{aligned} \nabla \boldsymbol{I}(x,y) &= \tfrac{\partial \boldsymbol{I}(x,y)}{\partial x}\mathbf{i} + \tfrac{\partial \boldsymbol{I}(x,y)}{\partial y}\mathbf{j} = \ldots \\ \ldots &= \left[\tfrac{\mathbf{I}(x+1,y)-\mathbf{I}(x-1,y)}{2}\right]\mathbf{i} + \left[\tfrac{\mathbf{I}(x,y+1)-\mathbf{I}(x,y-1)}{2}\right]\mathbf{j}. \end{aligned} \tag{3.1}$$

$$\theta(x,y) = \arctan\left[\frac{\frac{\partial \boldsymbol{I}(x,y)}{\partial y}}{\frac{\partial \boldsymbol{I}(x,y)}{\partial x}}\right]. \tag{3.2}$$

- Supressão não máxima: para cada *pixel* calcula-se a direção que melhor se aproxima da direção do gradiente. Se o contraste de cada *pixel* for menor que o contraste de um de seus vizinhos na direção do gradiente, então define-se para este *pixel* o valor zero.
- Limite de borda: o método define um limite alto e um baixo. Se o *pixel* tiver um valor acima do limite inferior e for vizinho de uma borda, este *pixel* é definido como uma borda mas quando o vizinho não for uma borda este mesmo *pixel* não é uma borda. Agora, caso o valor seja maior que o limite superior o *pixel* é definido como uma borda.
- Rastreamento: as bordas que não estiverem conectadas a bordas consideradas mais fortes são desconsideradas.

O resultado do método de *Canny* aplicado à Figura 30, pode ser observado na Figura 32.

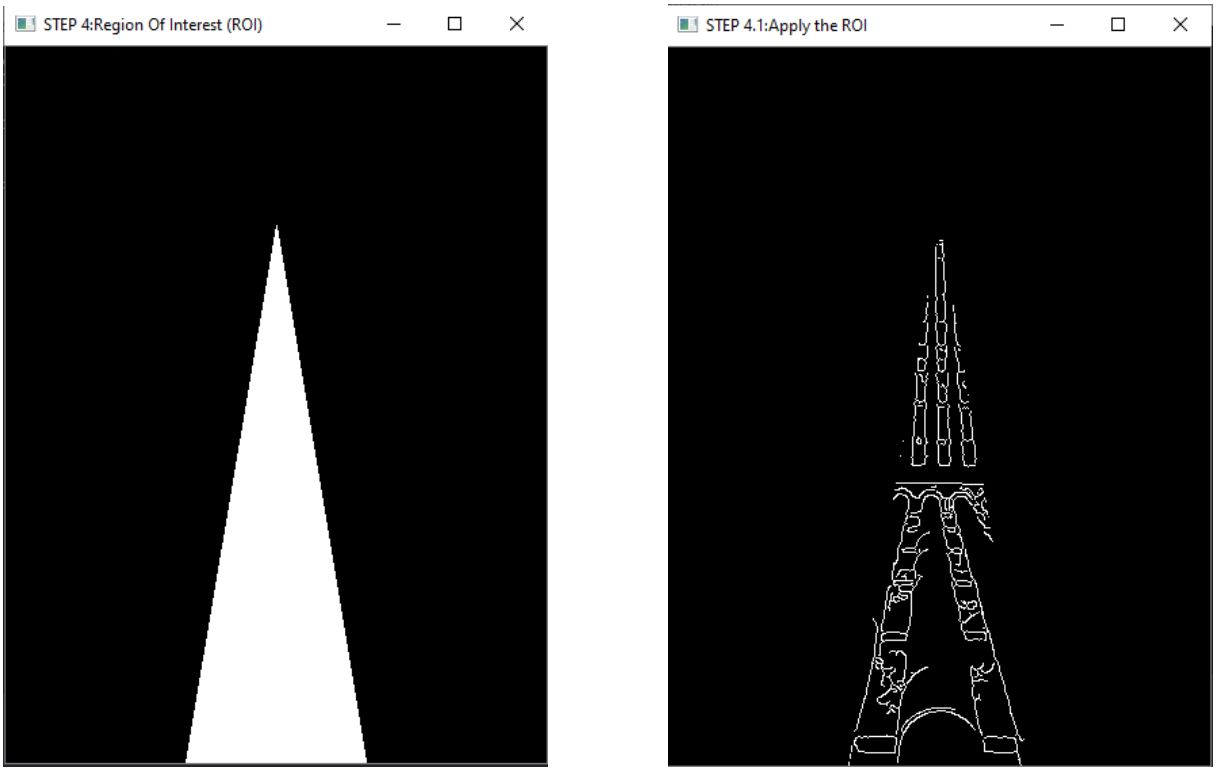
Figura 32 – Evidenciando a intensidade dos *pixels* desejados pelo método de *Canny*.



A imagem com as bordas detectadas ainda contém muita informação. Neste caso, deseja-se obter apenas as informações relacionadas ao gasoduto. Portanto, é criada uma região de interesse (ROI-*region of interest*) que deve ter o mesmo tamanho da imagem. A região da imagem que o gasoduto ocupa pode ser definida, como um triângulo, com base igual a um terço (1/3) da largura da imagem e altura igual à quatro quintos (4/5) da altura da imagem, conforme Figura 33a.

A ROI pode ser considerada como uma máscara, onde o *pixel* que não pertence à ROI recebe o valor zero (cor preta na escala de cinza) e o pertencente a ROI recebe o valor 255 (cor branca). Ao aplicar a ROI (máscara) na imagem desejada (Figura 32), somente as bordas dentro da ROI ficaram, as demais são cobertas pela máscara, conforme a Figura 33b.

Figura 33 – Criação e aplicação de uma ROI.



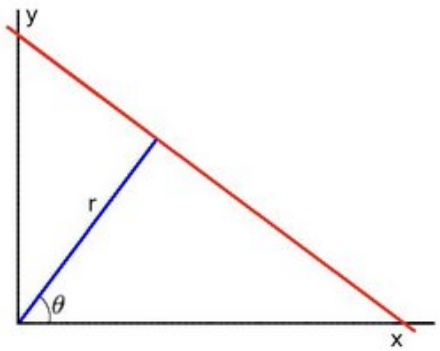
(a) Criação da ROI.

(b) Aplicação da ROI na imagem.

O passo seguinte é transformar estas bordas em linhas. De acordo com Mukhopadhyay e Chaudhuri (2015), um método eficiente e popular para detectar linhas é através da Transformação de *Hough* (HT).

Uma linha pode ser definida por meio de dois parâmetros, tanto em coordenadas cartesianas (m inclinação da linha e b ponto onde cruza o eixo horizontal) como em coordenadas polares, sendo r a distância da origem até a linha e θ o ângulo entre o eixo horizontal e a normal da origem até a linha, conforme a Figura 34.

Figura 34 – Exemplo de representação de uma reta ou linha em coordenadas cartesianas e polares.



Fonte: Adaptado de (OPENCV, 2020).

Para a HT utiliza-se a representação da linha em coordenadas polares (Equação (3.3)):

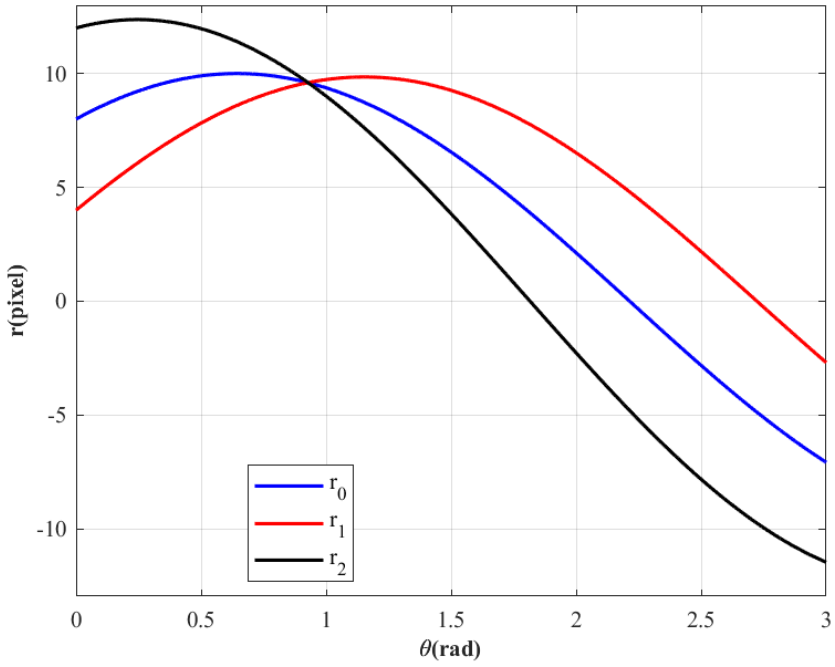
$$r = x\cos(\theta) + ysen(\theta) \tag{3.3}$$

que pode ser convertida em coordenadas cartesianas, como segue:

$$y = \left(-\frac{\cos(\theta)}{sen(\theta)}\right)x + \left(\frac{r}{sen(\theta)}\right). \tag{3.4}$$

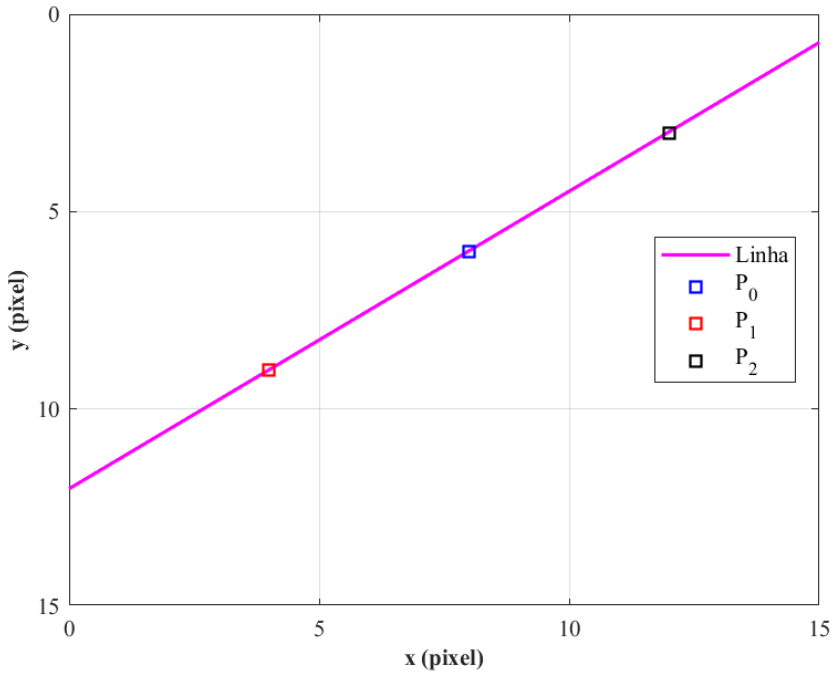
Para exemplificar a HT, supõe-se que uma imagem com dimensão 15x15 *pixels* e que os *pixels* nas posições $P_0(8,6)$, $P_1(4,9)$ e $P_2(12,3)$ foram marcados como bordas. E escrevendo-os em coordenadas polares e variando $0 < \theta < 2\pi$ e considerando somente valores com $r > 0$ e fazendo um gráfico com estes pontos no plano $(r\ \theta)$, obtém-se a Figura 35.

Figura 35 – Representação dos pontos $P_0(8,6)$, $P_1(4,9)$ e $P_2(12,3)$ no plano $(r-\theta)$.



De acordo com a Figura 35, as três curvas se interceptam no ponto $r = 9,6$ e $\theta = 0,925$, estes valores representam uma linha em coordenadas polares que passa pelos três pontos, como pode-se observar na Figura 36, onde a curva em magenta é a obtida para $r = 9,6$ e $\theta = 0,925$.

Figura 36 – Representação dos pontos $P_0(8,6)$, $P_1(4,9)$ e $P_2(12,3)$ e da linha gerada por $r = 9,6$ e $\theta = 0,925$.



Quanto mais intersecções de curvas mais pontos esta linha terá. Assim pode-se estabelecer um limite mínimo para o número de intersecções para detectar uma linha, e o algoritmo pode retorna os valores de r e θ ou dois pontos, ponto inicial (x_i,y_i) e ponto final (x_f,y_f).

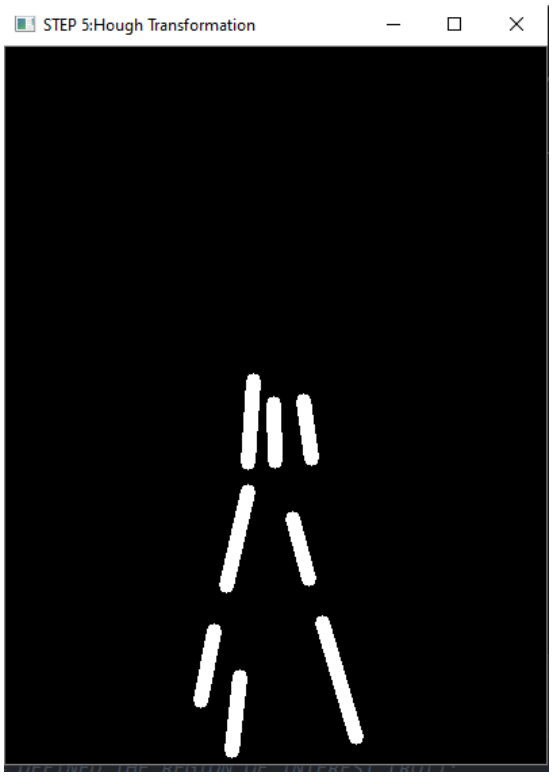
Aplicando a HT na Figura 33b, obtiveram-se dezesseis pontos, oito pontos iniciais, oito pontos finais e oito valores de r e θ, conforme a Tabela 4.

Tabela 4 – Pontos obtidos pela HT. Pontos iniciais (x_i,y_i), pontos finais (x_f,y_f) e coordenadas polares (r,θ).

x_i $(pixel)$	y_i $(pixel)$	x_f $(pixel)$	y_f $(pixel)$	r $(pixel)$	θ (rad)
212	348	224	393	-108	2.8623
167	520	173	465	-194	3.1241
163	398	179	328	248	0.2269
198	263	199	306	-197	3.1067
234	425	259	510	200	0.0698
144	483	154	431	221	0.1047
179	307	183	246	214	0.0873
220	261	226	304	196	0.0524

A critério de verificação pode-se sobrepor as oitos linhas identificadas na imagem original, por meio dos pontos iniciais e finais, conforme a Figura 37.

Figura 37 – Resultado da HT aplicado na Figura 33.



(a) Resultado da HT na ROI.

(b) Resultado da HT sobre a imagem original.

De acordo com Cerón, Mondragón B. e Prieto (2014), a imagem pode retornar linhas segmentadas, assim o pós-processamento é necessário para fazer a junção das linhas segmentadas. O pós-processamento pode ser feito tanto com as linhas geradas em coordenadas polares ou cartesianas.

Neste caso, as oito linhas geradas são transformadas em apenas duas, uma a esquerda e uma a direita. No caso das coordenadas polares, as linhas com $\theta > \frac{\pi}{2}$ são as linhas à esquerda (r_e, θ_e) e as demais são as linhas à direita (r_d, θ_d). Fazendo a média das linhas à esquerda e à direita, obtém-se a trajetória a ser seguida pelo quadricóptero:

$$\begin{cases} r_e = -166,33pixel \\ \theta_e = 3,0310rad \end{cases} \qquad \begin{cases} r_d = 215,80pixel \\ \theta_e = 0,1082rad \end{cases} . \tag{3.5}$$

Seguindo o mesmo procedimento mas para o caso de coordenadas cartesianas, pode-se determinar qual linha está à esquerda e qual está à direita, os pontos da Tabela 4 são convertidos em retas, por:

$$y = mx + b \tag{3.6}$$

onde $m = \frac{y_f - y_i}{x_f - x_i}$ e b o ponto onde a reta cruza o eixo horizontal x. Assim, quando m for menor que zero, a reta está à esquerda e quando for maior, a reta está à direita. Calculando a média dos parâmetros (m e b) à esquerda e à direita, obtêm-se duas retas descritas pela

Equação (3.7):

$$\begin{aligned} y_L &= -8,49791667x_L + 1857,62708333 \\ y_R &= 14,32916667x_R - 2596,06666667. \end{aligned} \tag{3.7}$$

A critério de verificação pode-se plotar as duas retas obtidas sobre a imagem original, conforme a Figura 38.

Figura 38 – Resultado final da identificação do gasoduto. Imagem original com as duas retas obtidas.

3.2 Controle Servo Visual

Existem diferentes tipos de abordagens para obtenção de informações por meio de imagens, geralmente classificadas em três categorias (CHAUMETTE; HUTCHINSON, 2006):

* Controle servo visual baseado em posição (PBVS - *Position-Based Visual Servoing*): as informações são baseadas em parâmetros retirados do objeto em 3-D;
* Controle servo visual baseado em imagem (IBVS - *Image-Based Visual Servoing*): as informações são baseadas em parâmetros retirados das imagens em 2-D;

* Controle servo visual híbrido (HVS - *Hybrid Visual Servoing*): uma combinação das duas anteriores.

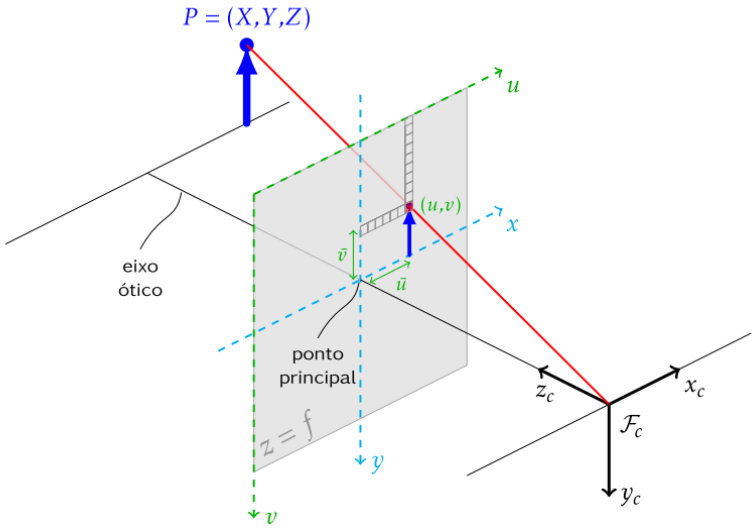
De acordo com Boltov et al. (2019), o intuito do modelo IBVS é fazer com que o plano da imagem da câmera siga um plano da imagem desejado assim as informações utilizadas obtidas diretamente do plano da imagem. Por outro lado, os modelos PBVS e o HVS possuem uma forte dependência dos parâmetros da câmera, uma vez, que os objetos são reconstruídos a partir da imagem. Os parâmetros da câmera são obtidos por meio de uma identificação prévia da câmera.

Conforme apresentado por Araar e Aouf (2014) e Araar e Aouf (2013), o controle PBVS demonstrou ser uma melhor opção para o rastreio de linhas paralelas, pois converge mais rapidamente para posição desejada.

3.2.1 Projeção do Espaço Tridimensional para o Plano da Imagem

A Figura 39 descreve o modelo de projeção perspectiva do espaço tridimensional para um espaço bidimensional, plano da imagem. A imagem é formada em um plano a uma distância f a frente da origem da câmera.

Figura 39 – Projeção perspectiva do espaço tridimensional para o bidimensional.



Fonte: Perrone (2013)

A projeção de um objeto pode ser feita por meio de semelhanças de triângulos, assim as coordenadas do ponto $\boldsymbol{P}$, no plano da imagem ($z = f$), podem ser escritas pelos *pixels* (x, y):

$$\begin{bmatrix} x \\ y \end{bmatrix} = \frac{f}{Z} \begin{bmatrix} X \\ Y \end{bmatrix}, \tag{3.8}$$

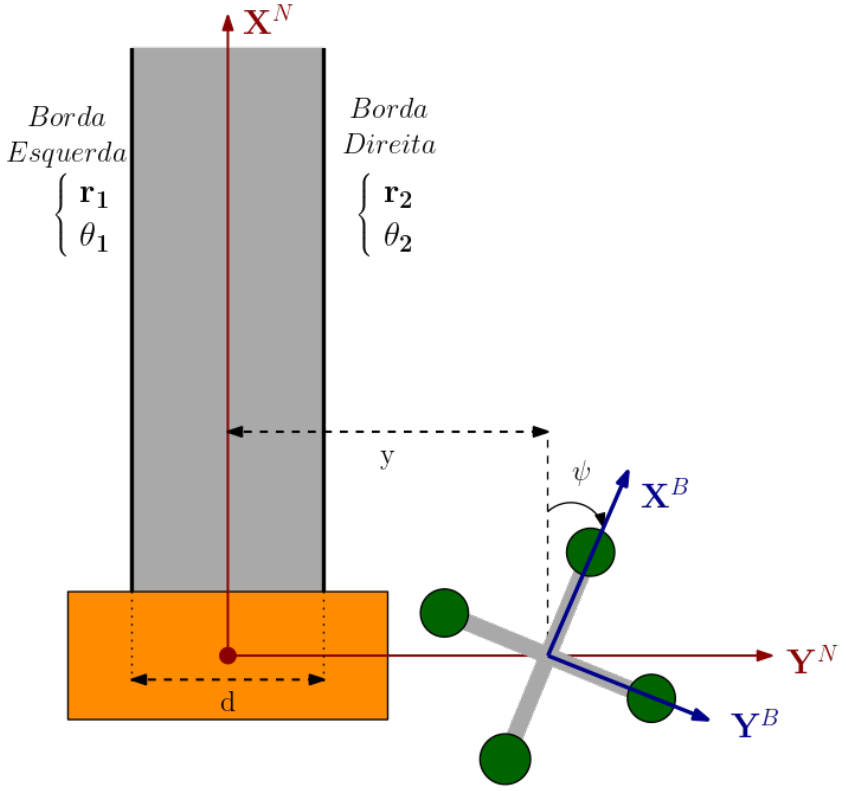
onde f é comprimento focal da câmera.

3.2.2 Controle Servo Visual Baseado em Posição

O principal desafio é encontrar uma relação entre os parâmetros medidos pela câmera com o objeto real. Neste caso, o desafio se torna um pouco mais fácil, pois o objeto é bem conhecido.

Supondo que o centro do gasoduto é a origem do sistema de coordenada inercial de navegação. Partindo do cenário onde o veículo não esta alinhado com o duto, ou seja, o veículo está deslocado no eixo y e rotacionado no eixo z (ψ), conforme a Figura 40.

Figura 40 – Posição do quadricóptero em relação as bordas dos duto.



A projeção das bordas do duto no plano da imagem da câmera é feita por meio da transformação de *Hough*, supondo que ϕ e θ são pequenos, próximos a zero. Aplicando a Equação (3.8), tem-se:

$$r_1 = f\frac{-y + \frac{d}{2}}{h} \tag{3.9a}$$

$$\theta_1 = -\psi \tag{3.9b}$$

$$r_2 = f\frac{-y - \frac{d}{2}}{h} \tag{3.9c}$$

$$\theta_2 = -\psi \tag{3.9d}$$

onde f é o comprimento focal, d é o diâmetro do duto e h é a distância entre a câmera e o duto.

O intuito é obter uma relação, com base na Equação (3.9), onde podem-se escrever as variáveis h, y e ψ em função dos demais termos, ou seja, $h = \mathcal{G}_{h(r_1,r_2,\theta_1,\theta_2)}$, $y = \mathcal{G}_{y(r_1,r_2,\theta_1,\theta_2)}$ e $\psi = \mathcal{G}_{\psi(r_1,r_2,\theta_1,\theta_2)}$. O ideal é obter estas variáveis somente em função dos parâmetros da câmera, mas neste caso como as dimensões do duto são conhecidas, podem-se utilizá-las sem problemas.

Dividindo a Equação (3.9a) pela Equação (3.9c) e isolando y, tem-se:

$$y = \frac{d\,(r_2 + r_1)}{2\,(r_2 - r_1)} \tag{3.10}$$

substituindo a Equação (3.10) na Equação (3.9a) ou Equação (3.9c) e isolando h, tem-se:

$$h = \frac{df}{r_1 - r_2} \tag{3.11}$$

obtendo assim a posição do veículo com relação ao duto. A orientação, ψ, pode ser obtida por meio da média de θ_1 e θ_2, assim:

$$\psi = -\frac{\theta_1 + \theta_2}{2}. \tag{3.12}$$

A Equação (3.10), Equação (3.11) e Equação (3.12), descrevem a posição e a atitude do veículo com a relação ao duto. Assim, pode-se escrever a trajetória desejada da seguinte maneira:

$$\begin{aligned} y_d &= y_q - y \\ z_d &= h_p + h \\ \psi_d &= \psi_q - \psi \end{aligned} \tag{3.13}$$

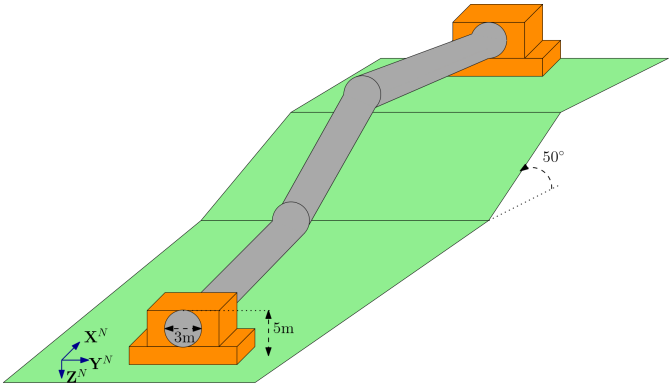
onde y_q é a posição do veículo no eixo y, h_p é a altura do duto com relação ao solo e ψ_q é a atitude do veículo ao redor do eixo z. A posição desejada do veículo ao longo do eixo x não foi definida. Como o duto é contínuo, pode-se definir uma velocidade constante para o veículo ou uma reta contínua ao longo do eixo x.

3.3 Simulação

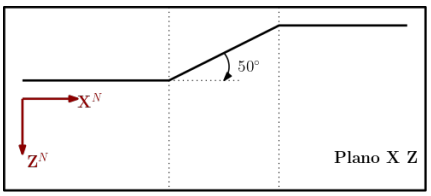
Para a realização da simulação, assume-se que o duto sempre estará dentro do campo de visão da câmera.

O intuito é fazer com que o veículo siga entre as duas bordas do duto. Supondo que o duto a ser seguido possua um perfil igual ao apresentado na Figura 41.

Figura 41 – Representação do duto a ser seguido pelo quadricóptero.



(a) Representação tridimensional do duto.



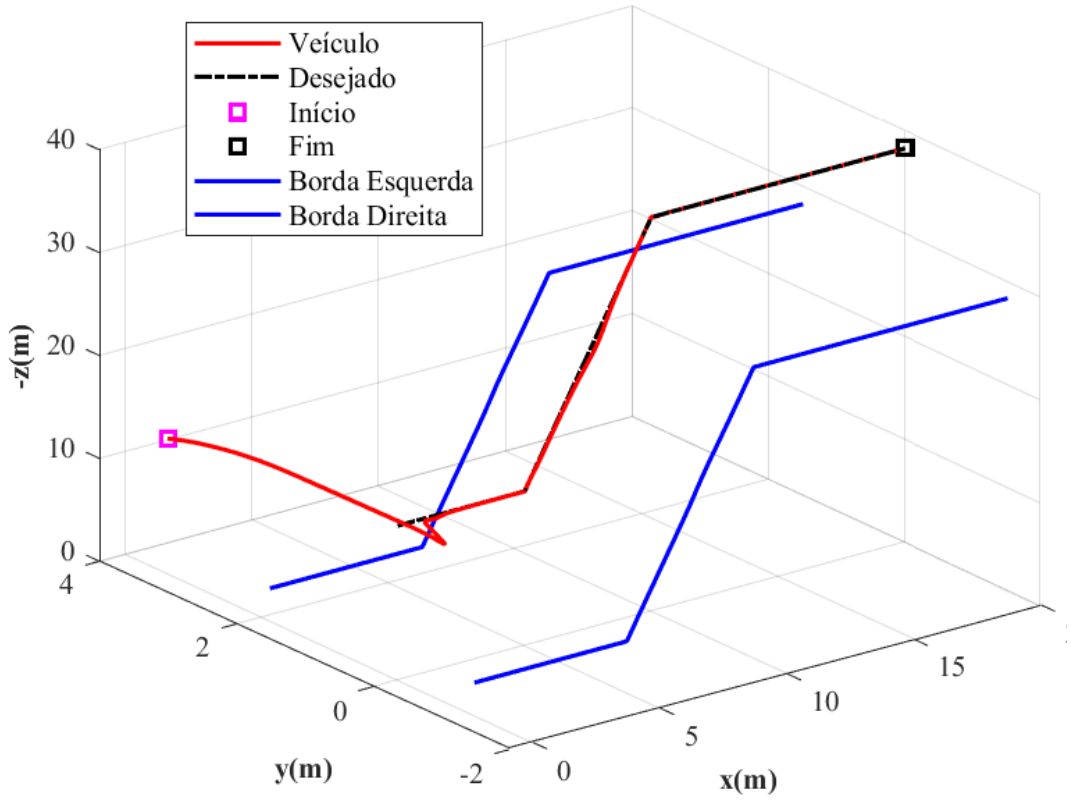
(b) Representação do plano XZ do duto.

Utilizando o mesmo controle desenvolvido na seção 2.2, e o caso do duto contínuo ao longo do eixo x, a posição desejada é $x_d = 0,1 \cdot t$, com t sendo o tempo de simulação.

Supõe-se que o veículo somente começará a seguir o duto quando atingir a altura desejada, que, neste caso, é de $15m$, ou seja $10m$ acima do duto. O ideal é que o veículo mantenha essa distância vertical do duto, para que os propulsores não dispersem o gás em caso de vazamento. As condições iniciais do veículo são: $x = -1m$, $y = 3m$, $z = -15m$ e $\psi = 30°$.

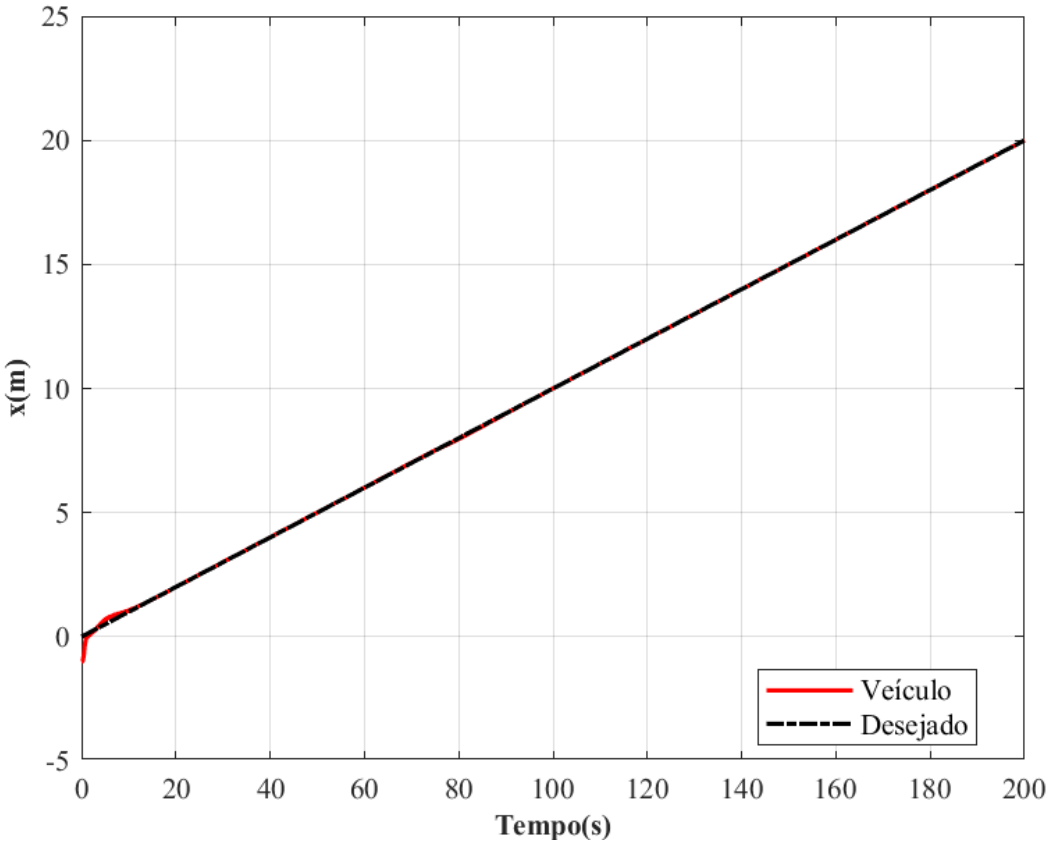
A Figura 42 apresenta a simulação do veículo no plano tridimensional. O sinal do eixo z foi invertido para facilitar a visualização. As bordas do duto são representadas pela curva azul. Pode-se observar que o veículo convergiu para o centro e seguiu o duto.

Figura 42 – Trajetória do veículo em relação ao duto. Para facilitar o visualização o eixo z foi invertido.



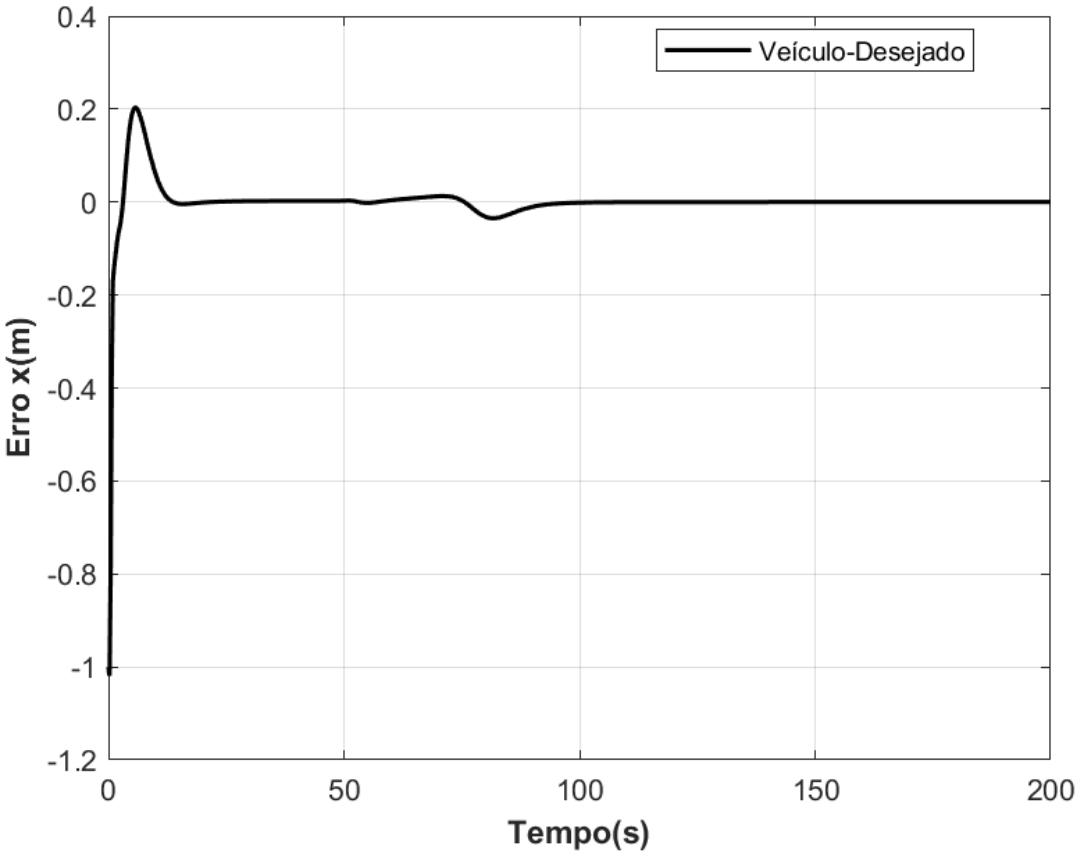
A Figura 43 apresenta a trajetória desejada em x, representada pela curva preta. Pode-se notar um erro nos instantes iniciais da simulação, o qual se origina da diferença entre a condição inicial do veículo em relação ao duto, como evidenciado na Figura 44.

Figura 43 – Deslocamento do veículo (curva vermelha) e deslocamento desejado (curva preta) em x.



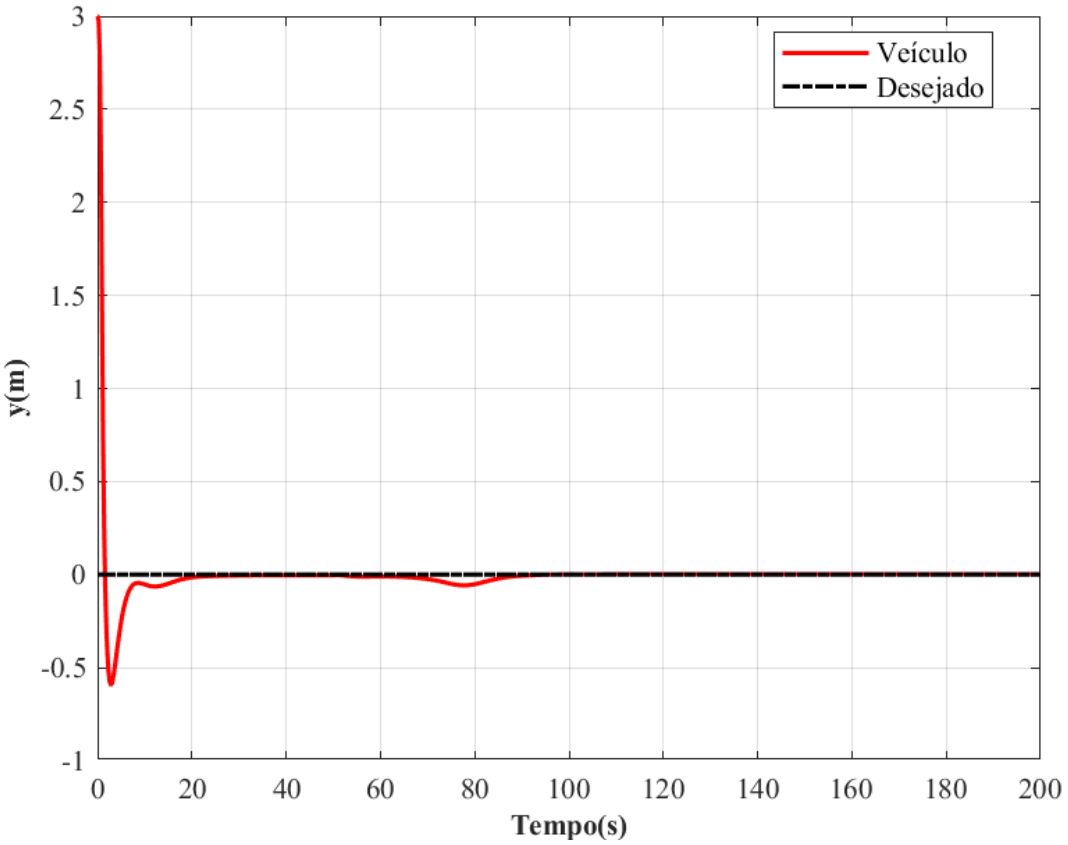
A Figura 44 apresenta o erro de deslocamento do veículo ao longo do eixo x. Pode-se notar uma oscilação no instante inicial e aos $80s$. O erro é devido a mudança de trajetória em z, que afetou o esforço dos propulsores, consequentemente, afetou o ângulo θ acarretando no erro em x.

Figura 44 – Erro entre o deslocamento do veículo e o desejado em x.



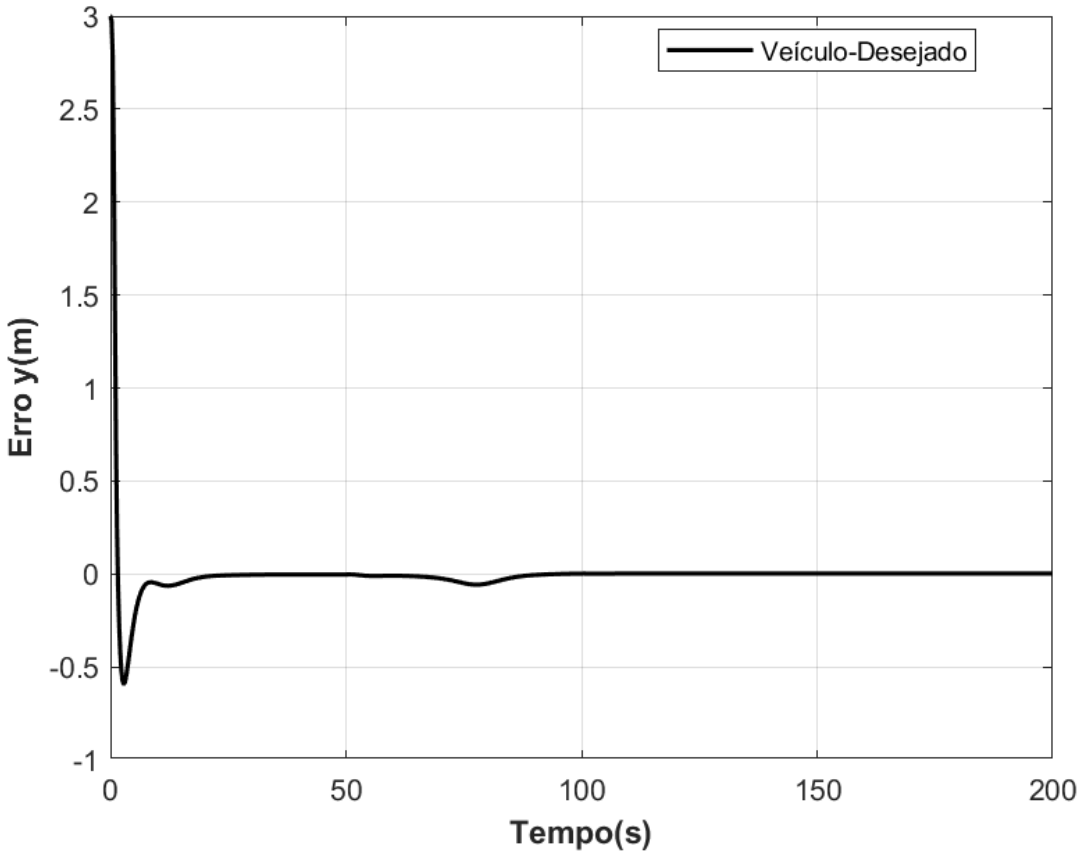
A Figura 45 apresenta a trajetória do veículo em y, representada pela curva vermelha. Pode-se observar que o veículo leva aproximadamente $20s$ para convergir à trajetória desejada. Esse tempo está relacionado aos ganhos do controlador. Aumentar os ganhos faz com que o veículo convirja mais rapidamente, mas, por outro lado, causa a saturação dos atuadores. Nesse caso, os propulsores atingiram a saturação nos primeiros dois segundos, devido ao erro na condição inicial.

Figura 45 – Deslocamento do veículo (curva vermelha) e o deslocamento desejado (curva preta) em y.



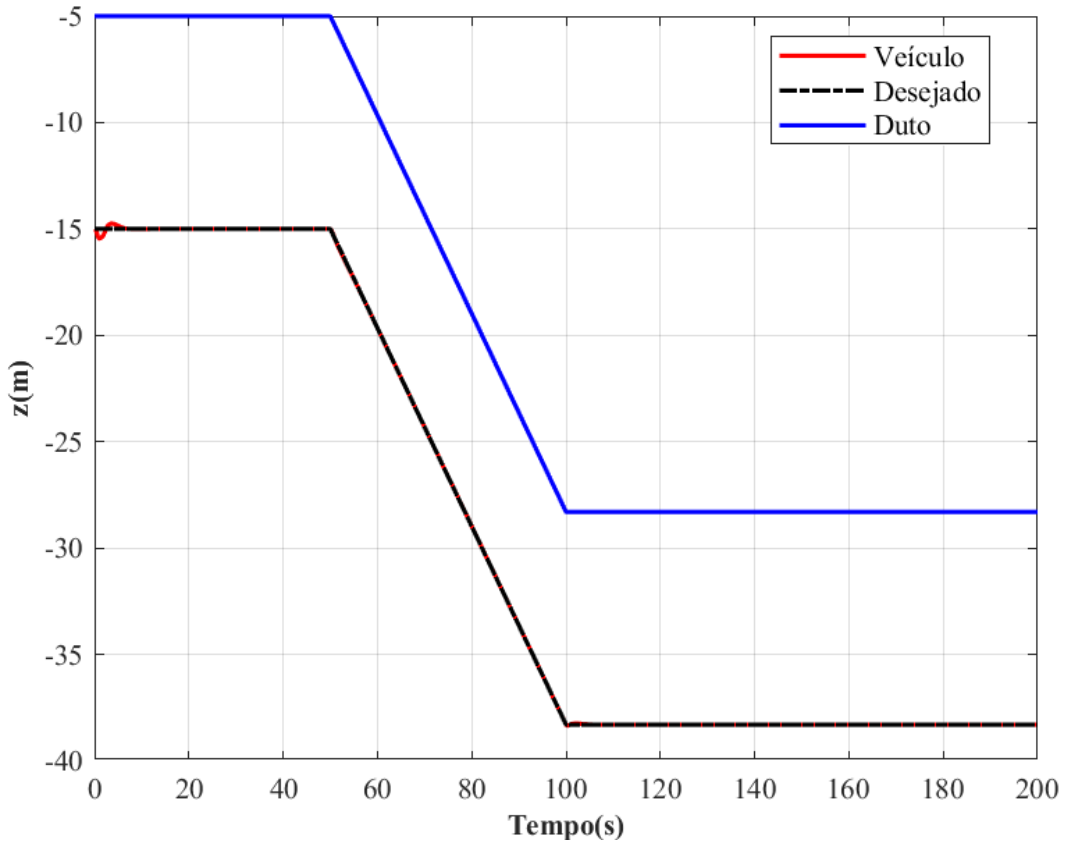
A Figura 46 apresenta o erro de deslocamento ao longo do eixo y. Assim como o deslocamento em x, a posição do veículo em y apresentou algumas oscilações, com amplitude máxima de $5cm$ nos $80s$ de simulação. O estado y leva um pouco mais de tempo para estabilizar em comparação com o estado x, isso ocorreu porque o ângulo ϕ não convergiu completamente para o valor desejado.

Figura 46 – Erro entre o deslocamento do veículo e o desejado em y.



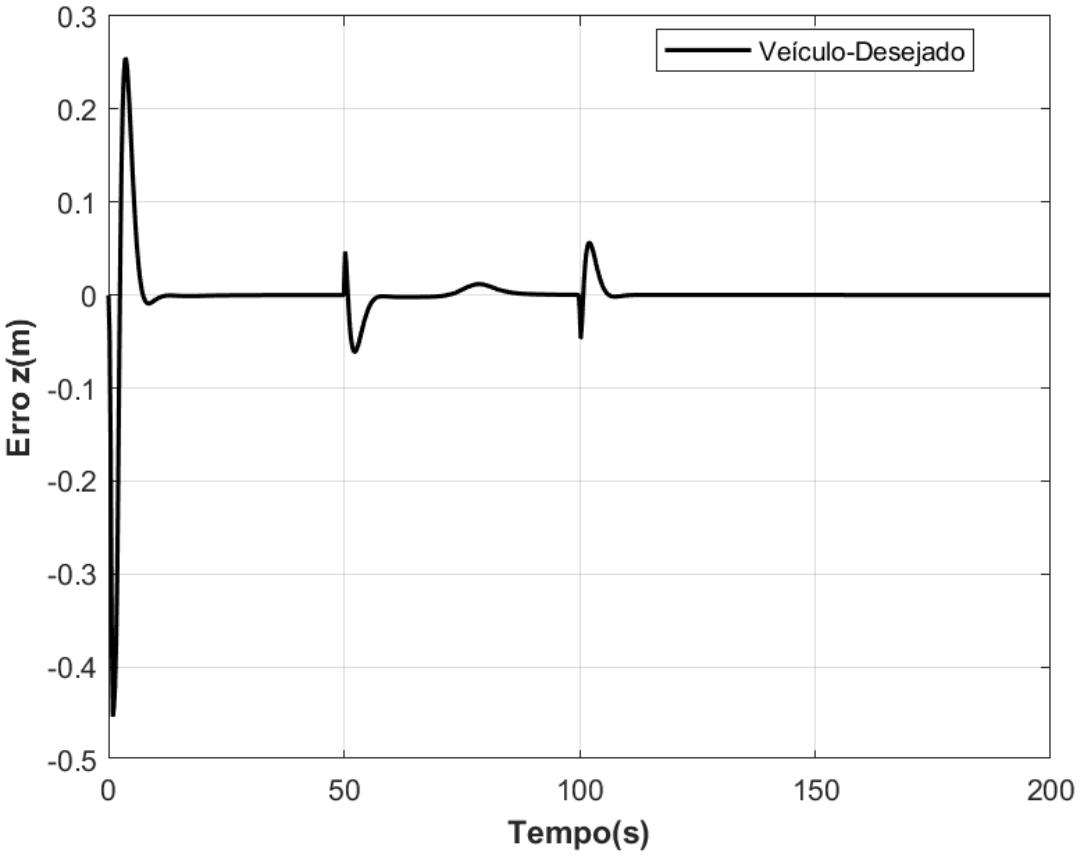
A Figura 47 apresenta a trajetória do veículo em z (curva vermelha), as bordas do duto (curva azul) e o deslocamento desejado (curva preta). Pode-se notar que o veículo manteve a distância vertical de $10m$ do duto.

Figura 47 – Posição do duto (curva azul), o deslocamento do veículo (curva vermelha) e o deslocamento desejado (curva preta) em z.



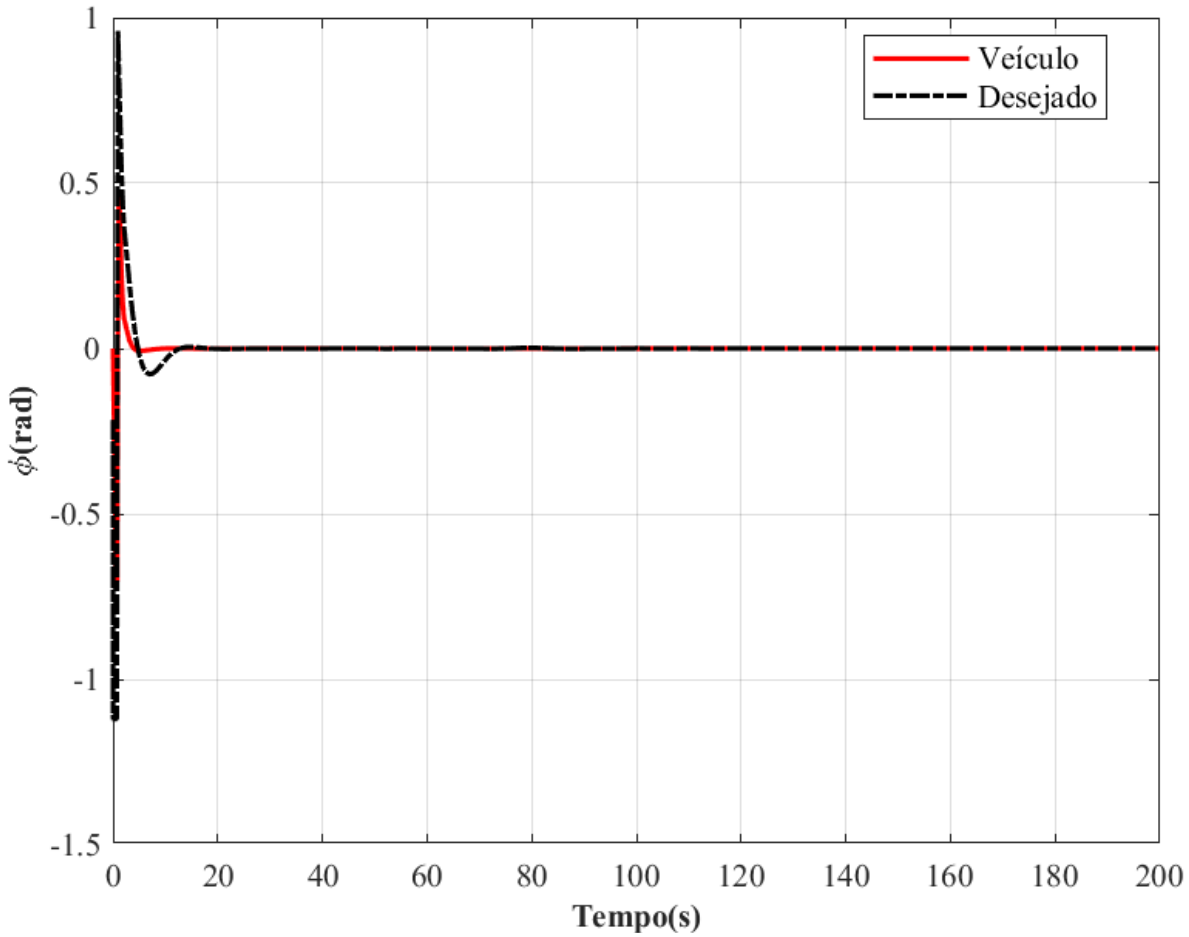
A Figura 48 apresenta o erro de deslocamento do veículo ao longo do eixo z. O valor máximo é de aproximadamente $6cm$, apresentando picos nos instantes de transição da trajetória do veículo ao longo do eixo z.

Figura 48 – Erro entre o deslocamento do veículo e o desejado em z.



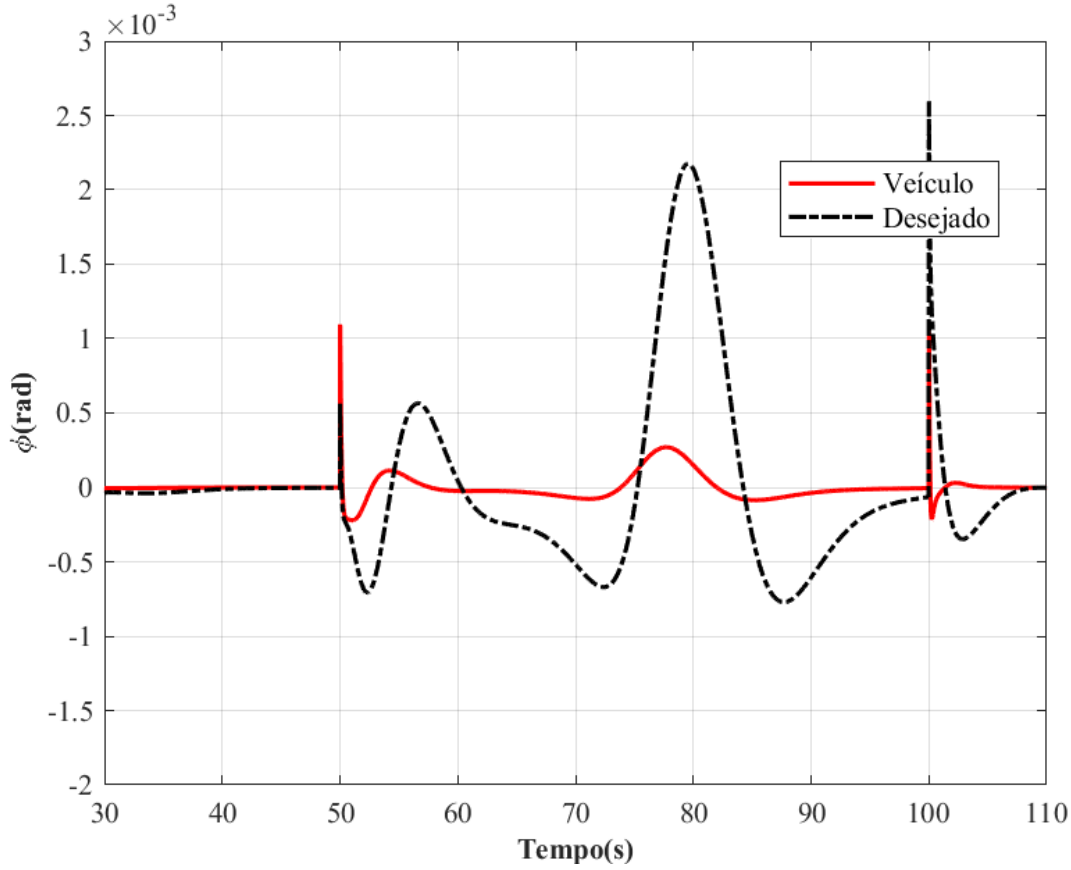
Na apresentação dos ângulos de Euler, ϕ e θ, é exibida uma visão mais detalhada no intervalo de simulação entre os $30s$ e os $110s$, em vez de apresentar um segundo gráfico com o erro. Todos os gráficos mostram uma oscilação nos momentos iniciais, decorrente do erro na condição inicial, como evidenciado na Figura 49, Figura 51 e Figura 53a.

Figura 49 – Comportamento do ângulo de *Euler*, ϕ. Ângulo do veículo (curva vermelha) e ângulo desejado (curva preta), resposta do controle de posição.



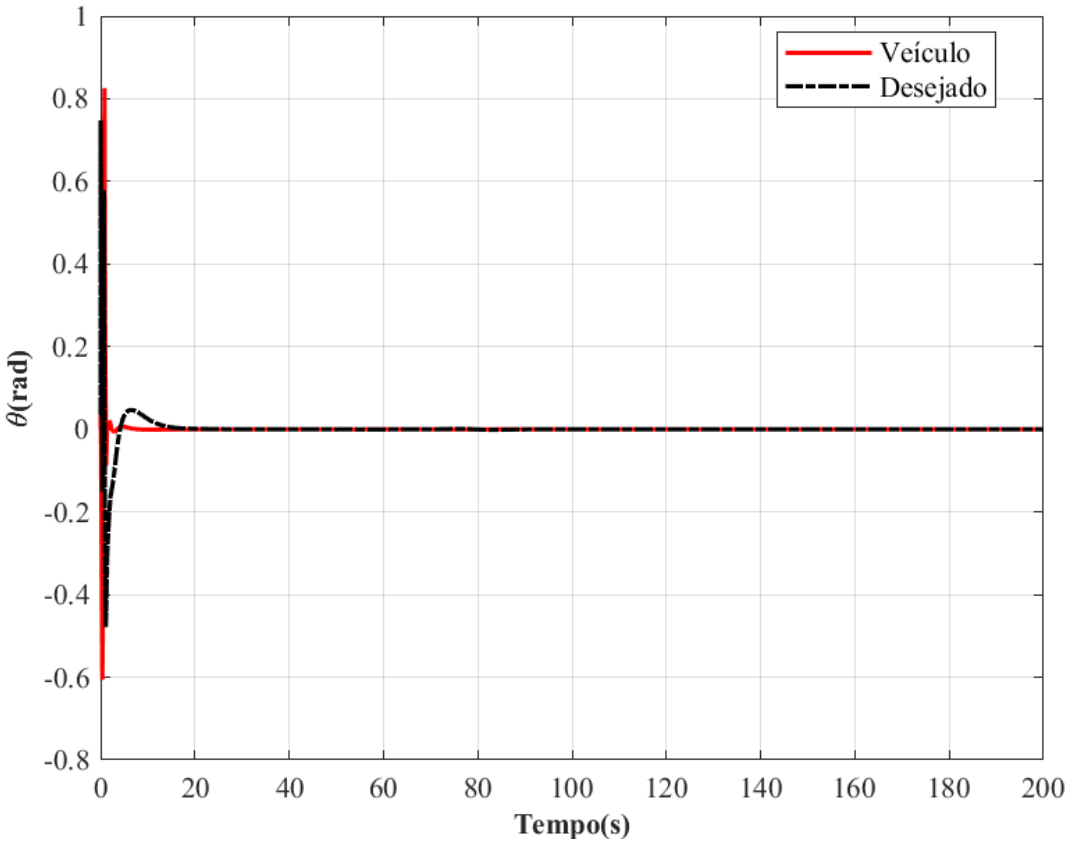
Pode-se notar que na Figura 50, o ângulo ϕ não convergiu para o valor desejado no intervalo de 50 segundos até 110 segundos, resultando em erros de aproximadamente $0,15°$ nos instantes de 80 segundos e 100 segundos. Esses erros podem ser resolvidos ajustando o ganho do controlador de atitude. Como os ganhos são constantes, esse ajuste pode levar à saturação dos atuadores no caso de uma mudança brusca na trajetória.

Figura 50 – Comportamento do ângulo ϕ, nos instantes finais da simulação.



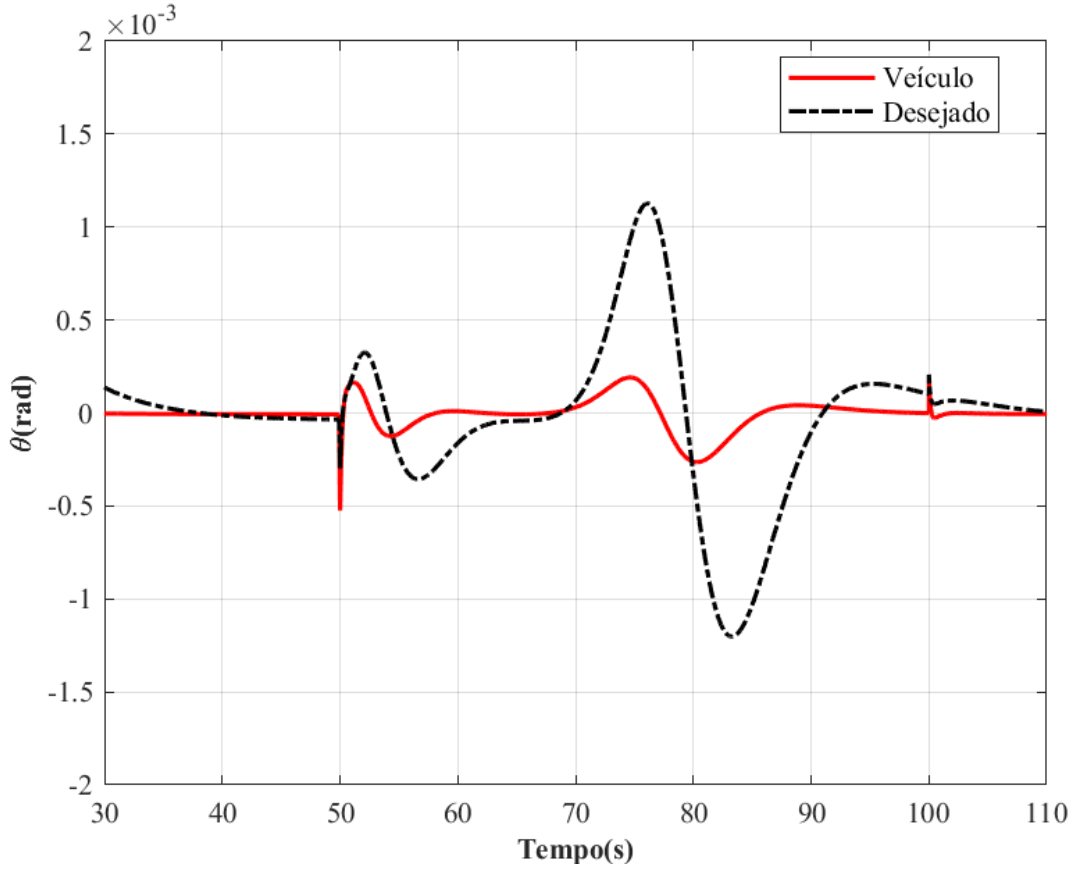
A Figura 51 apresenta o ângulo θ. Conforme dito anteriormente, nos instantes iniciais o sinal apresentou uma oscilação.

Figura 51 – Comportamento do ângulo de *Euler*, θ. Ângulo do veículo (curva vermelha) e ângulo desejado (curva preta), resposta do controle de posição.



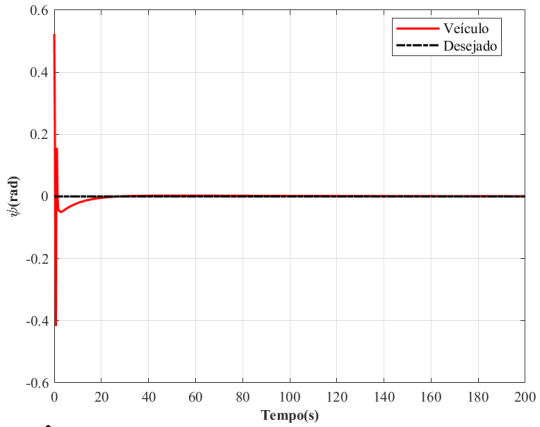
Pode-se notar na Figura 52 que, entre os instantes de $70s$ e $90s$ da simulação, o ângulo θ não convergiu para o valor desejado, apresentando um erro máximo de $0,090°$. Neste intervalo de tempo, o veículo estava se estabilizando após a mudança de trajetória ocorrida no instante de $50s$.

Figura 52 – Comportamento do ângulo θ, nos instantes finais da simulação.

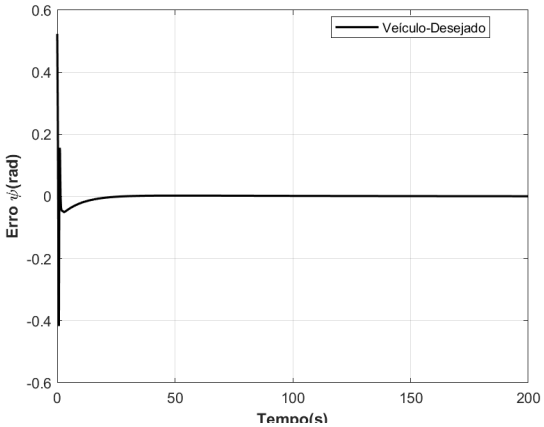


O ângulo ψ apresentou um erro inicial, com condição inicial igual a 28°, conforme ilustrado na Figura 53a. Após 20 segundos de simulação, o erro convergiu praticamente para zero, como demonstrado na Figura 53b.

Figura 53 – Comportamento do ângulo de *Euler*, ψ.



(a) Ângulo do veículo (curva vermelha) e ângulo desejado (curva preta).

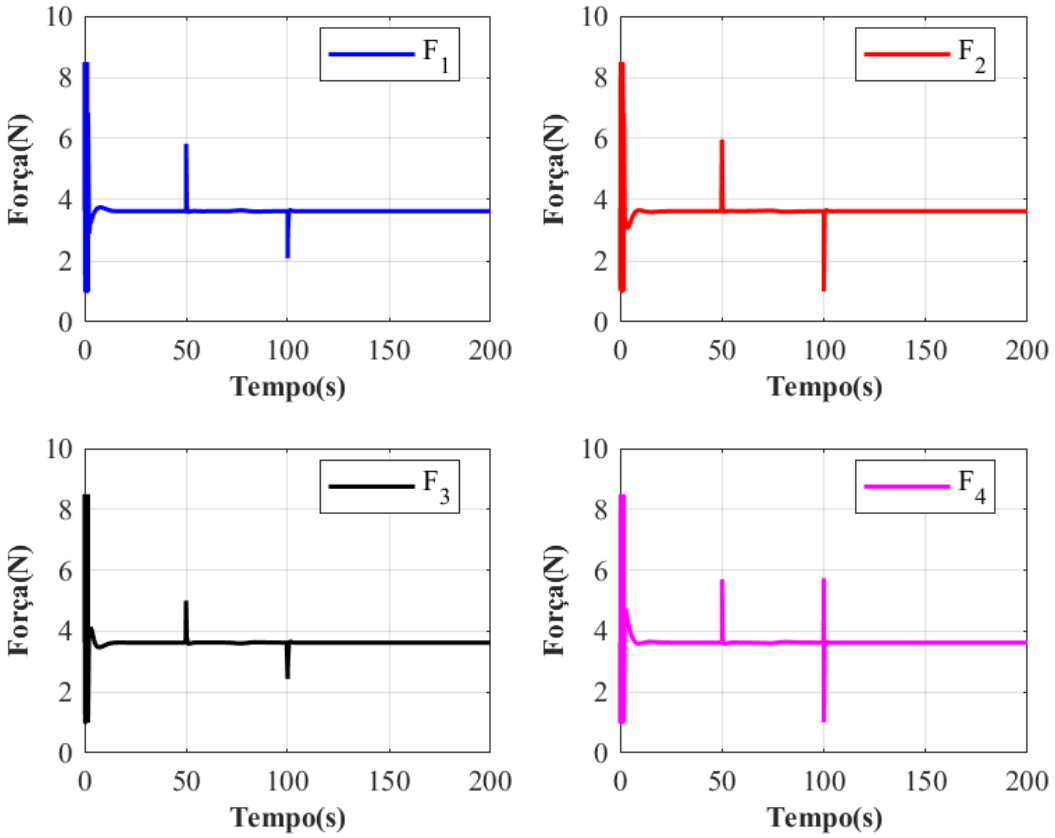


(b) Erro entre o deslocamento do veículo e o desejado em ψ.

A Figura 54 apresenta o empuxo requerido em cada um dos propulsores. Na maior parte do tempo, os esforços mantiveram-se próximos a $3,6N$, o que representa um quarto

do peso do veículo. Fora desta região, nos instantes em que o veículo fez mudança na trajetória em z e nos instantes iniciais devido ao erro de condição inicial.

Figura 54 – Esforço requerido em cada um dos quatro motores.



3.4 Discussão

Apesar dos erros de atitude e posição, pode-se dizer que o PBVS em conjunto com o controle desenvolvido são capazes de realizar o rastreamento do duto. O intuito é que o veículo siga o duto e que ele sempre permaneça dentro do campo de visão da câmera assim o veículo pode apresentar algum erro de rastreamento, desde que o duto permaneça no campo de visão da câmera

O esforço de controle apresenta algumas oscilações nos primeiros instantes, proveniente do erro de estado inicial. Estas oscilações demonstram que o controle projetado não é adequado para trajetórias descontinuas. Estas oscilações poderiam ser evitadas usando uma trajetórias sem descontinuidades ou adaptando os parâmetros do controlador, matrizes $\boldsymbol{K}$ e $\boldsymbol{K_A}$, de acordo com o erro entre o veículo e a trajetória desejada.

4 Identificação do Vazamento de Gás

Atualmente, existem inúmeros sensores comerciais capazes de detectar vazamento de gases como CH_4 e CO_2. Estes sensores podem ser divididos em resistivos, laser e por imagem óptica de gás (OGI - *Optical Gas Imaging*).

Os sensores do tipo resistivo são baseados em semicondutores de dióxido de estanho, por exemplo: Arduino sensor MQ-2. O sensor ao entrar em contato com o gás sofre alteração na sua condutividade e consequentemente altera a corrente de saída. A mudança da condutividade está relacionada à quantidade e o tipo de gás. Quanto maior a concentração de gás maior é a condutividade (SHI et al., 2016).

Já os sensores do tipo laser são baseados em um feixe de luz com um comprimento de onda conhecido. O feixe de luz em contato com o gás sofre uma alteração no comprimento de onda devido a absorção pelo gás. O dispositivo necessita de um emissor e um receptor para comparar a alteração no comprimento de onda. Esta alteração é associada com o tipo e quantidade de gás presente (WERLE, 1998).

Os sensores do tipo OGI são baseados em imagens em infravermelho (IR - *infrared*). Uma câmera capta imagens no espectro do infravermelho com comprimento de onda entre $0,5$ e 15 micrômetros. Semelhante ao laser, o gás absorve certo comprimento do infravermelho, que varia de acordo com o gás, afetando a imagem captada pela câmera (FLIR System, 2016).

O sensor do tipo resistivo precisa estar em contato com o gás, o que torna quase impossível o uso em um quadricóptero, pois as hélices dispersariam o gás. O sensor baseado em laser realiza medidas pontuais, fazendo com que o veículo necessite mais tempo de operação (BRETSCHNEIDER; SHETTI, 2014). Resta o sensor do tipo OGI que consegue realizar um mapeamento maior à cada sobrevoo do duto, sendo o sensor escolhido para embarcar no veículo.

Primeiro precisa-se detectar a presença de um vazamento. Utilizando o fragmento do vídeo apresentado no trabalho Ravikumar, Wang e Brandt (2017) em escala de cinzas. O vídeo foi editado, para que no início não apresente nenhum vazamento, conforme apresentado na Figura 55.

De acordo com Jadin e Ghazali (2014), a imagem gerada pela câmera pode ser convertida em RGB e depois convertida em escala de cinzas. A imagem em escala de cinzas é um vetor onde os dois primeiros índices referem à posição no plano da imagem e o último é a escala de cinza. O passo seguinte é converter está imagem em binário (binarização), onde a cor é transformada em valores lógicos (0 e 1). Na binarização é

Figura 55 – Quadros da filmagem, primeiro quadro sem vazamento, quadro do meio da filmagem com vazamento e último quadro com vazamento.

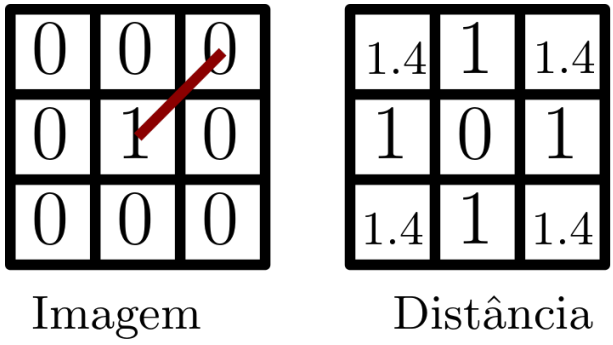
necessário escolher quanto porcento do fundo da imagem é mais escura ou mais brilhante que o objeto principal. Neste caso, define-se que o fundo da imagem é $0,8$ mais escuro que o objeto principal. Este valor deve ser ajustado previamente com base em alguma amostra, obtendo a Figura 56.

Figura 56 – Binarização dos quadros da filmagem.

A imagem ainda possui muita informação, assim escolhe-se uma ROI. Conforme detalhada nas seções anteriores, o duto estará no centro da imagem portanto, a ROI escolhida corresponderá à parte central da imagem.

O passo seguinte é calcular a distância média entre os *pixels* com valor 1 até o valor 0, conforme a Figura 57.

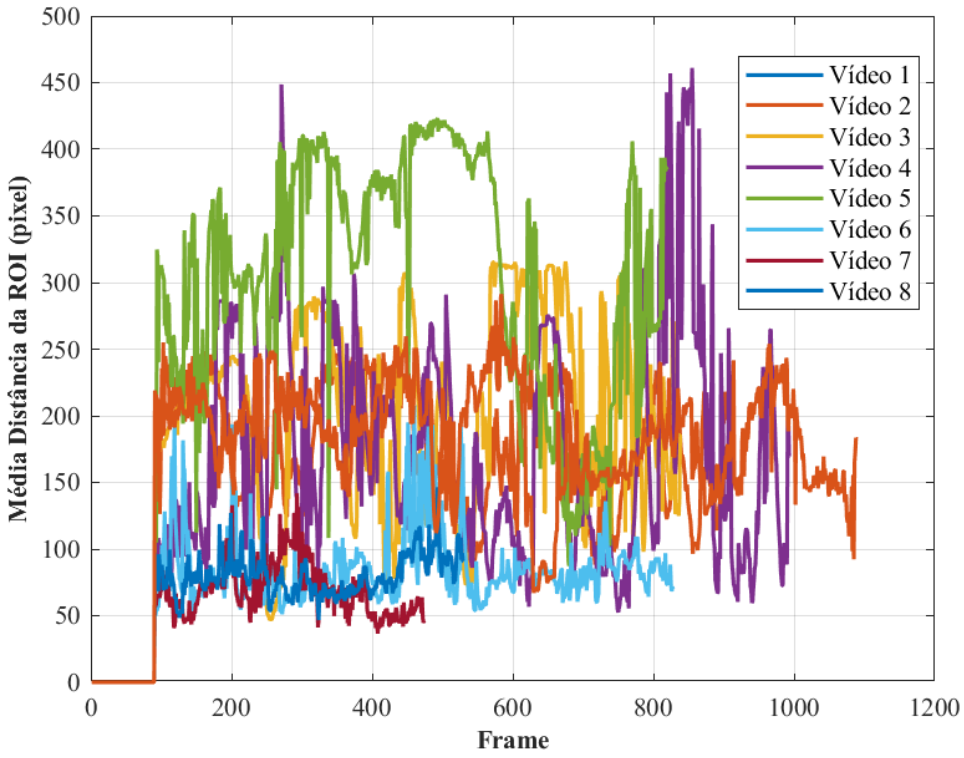
Figura 57 – Exemplo do cálculo da distância entre os *pixels* .



Por fim, pode-se plotar um gráfico da média das distâncias pelo número de quadros

da filmagem. No repositório encontrado (RAVIKUMAR; WANG; BRANDT, 2017), havia oito vídeos e em todos foram utilizados o mesmo processo, obtendo a Figura 58.

Figura 58 – Média da distância entre os *pixels* pelo número de quadros do vídeo.



Analisando a Figura 58, pode-se concluir que quando a média tiver um valor maior que 35 *pixel* o quadro possui um vazamento. Nos primeiros 90 quadros de todos os vídeos não há vazamento, pois este foi o trecho editado.

4.1 Identificação do Gás

A luz é uma onda eletromagnética com comprimento de onda que engloba o espectro visível, o infravermelho e mais outros espectros que não são relevantes aqui.

O espectro do infravermelho pode ser subdividido em três: onda curta (SWIR - *shortwave infrared*, com $1,0$ até $2,5$ micrômetros), onda média (MWIR - *midwave infrared*, com 3 até 5 micrômetros) e onda longa (LWIR - *longwave infrared*, 7 até 14 micrômetros) (SCAFUTTO et al., 2021).

O gás metano e o dióxido de carbono não são visíveis ao olho humano, ou seja não absorvem a luz visível, cujo o espectro visível tem comprimento de onda entre $0,4$ e $0,75$ micrômetros. A maioria dos materiais emitem e absorvem o comprimento de onda do infravermelho. O aumento do movimento vibracional e translacional dos átomos resulta no aumento da temperatura e a emissão do infravermelho, o contrário também é

válido, a absorção do infravermelho resulta em um aumento da temperatura (RESNICK; HALLIDAY; WALKER, 2016).

De acordo com o fabricante de câmeras, FLIR System (2016), gases de hidrocarbonetos como o metano absorvem o infravermelho próximo à $3,3$ e $7,0$ micrômetros. Já o CO_2 absorve o infravermelho $4,5$ e 15 micrômetros, conforme a Figura 59 e a Figura 60, obtidas do site *American National Institute of Standards and Technology* (NIST) (WEBBOOK, 2018).

Figura 59 – Comprimento de onda absorvida pelo metano.

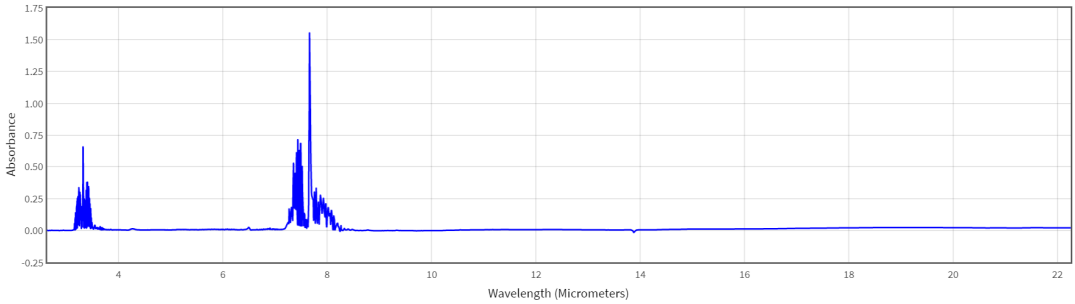
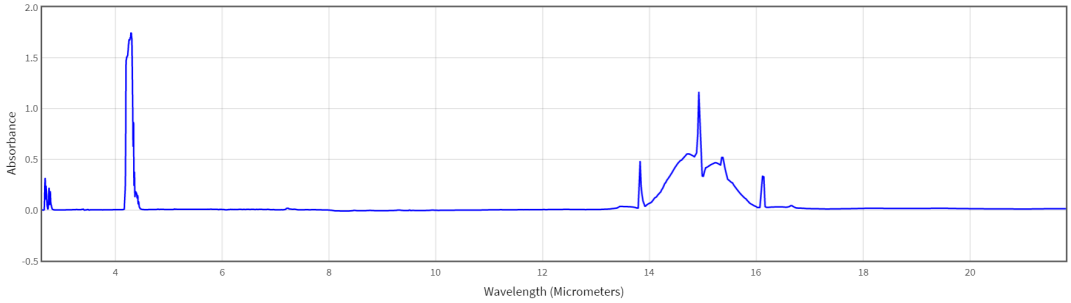


Figura 60 – Comprimento de onda absorvida pelo CO_2.



As câmeras infravermelhas medem a radiância e não a temperatura. No entanto, o *software* embarcado na câmera converte a radiância em temperatura usando a emissividade conhecida de um objeto alvo e aplicando os dados de calibração interna para a resposta espectral da câmera (FLIR System, 2012).

A emissividade de um material ($0 < \epsilon < 1$) é a capacidade de emitir energia eletromagnética, neste caso infravermelho. Esta propriedade é calculada com relação ao corpo negro na mesma temperatura. O corpo negro é um objeto ideal, que absorve e emite toda a radiação incidente ($\epsilon = 1$).

A distribuição espectral de radiação ao longo do comprimento de onda é dada pela lei de *Planck*, que afirma que a radiância emitida por um corpo negro, a uma temperatura absoluta T em função de um comprimento de onda λ é dada por:

$$I_{\mathrm{bb}}(\lambda, T) = \frac{2\pi hc^2\lambda^3}{e^{\left(\frac{hc\lambda}{\sigma T}\right)} - 1} \tag{4.1}$$

onde $\sigma = 5,669 \cdot 10^{-8} Wcm^{-2}K^{-4}$ é a constante de *Stefan-Boltzmann*, c é a velocidade da luz e $h = 6,62607015 \times 10^{-34} Js$ é a constante de *Planck*.

Existe uma relação (lei de *Wien*) simples entre a temperatura e o comprimento de onda, válida para $\lambda T < 5000(\mu mK)$ (BITTENCOURT, 2012),

$$\lambda_{\max} = \frac{2,897}{T}. \tag{4.2}$$

A radiância real do gás I_g, com emissividade ϵ_g, comprimento de onda λ e temperatura T_g, pode ser escrita em função da radiância do corpo negro,

$$I_g(\lambda, T_g) = \epsilon_g(\lambda) I_{\text{bb}}(\lambda, T_g) \tag{4.3}$$

Portanto, a temperatura aparente de cada *pixel* pode ser estimada,

$$T = \frac{hc\lambda}{\sigma \log\left(\frac{2c^2\lambda^3}{I(\lambda)} + 1\right)} \tag{4.4}$$

onde $I(\lambda)$ é a radiância para o *pixel* em teste.

A maioria dos objetos possuem uma emissividade constante em uma determinada temperatura. Para objetos que interagem com a faixa espectral do infravermelho, a emissividade varia em função do comprimento de onda. Assim, dois objetos com a mesma temperatura mas com emissividade diferentes apresentam temperaturas diferentes, na câmera (BÄHRECKE, 2015). Portanto, a emissividade precisa ser estimada. A maioria das câmeras térmicas possuem tabelas com valores de emissividade, onde o operador define o valor.

Aplica-se a lei de *Kirchhoff*, onde a emissividade e absortividade são iguais para qualquer objeto em equilíbrio térmico. A estimativa é feita comparando o objeto com o fundo da imagem. De acordo com Kastek, Piatkowski e Trzaskawka (2011), a estimativa da emissividade de cada *pixel* , pode ser escrita como:

$$\epsilon(\lambda) = \epsilon_g(\lambda)\Delta T + \tau_g(\lambda)\epsilon_b(\lambda) \tag{4.5}$$

onde ΔT é a temperatura aparente entre o gás e o fundo da imagem, $\tau_g(\lambda)$ é a transmissão do gás e $\epsilon_b(\lambda)$ é a emissividade do fundo da imagem.

Para cada *pixel* da imagem a emissividade é calculada da seguinte maneira: a temperatura de cada *pixel* conforme a Equação (4.4), depois a radiação de corpo negro pela Equação (4.1), obtendo a emissividade do gás, ϵ_g, pela Equação (4.3) e por fim aplica-se a Equação (4.5). Assim pode-se realçar a plume de gás de acordo o tipo de gás presente.

4.2 Discussão

O algoritmo desenvolvido para a detecção de gases demonstrou eficácia notável com base nos vídeos utilizados como parte da validação experimental. A calibração meticulosa dos parâmetros do algoritmo foi conduzida considerando a aquisição de imagens específicas, no entanto, é imprescindível ressaltar que persiste a necessidade para a condução de testes adicionais em variadas condições operacionais. Tais testes deverão abranger a avaliação do desempenho do algoritmo sob diferentes cenários, contemplando variações na taxa de fluxo dos gases, nas características dos gases em análise, bem como nas condições atmosféricas reinantes, a fim de se estabelecer uma rotina que assegure detecções de vazamentos com a máxima precisão.

Cabe ressaltar que, até o presente momento, a identificação precisa do tipo de gás em questão não pôde ser realizada. A limitação observada decorre, em grande parte, da insuficiência das informações contidas na filmagem disponibilizada, a qual foi adquirida em formato JPEG. É válido salientar que, comumente, os registros de câmeras térmicas são armazenados em formatos como RJPEG ou equivalentes, nos quais são incorporados os dados relacionados à radiância e/ou temperatura associados a cada pixel, além das informações de vídeo necessárias para análises mais detalhadas.

5 Montagem do Veículo e Simulação no software ArduPilot

Após extensa pesquisa, especialmente no que diz respeito aos aspectos relacionados aos preços, foi possível identificar um conjunto integral de componentes destinados à construção de um veículo da categoria quadricóptero. Este conjunto engloba todos os elementos essenciais necessários à montagem do veículo, conforme ilustrado na Figura 61. Os componentes, tais como os motores, as hélices e os cabos, estão detalhadamente apresentados na Figura 61a. Por sua vez, a placa de controle encontra-se representada na Figura 61b. A estrutura adquirida para a construção do veículo corresponde ao modelo ZD550, confeccionada em fibra de carbono e com uma extensão de $550mm$, conforme demonstrado na Figura 61c.

Figura 61 – Conjunto adquirido para a montagem do quadricóptero.

(a) Acessórios, motores, hélice e cabos do quadricóptero.

(b) Placa de controle Pixhawk.

(c) Estrutura de fibra de carbono modelo ZD550.

O sistema é composto por uma bateria com uma tensão nominal de $14,8$ V e capacidade de 4000 mAh, conforme Figura 62a. Além disso, inclui um controle de radiofrequência, o RadioLink AT10, com 12 canais de comunicação, e os receptores correspondentes, conforme Figura 62b.

Figura 62 – Bateria de $14,8V$, 4000mAh e controle de radio frequência.

(a) Bateria $14,8V$ e 4000mAh.

(b) Controle radio frequência, RadioLink AT10.

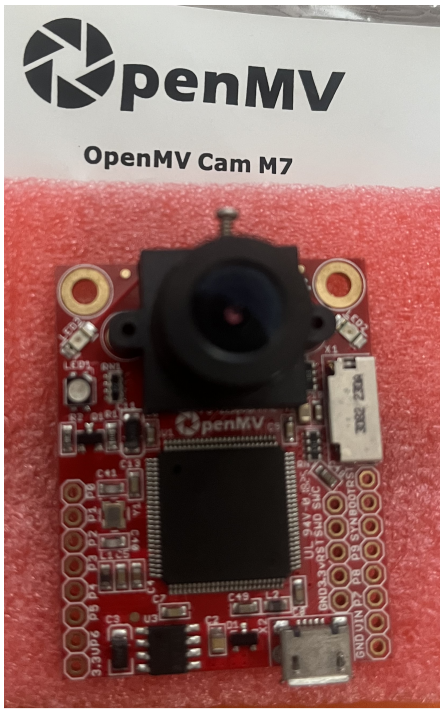
A montagem do veículo, juntamente com a incorporação dos acessórios, da placa de controle Pixhawk, da bateria e dos receptores do controle de radiofrequência, foi executada conforme ilustrado na Figura 63. Dessa forma, o veículo torna-se apto a realizar voos controlados por um operador, uma vez que a placa Pixhawk opera em conjunto com o software embarcado ArduPilot (ArduPilot, 2023). O usuário deve efetuar o download do programa a partir do sítio eletrônico, armazenando-o em um cartão de memória do tipo micro SD e procedendo de acordo com as etapas indicadas. Este software possibilita o controle do veículo e a execução de determinadas tarefas de maneira autônoma, tais como a realização de pousos, decolagens e movimentos circulares.

Figura 63 – Montagem do veículo.

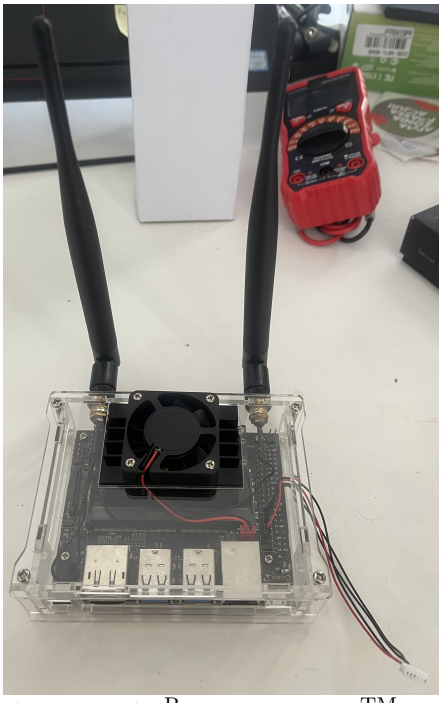
Para a captura das imagens do duto, foi adquirida a câmera OpenMV Cam H7, conforme ilustrado na Figura 64a. Este modelo de câmera estava disponível de projetos

anteriores. A mencionada câmera é equipada com um processador STM32H743VI ARM Cortex M7 e opera com algoritmos implementados em Python, conforme documentado por (OpenMV, 2023). A intenção inicial era utilizar a câmera para identificação e geração de trajetórias; no entanto, não foi possível estabelecer comunicação eficaz entre a câmera e a placa Pixhawk. Consequentemente, optou-se pela aquisição de uma placa de desenvolvimento Nvidia^{R} Jetson Nano^{TM}. Adicionalmente, para simplificar a comunicação com a placa, adquiriu-se um módulo *wifi*. O conjunto resultante, composto pela placa e pelo módulo *wifi*, juntamente com as duas antenas, é ilustrado na Figura 64b.

Figura 64 – Placa Nvidia^{R} Jetson Nano^{TM} e modulo wifi. Câmera OpenMV Cam H7



(a) Câmera OpenMV Cam H7.



(b) Placa Nvidia^{R} Jetson Nano^{TM} + modulo *wifi* + capa acrílica.

A placa Pixhawk desempenha um papel crucial no funcionamento do veículo autônomo, abrangendo diversas funcionalidades, tais como a leitura dos sensores, a estimação dos estados utilizando um filtro de *Kalman*, o controle de voo, a execução de códigos e tarefas pré-definidas, como operações de pouso e decolagem. É nesse dispositivo que o software ArduPilot é executado.

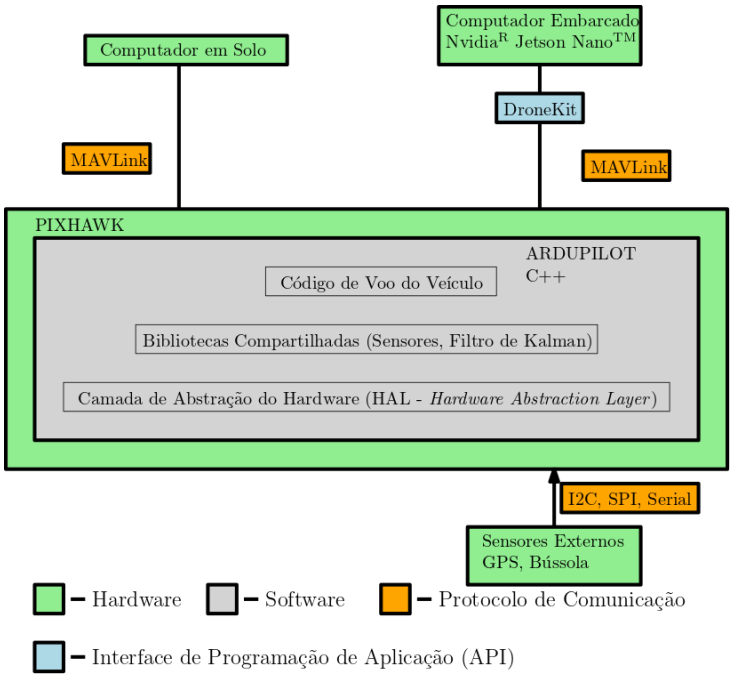
O ArduPilot é um projeto de piloto automático de código aberto que goza de uma vasta comunidade online e, portanto, está em constante evolução. Seu primeiro repositório foi estabelecido em 2009 e foi implantado em mais de um milhão de veículos, abrangendo uma diversidade de tipos de veículos, incluindo aeronaves convencionais, planadores, multirotores, helicópteros, veleiros, embarcações motorizadas, submarinos e veículos terrestres. Importante notar que o ArduPilot não fabrica hardware específico, sendo compatível com uma ampla variedade de placas, graças à sua camada de abstração

de hardware (HAL - sigla em inglês para *Hardware Abstraction Layer*), responsável por adaptar o código ao hardware subjacente. Essa característica demonstra a confiabilidade do sistema, que foi amplamente testado (ArduPilot, 2023).

O protocolo de comunicação adotado para a troca de informações é o MAVLink, um protocolo de mensagens extremamente leve, otimizado para comunicação com drones e entre os componentes integrados de drones. O MAVLink segue um padrão híbrido de publicação-assinatura e ponto a ponto: os dados são transmitidos e publicados como tópicos, enquanto os subprotocolos de configuração, como o protocolo de missão ou o protocolo de parâmetros, operam no modo ponto a ponto com suporte a retransmissão (Dronecode Project, 2023).

O esquema representado na Figura 65 demonstra a arquitetura de comunicação entre as placas Pixhawk e Jetson Nano™. A Interface de Programação de Aplicação (DroneKit) é responsável por converter os códigos escritos em Python para o formato MAVLink, facilitando a comunicação e o controle do veículo autônomo.

Figura 65 – Esquema de comunicação do veículo com as placas Pixhawk e Nvidia® Jetson Nano™.



5.1 Adição do controle *backstepping* com modos deslizantes no ArduPilot

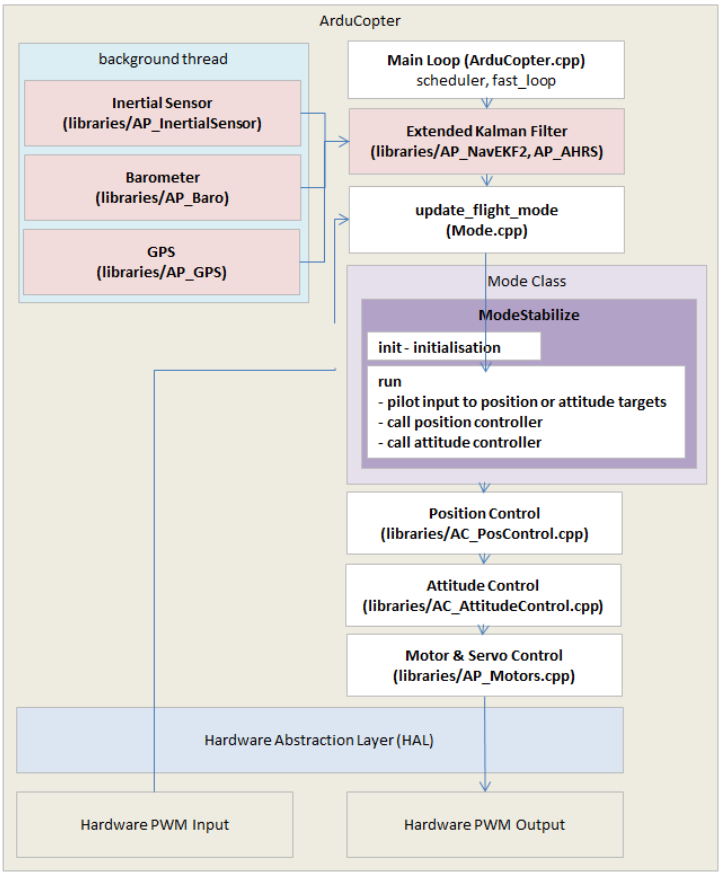
O ArduPilot possui algumas funções pré-definidas, estas funções são chamadas de modos de voo (MV) e estão declaradas dentro do arquivo («*mode.cpp*») (ArduPilot Dev. Team, 2023). Atualmente possui 23 MV no total que são:

- STABILIZE: o veículo fica neste modo esperando novos comandos;
- ACRO: permite ao usuário controle a velocidade angular do veículo;
- ALT_HOLD: estabiliza a altura do veículo;
- AUTO: seguimento automático de pontos fornecidos pelo usuário;
- GUIDED: seguimento automático de posição e/ou velocidade fornecidas pelo usuário via MAVLink;
- LOITER: controle automático das acelerações no referencial móvel;
- RTL: retorno automático para o ponto de decolagem;
- CIRCLE: voa em circulo com raio definido pelo usuário;
- LAND: pouso automático controlando a posição horizontal;
- DRIFT: controle de posição semi-automático controlando rotação em torno do eixo z;
- SPORT: controle automático das acelerações no referencial inercial;
- FLIP: rotação completa em torno do eixo x;
- AUTOTUNE: calculo automático dos parâmetros do controle de rotação em torno dos eixos x e y;
- POSHOLD: mantém a posição atual;
- BRAKE: caso de alguma falha o veículo para e mantém-se parado;
- THROW: caso ocorra algum problema como bateria fraca o veículo retorna ao ponto de decolagem;
- AVOID_ADSB: evita obstáculos, necessário alguns sensores;
- GUIDED_NOGPS: voo sem o GPS, mas somente controle de atitude e altitude;
- SMART_RTL: retorno para o ponto de decolagem mas seguindo o mesmo caminho percorrido;
- FLOWHOLD: mantém a posição atual com ajuda do sensor óptico;
- FOLLOW: segue outro veículo ou uma estação no solo;
- ZIGZAG: voa de um ponto até outro, percorrendo um área predefinida;
- SYSTEMID: identificação automática do sistema e dos sinais de controle;

- AUTOROTATE: rotação automática em torno do eixo z.

Durante a execução de um software, uma das primeiras verificações consiste em determinar qual Modo de Voo (MV) está ativo, uma vez que, dependendo do MV ativo, outros arquivos, funções e bibliotecas são empregados em uma sequência específica. Por exemplo, a Figura 66 ilustra um esquema do MV denominado STABILIZE, que é o estado no qual o veículo se encontra ao ser inicializado. O veículo demanda alguns minutos para se preparar para a decolagem, visto que requer a ativação das bibliotecas relacionadas aos sensores, sendo que o GPS demanda um tempo adicional para iniciar a coleta de dados. O passo subsequente envolve a invocação das bibliotecas de controle de posição («*AC_PosControl.cpp*») e de atitude («*AC_AttitudeControl.cpp*»). Adicionalmente, a biblioteca responsável pelo acionamento dos motores («*Ap_MotorsMatrix.cpp*») desempenha a função de converter a força vertical e os momentos de rotação em forças individuais para cada um dos motores (ArduPilot Dev. Team, 2023).

Figura 66 – Esquema de funcionamento do ArduPilot, modo de voo STABILIZE.

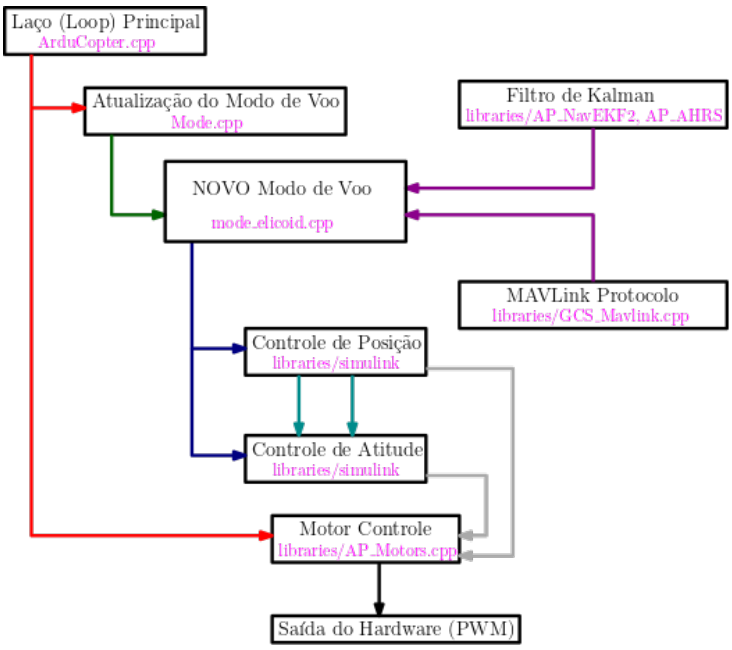


Fonte: ArduPilot Dev. Team (2023)

Assim, desenvolveu-se um novo MV denominado ELICOID. O principal objetivo deste MV consiste em receber a trajetória desejada do computador embarcado, adquirir a posição e a atitude atual da embarcação, calcular o erro de posição e atitude e, em

seguida, encaminhá-los para o controlador de posição e atitude, que por sua vez envia os comandos necessários à biblioteca de controle do motor. De maneira simplificada, o esquema do Modo ELICOID é apresentado na Figura 67 e o código correspondente pode ser encontrado no Apêndice D.

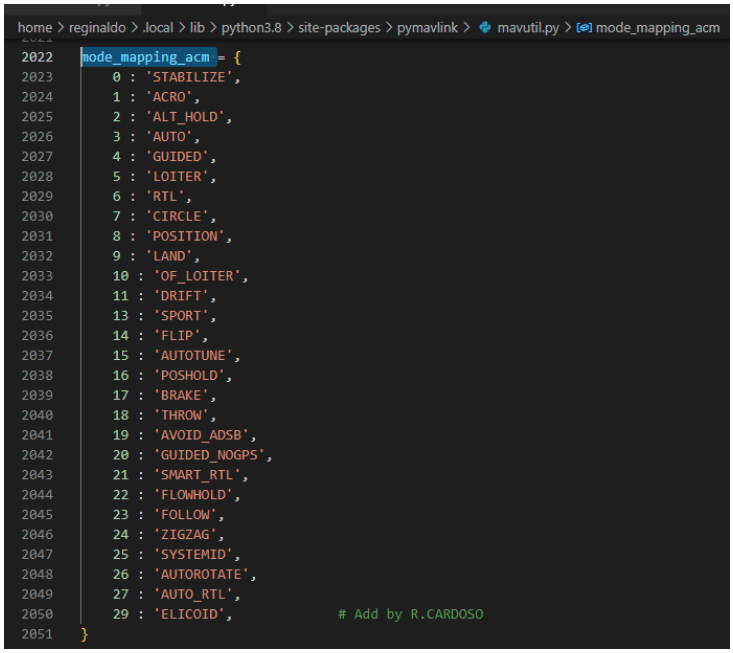
Figura 67 – Esquema de funcionamento do modo de voo criado.



O MV ELICOID precisa ser declarado no arquivo («*mode.h*»), conforme Apêndice E. Criou-se algumas varáveis para comutar o controle e o tipo de simulação. Onde as variáveis *"SwitchVariable_APC"* e *"SwitchVariable_AIC"* representa a simulação com os controles PID e o integral *backstepping* com modos deslizantes, já as variáveis *"SwitchVariable_EPC"* e *"SwitchVariable_EIC"* a mesma simulação mas com entrada externa da trajetória desejada.

O novo MV precisa ser declarado também dentro da variável («*mode_mapping_acm*») no arquivo localizado na pasta («*'(local de instalação do python)'/python3.8/site-packages/pymavlink/mavutil.py*»), conforme Figura 68. Assim o MV fica visível dentro do protocolo MAVLink e pode ser habilitado por programas externos.

Figura 68 – Novo modo de voo adicionado dentro da variável "mode_mapping_acm".



O único MV habilitado para receber informações externas via protocolo MAVLink é o modo de operação GUIDED, sendo que a biblioteca encarregada dessa comunicação é a «*GCS_Mavlink.cpp*>". Dessa forma, para possibilitar que o MV ELICOID receba informações de posição, velocidade, ângulo e velocidade angular em torno do eixo z, foi necessário efetuar um total de 22 modificações na referida biblioteca, conforme detalhado no Apêndice E.

O passo subsequente consiste na conversão do controlador desenvolvido na seção 2.2 do ambiente Simulink^{R} para o código em linguagem C++. O próprio Simulink^{R} dispõe de uma biblioteca denominada Simulink^{R} Coder, que realiza a conversão de arquivos para as linguagens C ou C++.

Como resultado, foram geradas duas novas bibliotecas em C++, uma para o controle de posição («*PosContIBSMC.cpp*>") e outra para o controle de atitude («*AttContIBSMC.cpp*>") do veículo, conforme detalhado no Apêndice F. Ambas as bibliotecas apresentam uma função principal denominada «*step*>", responsável pela atualização do controle.

O algoritmo implementado em linguagem Python é responsável por enviar a trajetória desejada ao veículo e seu código pode ser encontrado no Apêndice G.

5.2 Simulação

Para a simulação das modificações implementadas no ArduPilot, empregou-se o SITL (*Software in the Loop*), um simulador que abrange uma variedade de tipos de veículos

e viabiliza a compilação do ArduPilot sem a necessidade de hardware físico. Para efetuar a simulação, é imprescindível atualizar a biblioteca («*SIM_Frame.cpp*») com os parâmetros físicos do veículo, como massa e momento de inércia.

O modelo dos propulsores empregados no ArduPilot é normalizado, com variação de 0 até 1 para as forças geradas e de -1 a 1 para os momentos (ArduPilot Dev. Team, 2021a), conforme apresentado no Apêndice B.

O ArduPilot incorpora um conjunto de controladores PID (*proportional integral derivative*) em cascata para o controle de quadricópteros. Um controlador PID é empregado para o plano horizontal, outro para o plano vertical, e um terceiro para o controle da atitude. Com o objetivo de testar o controle desenvolvido sem considerar influências do solo, o pouso e a decolagem do veículo foram realizados utilizando o controle do ArduPilot. Dessa forma, o veículo decola, e quando atinge uma altitude de 10 metros, o controle desenvolvido executa o rastreamento da trajetória desejada,

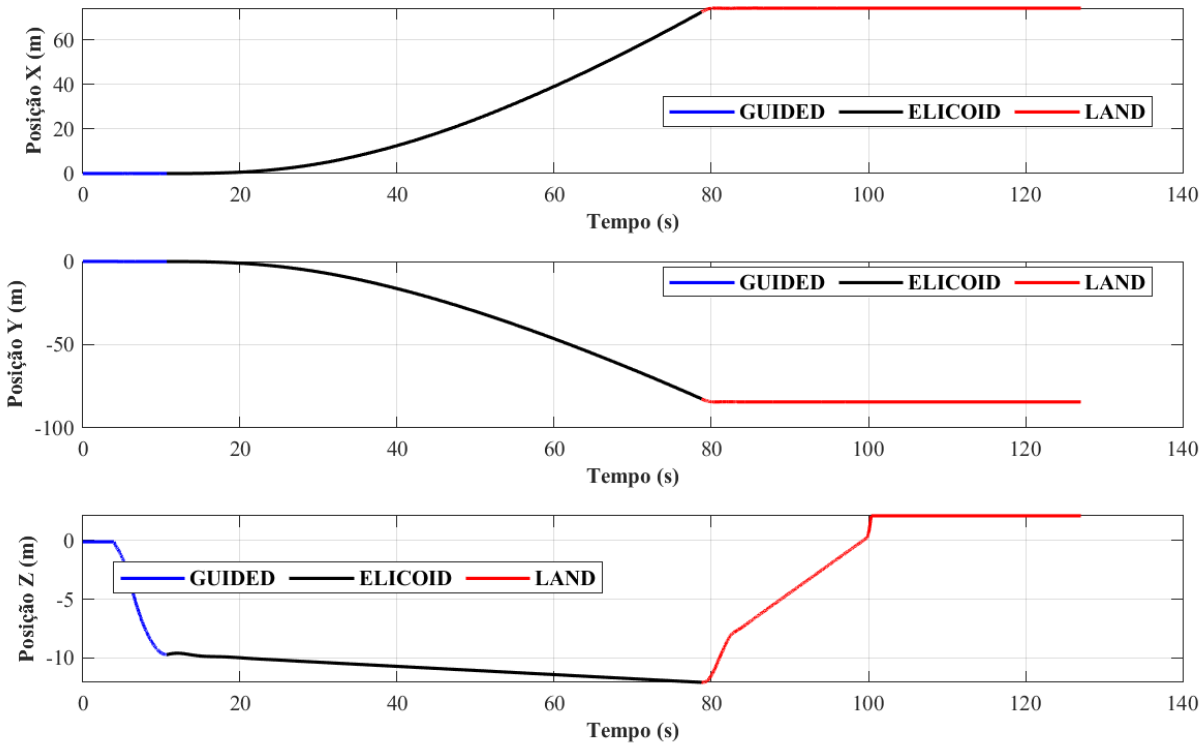
$$\begin{aligned} x_d &= x_0 + (5sen(0,2t)) \\ y_d &= y_0 + ((5cos(0,2t)) - 5) \\ z_d &= -z_0 + (0,05t) \\ \dot{x}_d &= cos(0,2t) \\ \dot{y}_d &= -sin(0,2t) \\ \dot{z}_d &= 0,05 \end{aligned} \tag{5.1}$$

onde x_0, y_0 e z_0 é a posição do veículo após a decolagem.

5.2.1 Simulação: decolagem, sobrevoo e pouso

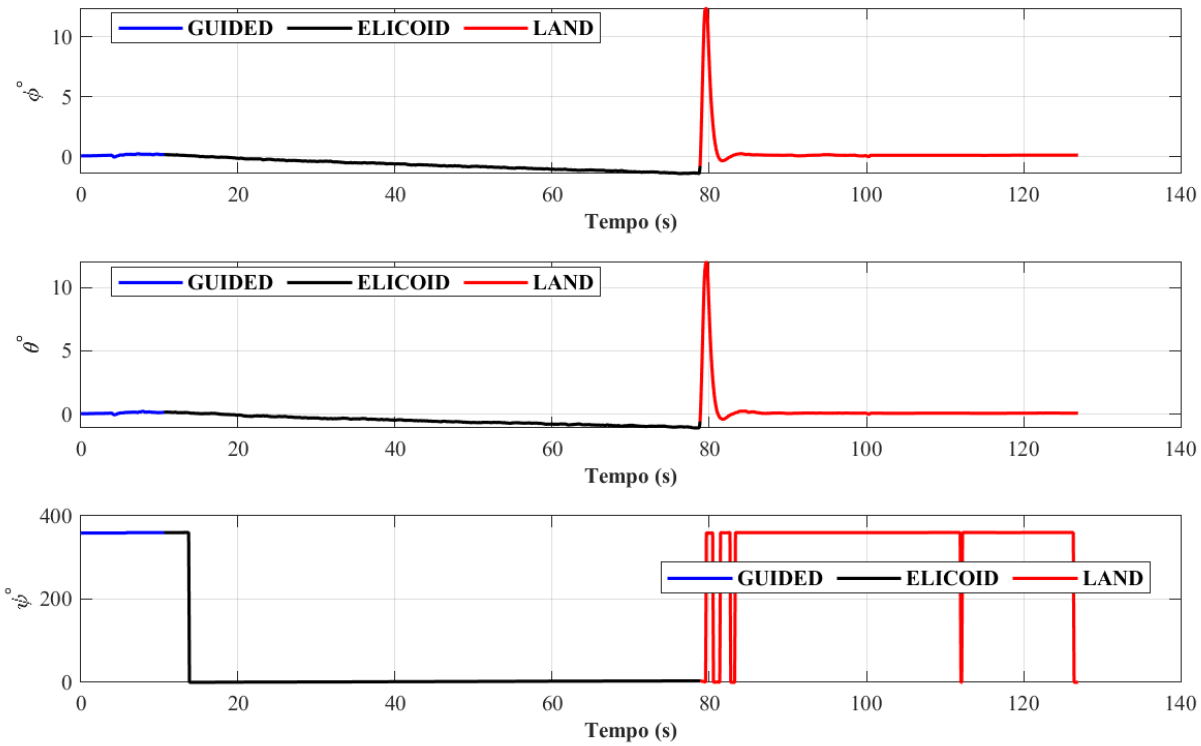
A Figura 69 ilustra a trajetória completa do veículo, abrangendo desde o momento da decolagem até a fase de pouso. A curva em azul representa a decolagem do veículo, a qual foi simulada utilizando o controlador ArduPilot no modo GUIDED. A curva em preto corresponde ao acompanhamento da trajetória desejada utilizando o controlador proposto no modo ELICOID. Por fim, a curva em vermelho retrata a fase de pouso do veículo, a qual foi controlada pelo ArduPilot no modo LAND.

Figura 69 – Posição do veículo durante a decolagem (modo GUIDED, curva em azul), o deslocamento desejado (modo ELICOID, curva em preto) e o pouso (modo LAND, curva em vermelho).



A Figura 70 mostra a variação completa da atitude do veículo, desde o momento da decolagem até seu pouso. Observa-se que a atitude do veículo apresenta variações mínimas, de menos de 2°, nos ângulos ϕ e θ. No entanto, em relação ao ângulo ψ, observa-se uma variação abrupta, indo de 360° a 0° praticamente instantaneamente, permanecendo em 0° pelo restante da simulação com o controle desenvolvido, antes de retornar a 360° ao entrar no modo LAND.

Figura 70 – Atitude do veículo durante a decolagem (modo GUIDED, curva em azul), o deslocamento desejado (modo ELICOID, curva em preto) e o pouso (modo LAND, curva em vermelho).



A Figura 71 ilustra a evolução do esforço de controle do veículo, desde o momento da decolagem até o momento da aterrissagem. É observável que os três primeiros esforços de controle, denominados momentos Mx, My, e Mz, foram concebidos e forçados por meio de saturação para variar no intervalo de -1 a 1, conforme documentado em (ArduPilot Dev. Team, 2021b) a placa suporta somente valores neste intervalo. Contudo, a análise revela que tal premissa não foi concretizada. Destaca-se, em particular a Figura 72, onde os momentos Mx e My, cujo valores aproximaram consideravelmente de 100, pois nesta simulação não foram forçados por meio de saturadores. Por outro lado, o esforço de controle Fz oscilou dentro da faixa de operação especificada, que abrange os valores de 0 a 1.

Figura 71 – Esforço normalizado e saturado do veículo durante a decolagem (modo GUIDED, curva em azul), o deslocamento desejado (modo ELICOID, curva em preto) e o pouso (modo LAND, curva em vermelho).

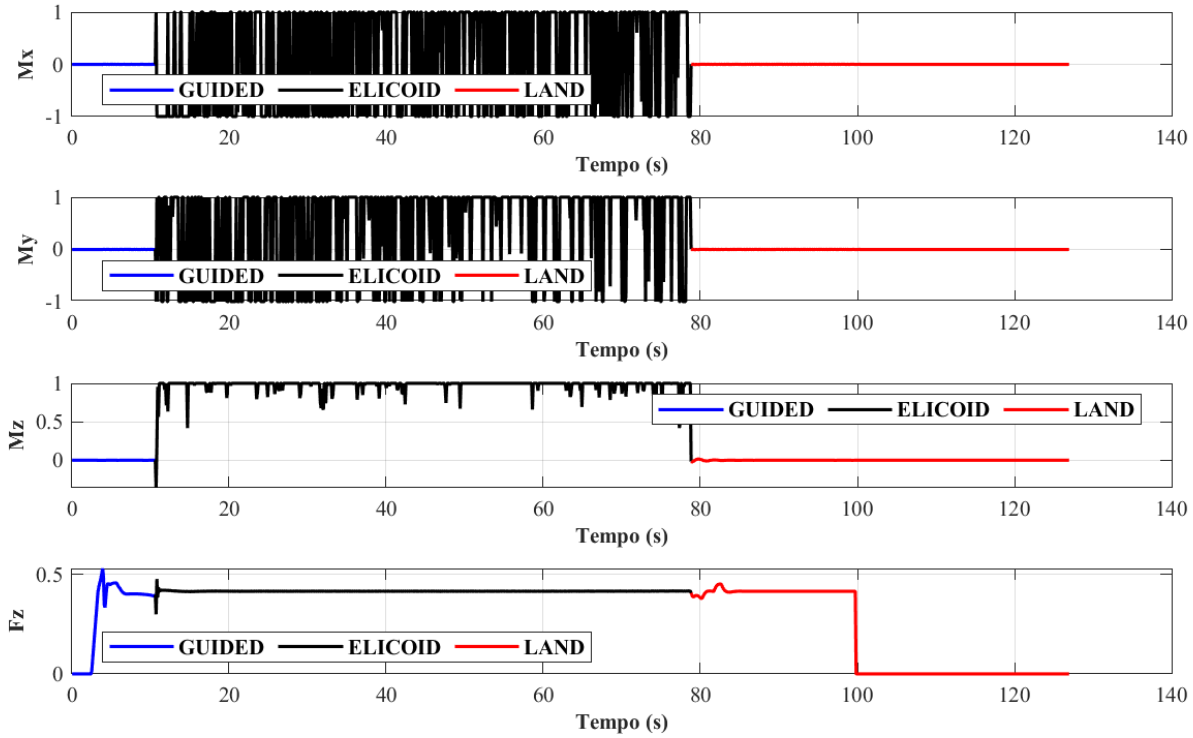
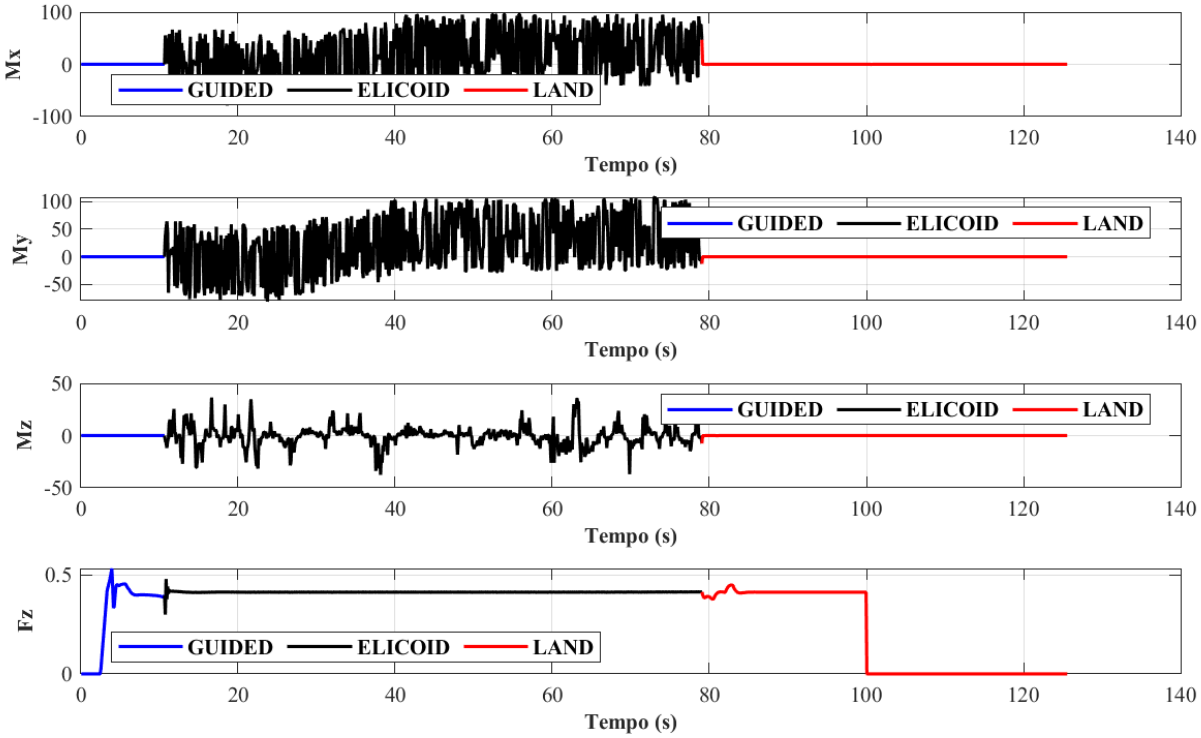


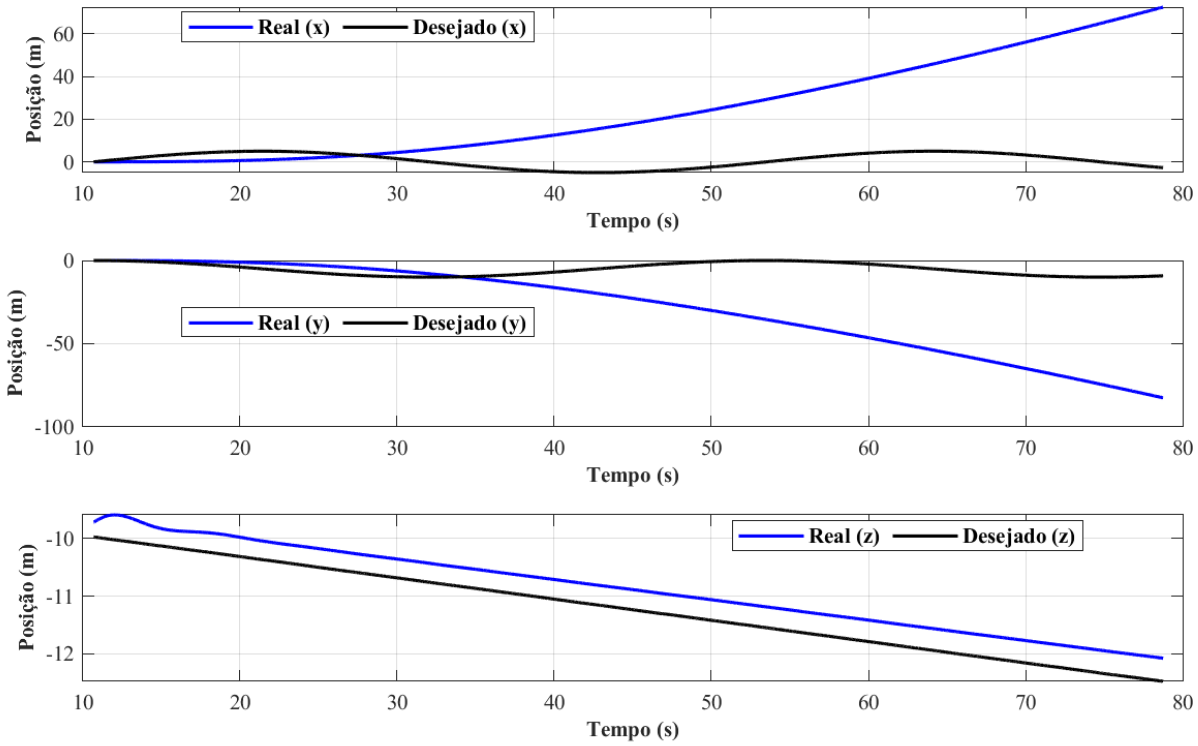
Figura 72 – Esforço normalizado sem saturação do veículo durante a decolagem (modo GUIDED, curva em azul), o deslocamento desejado (modo ELICOID, curva em preto) e o pouso (modo LAND, curva em vermelho).



5.3 Simulação: controle integral *backstepping* com modos deslizantes na posição e na atitude

Analisando exclusivamente o segmento da simulação em que se emprega o controle integral *backstepping* com modos deslizantes desenvolvido. A Figura 73 apresenta a posição do veículo (representada pela curva em azul) em relação à posição desejada (curva em preto). Observa-se que os erros nas coordenadas x e y foram de aproximadamente 60 e 90 metros, respectivamente, enquanto na coordenada z de $0,3425$ metros.

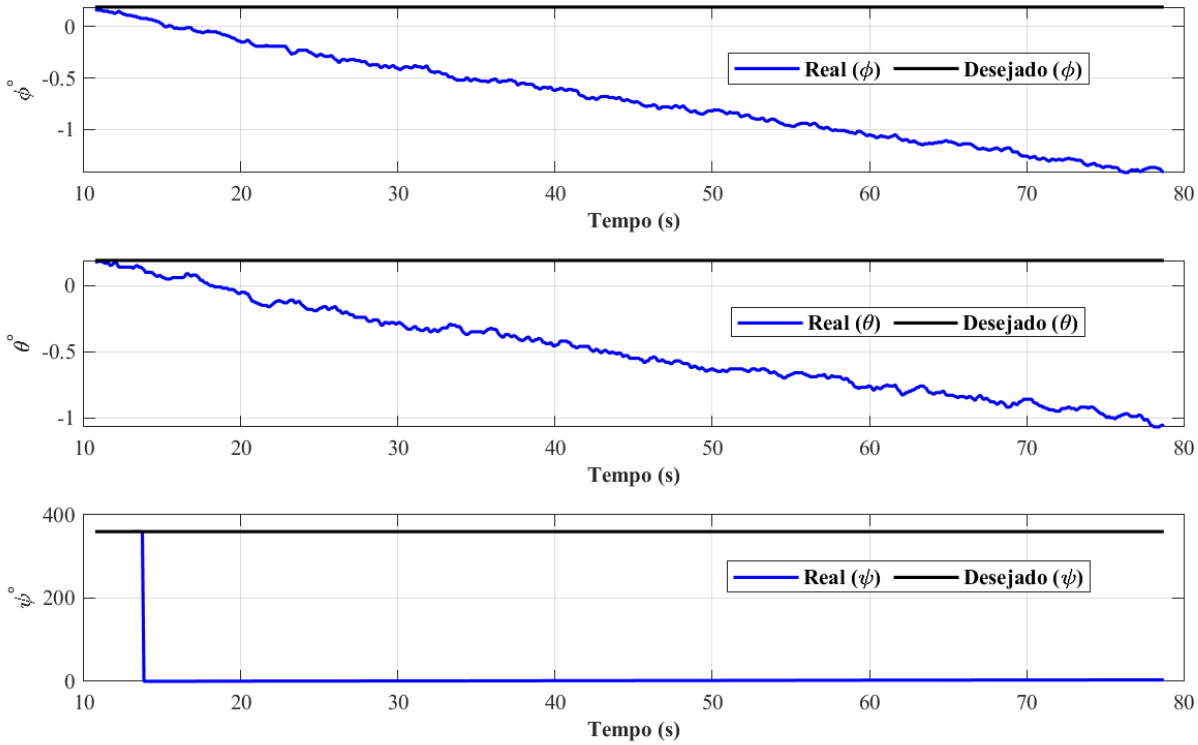
Figura 73 – Posição do veículo durante a simulação com o controle integral *backstepping* com modos deslizantes.



Os erros nas variáveis x e y estão intrinsecamente relacionados ao controle de atitude, uma vez que esses estados são subatuados, dependendo da convergência da atitude do veículo para os valores desejados.

A Figura 74 apresenta as atitudes do veículo (em azul) e as atitudes desejadas (em preto). Apesar de os erros serem próximos de $1,0°$ nas direções ϕ e θ, esses desvios foram suficientes para ocasionar um erro de posição de aproximadamente 60 metros nas coordenadas x e y do veículo.

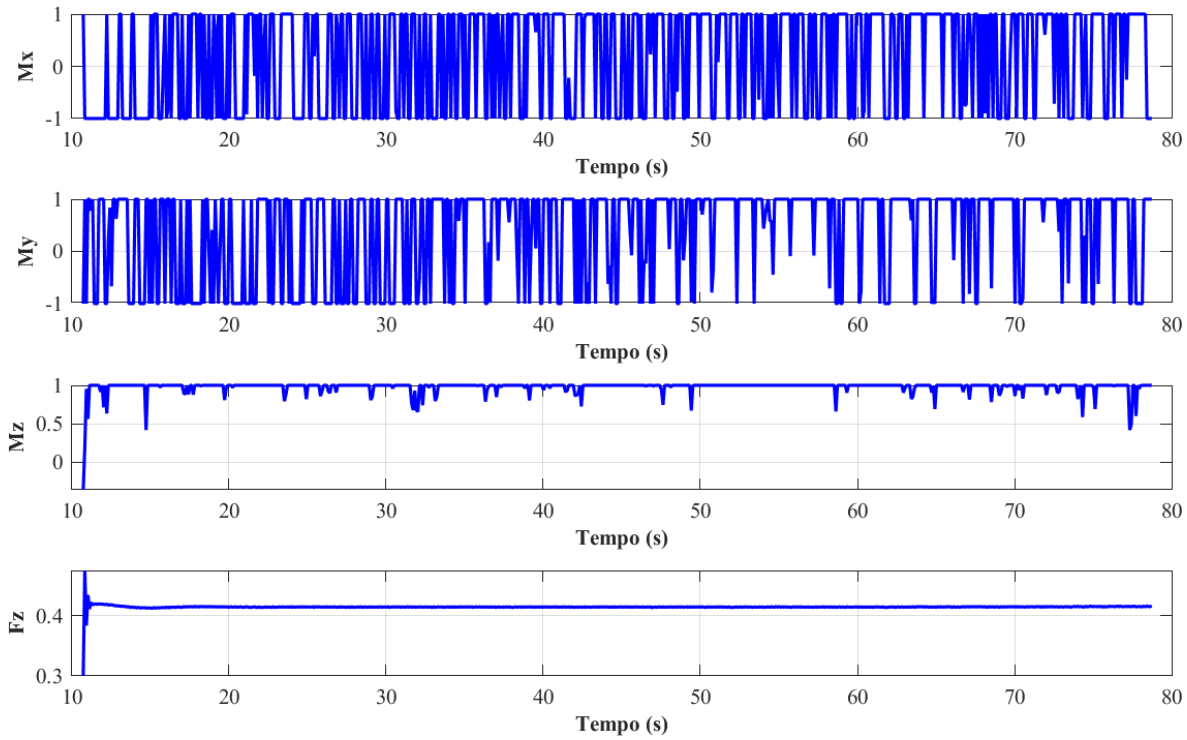
Figura 74 – Atitude do veículo durante a simulação com o controle integral *backstepping* com modos deslizantes.



Analisando as informações apresentadas na Figura 73 e na Figura 74, é possível inferir que o veículo em questão parece estar equipado apenas com um controlador de posição, não sendo evidente a presença de um controlador de atitude. Isso se torna aparente ao observarmos o comportamento do veículo na direção do eixo z, onde o mesmo é diretamente influenciado pelo sinal de controle Fz, gerado pelo controlador de posição.

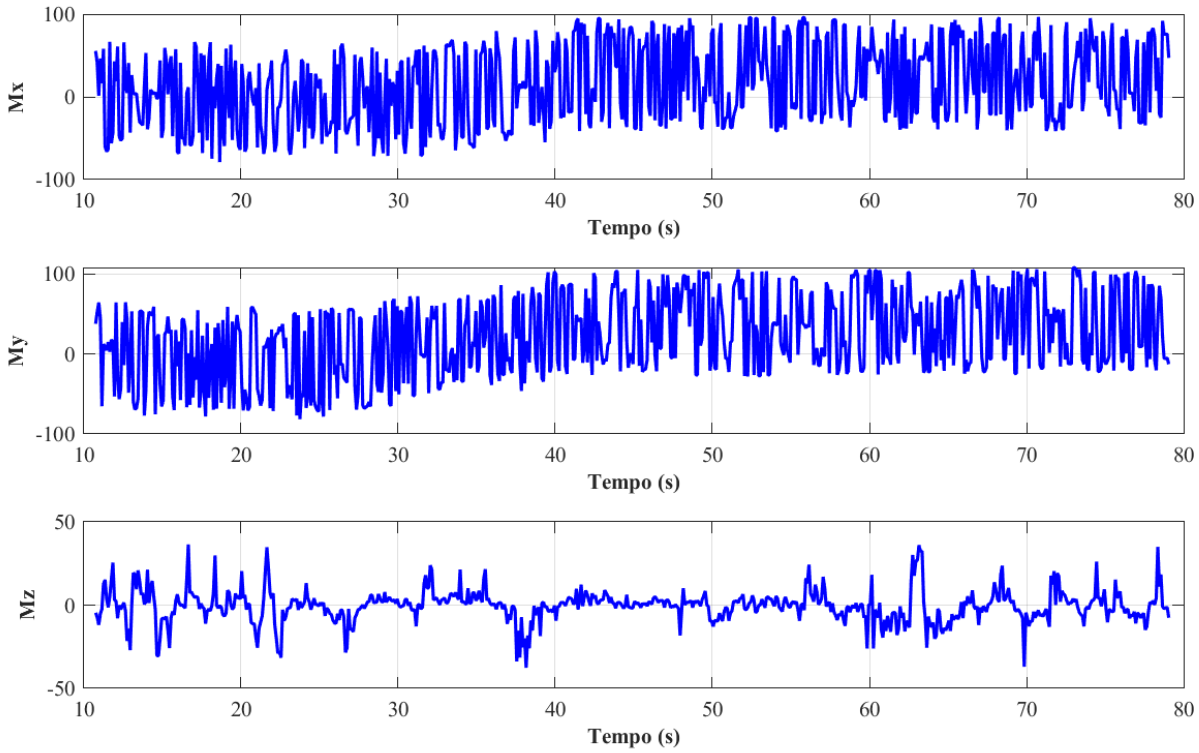
Analisando a Figura 75 pode-se notar que os momentos Mx e My gerados pelo controle de atitude apresentam oscilações entre os extremos da saturação. Já o momento Mz ficou na maior parte da simulação saturado em 1. É relevante notar que o esforço de controle Fz exibe uma oscilação nos instantes iniciais, mas dentro da faixa de operação, no restante da simulação se manteve praticamente constante em $0,41$.

Figura 75 – Esforço do veículo, normalizado e saturado, durante a simulação com o controle integral *backstepping* com modos deslizantes.



A Figura 76 apresenta os momentos Mx, My e Mz do veículo sem o saturador. Pode-se notar que Mx e My oscilam entre -50 a 50 até os 30 segundos de simulação e de 40 segundos até o final da simulação oscilam entre -20 a 80. Já o Mz na maior parte do tempo oscilou entre -20 a 20, com alguns picos de 40 e -40.

Figura 76 – Momentos Mx, My e Mz, normalizados mas sem o saturador, com o controle de posição integral *backstepping* com modos deslizantes.

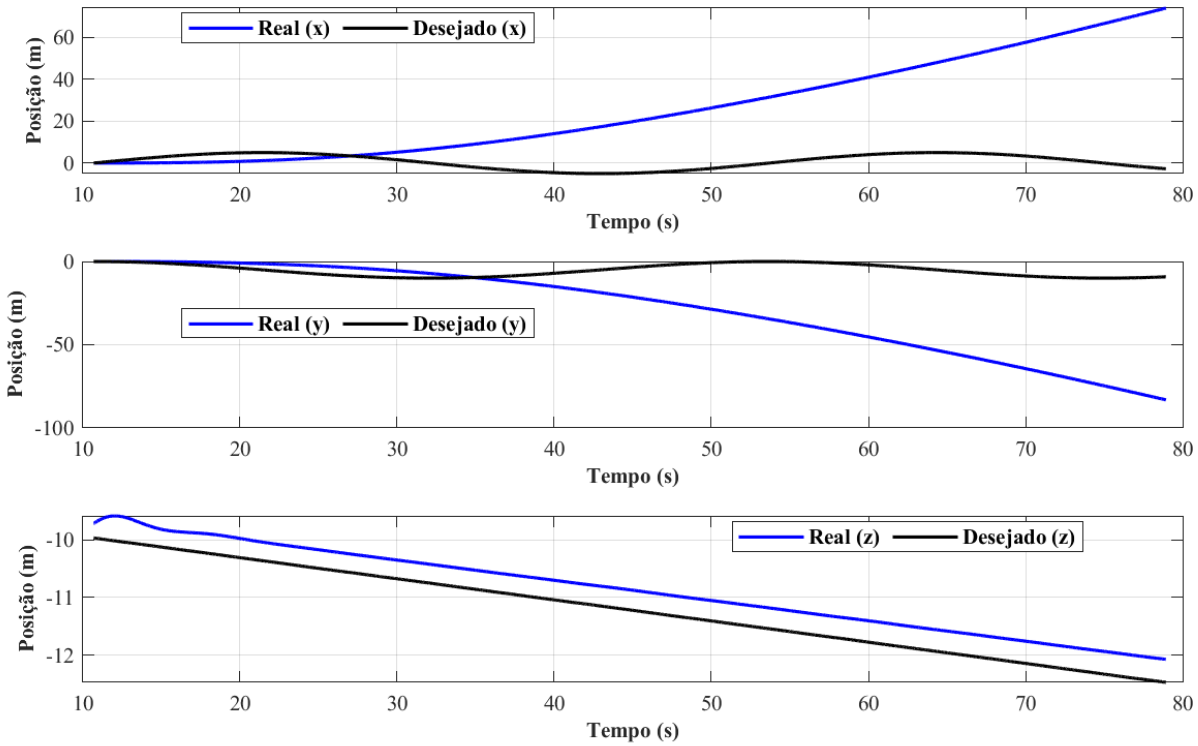


5.4 Simulação: controle integral *backstepping* com modos deslizantes na atitude e controle integral *backstepping* na posição

Uma nova simulação foi conduzida, na qual empregou-se a combinação do controle desenvolvido, integral *backstepping* com modos deslizantes para a regulação da atitude, enquanto o controle da posição foi utilizado um controle integral *backstepping*, o qual possui a mesma lei de controle mas sem o termo do descontínuo, ou seja na Equação (2.53) sem o termo $(-\boldsymbol{Gs})$.

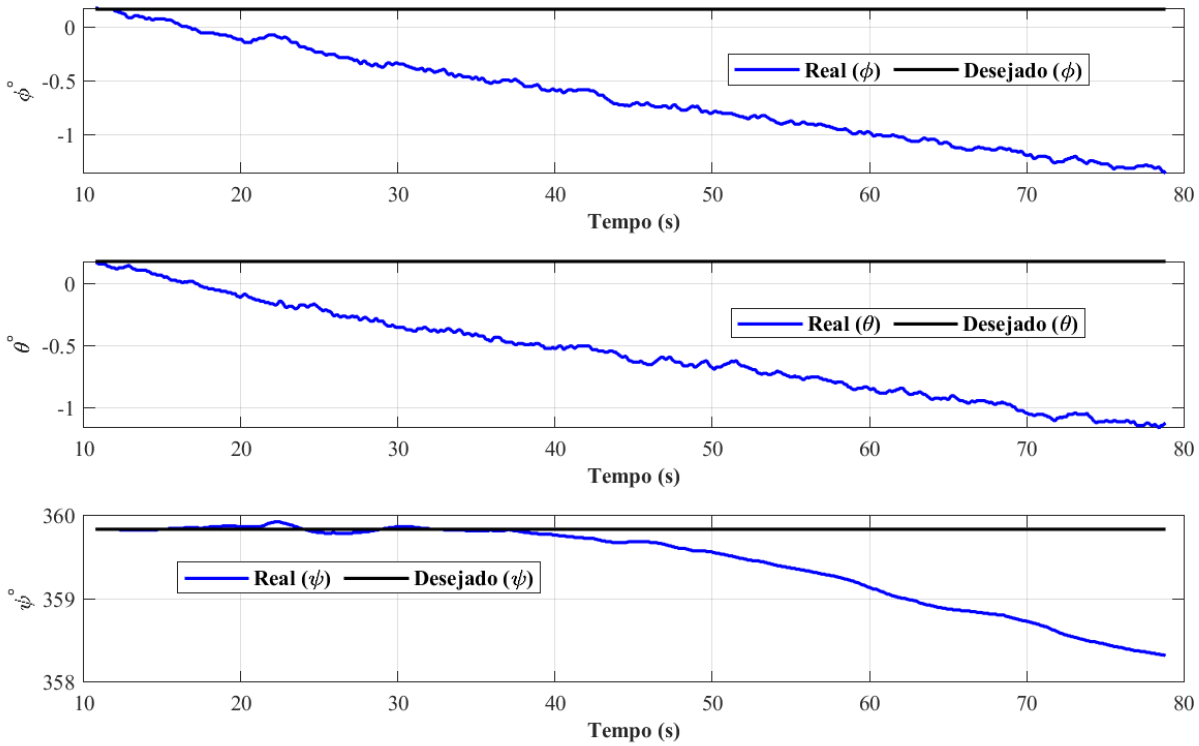
A Figura 77 ilustra os resultados da simulação em termos da posição do veículo (representada pela curva em azul) e a posição desejada (representada pela curva em preto). Observa-se que o erro de posicionamento nas direções x, y e z mantiveram-se praticamente iguais em comparação com os resultados apresentados na Figura 73.

Figura 77 – Posição do veículo durante a simulação com o controle de posição integral *backstepping* e com o integral *backstepping* com modos deslizantes controlando a atitude.



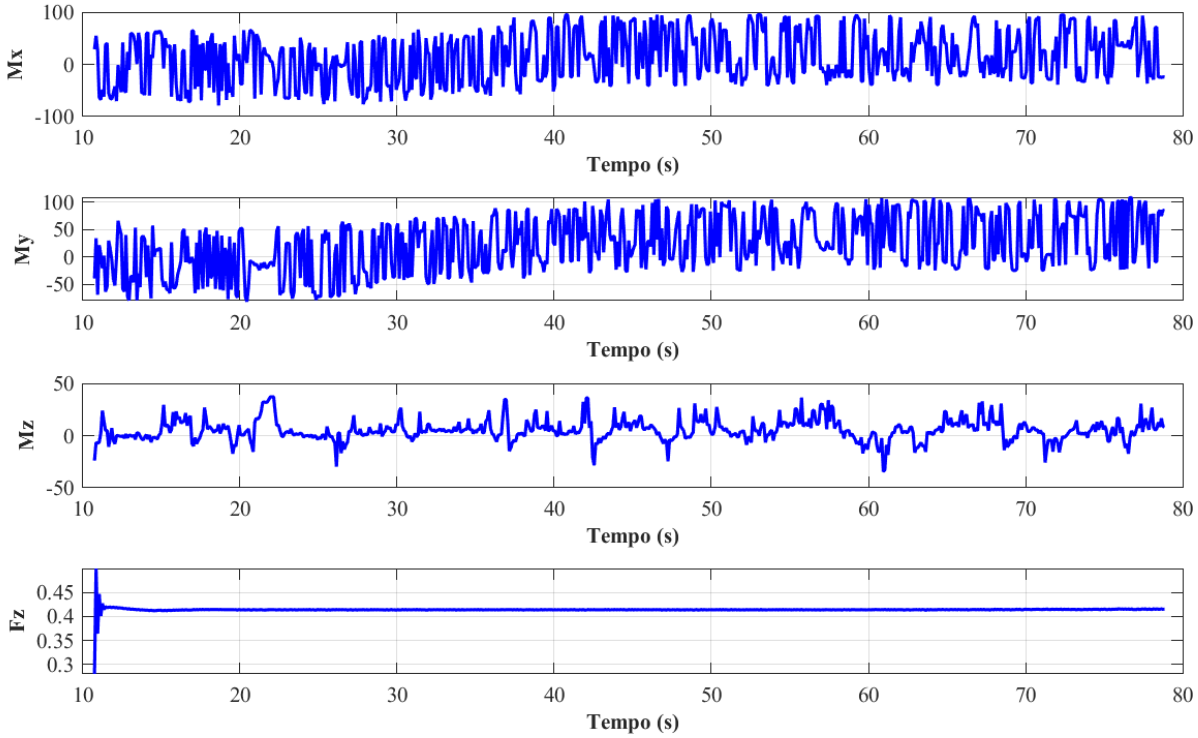
A Figura 78 apresenta a evolução da atitude do veículo (representada pela curva em azul) em comparação com a atitude desejada (curva em preto). Pode-se notar que os ângulos ϕ e θ mantiveram-se praticamente iguais em comparação com os resultados apresentados na Figura 74. Já ψ apresentou um erro menor nos 40 primeiros segundos de simulação, inferior a $0,5°$, mas nos instantes finais da simulação este erro quase chegou a $2,0°$.

Figura 78 – Atitude do veículo durante a simulação com o controle de posição integral *backstepping* e com o integral *backstepping* com modos deslizantes controlando a atitude.



A Figura 79 apresenta o esforço exercido pelo controlador integral *backstepping*, responsável pelo controle da posição, bem como pelo controlador integral *backstepping* com modos deslizantes, que controla a atitude. Apesar do erro em ψ o momento Mz responsável por corrigir este erro não apresentou nenhuma reação, manteve-se praticamente inalterado, conforme Figura 79. O que não ocorre com os momentos Mx e My que após os 30 segundos de simulação, apresentam uma mudança na amplitude de oscilação deslocando-se em aproximadamente de -50 a 50 para -20 a 80, parecido com a Figura 76.

Figura 79 – Esforço do veículo, normalizado e sem saturador, durante a simulação com o controle de posição integral *backstepping* e com o integral *backstepping* com modos deslizantes controlando a atitude.



5.5 Discussão

A despeito de não ter sido obtido um resultado satisfatório que permitisse a sua implementação em um veículo real, o presente estudo possibilitou uma compreensão abrangente do funcionamento do sistema ArduPilot. Adicionalmente, viabilizou a elaboração de um algoritmo de controle de tipo integral com o método *backstepping* e modos deslizantes. Neste contexto, realizou-se uma estimativa dos parâmetros do veículo para a posterior utilização destes na realimentação do controlador.

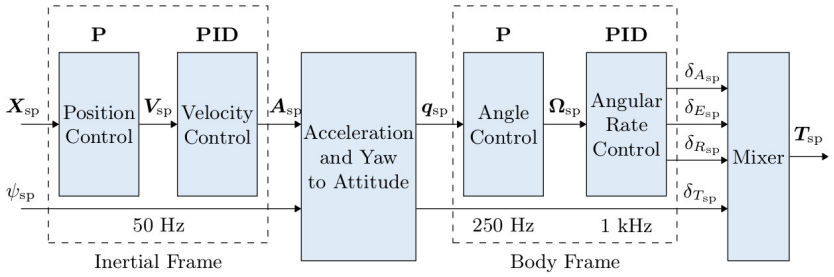
Importante ressaltar que a compreensão aprofundada do controle de atitude no ArduPilot ainda se mostra incompleta. Este fato decorre, em parte, das informações contidas na documentação do sistema, as quais indicam uma saturação nos valores de controle, variando de -1 a 1, em momentos específicos. Tais comportamentos, entretanto, não foram corroborados pelos resultados obtidos nas simulações conduzidas.

Durante todas as simulações realizadas, observou-se que o veículo não alterou o sentido de seu deslocamento nos eixos x e y, mesmo quando o controle foi ajustado, indicando uma aparente falta de recebimento dos sinais enviados pelo controlador. A eficiência do controle de atitude torna-se crucial, devendo superar a velocidade do controle

de posição; esse ajuste é alcançado por meio da técnica conhecida como separação de escala de tempo (*time-scale separation*). A aplicação dessa abordagem torna-se imperativa em contextos práticos, uma vez que, em simulações ideais, presume-se que o controle e o subsistema de atitude são mais rápidos do que os correspondentes de posição (JR; SANTOS, 2023). Esta técnica não foi incorporada no desenvolvimento do método de controle, uma vez que se confiava na capacidade do ArduPilot de contornar este problema.

Cabe destacar que o ArduPilot e seu concorrente, o PX4 AutoPilot, ostentam notável semelhança em diversos aspectos, incluindo a compartilhada utilização de diversas bibliotecas. Assim, é fundamental salientar que o PX4 AutoPilot se diferencia em sua arquitetura de controle, conforme ilustrado na Figura 80. Uma característica notável é a existência de múltiplas frequências de amostragem, ao contrário do ArduPilot, que, conforme a documentação disponível ArduPilot Dev. Team (2020), não contempla tal variação.

Figura 80 – Arquitetura de controle do PX4 AutoPilot.



Fonte: PX4 (2023)

Durante a elaboração deste estudo, foram identificadas diversas discrepâncias entre a documentação do sistema ArduPilot e sua implementação de software correspondente. Um exemplo notável dessas discrepâncias reside na disposição dos motores, que, quando confrontada com a documentação de referência, revelou uma orientação inversa. Tal discrepância foi abordada mediante a tentativa e erro para alcançar a configuração correta.

6 Conclusão

A combinação das duas técnicas de controle, modos deslizantes com o *backstepping*, apresentou um desempenho satisfatório, sendo capaz de realizar o seguimento da trajetória desejada. Em alguns momentos da simulação, o controlador causou oscilações nos atuadores, mas que logo em seguida foram estabilizadas. O controlador foi construído supondo desconhecimento dos parâmetros do veículo. O próprio controlador foi responsável por realizar a estimação que apresentou erros. O controlador mostrou ser robusto à variação de parâmetros, deixando a desejar na presença de trajetórias descontínuas, o que causou a saturação.

A aplicação da técnica de GA para obtenção dos parâmetros do controlador demonstrou ser eficaz pois foi possível obter os valores dos parâmetros, sem a necessidade de realizar uma busca exaustiva para cada parâmetro. O GA também demonstrou ser capaz de ajustar o controlador, quando praticamente transformou um controle integral *backstepping* para um controlador puramente *backstepping* nas direções de x e y.

O algoritmo de detecção de bordas para a identificação do gasoduto não identificou com precisão as bordas do duto, mas pode-se dizer que o gasoduto sempre estará dentro do plano da imagem da câmera, se o veículo seguir as bordas identificadas pelo algoritmo, sendo o erro insignificante.

O controle PBVS desenvolvido em conjunto com o controle de modos deslizantes e *backstepping* mostrou ser capaz de seguir o duto. A relação desenvolvida entre os parâmetros obtidos pela imagem e os parâmetros reais do duto foram suficientes para que o veículo, saindo de uma condição inicial diferente da desejada sobre o duto, convirja rapidamente para a desejada. O veículo manteve a distância do duto fixa, mesmo supondo uma inclinação no duto.

O algoritmo desenvolvido para a identificação da presença de vazamento na filmagem, com base em filmagens obtidas na internet, demonstrou ser promissor. Precisa de mais testes com variação de parâmetros como atmosféricos, vazão e tipo de gás.

A identificação do tipo e da pluma de gás ainda não foi possível de realizar, pois não se encontrou nenhuma filmagem com dados disponíveis para testes, mas foi possível desenvolver um estudo da identificação do gás e da pluma na filmagem de câmeras de infravermelho.

A execução da estratégia de controle proposta no simulador implementado em C++ em associação com o emulador da placa Pixhawk, deixou a desejar, uma vez que não foi possível o controle eficaz do veículo. A simulação apresentou notáveis incongruências em

relação à documentação oficial do ArduPilot, como a saturação dos atuadores. Adicionalmente, foi concebido um novo modo de voo para o ArduPilot, o qual permite a ativação direta dos controladores de posição e atitude, convertidos usando a biblioteca Simulink^{R} Coder, para linguagem de programação C++.

Apesar de a simulação do ArduPilot com o controlador desenvolvido precisar de algumas modificações, é importante salientar os resultados alcançados, os quais incluem:

- A integração das técnicas de controle *backstepping* e modos deslizantes, envolvendo a equalização da derivada do erro de posição do *backstepping* com o termo descontínuo do controle de modo deslizante.
- A aplicação do método de detecção de bordas para identificar as bordas do gasoduto em conjunto com o controle servo visual baseado em posição;
- O desenvolvimento de um algoritmo capaz de identificar a presença de uma pluma de gás na filmagem de câmeras de infravermelho;
- O desenvolvimento do modo de voo dentro do ArduPilot que habilita o controlador desenvolvido sem a necessidade fazer alterações no controlador pré-existente e capaz de chavear o controlador, tanto para a posição como para a atitude, por meio de uma variável.

6.1 Trabalhos Futuros

Os passos subsequentes se configuram na implementação da metodologia de controle delineada na seção 2.2 em um veículo quadricóptero. Adicionalmente, será implementado o algoritmo de rastreamento do gasoduto, conforme descrito na seção 3.1.

Vale salientar que a execução destas etapas encontra-se em curso, como o processo de transposição do controle implementado inicialmente em ambiente Simulink^{R} para a linguagem C++, que é a linguagem de programação pertinente à placa Pixhawk.

Melhorar a simulação em C++ do controlador, para poder fazer testes no veículo real e assim, testar o algoritmo de detecção das bordas do gasoduto.

Referências

AL-YOUNES, Y.; JARRAH, M. A.; SUKKARIEH, S. Adaptive Integral Backstepping Controller for an autonomous rotorcraft. In: *2009 6th International Symposium on Mechatronics and its Applications*. IEEE, 2009. p. 1–7. ISBN 978-1-4244-3480-0. Disponível em: <http://ieeexplore.ieee.org/document/5164795/>.

ALMAKHLES, D. J. Robust Backstepping Sliding Mode Control for a Quadrotor Trajectory Tracking Application. *IEEE Access*, IEEE, v. 8, p. 5515–5525, 2020. ISSN 21693536.

ANTONIO-TOLEDO, M. E. et al. Real-Time Integral Backstepping with Sliding Mode Control for a Quadrotor UAV. *IFAC-PapersOnLine*, v. 51, n. 13, p. 549–554, 2018. ISSN 24058963.

ARAAR, O.; AOUF, N. Visual servoing of a quadrotor UAV for the tracking of linear structured infrastructures. *Proceedings - 2013 IEEE International Conference on Systems, Man, and Cybernetics, SMC 2013*, IEEE, p. 3310–3315, 2013.

ARAAR, O.; AOUF, N. Visual servoing of a Quadrotor UAV for autonomous power lines inspection. *2014 22nd Mediterranean Conference on Control and Automation, MED 2014*, n. November 2014, p. 1418–1424, 2014.

ArduPilot. *ArduPilot, Versatile, Trusted, Open*. 2023. Disponível em: <https://ardupilot.org/>. Acesso em: 09 fev 2023.

ArduPilot Dev. Team. *Copter Attitude Control*. 2020. Disponível em: <https://ardupilot.org/dev/docs/apmcopter-programming-attitude-control-2.html>. Acesso em: 10 fev 2020.

ArduPilot Dev. Team. *Copter Motors Library*. 2021. Disponível em: <https://ardupilot.org/dev/docs/code-overview-copter-motors-library.html>. Acesso em: 5 fev 2021.

ArduPilot Dev. Team. *Motor Thrust Scaling*. 2021. Disponível em: <https://ardupilot.org/copter/docs/motor-thrust-scaling.html#motor-thrust-scaling>. Acesso em: 10 fev 2021.

ArduPilot Dev. Team. *Adding a New Flight Mode to Copter*. 2023. Disponível em: <https://ardupilot.org/dev/docs/apmcopter-adding-a-new-flight-mode.html>. Acesso em: 09 fev 2023.

BÄHRECKE, N. *Automatic Classification and Visualisation of Gas from Infrared Video Data*. 104 p. Dissertação (Master of Science Thesis in Medical Engineering) — KTH Technology and Health, 2015.

BARKANOV, B. *Routledge Handbook of Russian Foreign Policy*. Abingdon, Oxon ; New York, NY : Routledge, 2018.: Routledge, 2018. 138–152 p. ISBN 9781315536934. Disponível em: <https://www.taylorfrancis.com/books/9781134994168>.

BENSALAH, C.; M'SIRDI, N. K.; NAAMANE, A. Full modelling and sliding mode control for a quadrotor UAV in visual servoing task. *12th International Conference on Integrated Modeling and Analysis in Applied Control and Automation, IMAACA 2019*, p. 48–57, 2019.

BHATIA, A. K. et al. Projection Modification Based Robust Adaptive Backstepping Control for Multipurpose Quadcopter UAV. *IEEE Access*, v. 7, p. 154121–154130, 2019. ISSN 2169-3536. Disponível em: <https://ieeexplore.ieee.org/document/8863341/>.

BIRCHBAUER, J. A.; WAKOLBINGER, S.; HORNACEK, M. Taking-off with uavs: Just hype or a future key technology in pipeline integrity management. *Pipeline Technology Journal*, v. 1, p. 7–15, feb 2017. ISSN 2196-4300.

BITTENCOURT, T. d. M. G. *Pré-processamento digital de imagens obtidas na faixa espectral do infravermelho distante*. 140 p. Dissertação (Mestrado em Engenharia) — Universidade de São Paulo, 2012. Disponível em: <http://www.teses.usp.br/teses/disponiveis/18/18152/tde-23102012-090741/pt-br.php>.

BOLTOV, Y. et al. Performance evaluation of real-time system for vision-based navigation of small autonomous mobile robots. *Conference Proceedings of 2019 10th International Conference on Dependable Systems, Services and Technologies, DESSERT 2019*, IEEE, p. 218–222, 2019.

BOON, M. A.; DRIJFHOUT, A. P.; TESFAMICHAEL, S. Comparison of a fixed-wing and multi-rotor UAV for environmental mapping applications: A case study. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences - ISPRS Archives*, v. 42, n. 2W6, p. 47–54, 2017. ISSN 16821750.

BOUABDALLAH, S.; NOTH, A.; SIEGWART, R. PID vs LQ control techniques applied to an indoor micro quadrotor. In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*. IEEE, 2004. v. 3, p. 2451–2456. ISBN 0-7803-8463-6. Disponível em: <http://ieeexplore.ieee.org/document/1389776/>.

BOUABDALLAH, S.; SIEGWART, R. Backstepping and Sliding-mode Techniques Applied to an Indoor Micro Quadrotor. n. April, p. 2259–2264, 2005.

BOUABDALLAH, S.; SIEGWART, R.; CAPRARI, G. Design and control of an indoor coaxial helicopter. *IEEE International Conference on Intelligent Robots and Systems*, n. April, p. 2930–2935, 2006.

BRANDT, A. R. et al. Methane Leaks from North American Natural Gas Systems. *Science*, v. 343, n. 6172, p. 733–735, feb 2014. ISSN 0036-8075. Disponível em: <https://www.sciencemag.org/lookup/doi/10.1126/science.1247045>.

BRETSCHNEIDER, T. R.; SHETTI, K. Uav-based gas pipeline leak detection. In: *Asian Conference on Remote Sensing (Nay Pyi Taw, Myanmar)*. [S.l.: s.n.], 2014. p. 27–31.

CANNY, J. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8, n. 6, p. 679–698, 1986.

CARDOSO, R. et al. Backstepping and integrative sliding mode control for trajectory tracking of a hybrid remotely operated vehicle. In: *2017 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. [S.l.: s.n.], 2017. p. 116–121.

CAVALLARO, S. L. H. *Modelagem, simulação e controle de um VANT do tipo quadricóptero*. 153 p. Dissertação (Mestrado em Engenharia Mecânica) — Universidade de São Paulo, 2019. Disponível em: <http://www.teses.usp.br/teses/disponiveis/3/3152/tde-01032019-155058/publico/SilvioLuisHoriCavallaroCorr19.pdf>.

CERÓN, A.; Mondragón B., I. F.; PRIETO, F. Towards visual based navigation with power line detection. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, v. 8887, p. 827–836, 2014. ISSN 16113349.

CHAUMETTE, F.; HUTCHINSON, S. Visual servo control. i. basic approaches. *IEEE Robotics and Automation Magazine*, v. 13, n. 4, p. 82–90, 2006. ISSN 10709932.

CHUNG, B. W. *Processing for Images and Computer Vision with OpenCV*. [S.l.: s.n.], 2017. ISBN 9781484227749.

CONSUMER ENERGY ALLIANCE. *The Importance of Pipeline Infrastructure*. 2017. Disponível em: <https://consumerenergyalliance.org/2017/05/importance-pipeline-infrastructure/>. Acesso em: 27 abr 2020.

DAVIDI, A.; BERMAN, N.; AROGETI, S. Formation flight using multiple integral backstepping controllers. *Proceedings of the 2011 IEEE 5th International Conference on Cybernetics and Intelligent Systems, CIS 2011*, IEEE, p. 317–322, 2011.

DAVIES, E. *The generalized Hough transform*. [S.l.: s.n.], 2017. 299–339 p. ISBN 9780128092842.

Dronecode Project. *MAVLink Developer Guide*. 2023. Disponível em: <https://mavlink.io/en/>. Acesso em: 09 fev 2023.

ELIKER, K.; ZHANG, W. Finite-time Adaptive Integral Backstepping Fast Terminal Sliding Mode Control Application on Quadrotor UAV. *International Journal of Control, Automation and Systems*, v. 18, n. 2, p. 415–430, 2020. ISSN 20054092.

EMRAN, B.; TANNANT, D.; NAJJARAN, H. Low-Altitude Aerial Methane Concentration Mapping. *Remote Sensing*, v. 9, n. 8, p. 823, aug 2017. ISSN 2072-4292. Disponível em: <https://www.mdpi.com/2072-4292/9/8/823>.

FAN, Y.; CAO, Y.; LI, T. Adaptive integral backstepping control for trajectory tracking of a quadrotor. In: *2017 4th International Conference on Information, Cybernetics and Computational Social Systems (ICCSS)*. IEEE, 2017. p. 619–624. ISBN 978-1-5386-3257-4. Disponível em: <http://ieeexplore.ieee.org/document/8091489/>.

FLIR System. A resource guide for using infrared in the research and development industry the ultimate infrared handbook for r&d professionals contents. 2012. Disponível em: <https://www.flirmedia.com/MMC/THG/Brochures/T559243/T559243_EN.pdf>.

FLIR System. *Optical Gas Imaging Brochure*. [S.l.], 2016. Disponível em: <https://www.flir.com/support/products/gf343#Resources>.

GOLLAPUDI, S. *Learn Computer Vision Using OpenCV*. [S.l.: s.n.], 2019. ISBN 9781484242605.

GRAVEL, J.-F. et al. Oil Pipeline Standoff Leak Detection: A Novel Approach for Airborne Remote Detection of Small Leaks. In: *Volume 3: Operations, Monitoring and Maintenance; Materials and Joining*. ASME, 2016. p. V003T04A016. ISBN 978-0-7918-5027-5. Disponível em: <http://proceedings.asmedigitalcollection.asme.org/proceeding.aspx?doi=10.1115/IPC2016-64704>.

GRUPA, A. d. A. S. Introduction to pigging and case study of an onshor cruid oil trunkline. *International Journal of Latest Technology in Engineering, Management & Applied Science-IJLTEMAS*, v. 5 issue 2, p. 18–25, Feb 2016. ISSN 2278–2540.

HE, Z.; ZHAO, L.; ZHAO, L. Robust chattering free backstepping/backstepping sliding mode control for quadrotor hovering. In: *2016 IEEE Information Technology, Networking, Electronic and Automation Control Conference*. IEEE, 2016. p. 616–620. ISBN 978-1-4673-9194-8. Disponível em: <http://ieeexplore.ieee.org/document/7560434/>.

HOLLANDER, H. D.; BOLLERMANN, B. Why is pipeline leak detection more than just installing a flowmeter at inlet and outlet? *2th Pipeline Technology Conference*, p. 1–9, Apr 2007. ISSN 2510-6716.

IWASZENKO, S. et al. Detection of Natural Gas Leakages Using a Laser-Based Methane Sensor and UAV. *Remote Sensing*, v. 13, n. 3, p. 510, jan 2021. ISSN 2072-4292. Disponível em: <https://www.mdpi.com/2072-4292/13/3/510>.

JADIN, M. S.; GHAZALI, K. H. Gas leakage detection using thermal imaging technique. In: *Proceedings - UKSim-AMSS 16th International Conference on Computer Modelling and Simulation, UKSim 2014*. [S.l.: s.n.], 2014. p. 302–306. ISBN 9781479949236.

JIA, Z. et al. Integral backstepping sliding mode control for quadrotor helicopter under external uncertain disturbances. *Aerospace Science and Technology*, Elsevier Masson SAS, v. 68, p. 299–307, sep 2017. ISSN 12709638. Disponível em: <http://dx.doi.org/10.1016/j.ast.2017.05.022https://linkinghub.elsevier.com/retrieve/pii/S1270963817308878>.

JIANG, T.; SONG, T.; LIN, D. Integral Sliding Mode based Control for Quadrotors with Disturbances: Simulations and Experiments. *International Journal of Control, Automation and Systems*, v. 17, n. 8, p. 1987–1998, 2019. ISSN 20054092.

JR, J. A. R.; SANTOS, D. A. Robust collision-free guidance and control for underactuated multirotor aerial vehicles. *Drones*, v. 7, n. 10, 2023. ISSN 2504-446X. Disponível em: <https://www.mdpi.com/2504-446X/7/10/611>.

KASTEK, M.; PIATKOWSKI, T.; TRZASKAWKA, P. Infrared imaging fourier transform spectrometer as the stand-off gas detection system. *Metrology and Measurement Systems*, v. 18, n. 4, p. 607–620, 2011. ISSN 08608229.

KIM, J. et al. Unmanned aerial vehicles in agriculture: A review of perspective of platform, control, and applications. *IEEE Access*, v. 7, p. 105100–105115, 2019. ISSN 21693536.

KOKOTOVIC, P.; ARCAK, M. Nonlinear and Adaptive Control: An Abbreviated Status Report. 2001.

KRSTIC, M.; KANELLAKOPOULOS, I.; KOKOTOVIC, P. *Nonlinear and Adaptive Control Design*. [S.l.]: Wiley-Interscience, 1995. 576 p. ISBN 0471127329.

LABBADI, M.; CHERKAOUI, M. Robust adaptive backstepping fast terminal sliding mode controller for uncertain quadrotor UAV. *Aerospace Science and Technology*, Elsevier Masson SAS, v. 93, p. 105306, oct 2019. ISSN 12709638. Disponível em: <https://doi.org/10.1016/j.ast.2019.105306https://linkinghub.elsevier.com/retrieve/pii/S1270963819300318>.

LI, H. Z. et al. Gathering pipeline methane emissions in utica shale using an unmanned aerial vehicle and ground-based mobile sampling. *Atmosphere*, v. 11, n. 7, p. 1–13, 2020. ISSN 20734433.

LIU, H.; GAO, G. Dynamic modeling and analyzing for a novel x-quadrotor. In: *2017 3rd International Conference on Computational Intelligence & Communication Technology (CICT)*. IEEE, 2017. p. 1–4. ISBN 978-1-5090-6218-8. Disponível em: <http://ieeexplore.ieee.org/document/7977292/>.

Magnussen, O.; Hovland, G.; Ottestad, M. Multicopter uav design optimization. In: *2014 IEEE/ASME 10th International Conference on Mechatronic and Embedded Systems and Applications (MESA)*. [S.l.: s.n.], 2014. p. 1–6. ISSN null.

MATHWORKS. *Wind Shear Model: calculate wind shear conditions*. 2023. Disponível em: <https://www.mathworks.com/help/aeroblks/windshearmodel.html>. Acesso em: 10 jul 2023.

MCHUGH, S. *Understanding Photography: Master Your Digital Camera and Capture That Perfect Photo*. No Starch Press, 2018. ISBN 9781593278953. Disponível em: <https://books.google.com.br/books?id=l0RBDwAAQBAJ>.

MUKHOPADHYAY, P.; CHAUDHURI, B. B. A survey of Hough Transform. *Pattern Recognition*, Elsevier, v. 48, n. 3, p. 993–1010, mar 2015. ISSN 00313203. Disponível em: <http://dx.doi.org/10.1016/j.patcog.2014.08.027https://linkinghub.elsevier.com/retrieve/pii/S0031320314003446>.

NGUYEN, A. T.; XUAN-MUNG, N.; HONG, S. K. Quadcopter adaptive trajectory tracking control: A new approach via backstepping technique. *Applied Sciences (Switzerland)*, v. 9, n. 18, p. 1–17, 2019. ISSN 20763417.

NOORALISHAHI, P.; LÓPEZ, F.; MALDAGUE, X. A drone-enabled approach for gas leak detection using optical flow analysis. *Applied Sciences (Switzerland)*, v. 11, n. 4, p. 1–19, 2021. ISSN 20763417.

NOURMOHAMMADI, A.; JAFARI, M.; ZANDER, T. O. A survey on unmanned aerial vehicle remote control using brain-computer interface. *IEEE Transactions on Human-Machine Systems*, IEEE, v. 48, n. 4, p. 337–348, 2018. ISSN 21682291.

OLIVEIRA, A. C. F. de. *Modelagem Dinâmica e controle para navegação de um veículo aéreo não tripupulado do tipo quadricóptero*. 128 p. Dissertação (Mestrado em Engenharia Mecânica) — Universidade Federal do ABC, 2019.

OLIVEIRA, T. R. de. *Controle por Modos Deslizantes de Sistemas Incertos com Direção de Controle Desconhecida*. 209 p. Dissertação (Mestrado em Engenharia Elétrica) — Universidade Federal do Rio de Janeiro, COPPE, 2006. Disponível em: <http://www.pee.ufrj.br/index.php/pt/producao-academica/dissertacoes-de-mestrado/2006-1/2006072101-2006072101/file>.

OPENCV. *OpenCV2:Hough Line Transform*. 2020. Disponível em: <https://docs.opencv.org/master/d9/db0/tutorial_hough_lines.html>. Acesso em: 27 abr 2020.

OpenMV. *OpenMV Cam H7*. 2023. Disponível em: <https://openmv.io/collections/products/products/openmv-cam-h7>. Acesso em: 09 fev 2023.

PERRONE, R. *Controle servo visual de veículos aéreos multirrotores.* 142 p. Dissertação (Mestrado em Mecatrônica) — Universidade Federal da Bahia, 2013.

PERRUQUETTI, W.; BARBOT, J.-P. *Sliding Mode Control In Engineering*. CRC Press, 2002. v. 39. 951–954 p. ISSN 00051098. ISBN 9780203910856. Disponível em: <https://www.taylorfrancis.com/books/9780203910856>.

PFEIFER, E. *Projeto e controle de um UAV quadrirotor*. 153 p. Dissertação (Mestrado em Engenharia) — Universidade de São Paulo, 2013. Disponível em: <https://teses.usp.br/teses/disponiveis/3/3139/tde-06072014-223705/publico/Dissertacao_Erick.pdf>.

PX4, D. T. *Controller Diagrams*. 2023. Disponível em: <https://docs.px4.io/v1.12/en/flight_stack/controller_diagrams.html>. Acesso em: 10 sep 2023.

QI, Y. et al. Autonomous landing solution of low-cost quadrotor on a moving platform. *Robotics and Autonomous Systems*, Elsevier B.V., v. 119, p. 64–76, 2019. ISSN 0921-8890. Disponível em: <https://doi.org/10.1016/j.robot.2019.05.004>.

RAMADEVI, R.; JAIGANESH, J.; KRISHNAMOORTHY, N. R. Leak detection methods-a technical review. In: KUMAR, A.; MOZAR, S. (Ed.). *ICCCE 2018*. Singapore: Springer Singapore, 2019. p. 125–139. ISBN 978-981-13-0212-1.

RAVIKUMAR, A. P.; WANG, J.; BRANDT, A. R. Are optical gas imaging technologies effective for methane leak detection? *Environmental Science & Technology*, v. 51, n. 1, p. 718–724, 2017. PMID: 27936621. Disponível em: <https://doi.org/10.1021/acs.est.6b03906>.

RENSHAW, P. F.; WIGGINS, M. W. The predictive utility of cue utilization and spatial aptitude in small Visual Line-Of-Sight rotary-wing Remotely Piloted Aircraft operations. *International Journal of Industrial Ergonomics*, Elsevier B.V, v. 61, p. 47–61, 2017. ISSN 18728219. Disponível em: <http://dx.doi.org/10.1016/j.ergon.2017.05.014>.

RESNICK, R.; HALLIDAY, D.; WALKER, J. *Fundamentos De Física - Volume 4 - Óptica E Física*. [S.l.]: LTC, 2016. ISBN 9788521630388.

RODRÍGUEZ-ABREO, O. et al. Genetic algorithm-based tuning of backstepping controller for a quadrotor-type unmanned aerial vehicle. *Electronics (Switzerland)*, v. 9, n. 10, p. 1–24, 2020. ISSN 20799292.

ROLDÁN, J. J. et al. Mini-UAV based sensory system for measuring environmental variables in greenhouses. *Sensors (Switzerland)*, v. 15, n. 2, p. 3334–3350, 2015. ISSN 14248220.

RW ENGENHARIA. *RW Engenharia*. 2016. Disponível em: <https://www.rwengenharia.eng.br/reuso-de-agua-na-industria/water-pipe-51758_960_720/>. Acesso em: 27 abr 2020.

SCAFUTTO, R. D. M. et al. An evaluation of airborne swir imaging spectrometers for ch4 mapping: Implications of band positioning, spectral sampling and noise. *International Journal of Applied Earth Observation and Geoinformation*, Elsevier B.V., v. 94, n. May 2020, p. 102233, 2021. ISSN 03032434. Disponível em: <https://doi.org/10.1016/j.jag.2020.102233>.

SENDOBRY, A. *Control System Theoretic Approach to Model Based Navigation*. 119 p. Tese (PhD Thesis) — Technische Universität Darmstadt:, 2013.

SHI, L. et al. Hazardous gas detection four-rotor UAV system development. In: *2016 IEEE International Conference on Mechatronics and Automation*. IEEE, 2016. p. 2461–2465. ISBN 978-1-5090-2396-7. Disponível em: <http://ieeexplore.ieee.org/document/7558952/>.

SILVA, C. A. N. da. *Controle não linear para um veículo áereo não tripulado: aspectos teóricos e numéricos*. 145 p. Dissertação (Mestrado em Engenharia Mecânica) — Universidade federal do ABC, 2018.

SLOTINE, J.; LI, W. *Applied Nonlinear Control*. [S.l.]: Prentice Hall, 1991. ISBN 9780130408907.

SUTTER, J. D.; BERLINGER, J. *Obama: Climate agreement 'best chance we have' to save the planet*. 2015. 1–5 p. Disponível em: <http://edition.cnn.com/2015/12/12/world/global-climate-change-conference-vote/>.

TANG, Y.-R.; LI, Y. Dynamic modeling for high-performance controller design of a uav quadrotor. In: *2015 IEEE International Conference on Information and Automation*. [S.l.: s.n.], 2015. p. 3112–3117.

UTKIN, V.; GULDNER, J.; SHIJUN, M. *Sliding Mode Control in Electro-mechanical Systems*. [S.l.]: Taylor & Francis, 1999. (Automation and Control Engineering). ISBN 9780748401161.

VIJAYARANI, S.; VINUPRIYA. Performance Analysis of Canny and Sobel Edge Detection Algorithms in Image Mining. *International Journal of Innovative Research in Computer and Communication Engineering (An ISO Certified Organization)*, v. 3297, n. 8, p. 1760–1767, 2007. ISSN 2320-9798. Disponível em: <www.ijircce.com>.

VOOS, H. Nonlinear state-dependent riccati equation control of a quadrotor uav. *IEEE International Conference on Control Applications, 2006*, p. 2547–2552, Out. 2006.

WALK, T. Technology update on leak detection systems. *5th Pipeline Technology Conference*, p. 1–9, Apr 2010. ISSN 2510-6716.

WANG, N. et al. Backpropagating Constraints-Based Trajectory Tracking Control of a Quadrotor with Constrained Actuator Dynamics and Complex Unknowns. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, IEEE, v. 49, n. 7, p. 1322–1337, 2019. ISSN 21682232.

WEBBOOK, N. I. of S. T. N. C. *NIST Standard Reference Database Number 69*. 2018. Disponível em: <https://webbook.nist.gov/chemistry/#CreditsControl>. Acesso em: 10 fev 2020.

WERLE, P. A review of recent advances in semiconductor laser based gas monitors. *Spectrochimica Acta Part A: Molecular and Biomolecular Spectroscopy*, v. 54, n. 2, p. 197–236, feb 1998. ISSN 13861425. Disponível em: <http://www.ingentaconnect.com/content/els/13861425/1998/00000054/00000002/art00227%5Cnhttp://www.ino.it/home/pwwerle/Download/1998-SAA.pdfhttps://linkinghub.elsevier.com/retrieve/pii/S1386142597002278>.

XIE, W. et al. Adaptive backstepping control of a quadcopter with uncertain vehicle mass, moment of inertia, and disturbances. *IEEE Transactions on Industrial Electronics*, v. 69, n. 1, p. 549–559, jan 2022. ISSN 0278-0046. Disponível em: <https://ieeexplore.ieee.org/document/9345483/>.

YAN, Y. et al. An Optimization Model for UAV Inspection Path of Oil and Gas Pipeline Network. In: *Volume 3: Operations, Monitoring, and Maintenance; Materials and Joining*. American Society of Mechanical Engineers, 2018. p. V003T04A007. ISBN 978-0-7918-5188-3. Disponível em: <https://asmedigitalcollection.asme.org/IPC/proceedings/IPC2018/51883/Calgary,Alberta,Canada/276956>.

YANG, S. et al. Natural Gas Fugitive Leak Detection Using an Unmanned Aerial Vehicle: Measurement System Description and Mass Balance Approach. *Atmosphere*, v. 9, n. 10, p. 383, oct 2018. ISSN 2073-4433. Disponível em: <http://www.mdpi.com/2073-4433/9/10/383>.

YI, S.; WATANABE, K.; NAGAI, I. Control of an Over-actuated Quadrotor Manipulator Based on Backstepping Integral Sliding Mode. In: *2021 International Conference on Advances in Computing, Communication, and Control (ICAC3)*. IEEE, 2021. p. 1–6. ISBN 978-1-6654-2634-3. Disponível em: <https://ieeexplore.ieee.org/document/9697191/>.

ZHANG, S.; CHEN, C. Backstepping Based Nonlinear Integral Sliding Mode Control for Quadrotors Under External Disturbances. In: *2019 Chinese Control Conference (CCC)*. IEEE, 2019. v. 2019-July, p. 8355–8359. ISBN 978-9-8815-6397-2. ISSN 21612927. Disponível em: <https://ieeexplore.ieee.org/document/8865215/>.

Apêndices

APÊNDICE A – Dados do ensaio do motor

Os dados apresentados na Tabela 5 foram obtidos com o uso do wattímetro ToolkitRC WM150 e multímetro JCSOLDER (Figura 81a), bateria(Figura 81c), motor e ESC (Figura 81b), hélice(Figura 81d) e uma balança de tara máxima de 10 Kg.

Tabela 5 – Dados obtidos nos ensaios do motor.

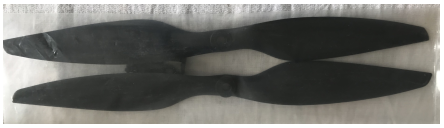
PWM (μs)	U_{out} (V)	I_A (A)	Potência (W)	ω_i (RPM)	Força (N)	Momento (Nm)
1137	0,521	0,096	1,0	507	0,04	0,000
1157	0,766	0,108	1,2	1082	0,07	0,000
1176	0,971	0,144	2,0	1405	0,14	0,000
1196	1,163	0,164	2,2	931	0,21	0,000
1275	1,888	0,344	5,6	2479	0,55	0,000
1353	2,625	0,684	10,8	3554	1,27	0,353
1431	7,193	1,310	21,0	4509	2,33	0,753
1510	7,375	2,146	35,0	5345	3,56	1,364
1588	7,698	3,192	51,6	5358	4,83	2,197
1667	7,918	4,600	75,6	8114	6,32	3,481
1745	8,125	6,376	102,6	7577	7,88	4,587
1824	8,353	8,526	135,8	8217	9,48	5,886
1902	8,688	10,828	171,6	8162	11,11	6,787
1922	8,588	11,294	178,2	8769	11,45	6,977
1941	8,610	11,348	179,6	8724	11,70	6,945
1961	8,513	11,244	175,8	8725	11,46	6,848
1980	8,553	11,234	174,6	8667	11,45	6,881
2000	8,598	11,386	180,2	4769	11,64	6,913

Figura 81 – Equipamentos utilizados para o ensaio do Motor Modelo MN3508 KV580, ESC XRotor-40A e hélice 12x45.

(a) Wattímetro ToolkitRC WM150 e multímetro JCSOLDER.

(b) Motor Modelo MN3508 KV580 e ESC XRotor-40A.

(c) Bateria $14,8V$ e 4000 mAh.



(d) Hélice modelo 12x45.

APÊNDICE B – Modelo dos Atuadores Normalizado

Os dados obtidos do motor com a hélice precisam ser normalizados para serem utilizados no software ArduPilot (ArduPilot Dev. Team, 2021b). Essa normalização possui dois objetivos principais: primeiro, evitar a região de *"dead-band"*, na qual o motor apresenta uma entrada (sinal PWM diferente de zero), mas não gera uma saída de velocidade igual a zero; segundo, converter os valores de força e momento para escalas de 0 a 1 e -1 a 1, respectivamente.

O procedimento de normalização é o mesmo tanto para a força quanto para o momento. O primeiro passo consiste em converter o valor obtido para uma escala de 0 a 1, conforme mostrado na Equação (B.1),

$$T_n = \frac{V}{V_{max}} \left(\frac{\sigma - \sigma_{min}}{\sigma_{max} - \sigma_{min}} \right) \tag{B.1}$$

onde: V é a tensão de entrada no motor, V_{max} é a tensão máxima, σ é o sinal do ESC em $[\mu s]$, σ_{min} é o valor mínimo do ESC para o motor começar a girar e σ_{max} o valor máximo.

O próximo passo é realizar o ajuste da Equação (B.1) aos dados experimentais. Esse ajuste é realizado por meio da introdução da variável M_{expo}, que controla a linearidade da curva e pode variar de 0 a 1. Neste caso específico, adotamos o valor $M_{expo} = 0,65$, conforme descrito na Equação (B.2),

$$F_n = (1 - M_{expo})\, T_n + M_{expo} T_n^2. \tag{B.2}$$

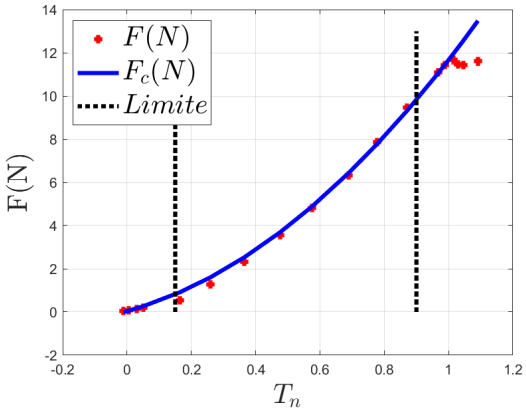
Finalmente, a foça ou o momento corrigido é calculado, conforme a Equação (B.3),

$$F_c = F_n F_{max} + F_{min} \tag{B.3}$$

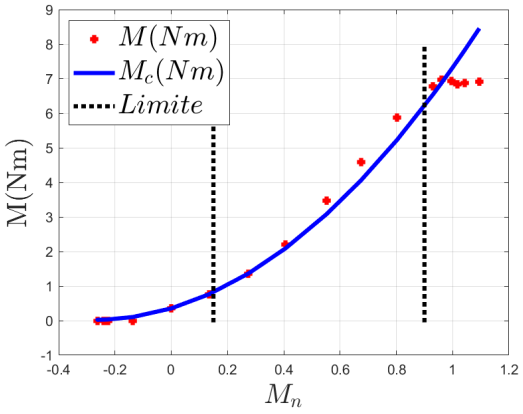
onde: F_{min} é a mínima força.

O resultado do processo de normalização é apresentado na Figura 82, na qual a Figura 82a exibe uma comparação entre os valores de força obtidos (pontos em vermelho) e a força normalizada (curva em azul). Já a Figura 82b apresenta uma comparação entre os valores de momento obtidos (pontos em vermelho) e o momento normalizado (curva em azul), sendo normalizado no intervalo de 0 a 1. É importante ressaltar que os valores negativos (de -1 a 0) possuem a mesma magnitude dos valores positivos, apenas com o sinal invertido, representando o sentido de rotação do motor.

Figura 82 – Normalização da força e momento do propulsor.



(a) Comparação da força obtida (pontos em vermelho) e força normalizada (curva em azul).



(b) Comparação do momento obtido (pontos em vermelho) e momento normalizado (curva em azul).

Com o intuito de evitar regiões de atuação extrema, como *dead-band* e a saturação, foram selecionados limites mínimo (15%) e máximo (90%) de atuação. Esses limites são representados pela linha pontilhada em preto na Figura 82.

Por fim, o momento é escrito em função da força realizando uma aproximação linear, Equação (B.4), apresentado na Figura 83a,

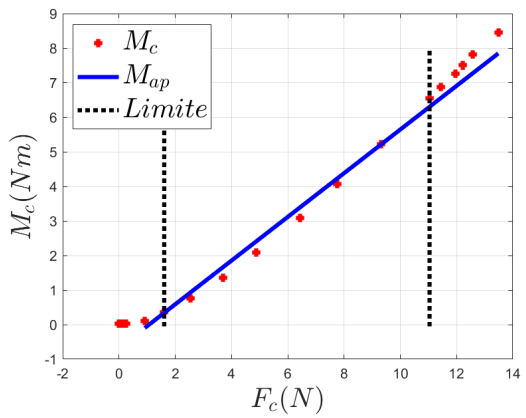
$$M_{ap} = K_m F_c + \delta \tag{B.4}$$

onde $K_m = 0,6322$ e $\delta = -0,6760$ são parâmetros de ajustes da curva. O sinal do ESC escrito em função da força, Equação (B.5), conforme Figura 83b

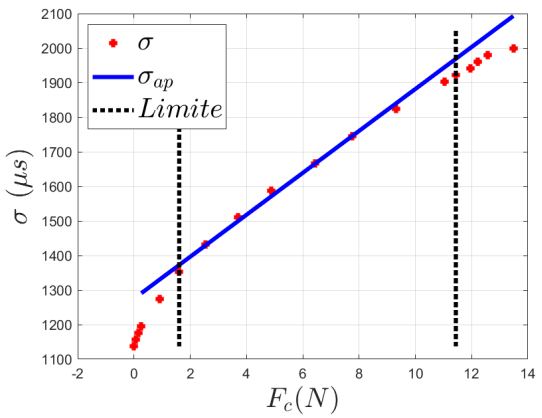
$$\sigma_{ap} = K_\sigma F_c + \delta_\sigma \tag{B.5}$$

onde $K_\sigma = 594,623$ e $\delta_\sigma = 1275$ são parâmetros de ajustes da curva.

Figura 83 – Aproximação linear do momento em função da força e da força em função do ESC.



(a) Aproximação do momento.

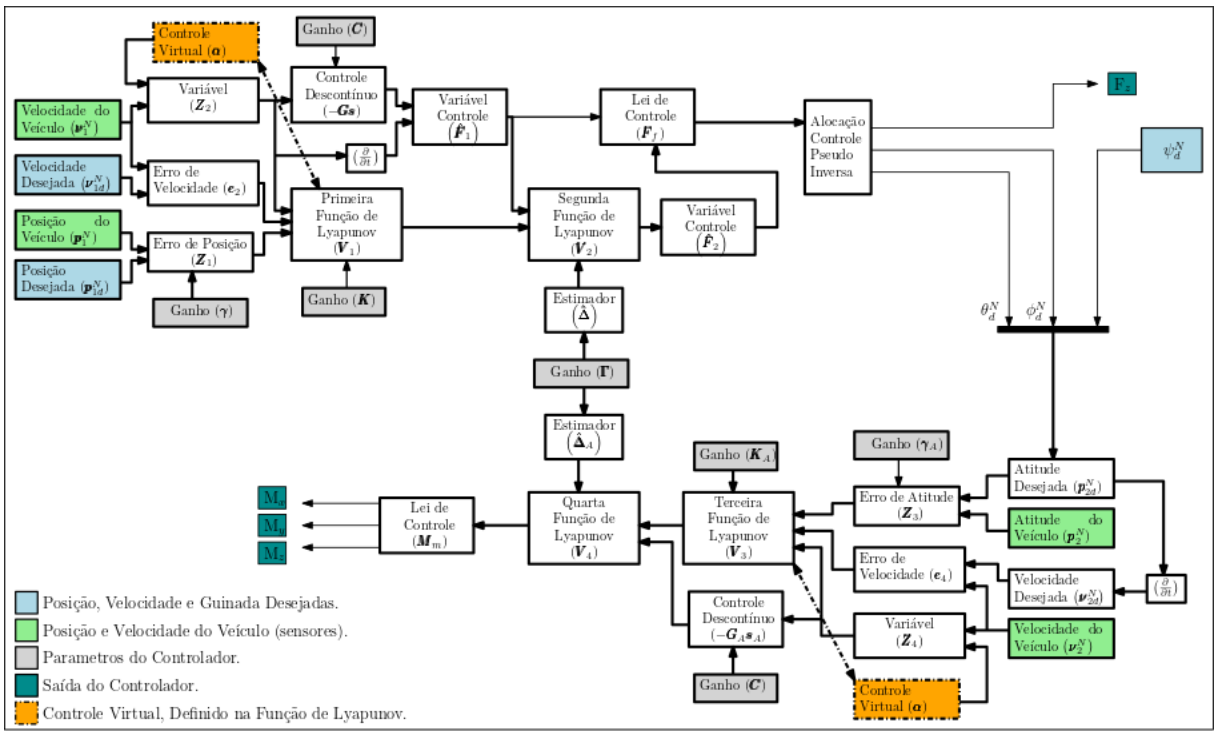


(b) Aproximação do ESC.

APÊNDICE C – Diagrama de Blocos

A Figura 84, representa o diagrama de blocos do controle integral *backstepping* com modos deslizantes desenvolvido.

Figura 84 – Diagrama de blocos do controle desenvolvido.



APÊNDICE D – Código do modo de voo ELICOID, desenvolvido em C++

```
#include "Copter.h"
#include <AC_CustomControl/AC_CustomControl_RC_POS.h>
#include <AC_AttitudeControl/AC_PosControl_IBSMC.h>
#include <stdio.h>
#include <AC_IBSMC/Pos/PosContIBSMC.h>
#include <AC_IBSMC/Pos/PosContIntBack.h>
#include <AC_IBSMC/Att/AttContIBSMC.h>
#include <AC_IBSMC/Att/AttContIBSMCv2.h>
#include <AP_Math/quaternion.h>
#include <SRV_Channel/SRV_Channel.h>
////////////////////////////////////////
////MAVLINK COMUNICATION TEST
#include <AP_Mission/AP_Mission.h>
#include <GCS_MAVLink/GCS.h>
#include <string>
#include <GCS_MAVLink/include/mavlink/v2.0/common/
   mavlink_msg_set_position_target_local_ned.h>
#include <iostream>
#include <unistd.h>
#include <cstring>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <iomanip>

#define MAX_BUFFER_SIZE 1024
#if MODE_ELICOID_ENABLED == ENABLED

static Vector3p elicoid_pos_target_cm;
bool elicoid_pos_terrain_alt;
static Vector3f elicoid_vel_target_cms;
static Vector3f elicoid_accel_target_cmss;
static uint32_t update_time_ms;
struct
{
```

```
  uint32_t update_time_ms;
  Quaternion attitude_quat;
  Vector3f ang_vel;
  float yaw_rate_cds;
  float climb_rate_cms;
  float thrust;
  bool use_yaw_rate;
  bool use_thrust;
} static elicoid_angle_state;

Vector3f ModeElicoid::ActualPos(){
  Vector3f current_position;
  if (ahrs.get_relative_position_NED_origin(current_position)){}
  return current_position;
}

Vector3f ModeElicoid::ActualVel(){
  Vector3f current_velocity;
  if (ahrs.get_velocity_NED(current_velocity)){}
  return current_velocity;
}

Vector3f ModeElicoid::ActualAtt(){
  Quaternion attitude_body;
  ahrs.get_quat_body_to_ned(attitude_body);
  Vector3f current_attitude = {attitude_body.get_euler_roll(),
     attitude_body.get_euler_pitch(), attitude_body.get_euler_yaw
     ()};
  return current_attitude;
}

bool ModeElicoid::allows_arming(AP_Arming::Method method) const{
  if (method == AP_Arming::Method::MAVLINK || method == AP_Arming
     ::Method::SCRIPTING) {
    return true;
  }
  return (copter.g2.guided_options & (uint32_t)Options::
     AllowArmingFromTX) != 0;
};
/*****************************************
set_destination - sets guided mode's target destination
```

```
Returns true if the fence is enabled and guided waypoint is
   within the fence
else return false if the waypoint is outside the fence
R.C.: USADO NA LINHA 1357 DO ARQUIVO GCS_Mavlink.cpp
*****************************************/
bool ModeElicoid::ELICOIDset_destination(const Vector3f &
   destination, bool use_yaw, float yaw_cd, bool use_yaw_rate,
   float yaw_rate_cds, bool relative_yaw, bool terrain_alt){
  if (terrain_alt){
    float origin_terr_offset;
    if (!wp_nav->get_terrain_offset(origin_terr_offset)){
      init(true);
      return false;
    }
  }
  else{
    pos_control->set_pos_offset_z_cm(0.0);
  }
  elicoid_pos_target_cm = destination.topostype();
  elicoid_pos_terrain_alt = terrain_alt;
  elicoid_vel_target_cms.zero();
  elicoid_accel_target_cmss.zero();
  update_time_ms = millis();
  return true;
}
/*****************************************
sets guided mode's target from a Location object
returns false if destination could not be set (probably caused by
    missing terrain data)
or if the fence is enabled and guided waypoint is outside the
   fence
R.C.: USADO NA LINHA 726 do arquiv GCS_Mavlink.cpp
*****************************************/
bool ModeElicoid::ELICOIDset_destination(const Location &dest_loc
   , bool use_yaw, float yaw_cd, bool use_yaw_rate, float
   yaw_rate_cds, bool relative_yaw)
{
  Vector3f pos_target_f;
  bool terrain_alt;
  if (!wp_nav->get_vector_NEU(dest_loc, pos_target_f, terrain_alt
     )){
    return false;
```

```
  }
  if (terrain_alt){
    float origin_terr_offset;
    if (!wp_nav->get_terrain_offset(origin_terr_offset)){
      init(true);
      return false;
    }
  else{
    pos_control->set_pos_offset_z_cm(0.0);
  }
  elicoid_pos_target_cm = pos_target_f.topostype();
  elicoid_pos_terrain_alt = terrain_alt;
  elicoid_vel_target_cms.zero();
  elicoid_accel_target_cmss.zero();
  update_time_ms = millis();
  return true;
}
/*****************************************
set_velaccel - sets guided mode's target velocity and
   acceleration
R.C.: USADO NA LINHA 1350 DO ARQUIVO GCS_Mavlink.cpp
*****************************************/
void ModeElicoid::ELICOIDset_velaccel(const Vector3f& velocity,
   const Vector3f& acceleration, bool use_yaw, float yaw_cd, bool
   use_yaw_rate, float yaw_rate_cds, bool relative_yaw, bool
   log_request){
  elicoid_pos_target_cm.zero();
  elicoid_pos_terrain_alt = false;
  elicoid_vel_target_cms = velocity;
  elicoid_accel_target_cmss = acceleration;
  update_time_ms = millis();
}
/*****************************************
set_destination_posvel - set guided mode position and velocity
   target
R.C.: USADO NA LINHA 1482 DO ARQUIVO GCS_Mavlink.cpp
*****************************************/
bool ModeElicoid::ELICOIDset_destination_posvel(const Vector3f&
   destination, const Vector3f& velocity, bool use_yaw, float
   yaw_cd, bool use_yaw_rate, float yaw_rate_cds, bool
   relative_yaw){
```

```
  return ELICOIDset_destination_posvelaccel(destination, velocity
      , Vector3f(), use_yaw, yaw_cd, use_yaw_rate, yaw_rate_cds,
      relative_yaw);
}
/*****************************************
set_destination_posvelaccel - set guided mode position, velocity
   and acceleration target
R.C.: USADO NA LINHA 1349 DO ARQUIVO GCS_Mavlink.cpp
*****************************************/
bool ModeElicoid::ELICOIDset_destination_posvelaccel(const
   Vector3f& destination, const Vector3f& velocity, const Vector3f
   & acceleration, bool use_yaw, float yaw_cd, bool use_yaw_rate,
   float yaw_rate_cds, bool relative_yaw){
  #if AP_FENCE_ENABLED
  const Location dest_loc(destination, Location::AltFrame::
     ABOVE_ORIGIN);
  if (!copter.fence.check_destination_within_fence(dest_loc)) {
    AP::logger().Write_Error(LogErrorSubsystem::NAVIGATION,
       LogErrorCode::DEST_OUTSIDE_FENCE);
    return false;
  }
  #endif
  update_time_ms = millis();
  elicoid_pos_target_cm = destination.topostype();
  elicoid_pos_terrain_alt = false;
  elicoid_vel_target_cms = velocity;
  elicoid_accel_target_cmss = acceleration;
  return true;
}
/*****************************************
*****************************************/
const Vector3p &ModeElicoid::ELICOIDget_target_pos() const{
  return elicoid_pos_target_cm;
}
/*****************************************
*****************************************/
const Vector3f &ModeElicoid::ELICOIDget_target_vel() const{
  return elicoid_vel_target_cms;
}
/*****************************************
*****************************************/
const Vector3f &ModeElicoid::ELICOIDget_target_accel() const{
```

```
  return elicoid_accel_target_cmss;
}
/*************************************
pos_control_run - runs the guided position controller
called from guided_run
*************************************/
void ModeElicoid::ELICOIDpos_control_run(){
  if (is_disarmed_or_landed()) {
    make_safe_ground_handling(copter.is_tradheli() && motors->
       get_interlock());
    return;
  }
  float terr_offset = 0.0f;
  if (elicoid_pos_terrain_alt && !wp_nav->get_terrain_offset(
     terr_offset)) {
    copter.failsafe_terrain_on_event();
    return;
  }
  motors->set_desired_spool_state(AP_Motors::DesiredSpoolState::
     THROTTLE_UNLIMITED);
  if (millis() - update_time_ms > ELICOIDget_timeout_ms()) {
    if ((auto_yaw.mode() == AutoYaw::Mode::RATE) || (auto_yaw.
       mode() == AutoYaw::Mode::ANGLE_RATE)) {
      auto_yaw.set_mode(AutoYaw::Mode::HOLD);
    }
  }
  float pos_offset_z_buffer = 0.0;
  if (elicoid_pos_terrain_alt) {
    pos_offset_z_buffer = MIN(copter.wp_nav->get_terrain_margin()
        * 100.0, 0.5 * fabsF(elicoid_pos_target_cm.z));
  }
  Vector3f current_position;
  Vector3f current_velocity;
  Vector3f current_attitude;
  current_position = ActualPos();
  current_velocity = ActualVel();
  current_attitude = ActualAtt();
  Vector3p pos = ELICOIDget_target_pos();
  Vector3f vel = ELICOIDget_target_vel();
  Vector3f accel = ELICOIDget_target_accel();
```

```
  pos_control->input_pos_xyz(elicoid_pos_target_cm, terr_offset,
     pos_offset_z_buffer);
  pos_control->update_xy_controller();
  pos_control->update_z_controller();
  attitude_control->input_thrust_vector_heading(pos_control->
     get_thrust_vector(), auto_yaw.get_heading());
}
/*****************************************
posvelaccel_control_run - runs the guided position, velocity and
   acceleration controller
called from guided_run
*****************************************/
void ModeElicoid::ELICOIDposvelaccel_control_run(){
  motors->set_desired_spool_state(AP_Motors::DesiredSpoolState::
     THROTTLE_UNLIMITED);
  uint32_t tnow = millis();
  if (tnow - update_time_ms > ELICOIDget_timeout_ms()) {
    elicoid_vel_target_cms.zero();
    elicoid_accel_target_cmss.zero();
    if ((auto_yaw.mode() == AutoYaw::Mode::RATE) || (auto_yaw.
       mode() == AutoYaw::Mode::ANGLE_RATE)) {
      auto_yaw.set_mode(AutoYaw::Mode::HOLD);
    }
  }
  pos_control->input_pos_vel_accel_xy(elicoid_pos_target_cm.xy(),
      elicoid_vel_target_cms.xy(), elicoid_accel_target_cmss.xy(),
      false);
  Vector3f current_position;
  Vector3f current_velocity;
  current_position = ActualPos();
  current_velocity = ActualVel();
  if (elicoid_pos_terrain_alt) {
    INTERNAL_ERROR(AP_InternalError::error_t::flow_of_control);
  }

  float pz = -elicoid_pos_target_cm.z;
  pos_control->input_pos_vel_accel_z(pz, elicoid_vel_target_cms.z
     , elicoid_accel_target_cmss.z, false);

  pos_control->update_xy_controller();
  pos_control->update_z_controller();
```

```
  // call attitude controller with auto yaw
  attitude_control->input_thrust_vector_heading(pos_control->
     get_thrust_vector(), auto_yaw.get_heading());
}
/*****************************************
return guided mode timeout in milliseconds. Only used for
   velocity, acceleration,
angle control, and angular rates
*****************************************/
uint32_t ModeElicoid::ELICOIDget_timeout_ms() const{
  return MAX(copter.g2.guided_timeout, 0.1) * 1000;
}
/*****************************************
///(PID POSITION + IBSMC ATTITUDE + TARJ.DES. MAVLINK)
*****************************************/
#if SwitchVariable_APC == ENABLED
/*****************************************
Function responsable to initialize the PID control
*****************************************/
bool ModeElicoid::init(bool ignore_checks){
  if (!pos_control->is_active_z()){
    pos_control->init_z_controller();
  }
  if (!pos_control->is_active_xy()){
    pos_control->init_xy_controller();
  }
  pos_control->set_max_speed_accel_z(-get_pilot_speed_dn(), g.
     pilot_speed_up, g.pilot_accel_z);
  pos_control->set_correction_speed_accel_z(-get_pilot_speed_dn()
     , g.pilot_speed_up, g.pilot_accel_z);
  pos_control->set_max_speed_accel_xy(wp_nav->
     get_default_speed_xy(), wp_nav->get_wp_acceleration());
  pos_control->set_correction_speed_accel_xy(wp_nav->
     get_default_speed_xy(), wp_nav->get_wp_acceleration());
  return true;
}
/*****************************************
Function responsable to run the Auto Helicoid Trajectory with the
    PID control
*****************************************/
void ModeElicoid::run(){
```

```
motors->set_desired_spool_state(AP_Motors::DesiredSpoolState::
   THROTTLE_UNLIMITED);
uint32_t tnow = millis();
if (tnow - update_time_ms > ELICOIDget_timeout_ms()){
  elicoid_vel_target_cms.zero();
  elicoid_accel_target_cmss.zero();
  if ((auto_yaw.mode() == AutoYaw::Mode::RATE) || (auto_yaw.
     mode() == AutoYaw::Mode::ANGLE_RATE)){
    auto_yaw.set_mode(AutoYaw::Mode::HOLD);
  }
}
Vector3f current_position;
Vector3f current_velocity;
current_position = ActualPos();
current_velocity = ActualVel();
Vector3f current_attitude;
current_attitude = ActualAtt();

float px = elicoid_pos_target_cm.x / 100.0f;
float py = elicoid_pos_target_cm.y / 100.0f;
float pz = elicoid_pos_target_cm.z / 100.0f;
Vector3f target_position_xyz = {px, py, pz};
Vector3f target_velocity_xyz = ELICOIDget_target_vel() / 100.0f
   ;

pos_control->set_max_speed_accel_z(-get_pilot_speed_dn(), g.
   pilot_speed_up, g.pilot_accel_z);
float zero2 = 0;
pos_control->input_pos_vel_accel_z(target_position_xyz[2],
   zero2, 0.0f);
Vector2p pos_target_xy = {target_position_xyz[0],
   target_position_xyz[1]};
Vector2f vel_target_xy = {target_velocity_xyz[0],
   target_velocity_xyz[1]};
Vector2f acc_target_xy = {0.0f, 0.0f};
pos_control->input_pos_vel_accel_xy(pos_target_xy,
   vel_target_xy, acc_target_xy);
pos_control->update_xy_controller();
pos_control->update_z_controller();

attitude_control->set_throttle_out(motors->get_throttle(), true
   , g.throttle_filt);
```

```
  static AttContIBSMCModelClass AttContIBSMC_Obj;
  Vector3f gyro_latest = ahrs.get_gyro_latest();
  float arg_Pn2[3]{current_attitude.x, current_attitude.y,
     current_attitude.z};

  float yaw_ibsmc = 0.0;
  float arg_Pn2d[3]{roll_ibsmc, pitch_ibsmc, yaw_ibsmc};
  float arg_dPn2[3]{gyro_latest.x, gyro_latest.y, gyro_latest.z};
  float arg_OutputATT[3];
  AttContIBSMC_Obj.step(arg_Pn2, arg_Pn2d, arg_dPn2,
     arg_OutputATT);
  float mx_ibsmc = arg_OutputATT[0];
  float my_ibsmc = arg_OutputATT[1];
  float mz_ibsmc = arg_OutputATT[2];

  motors->set_roll(mx_ibsmc);
  motors->set_pitch(my_ibsmc);
  motors->set_yaw(mz_ibsmc);
}
#endif
/*****************************************
(IB POSITION + IBSMC ATTITUDE + TARJ.DES. MAVLINK)
*****************************************/
#if SwitchVariable_AIC == ENABLED
/*****************************************
Function responsable to initialize the IBSMC control
*****************************************/
bool ModeElicoid::init(bool ignore_checks){
  static PosContIntBackModelClass PosContIntBack_Obj;
  PosContIntBack_Obj.initialize();
  static AttContIBSMCv2ModelClass AttContIBSMCv2_Obj;
  AttContIBSMCv2_Obj.initialize();
  return true;
}
/*****************************************
ELICOID_AIC_posvel_IBSMC_run - runs the elicoid position and
   velocity controller IBSMC
called from elicoid_run AIC
*****************************************/
void ModeElicoid::ELICOID_AIC_posvel_IBSMC_run(){
  motors->set_desired_spool_state(AP_Motors::DesiredSpoolState::
     THROTTLE_UNLIMITED);
```

```
// set velocity to zero and stop rotating if no updates
   received for 3 seconds
uint32_t tnow = millis();
if (tnow - update_time_ms > ELICOIDget_timeout_ms()){
  elicoid_vel_target_cms.zero();
  elicoid_accel_target_cmss.zero();
  if ((auto_yaw.mode() == AutoYaw::Mode::RATE) || (auto_yaw.
     mode() == AutoYaw::Mode::ANGLE_RATE)) {
    auto_yaw.set_mode(AutoYaw::Mode::HOLD);
  }
}
static PosContIntBackModelClass PosContIntBack_Obj;
Vector3f current_position;
Vector3f current_velocity;
Vector3f current_attitude;
current_position = ActualPos();
current_velocity = ActualVel();
current_attitude = ActualAtt();

float px = elicoid_pos_target_cm.x / 100.0f;
float py = elicoid_pos_target_cm.y / 100.0f;
float pz = elicoid_pos_target_cm.z / 100.0f;
Vector3f target_position_xyz = {px, py, pz};
Vector3f target_velocity_xyz = ELICOIDget_target_vel() / 100.0f
   ;

float arg_Pn1d[3]{(float)target_position_xyz[0], (float)
   target_position_xyz[1], (float)target_position_xyz[2]};
float arg_dPn1d[3]{target_velocity_xyz[0], target_velocity_xyz
   [1], target_velocity_xyz[2]};
float arg_Pn1[3]{current_position[0], current_position[1],
   current_position[2]};
float arg_dPn1[3]{current_velocity[0], current_velocity[1],
   current_velocity[2]};
float arg_Pn2[3]{current_attitude.x, current_attitude.y,
   current_attitude.z};
float arg_Output[3];
PosContIntBack_Obj.step(arg_Pn2, arg_Pn1, arg_Pn1d, arg_dPn1,
   arg_dPn1d, arg_Output);
float throttle_intback = arg_Output[0];
float roll_intback = arg_Output[1];
float pitch_intback = arg_Output[2];
```

```
  float yaw_intback = 0.0f;

  static AttContIBSMCv2ModelClass AttContIBSMCv2_Obj;
  Vector3f gyro_latest = ahrs.get_gyro_latest();
  float arg_Pn2ATT[3]{current_attitude.x, current_attitude.y,
     current_attitude.z};
  float arg_Pn2d[3]{roll_intback, pitch_intback, yaw_intback};
  float arg_dPn2[3]{gyro_latest.x, gyro_latest.y, gyro_latest.z};
  float arg_OutputATT[3];
  AttContIBSMCv2_Obj.step(arg_Pn2ATT, arg_Pn2d, arg_dPn2,
     arg_OutputATT);
  float mx_ibsmc = arg_OutputATT[0];
  float my_ibsmc = arg_OutputATT[1];
  float mz_ibsmc = arg_OutputATT[2];

  motors->set_throttle(throttle_intback);
  motors->set_roll(mx_ibsmc);
  motors->set_pitch(my_ibsmc);
  motors->set_yaw(mz_ibsmc);
}
/*****************************************
Function responsable to run the Auto Helicoid Trajectory with the
    IBSMC control
*****************************************/
void ModeElicoid::run(){
  ELICOID_AIC_posvel_IBSMC_run();
}
#endif
/*****************************************
EIC (IBSMC POSITION E ATTITUDE + TARJ.DES. MAVLINK)
*****************************************/
#if SwitchVariable_EIC == ENABLED
/*****************************************
Function responsable to initialize the IBSMC control
*****************************************/
bool ModeElicoid::init(bool ignore_checks){
  static PosContIBSMCModelClass PosContIBSMC_Obj;
  PosContIBSMC_Obj.initialize();
  static AttContIBSMCModelClass AttContIBSMC_Obj;
  AttContIBSMC_Obj.initialize();
  gcs().send_text(MAV_SEVERITY_INFO, "Integral Adpative
     Backstepping Sliding Mode Control initialization is done.");
```

```
  return true;
}
/*************************************
ELICOIDposvelaccel_IBSMC_run - runs the elicoid position and
   velocity controller IBSMC
called from elicoid_run EIC
*************************************/
void ModeElicoid::ELICOIDposvelaccel_IBSMC_run(){
  motors->set_desired_spool_state(AP_Motors::DesiredSpoolState::
     THROTTLE_UNLIMITED);
  uint32_t tnow = millis();
  if (tnow - update_time_ms > ELICOIDget_timeout_ms()) {
    elicoid_vel_target_cms.zero();
    elicoid_accel_target_cmss.zero();
    if ((auto_yaw.mode() == AutoYaw::Mode::RATE) || (auto_yaw.
       mode() == AutoYaw::Mode::ANGLE_RATE)) {
      auto_yaw.set_mode(AutoYaw::Mode::HOLD);
    }
  }
  static PosContIBSMCModelClass PosContIBSMC_Obj;
  Vector3f current_position;
  Vector3f current_velocity;
  Vector3f current_attitude;
  current_position = ActualPos();
  current_velocity = ActualVel();
  current_attitude = ActualAtt();
  float px = elicoid_pos_target_cm.x / 100.0f;
  float py = elicoid_pos_target_cm.y / 100.0f;
  float pz = elicoid_pos_target_cm.z / 100.0f;

  Vector3f target_velocity_xyz = ELICOIDget_target_vel()/100.0f;

  float arg_Pn1d[3]{px, py, pz};
  float arg_dPn1d[3]{target_velocity_xyz[0], target_velocity_xyz
     [1], target_velocity_xyz[2]};
  float arg_Pn1[3]{current_position[0], current_position[1],
     current_position[2]};
  float arg_dPn1[3]{current_velocity[0], current_velocity[1],
     current_velocity[2]};
  float arg_Pn2[3]{current_attitude.x, current_attitude.y,
     current_attitude.z};
  float arg_Output[3];
```

```
  static AttContIBSMCv2ModelClass AttContIBSMC_Obj;
  Vector3f gyro_latest = ahrs.get_gyro_latest();
  float yaw_ibsmc = DEG_TO_RAD*(auto_yaw.get_heading().
     yaw_angle_cd);
  float arg_Pn2d[3]{roll_ibsmc, pitch_ibsmc, yaw_ibsmc};
  float arg_dPn2[3]{ gyro_latest.x, gyro_latest.y, gyro_latest.z
     };
  float arg_OutputATT[3];
  AttContIBSMC_Obj.step(arg_Pn2, arg_Pn2d, arg_dPn2,
     arg_OutputATT);
  float mx_ibsmc = arg_OutputATT[0];
  float my_ibsmc = arg_OutputATT[1];
  float mz_ibsmc = arg_OutputATT[2];
  motors->set_throttle(throttle_ibsmc);
  motors->set_roll(mx_ibsmc);
  motors->set_pitch(my_ibsmc);
  motors->set_yaw(mz_ibsmc);
}
void ModeElicoid::run(){
  ELICOIDposvelaccel_IBSMC_run();
}

#endif
#endif
```

APÊNDICE E – Códigos em C++, modificados do ArduPilot e desenvolvidos

Nesta seção apresenta-se as modificações feitas no software ArduPilot.

Código modificado da biblioteca mode.h

Esta biblioteca é muito extensa, mais de 2000 linhas de código, portanto será apresentada somente as mudanças realizadas. O trecho de código a seguir foi acrescentado no final do código já existente.

```
////////////////////////////////////////////
//// HELICOIDAL MODE DESIGNED BY R. CARDOSO
////////////////////////////////////////////
#if MODE_ELICOID_ENABLED == ENABLED
class ModeElicoid : public Mode {
public:
  // inherit constructor
  using Mode::Mode;
  Number mode_number() const override { return Number::ELICOID; }
  ////////////////////////////////////////////
  //// Variable to define which case run  ////
  ////////////////////////////////////////////
  // (PID POSITION + IBSMC ATTITUDE + TARJ.DES. MAVLINK)
  #ifndef SwitchVariable_APC
  # define SwitchVariable_APC DISABLED
  #endif
  // (IB POSITION + IBSMC ATTITUDE + TARJ.DES. MAVLINK)
  #ifndef SwitchVariable_AIC
  # define SwitchVariable_AIC ENABLED
  #endif
  // (IBSMC POSITION E ATTITUDE + TARJ.DES. MAVLINK)
  #ifndef SwitchVariable_EIC
  # define SwitchVariable_EIC DISABLED
  #endif
  Vector3f ActualPos();
  Vector3f ActualVel();
  Vector3f ActualAtt();
```

```
    ////////////////////////////////////////////
    ////////////////////////////////////////////
    #if SwitchVariable_APC == ENABLED
    //----------------------------------------
    bool init(bool ignore_checks) override;
    virtual void run() override;
    //----------------------------------------
    #endif
    ////////////////////////////////////////////
    ////////////////////////////////////////////
    #if SwitchVariable_AIC == ENABLED
    //----------------------------------------
    bool init(bool ignore_checks) override;
    virtual void run() override;
    //----------------------------------------
    #endif
    ////////////////////////////////////////////
    ////////////////////////////////////////////
    #if SwitchVariable_EIC == ENABLED
    //----------------------------------------
    bool init(bool ignore_checks) override;
    virtual void run() override;
    //----------------------------------------
    #endif
    ////////////////////////////////////////////
    ////////////////////////////////////////////
    bool requires_GPS() const override { return true; }
    bool has_manual_throttle() const override { return false; }
    bool allows_arming(AP_Arming::Method method) const override;
    bool is_autopilot() const override { return true; }

    bool has_user_takeoff(bool must_navigate) const override {
       return true; }
    bool in_guided_mode() const override { return true; }

    bool ELICOIDset_destination(const Vector3f &destination, bool
       use_yaw = false, float yaw_cd = 0.0, bool use_yaw_rate =
       false, float yaw_rate_cds = 0.0, bool yaw_relative = false,
       bool terrain_alt = false);
    bool ELICOIDset_destination(const Location &dest_loc, bool
       use_yaw = false, float yaw_cd = 0.0, bool use_yaw_rate =
       false, float yaw_rate_cds = 0.0, bool yaw_relative = false);
```

```
  void ELICOIDset_velaccel(const Vector3f &velocity, const
     Vector3f &acceleration, bool use_yaw = false, float yaw_cd =
     0.0, bool use_yaw_rate = false, float yaw_rate_cds = 0.0,
     bool yaw_relative = false, bool log_request = true);
  bool ELICOIDset_destination_posvel(const Vector3f &destination,
      const Vector3f &velocity, bool use_yaw = false, float yaw_cd
      = 0.0, bool use_yaw_rate = false, float yaw_rate_cds = 0.0,
     bool yaw_relative = false);
  bool ELICOIDset_destination_posvelaccel(const Vector3f &
     destination, const Vector3f &velocity, const Vector3f &
     acceleration, bool use_yaw = false, float yaw_cd = 0.0, bool
     use_yaw_rate = false, float yaw_rate_cds = 0.0, bool
     yaw_relative = false);
  // get position, velocity and acceleration targets
  const Vector3p &ELICOIDget_target_pos() const;
  const Vector3f &ELICOIDget_target_vel() const;
  const Vector3f &ELICOIDget_target_accel() const;

  uint32_t ELICOIDget_timeout_ms() const;
  int8_t motorEnable; // whether to raise motor PWM
protected:
  const char *name() const override { return "ELICOID"; }
  const char *name4() const override { return "ELIC"; }
private:
  // enum for GUID_OPTIONS parameter
  enum class Options : int32_t {
    AllowArmingFromTX   = (1U << 0),
    // this bit is still available, pilot yaw was mapped to bit 2
        for symmetry with auto
    IgnorePilotYaw      = (1U << 2),
    SetAttitudeTarget_ThrustAsThrust = (1U << 3),
    DoNotStabilizePositionXY = (1U << 4),
    DoNotStabilizeVelocityXY = (1U << 5),
    WPNavUsedForPosControl = (1U << 6),
    AllowWeatherVaning = (1U << 7)
  };
  void ELICOIDpos_control_run();
  void ELICOIDposvelaccel_control_run();
  void ELICOID_AIC_posvel_IBSMC_run();
  void ELICOIDposvelaccel_IBSMC_run();
};
#endif
```

Código modificado da biblioteca GCS_Mavlink.cpp

Esta biblioteca é muito extensa, mais de 1500 linhas de código, portanto será apresentado somente algumas das 22 mudanças realizadas, as mais importantes. As demais somente habilita o modo ELICOID para receber informações via MAVLink, assim basta alterar todas as vezes que aparece o seguinte trecho de código:

```
#if MODE_GUIDED_ENABLED == ENABLED
```

para

```
#if MODE_GUIDED_ENABLED == ENABLED || MODE_ELICOID_ENABLED ==
   ENABLED
```

A principal alteração ocorreu na função: («*void GCS_MAVLINK_Copter :: handleMessage(const mavlink_message_t &msg)*»), dentro do comando («*case MAVLINK_MSG_ID_SET_POSITION_TARGET_LOCAL_NED:*»), na LINHA 1342, responsável por habilitar o envio de posição, velocidade e aceleração com relação a origem:

```
if (!pos_ignore && !vel_ignore){
  copter.mode_elicoid.ELICOIDset_destination_posvelaccel(
     pos_vector, vel_vector, accel_vector, !yaw_ignore, yaw_cd, !
     yaw_rate_ignore, yaw_rate_cds, yaw_relative);
} else if (pos_ignore && !vel_ignore){
  copter.mode_elicoid.ELICOIDset_velaccel(vel_vector,
     accel_vector, !yaw_ignore, yaw_cd, !yaw_rate_ignore,
     yaw_rate_cds, yaw_relative);
}
else if (!pos_ignore && vel_ignore && acc_ignore){
  copter.mode_elicoid.ELICOIDset_destination(pos_vector, !
     yaw_ignore, yaw_cd, !yaw_rate_ignore, yaw_rate_cds,
     yaw_relative, false);
}
else
{
  // input is not valid so stop
  copter.mode_elicoid.init(true);
}
```

e dentro do comando («*case MAVLINK_MSG_ID_SET_POSITION_TARGET_GLOBAL_INT:*»), na LINHA 1451, responsável por habilitar o envio de posição, velocidade e aceleração com relação as atuais::

```
if (!pos_ignore && !vel_ignore) {
  // convert Location to vector from ekf origin for posvel
     controller
  if (loc.get_alt_frame() == Location::AltFrame::ABOVE_TERRAIN) {
    copter.mode_guided.init(true);
    copter.mode_elicoid.init(true);
    break;
  }
  Vector3f pos_neu_cm;
  if (!loc.get_vector_from_origin_NEU(pos_neu_cm)) {
    copter.mode_guided.init(true);
    copter.mode_elicoid.init(true);
    break;
  }
  copter.mode_guided.set_destination_posvel(pos_neu_cm,
     vel_vector, !yaw_ignore, yaw_cd, !yaw_rate_ignore,
     yaw_rate_cds);
  copter.mode_elicoid.ELICOIDset_destination_posvel(pos_neu_cm,
     vel_vector, !yaw_ignore, yaw_cd, !yaw_rate_ignore,
     yaw_rate_cds);
} else if (pos_ignore && !vel_ignore) {
  copter.mode_guided.set_velaccel(vel_vector, accel_vector, !
     yaw_ignore, yaw_cd, !yaw_rate_ignore, yaw_rate_cds);
} else if (pos_ignore && vel_ignore && !acc_ignore) {
  copter.mode_guided.set_accel(accel_vector, !yaw_ignore, yaw_cd,
      !yaw_rate_ignore, yaw_rate_cds);
} else if (!pos_ignore && vel_ignore && acc_ignore) {
  copter.mode_guided.set_destination(loc, !yaw_ignore, yaw_cd, !
     yaw_rate_ignore, yaw_rate_cds);
  copter.mode_elicoid.ELICOIDset_destination(loc, !yaw_ignore,
     yaw_cd, !yaw_rate_ignore, yaw_rate_cds);
} else {
  // input is not valid so stop
  copter.mode_guided.init(true);
  copter.mode_elicoid.init(true);
}
```

APÊNDICE F – Códigos em C++, gerados pelo Simulink^{R} Coder

Neste anexo apresenta-se os códigos gerados pelo Simulink^{R}, por questão de espaço somente a função step, responsável por fazer a atualização dos controladores.

Função *step* do controle de posição, PosContIBSMC.cpp

```
void  PosContIBSMCModelClass::step(real32_T arg_Pn2[3], real32_T
   arg_Pn1[3], real32_T arg_Pn1d[3], real32_T arg_dPn1[3],
   real32_T arg_dPn1d[3], real32_T arg_outputPOS[3])
{
 real32_T Fz_aux;
 real32_T phi_d_aux;
 real32_T theta_d_aux;
 real32_T SMCpos1;
 real32_T SMCpos2;
 real32_T SMCpos3;
 PosContIBSMC_B.SomaEP1 = arg_Pn1[0] - arg_Pn1d[0];
 //***** DiscreteIntegrator:'DTIntEP1'*******************//
 PosContIBSMC_B.DTIntEP1 = PosContIBSMC_DW.DTIntEP1_DSTATE;
 PosContIBSMC_DW.DTIntEP1_DSTATE += PosContIBSMC_P.
    DTIntEP1_gainval * PosContIBSMC_B.SomaEP1;
 //***** DiscreteIntegrator:'FIM'************************//
 PosContIBSMC_B.RPO_GAMA_P1_IntEP1 = PosContIBSMC_P.RPO_GAMA_P1 *
     PosContIBSMC_B.DTIntEP1;
 PosContIBSMC_B.Z1_1 = PosContIBSMC_B.SomaEP1 + PosContIBSMC_B.
    RPO_GAMA_P1_IntEP1;
 PosContIBSMC_B.RPO_KP_1_Z1_1 = PosContIBSMC_P.RPO_KP_1 *
    PosContIBSMC_B.Z1_1;
 PosContIBSMC_B.SomadEP1 = arg_dPn1[0] - arg_dPn1d[0];
 PosContIBSMC_B.RPO_GAMA_P1_EP1 = PosContIBSMC_P.RPO_GAMA_P1 *
    PosContIBSMC_B.SomaEP1;
 PosContIBSMC_B.Z2_1 = (PosContIBSMC_B.RPO_KP_1_Z1_1 +
    PosContIBSMC_B.SomadEP1) + PosContIBSMC_B.RPO_GAMA_P1_EP1;
 SMCpos1 = PosContIBSMC_P.LAMBDA * PosContIBSMC_P.LAMBDA *
    PosContIBSMC_B.Z2_1;
```

```
PosContIBSMC_B.SMCpos_1 = -SMCpos1 / ((PosContIBSMC_P.LAMBDA *
   PosContIBSMC_P.EPISON1_POS) + std::abs(SMCpos1));
PosContIBSMC_B.dotDeltaP_1 = (PosContIBSMC_P.RPO_GGAMA_P1) *
   PosContIBSMC_B.Z2_1;
//***** DiscreteIntegrator:'DTIntDeltaP1' ***************//
PosContIBSMC_B.DeltaP1 = PosContIBSMC_DW.DTIntDeltaP1_DSTATE;
PosContIBSMC_DW.DTIntDeltaP1_DSTATE += PosContIBSMC_P.
   DTIntDeltaP1_gainval * PosContIBSMC_B.dotDeltaP_1;
//****** DiscreteIntegrator:'FIM' ************************//
//****** DiscreteDerivative:'DotZ1_1' ********************//
PosContIBSMC_B.Diff = (PosContIBSMC_B.Z1_1 - PosContIBSMC_DW.
   UD_DSTATE) / (1.0f / PosContIBSMC_P.TSamp_WtEt);
PosContIBSMC_DW.UD_DSTATE = PosContIBSMC_B.Z1_1;
//****** DiscreteDerivative:'FIM' ************************//
PosContIBSMC_B.RPO_KP_1_DZ1_1 = PosContIBSMC_P.RPO_KP_1 *
   PosContIBSMC_B.Diff;
PosContIBSMC_B.RPO_GAMA_P1_dEP1 = PosContIBSMC_P.RPO_GAMA_P1 *
   PosContIBSMC_B.SomadEP1;
PosContIBSMC_B.SOMA_BU_1 = arg_dPn1d[0] - PosContIBSMC_B.DeltaP1
    - PosContIBSMC_B.SMCpos_1 - PosContIBSMC_B.Z1_1 -
PosContIBSMC_B.RPO_KP_1_DZ1_1 - PosContIBSMC_B.RPO_GAMA_P1_dEP1;
PosContIBSMC_B.SomaEP2 = arg_Pn1[1] - arg_Pn1d[1];
//****** DiscreteIntegrator: 'DTIntEP2' ******************//
PosContIBSMC_B.DTIntEP2 = PosContIBSMC_DW.DTIntEP2_DSTATE;
PosContIBSMC_DW.DTIntEP2_DSTATE += PosContIBSMC_P.
   DTIntEP2_gainval * (PosContIBSMC_B.SomaEP2);
//****** DiscreteIntegrator: 'FIM' ************************//
PosContIBSMC_B.RPO_GAMA_P2_IntEP2 = PosContIBSMC_P.RPO_GAMA_P2 *
    PosContIBSMC_B.DTIntEP2;
PosContIBSMC_B.Z1_2 = PosContIBSMC_B.SomaEP2 + PosContIBSMC_B.
   RPO_GAMA_P2_IntEP2;
PosContIBSMC_B.RPO_KP_2_Z1_2 = PosContIBSMC_P.RPO_KP_2 *
   PosContIBSMC_B.Z1_2;
PosContIBSMC_B.SomadEP2 = arg_dPn1[1] - arg_dPn1d[1];
PosContIBSMC_B.RPO_GAMA_P2_EP2 = PosContIBSMC_P.RPO_GAMA_P2 *
   PosContIBSMC_B.SomaEP2;
PosContIBSMC_B.Z2_2 = (PosContIBSMC_B.RPO_KP_2_Z1_2 +
   PosContIBSMC_B.SomadEP2) + PosContIBSMC_B.RPO_GAMA_P2_EP2;
SMCpos2 = (PosContIBSMC_P.LAMBDA) * ((PosContIBSMC_P.LAMBDA) *
   PosContIBSMC_B.Z2_2);
PosContIBSMC_B.SMCpos_2 = -SMCpos2 / ((PosContIBSMC_P.LAMBDA *
   PosContIBSMC_P.EPISON2_POS) + std::abs(SMCpos2));
```

```
PosContIBSMC_B.dotDeltaP_2 = (PosContIBSMC_P.RPO_GGAMA_P2) *
   PosContIBSMC_B.Z2_2;
//****** DiscreteIntegrator:'DTIntDeltaP2' ***************//
PosContIBSMC_B.DeltaP2 = PosContIBSMC_DW.DTIntDeltaP2_DSTATE;
PosContIBSMC_DW.DTIntDeltaP2_DSTATE += PosContIBSMC_P.
   DTIntDeltaP2_gainval * PosContIBSMC_B.dotDeltaP_2;
//****** DiscreteIntegrator:'FIM' ************************//
//****** DiscreteDerivative:'DotZ1_2' ********************//
PosContIBSMC_B.Diff_b = (PosContIBSMC_B.Z1_2 - PosContIBSMC_DW.
   UD_DSTATE_i) / (1.0f / PosContIBSMC_P.TSamp_WtEt_d);
PosContIBSMC_DW.UD_DSTATE_i = PosContIBSMC_B.Z1_2;
//****** DiscreteDerivative:'FIM' ************************//
PosContIBSMC_B.RPO_KP_2_DZ1_2 = PosContIBSMC_P.RPO_KP_2 *
   PosContIBSMC_B.Diff_b;
PosContIBSMC_B.RPO_GAMA_P2_dEP2 = PosContIBSMC_P.RPO_GAMA_P2 *
   PosContIBSMC_B.SomadEP2;
PosContIBSMC_B.SOMA_BU_2 = arg_dPn1d[1] - PosContIBSMC_B.DeltaP2
    - PosContIBSMC_B.SMCpos_2 - PosContIBSMC_B.Z1_2 -
PosContIBSMC_B.RPO_KP_2_DZ1_2 - PosContIBSMC_B.RPO_GAMA_P2_dEP2;
PosContIBSMC_B.SomaEP3 = arg_Pn1[2] - arg_Pn1d[2];
//****** DiscreteIntegrator: 'DTIntEP3' ********************//
PosContIBSMC_B.DTIntEP3 = PosContIBSMC_DW.DTIntEP3_DSTATE;
PosContIBSMC_DW.DTIntEP3_DSTATE += PosContIBSMC_P.
   DTIntEP3_gainval * (PosContIBSMC_B.SomaEP3);
//****** DiscreteIntegrator: 'FIM' ************************//
PosContIBSMC_B.RPO_GAMA_P3_IntEP3 = PosContIBSMC_P.RPO_GAMA_P3 *
    PosContIBSMC_B.DTIntEP3;
PosContIBSMC_B.Z1_3 = PosContIBSMC_B.SomaEP3 + PosContIBSMC_B.
   RPO_GAMA_P3_IntEP3;
PosContIBSMC_B.RPO_KP_3_Z1_3 = PosContIBSMC_P.RPO_KP_3 *
   PosContIBSMC_B.Z1_3;
PosContIBSMC_B.SomadEP3 = arg_dPn1[2] - arg_dPn1d[2];
PosContIBSMC_B.RPO_GAMA_P3_EP3 = PosContIBSMC_P.RPO_GAMA_P3 *
   PosContIBSMC_B.SomaEP3;
PosContIBSMC_B.Z2_3 = (PosContIBSMC_B.RPO_KP_3_Z1_3 +
   PosContIBSMC_B.SomadEP3) + PosContIBSMC_B.RPO_GAMA_P3_EP3;
SMCpos3 = (PosContIBSMC_P.LAMBDA) * ((PosContIBSMC_P.LAMBDA) *
   PosContIBSMC_B.Z2_3);
PosContIBSMC_B.SMCpos_3 = -SMCpos3 / ((PosContIBSMC_P.LAMBDA *
   PosContIBSMC_P.EPISON3_POS) + std::abs(SMCpos3));
PosContIBSMC_B.dotDeltaP_3 = (PosContIBSMC_P.RPO_GGAMA_P3) *
   PosContIBSMC_B.Z2_3;
```

```
//****** DiscreteIntegrator:'DTIntDeltaP3' ******************//
PosContIBSMC_B.DeltaP3 = PosContIBSMC_DW.DTIntDeltaP3_DSTATE;
// Update for DiscreteIntegrator: '<Root>/DTIntDeltaP3'
PosContIBSMC_DW.DTIntDeltaP3_DSTATE += PosContIBSMC_P.
   DTIntDeltaP3_gainval * PosContIBSMC_B.dotDeltaP_3;
//****** DiscreteIntegrator:'FIM' ***************************//
//****** DiscreteDerivative:'DotZ1_3' ***********************//
PosContIBSMC_B.Diff_i = (PosContIBSMC_B.Z1_3 - PosContIBSMC_DW.
   UD_DSTATE_k) / (1.0f / PosContIBSMC_P.TSamp_WtEt_do);
PosContIBSMC_DW.UD_DSTATE_k = PosContIBSMC_B.Z1_3;
//****** DiscreteDerivative:'FIM' ***************************//
PosContIBSMC_B.RPO_KP_3_DZ1_3 = PosContIBSMC_P.RPO_KP_3 *
   PosContIBSMC_B.Diff_i;
PosContIBSMC_B.RPO_GAMA_P3_dEP3 = PosContIBSMC_P.RPO_GAMA_P3 *
   PosContIBSMC_B.SomadEP3;
PosContIBSMC_B.SOMA_BU_3 = arg_dPn1d[2] - PosContIBSMC_B.DeltaP3
    - PosContIBSMC_B.SMCpos_3 - PosContIBSMC_B.Z1_3 -
PosContIBSMC_B.RPO_KP_3_DZ1_3 - PosContIBSMC_B.RPO_GAMA_P3_dEP3;

Fz_aux = (((-PosContIBSMC_P.MASSA_TOTAL) * std::sin(arg_Pn2[0])
   * std::sin(arg_Pn2[2]) -
(PosContIBSMC_P.MASSA_TOTAL) * std::cos(arg_Pn2[0]) * std::cos(
   arg_Pn2[2]) * std::sin(arg_Pn2[1])) *
PosContIBSMC_B.SOMA_BU_1 +
((PosContIBSMC_P.MASSA_TOTAL) * std::cos(arg_Pn2[2]) * std::sin(
   arg_Pn2[0]) -
(PosContIBSMC_P.MASSA_TOTAL) * std::cos(arg_Pn2[0]) * std::sin(
   arg_Pn2[2]) * std::sin(arg_Pn2[1])) *
PosContIBSMC_B.SOMA_BU_2) +
(-PosContIBSMC_P.MASSA_TOTAL) * std::cos(arg_Pn2[0]) * std::cos(
   arg_Pn2[1]) * PosContIBSMC_B.SOMA_BU_3;

PosContIBSMC_B.Fz = Fz_aux / (4.0 * PosContIBSMC_P.FORCE_MAX);

phi_d_aux = (-PosContIBSMC_P.MASSA_TOTAL) * (PosContIBSMC_B.
   SOMA_BU_1 * std::sin(arg_Pn2[2]) -
PosContIBSMC_B.SOMA_BU_2 * std::cos(arg_Pn2[2])) / Fz_aux;
if (phi_d_aux >= 0.9848F)
{
 phi_d_aux = 0.9848F;
}
else if (phi_d_aux <= -0.9848F)
```

```
{
 phi_d_aux = -0.9848F;
}
PosContIBSMC_B.phi_d = std::asin(phi_d_aux);
if ((PosContIBSMC_B.SOMA_BU_3 <= 1.29246971E-26F) && (
   PosContIBSMC_B.SOMA_BU_3 >= -1.29246971E-26F)) {
 PosContIBSMC_B.theta_d = 0.0F;
} else {
 theta_d_aux = (PosContIBSMC_B.SOMA_BU_1 * std::cos(arg_Pn2[2])
    + PosContIBSMC_B.SOMA_BU_2 * std::sin(arg_Pn2[2])) /
    PosContIBSMC_B.SOMA_BU_3;
 PosContIBSMC_B.theta_d = std::atan(theta_d_aux);
}
arg_outputPOS[0] = constrain_float(PosContIBSMC_B.Fz,0,1);
arg_outputPOS[1] = constrain_float(calc_lowpass_alpha_dt(1.0f /
   PosContIBSMC_P.TSamp_WtEt_do, 8.0f) * PosContIBSMC_B.phi_d, -
   M_PI_2, M_PI_2);
arg_outputPOS[2] = constrain_float(calc_lowpass_alpha_dt(1.0f /
   PosContIBSMC_P.TSamp_WtEt_do, 8.0f) * PosContIBSMC_B.theta_d,
   -M_PI_2, M_PI_2);
//****** Salva as Variaveis *************************//
AP::logger().Write("TPI2", "TimeUS,Diff_ZP2,ZP2,DotDP2,DP2",
"Qffff",
AP_HAL::micros64(),
(double)PosContIBSMC_B.Diff_b,
(double)PosContIBSMC_B.Z1_2,
(double)PosContIBSMC_B.dotDeltaP_2,
(double)PosContIBSMC_B.DeltaP2);
AP::logger().Write("TPI3", "TimeUS,Diff_ZP3,ZP3,DotDP3,DP3",
"Qffff",
AP_HAL::micros64(),
(double)PosContIBSMC_B.Diff_i,
(double)PosContIBSMC_B.Z1_3,
(double)PosContIBSMC_B.dotDeltaP_3,
(double)PosContIBSMC_B.DeltaP3);
AP::logger().Write("TPI1", "TimeUS,Diff_ZP1,ZP1,DotDP1,DP1",
"Qffff",
AP_HAL::micros64(),
(double)PosContIBSMC_B.Diff,
(double)PosContIBSMC_B.Z1_1,
(double)PosContIBSMC_B.dotDeltaP_1,
(double)PosContIBSMC_B.DeltaP1);
```

```
 AP::logger().Write("SMP0", "TimeUS,RegP_Z21,RegP_Z22,RegP_Z23",
 "Qfff",
 AP_HAL::micros64(),
 (double)PosContIBSMC_B.Z2_1,
 (double)PosContIBSMC_B.Z2_2,
 (double)PosContIBSMC_B.Z2_3);
 AP::logger().Write("SMP1", "TimeUS,RegP_Z11,RegP_Z12,RegP_Z13",
 "Qfff",
 AP_HAL::micros64(),
 (double)PosContIBSMC_B.Z1_1,
 (double)PosContIBSMC_B.Z1_2,
 (double)PosContIBSMC_B.Z1_3);
 AP::logger().Write("SMP2", "TimeUS,RegP_B1,RegP_B2,RegP_B3",
 "Qfff",
 AP_HAL::micros64(),
 (double)PosContIBSMC_B.SOMA_BU_1,
 (double)PosContIBSMC_B.SOMA_BU_2,
 (double)PosContIBSMC_B.SOMA_BU_3);
 AP::logger().Write("SMP3", "TimeUS,RegP_S1,RegP_S2,RegP_S3",
 "Qfff",
 AP_HAL::micros64(),
 (double)PosContIBSMC_B.SMCpos_1,
 (double)PosContIBSMC_B.SMCpos_2,
 (double)PosContIBSMC_B.SMCpos_3);
 AP::logger().Write("SMP4", "TimeUS,RegP_Fz,RegP_PHI,RegP_THETA",
 "Qfff",
 AP_HAL::micros64(),
 (double)PosContIBSMC_B.Fz,
 (double)PosContIBSMC_B.phi_d,
 (double)PosContIBSMC_B.theta_d);
}
```

Função *step* do controle de atitude, AttContIBSMC.cpp

```
void  AttContIBSMCModelClass::step(real32_T arg_Pn2_[3], real32_T
    arg_Pn2d_[3], real32_T arg_dPn2[3], real32_T arg_OutputATT[3])
{
 real32_T SMC1_att_tmpZ1;
 real32_T SMC1_att_tmpBU1;
 real32_T SMC1_att_tmpZ3;
 real32_T SMC2_att_tmpZ2;
```

```
AttContIBSMC_B.SomaEA1 = arg_Pn2_[0] - arg_Pn2d_[0];
//***** DiscreteIntegrator: 'DTIntEA1'*************//
AttContIBSMC_B.DTIntEA1 = AttContIBSMC_DW.DTIntEA1_DSTATE;
AttContIBSMC_DW.DTIntEA1_DSTATE += AttContIBSMC_P.
   DTIntEA1_gainval * AttContIBSMC_B.SomaEA1;
//***** DiscreteIntegrator: 'FIM'******************//
AttContIBSMC_B.RAT_GAMA_A1_IntEA1 = AttContIBSMC_P.
   RAT_GAMA_A1_IntEA1_Gain * AttContIBSMC_B.DTIntEA1;
AttContIBSMC_B.SomaZ1_1 = AttContIBSMC_B.SomaEA1 +
   AttContIBSMC_B.RAT_GAMA_A1_IntEA1;
//***** DiscreteDerivative: 'dPn2d[0]'**************//
AttContIBSMC_B.Diff = (arg_Pn2d_[0] - AttContIBSMC_DW.UD_DSTATE)
    / (1.0f / AttContIBSMC_P.TSamp_WtEt);
AttContIBSMC_DW.UD_DSTATE = arg_Pn2d_[0];
//***** DiscreteDerivative: 'FIM' ******************//
AttContIBSMC_B.SomaderivEA1 = arg_dPn2[0] - AttContIBSMC_B.Diff;
AttContIBSMC_B.RAT_KA_1_Z1_1 = AttContIBSMC_P.RAT_KA_1_Z1_1_Gain
    * AttContIBSMC_B.SomaZ1_1;
AttContIBSMC_B.RAT_GAMA_A1_EA1 = AttContIBSMC_P.
   RAT_GAMA_A1_EA1_Gain * AttContIBSMC_B.SomaEA1;
AttContIBSMC_B.SomaZ2_1 = (AttContIBSMC_B.RAT_KA_1_Z1_1 +
   AttContIBSMC_B.RAT_GAMA_A1_EA1) + AttContIBSMC_B.SomaderivEA1;
SMC1_att_tmpZ1 = (AttContIBSMC_P.LAMBDA) * ((AttContIBSMC_P.
   LAMBDA) * AttContIBSMC_B.SomaZ2_1);
AttContIBSMC_B.SMC1_att = -SMC1_att_tmpZ1 / ((AttContIBSMC_P.
   LAMBDA * AttContIBSMC_P.EPISON1_ATT) + std::abs(SMC1_att_tmpZ1
   ));
AttContIBSMC_B.dotDeltaA1 = (AttContIBSMC_P.RAT_GGAMA_A1) *
   AttContIBSMC_B.SomaZ2_1;
//***** DiscreteIntegrator: 'DTIntDeltaA1'************//
AttContIBSMC_B.DTIntDeltaA1 = AttContIBSMC_DW.
   DTIntDeltaA1_DSTATE;
AttContIBSMC_DW.DTIntDeltaA1_DSTATE += AttContIBSMC_P.
   DTIntDeltaA1_gainval * AttContIBSMC_B.dotDeltaA1;
//***** DiscreteIntegrator: 'FIM' ******************//
//***** DiscreteDerivative: 'Z1_1'******************//
AttContIBSMC_B.Diff_h = (AttContIBSMC_B.SomaZ1_1 -
   AttContIBSMC_DW.UD_DSTATE_m) / (1.0f / AttContIBSMC_P.
   TSamp_WtEt_o);
AttContIBSMC_DW.UD_DSTATE_m = AttContIBSMC_B.SomaZ1_1;
//***** DiscreteDerivative: 'FIM' ******************//
```

```
SMC1_att_tmpBU1 = (2.0 * AttContIBSMC_P.LENGTH_ARM *
   AttContIBSMC_P.FORCE_MAX);
AttContIBSMC_B.BU_1 = (-AttContIBSMC_B.SMC1_att - AttContIBSMC_B
   .DTIntDeltaA1 -
(AttContIBSMC_P.BU_1_RAT_KA_1 * AttContIBSMC_B.Diff_h) +
AttContIBSMC_B.Diff - (AttContIBSMC_P.BU_1_RAT_GAMA_A1 *
   AttContIBSMC_B.SomaderivEA1) -
AttContIBSMC_B.SomaZ1_1) / SMC1_att_tmpBU1;
AttContIBSMC_B.SomaEA2 = arg_Pn2_[1] - arg_Pn2d_[1];
//***** DiscreteIntegrator: 'DTIntEA2'****************//
AttContIBSMC_B.DTIntEA2 = AttContIBSMC_DW.DTIntEA2_DSTATE;
AttContIBSMC_DW.DTIntEA2_DSTATE += AttContIBSMC_P.
   DTIntEA2_gainval * AttContIBSMC_B.SomaEA2;
//***** DiscreteIntegrator: 'FIM' *********************//
AttContIBSMC_B.RAT_GAMA_A2_intEA2 = AttContIBSMC_P.
   RAT_GAMA_A2_intEA2_Gain * AttContIBSMC_B.DTIntEA2;
AttContIBSMC_B.SomaZ1_2 = AttContIBSMC_B.SomaEA2 +
   AttContIBSMC_B.RAT_GAMA_A2_intEA2;
//***** DiscreteDerivative: 'dPn2d[1]'******************//
AttContIBSMC_B.Diff_n = (arg_Pn2d_[1] - AttContIBSMC_DW.
   UD_DSTATE_m5) / (1.0f / AttContIBSMC_P.TSamp_WtEt_m);
AttContIBSMC_DW.UD_DSTATE_m5 = arg_Pn2d_[1];
//**** DiscreteDerivative: 'FIM' **********************//
AttContIBSMC_B.SomaderivEA2 = arg_dPn2[1] - AttContIBSMC_B.
   Diff_n;
AttContIBSMC_B.RAT_KA_2_Z1_2 = AttContIBSMC_P.RAT_KA_2_Z1_2_Gain
    * AttContIBSMC_B.SomaZ1_2;
AttContIBSMC_B.RAT_GAMA_A2_EA2 = AttContIBSMC_P.
   RAT_GAMA_A2_EA2_Gain * AttContIBSMC_B.SomaEA2;
AttContIBSMC_B.SomaZ2_2 = (AttContIBSMC_B.RAT_KA_2_Z1_2 +
   AttContIBSMC_B.SomaderivEA2) + AttContIBSMC_B.RAT_GAMA_A2_EA2;
SMC2_att_tmpZ2 = (AttContIBSMC_P.LAMBDA) * ((AttContIBSMC_P.
   LAMBDA) * AttContIBSMC_B.SomaZ2_2);
AttContIBSMC_B.SMC2_att = -SMC2_att_tmpZ2 / ((AttContIBSMC_P.
   LAMBDA * AttContIBSMC_P.EPISON2_ATT) + std::abs(SMC2_att_tmpZ2
   ));
AttContIBSMC_B.dotDeltaA2 = (AttContIBSMC_P.RAT_GGAMA_A2) *
   AttContIBSMC_B.SomaZ2_2;
//***** DiscreteIntegrator: 'DTIntDeltaA2'****************//
AttContIBSMC_B.DTIntDeltaA2 = AttContIBSMC_DW.
   DTIntDeltaA2_DSTATE;
```

```
AttContIBSMC_DW.DTIntDeltaA2_DSTATE += AttContIBSMC_P.
   DTIntDeltaA2_gainval * AttContIBSMC_B.dotDeltaA2;
//***** DiscreteIntegrator: 'FIM' *************//
//***** DiscreteDerivative: 'Z1_2'*************//
AttContIBSMC_B.Diff_hu = (AttContIBSMC_B.SomaZ1_2 -
   AttContIBSMC_DW.UD_DSTATE_mz) / (1.0f / AttContIBSMC_P.
   TSamp_WtEt_c);
AttContIBSMC_DW.UD_DSTATE_mz = AttContIBSMC_B.SomaZ1_2;
//***** DiscreteDerivative: 'FIM' *************//
AttContIBSMC_B.BU_2 = (-AttContIBSMC_B.SMC2_att - AttContIBSMC_B
   .DTIntDeltaA2 -
(AttContIBSMC_P.BU_2_RAT_KA_2 * AttContIBSMC_B.Diff_hu) +
AttContIBSMC_B.Diff_n -
(AttContIBSMC_P.BU_2_RAT_GAMA_A2 * AttContIBSMC_B.SomaderivEA2)
   -
AttContIBSMC_B.SomaZ1_2) / SMC1_att_tmpBU1;
AttContIBSMC_B.SomaEA3 = (arg_Pn2_[2] - arg_Pn2d_[2]);
//***** DiscreteIntegrator: 'DTIntEA3'************//
AttContIBSMC_B.DTIntEA3 = AttContIBSMC_DW.DTIntEA3_DSTATE;
AttContIBSMC_DW.DTIntEA3_DSTATE += AttContIBSMC_P.
   DTIntEA3_gainval * AttContIBSMC_B.SomaEA3;
//***** DiscreteIntegrator: 'FIM'****************//
AttContIBSMC_B.RAT_GAMA_A3_intEA3 = AttContIBSMC_P.
   RAT_GAMA_A3_intEA3_Gain * AttContIBSMC_B.DTIntEA3;
AttContIBSMC_B.SomaZ1_3 = AttContIBSMC_B.SomaEA3 +
   AttContIBSMC_B.RAT_GAMA_A3_intEA3;
//***** DiscreteDerivative: 'dPn2d[2]'************//
AttContIBSMC_B.Diff_j = (arg_Pn2d_[2] - AttContIBSMC_DW.
   UD_DSTATE_h) / (1.0f / AttContIBSMC_P.TSamp_WtEt_d);
AttContIBSMC_DW.UD_DSTATE_h = arg_Pn2d_[2];
//***** DiscreteDerivative: 'dPn2d[2]'*************//
AttContIBSMC_B.SomaderivEA3 = arg_dPn2[2] - AttContIBSMC_B.
   Diff_j;
AttContIBSMC_B.RAT_KA_3_Z1_3 = AttContIBSMC_P.RAT_KA_3_Z1_3_Gain
    * AttContIBSMC_B.SomaZ1_3;
AttContIBSMC_B.RAT_GAMA_A3_EA3 = AttContIBSMC_P.
   RAT_GAMA_A3_EA3_Gain * AttContIBSMC_B.SomaEA3;
AttContIBSMC_B.SomaZ2_3 = (AttContIBSMC_B.RAT_KA_3_Z1_3 +
   AttContIBSMC_B.SomaderivEA3) + AttContIBSMC_B.RAT_GAMA_A3_EA3;
SMC1_att_tmpZ3 = (AttContIBSMC_P.LAMBDA) * ((AttContIBSMC_P.
   LAMBDA) * AttContIBSMC_B.SomaZ2_3);
```

```
 AttContIBSMC_B.SMC3_att = -SMC1_att_tmpZ3 / ((AttContIBSMC_P.
    LAMBDA * AttContIBSMC_P.EPISON3_ATT) + std::abs(SMC1_att_tmpZ3
    ));
 AttContIBSMC_B.dotDeltaA3 = (AttContIBSMC_P.RAT_GGAMA_A3) *
    AttContIBSMC_B.SomaZ2_3;
 //***** DiscreteIntegrator: 'DTIntDelta3' ************//
 AttContIBSMC_B.DTIntDelta3 = AttContIBSMC_DW.DTIntDelta3_DSTATE;
 AttContIBSMC_DW.DTIntDelta3_DSTATE += AttContIBSMC_P.
    DTIntDelta3_gainval * AttContIBSMC_B.dotDeltaA3;
 //***** DiscreteIntegrator: 'DTIntDelta3' *************//
 //***** DiscreteIntegrator: 'Z1_2'***************//
 AttContIBSMC_B.Diff_c = (AttContIBSMC_B.SomaZ1_3 -
    AttContIBSMC_DW.UD_DSTATE_e) / (1.0f / AttContIBSMC_P.
    TSamp_WtEt_n);
 AttContIBSMC_DW.UD_DSTATE_e = AttContIBSMC_B.SomaZ1_3;
 //***** DiscreteIntegrator: 'Z1_2'*****************//
 AttContIBSMC_B.BU_3 = (-AttContIBSMC_B.SMC3_att - AttContIBSMC_B
    .DTIntDelta3 - (AttContIBSMC_P.BU_3_RAT_KA_3 * AttContIBSMC_B.
    Diff_c) + AttContIBSMC_B.Diff_j - (AttContIBSMC_P.
    BU_3_RAT_GAMA_A3 * AttContIBSMC_B.SomaderivEA3) -
    AttContIBSMC_B.SomaZ1_3) / (2.0 * AttContIBSMC_P.MOMENT_MAX);
 arg_OutputATT[0] = (AttContIBSMC_B.BU_1);
 arg_OutputATT[1] = (AttContIBSMC_B.BU_2);
 arg_OutputATT[2] = (AttContIBSMC_B.BU_3);
}
```

APÊNDICE G – Códigos desenvolvido em Python

```
from pymavlink import mavutil
from dronekit import connect, VehicleMode, LocationGlobalRelative
import time
from math import sin, cos
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import os
import sys
from pathlib import Path
"""
# ----------------------------------------
# PARAMETROS DO CONTROLE IBSMC DE POSICAO
# ----------------------------------------
"""
m = 1.47
RPO_DELTA = 10.0
RPO_KP_1 = 9.11
RPO_KP_2 = 35.8922
RPO_KP_3 = 75.7405
# ----------------------------------------
RPO_GAMA_P1 = 0.43
RPO_GAMA_P2 = 5.9882
RPO_GAMA_P3 = 27.1481
# ----------------------------------------
RPO_GGAMA_P1 = 0.001
RPO_GGAMA_P2 = 0.001
RPO_GGAMA_P3 = 0.001
"""
# ----------------------------------------
# PARAMETROS DO CONTROLE IBSMC DE ATTITUDE
# ----------------------------------------
"""
RAT_DELTA = 10.0
RAT_KA_1 = 20.1771
```

```
RAT_KA_2 = 25.3564
RAT_KA_3 = 52.4907
# ----------------------------------------
RAT_GAMA_A1 = 7.3988
RAT_GAMA_A2 = 7.7288
RAT_GAMA_A3 = 4.1983
# ----------------------------------------
RAT_GGAMA_A1 = 0.001
RAT_GGAMA_A2 = 0.001
RAT_GGAMA_A3 = 0.001
# ----------------------------------------
MASSA_TOTAL = 1.477
LENGTH_ARM = 0.275
FORCE_MAX = 12.095
MOMENT_MAX = 1.025
# ----------------------------------------
PASSO_INT = 1/400
LAMBDA = (1/PASSO_INT)/5
# ----------------------------------------
EPISON1_POS = 0.01
EPISON2_POS = 0.01
EPISON3_POS = 0.01
# ----------------------------------------
EPISON1_ATT = 1.7453e-4     #0.01deg
EPISON2_ATT = 1.7453e-4
EPISON3_ATT = 1.7453e-4
"""
# ------------------------------------------------------------
# Funcao para criar e enviar uma mensagem LOCAL_POSITION_NED
# ------------------------------------------------------------
"""
def Send_Param_Valor(vehicle,LAMBDA,PASSO_INT,RPO_KP,RPO_GAMA_P,
   RPO_GGAMA_P,EPISON_POS,RPO_DELTA,RAT_KA,RAT_GAMA_A,RAT_GGAMA_A,
   EPISON_ATT,RAT_DELTA):
try:
vehicle.parameters["LAMBDA"] = LAMBDA
vehicle.parameters["PASSO_INT"] = PASSO_INT
# ------------------------------------------------------------
# ------------------------------------------------------------
vehicle.parameters["RPO_KP_1"] = RPO_KP[0]
vehicle.parameters["RPO_KP_2"] = RPO_KP[1]
vehicle.parameters["RPO_KP_3"] = RPO_KP[2]
```

```
# ------------------------------------------------------------
vehicle.parameters["RPO_GAMA_P1"] = RPO_GAMA_P[0]
vehicle.parameters["RPO_GAMA_P2"] = RPO_GAMA_P[1]
vehicle.parameters["RPO_GAMA_P3"] = RPO_GAMA_P[2]
# ------------------------------------------------------------
vehicle.parameters["RPO_GGAMA_P1"] = RPO_GGAMA_P[0]
vehicle.parameters["RPO_GGAMA_P2"] = RPO_GGAMA_P[1]
vehicle.parameters["RPO_GGAMA_P3"] = RPO_GGAMA_P[2]
# ------------------------------------------------------------
vehicle.parameters["EPISON1_POS"] = EPISON_POS[0]
vehicle.parameters["EPISON2_POS"] = EPISON_POS[1]
vehicle.parameters["EPISON3_POS"] = EPISON_POS[2]
# ------------------------------------------------------------
# ------------------------------------------------------------
vehicle.parameters["RAT_KA_1"] = RAT_KA[0]
vehicle.parameters["RAT_KA_2"] = RAT_KA[1]
vehicle.parameters["RAT_KA_3"] = RAT_KA[2]
# ------------------------------------------------------------
vehicle.parameters["RAT_GAMA_A1"] = RAT_GAMA_A[0]
vehicle.parameters["RAT_GAMA_A2"] = RAT_GAMA_A[1]
vehicle.parameters["RAT_GAMA_A3"] = RAT_GAMA_A[2]
# ------------------------------------------------------------
vehicle.parameters["RAT_GGAMA_A1"] = RAT_GGAMA_A[0]
vehicle.parameters["RAT_GGAMA_A2"] = RAT_GGAMA_A[1]
vehicle.parameters["RAT_GGAMA_A3"] = RAT_GGAMA_A[2]
# ------------------------------------------------------------
vehicle.parameters["EPISON1_ATT"] = EPISON_ATT[0]
vehicle.parameters["EPISON2_ATT"] = EPISON_ATT[1]
vehicle.parameters["EPISON3_ATT"] = EPISON_ATT[2]
# ------------------------------------------------------------
# ------------------------------------------------------------
vehicle.parameters["RPO_DELTA"] = RPO_DELTA
vehicle.parameters["RAT_DELTA"] = RAT_DELTA
except Exception as e:
print('Erro ao enviar comando:', str(e))
"""
# ------------------------------------------------------------
Funcao para leitura dos parametros do controlador.
pos:posicao att:attitude
# ------------------------------------------------------------
"""
def Read_IBSMC_Param(vehicle,posORatt):
```

```
try:
if posORatt == "pos":
print('\n**********\nPARAMETROS IBSMC
   POSICAO\n**********\n')
print('PASSO DE INTEGRACAO CONTROLE=%f, LAMBDA=%f'%(vehicle.
   parameters['PASSO_INT'],vehicle.parameters['LAMBDA']))
print('RPO_KP_1=%f, RPO_KP_2=%f, RPO_KP_3=%f\n' %(vehicle.
   parameters['RPO_KP_1'], vehicle.parameters['RPO_KP_2'], vehicle
   .parameters['RPO_KP_3']))
print('RPO_GAMA_P1=%f, RPO_GAMA_P2=%f, RPO_GAMA_P3=%f\n' %(
   vehicle.parameters['RPO_GAMA_P1'], vehicle.parameters['
   RPO_GAMA_P2'], vehicle.parameters['RPO_GAMA_P3']))
print('RPO_GGAMA_P1=%f, RPO_GGAMA_P2=%f, RPO_GGAMA_P3=%f\n' %(
   vehicle.parameters['RPO_GGAMA_P1'], vehicle.parameters['
   RPO_GGAMA_P2'], vehicle.parameters['RPO_GGAMA_P3']))
print('EPISON1_POS=%f, EPISON2_POS=%f, EPISON3_POS=%f\n' %(
   vehicle.parameters['EPISON1_POS'], vehicle.parameters['
   EPISON2_POS'], vehicle.parameters['EPISON3_POS']))
if posORatt == "att":
print('\n**********\nPARAMETROS IBSMC
   ATTITUDE\n**********\n')
print('PASSO DE INTEGRACAO CONTROLE=%f, LAMBDA=%f'%(vehicle.
   parameters['PASSO_INT'],vehicle.parameters['LAMBDA']))
print('RAT_KA_1=%f, RAT_KA_2=%f, RAT_KA_3=%f\n' %(vehicle.
   parameters['RAT_KA_1'], vehicle.parameters['RAT_KA_2'], vehicle
   .parameters['RAT_KA_3']))
print('RAT_GAMA_A1=%f, RAT_GAMA_A2=%f, RAT_GAMA_A3=%f\n' %(
   vehicle.parameters['RAT_GAMA_A1'], vehicle.parameters['
   RAT_GAMA_A2'], vehicle.parameters['RAT_GAMA_A3']))
print('RAT_GGAMA_A1=%f, RAT_GGAMA_A2=%f, RAT_GGAMA_A3=%f\n' %(
   vehicle.parameters['RAT_GGAMA_A1'], vehicle.parameters['
   RAT_GGAMA_A2'], vehicle.parameters['RAT_GGAMA_A3']))
print('EPISON1_ATT=%f, EPISON2_ATT=%f, EPISON3_ATT=%f\n' %(
   vehicle.parameters['EPISON1_ATT'], vehicle.parameters['
   EPISON2_ATT'], vehicle.parameters['EPISON3_ATT']))
except Exception as error:
print(error)
sys.exit(0)
"""
# ------------------------------------------------------------
# Funcao que constroi a trajetoria desejada:
#  O Z tem que ser positivo pois ira ser multiplicado por -100.0f
```

```
# ------------------------------------------------------------
"""
def TrajHelicoidalTempoIBSMCFrameCoord7(vehicle,xStartTrack,
    yStartTrack, zStartTrack,tempo):
Ax = 5.0
Ay = 5.0
Sz = 0.05
fx = 0.1
fy = 0.1
current_x = vehicle.location.local_frame.north
current_y = vehicle.location.local_frame.east
current_z = vehicle.location.local_frame.down
current_v = vehicle.velocity
# Create the desired trajectory in Z and XY
target_position_xyz = np.array([-(xStartTrack-(Ax*sin(fx*tempo)))
    +current_x, -(Ay+yStartTrack-(Ay*cos(fy*tempo)))+current_y, (
    zStartTrack+(Sz*tempo))+current_z])
# Creator the desired velocity in Z and XY
target_velocity_xyz = np.array([-(-Ax*fx*cos(fx*tempo))+current_v
    [0], -(Ay*fy*sin(fy*tempo))+current_v[1], Sz+current_v[2]])
# Creator the desired velocity in Z and XY
return (target_position_xyz[0], target_position_xyz[1],
    target_position_xyz[2], target_velocity_xyz[0],
    target_velocity_xyz[1], target_velocity_xyz[2])
"""
# ------------------------------------------------------------
# Funcao que constroi a trajetoria desejada
# ------------------------------------------------------------
"""
def TrajHelicoidalTempoIBSMCFrameCoord1(xStartTrack, yStartTrack,
     zStartTrack,tempo):
Ax = 5.0
Ay = 5.0
Sz = 0.05
fx = 0.01
fy = 0.01
# Create the desired trajectory in Z and XY
target_position_xyz = np.array([xStartTrack+(Ax*sin(fx*tempo)),
    yStartTrack+((Ay*cos(fy*tempo))-Ay), -(zStartTrack-(Sz*tempo))
    ])
# Creator the desired velocity in Z and XY
```

```
target_velocity_xyz = np.array([Ax*fx*cos(fx*tempo), -Ay*fy*sin(
   fy*tempo), Sz])
return (target_position_xyz[0], target_position_xyz[1],
   target_position_xyz[2], target_velocity_xyz[0],
   target_velocity_xyz[1], target_velocity_xyz[2])
"""
# ------------------------------------------------------------
# Funcao que constroi a trajetoria desejada
# ------------------------------------------------------------
"""
def TrajHelicoidal(inicio_simulacao):
fim_simulacao = time.time()
duration = fim_simulacao - inicio_simulacao
print("DURACAO: %f" %duration)
# Create the desired trajectory in Z and XY
target_position_xyz = np.array([-(1.0 * sin(0.1 * duration)), -
   (1.0 * cos(0.1 * duration)), -(0.05 * duration)])
# Creator the desired velocity in Z and XY
target_velocity_xyz = np.array([-0.1 * cos(0.1 * duration), 0.1 *
    sin(0.1 * duration), -0.05])
# Creator the desired velocity in Z and XY
return (target_position_xyz[0], target_position_xyz[1],
   target_position_xyz[2], target_velocity_xyz[0],
   target_velocity_xyz[1], target_velocity_xyz[2])
"""
# ------------------------------------------------------------
# Funcao para conectar ao veiculo
# ------------------------------------------------------------
"""
def connect_to_vehicle(connection_string):
try:
print('Conectando ao veiculo em: %s' % connection_string)
vehicle = connect(connection_string, baud=115200)
vehicle.wait_ready(True, raise_exception=False)
return vehicle
except Exception as e:
print('Erro ao conectar ao veiculo:', str(e))
return None
"""
# ------------------------------------------------------------
# Funcao para armar ou desarmar o veiculo
# ------------------------------------------------------------
```

```
"""
def arm_or_disarm(vehicle, armORdisarm):
if (armORdisarm == "arm"):
if vehicle.mode != VehicleMode("GUIDED"):
print("Flight Mode Incorreto:%s\n Mude para o GUIDED!"%vehicle.
    mode)
else:
print('Preparando para armar os motores...')
vehicle.armed = True
vehicle.flush()
while not vehicle.armed:
print("Esperando armar!")
time.sleep(0.1)
print('Iniciando a decolagem...')
if (armORdisarm == "disarm"):
print('Desarmando veiculo...')
vehicle.armed = False
vehicle.flush()
while vehicle.armed:
time.sleep(0.1)
print('veiculo desarmado com sucesso!')
"""
# ------------------------------------------------------------
# Funcao para decolar o veiculo ate uma altitude desejada
# ------------------------------------------------------------
"""
def takeoff(vehicle, target_altitude):
if (vehicle.mode != VehicleMode("GUIDED") and vehicle.armed ==
    False):
print("Flight Mode Incorreto:%s\n Mude para o GUIDED!\n"%vehicle.
    mode)
print("veiculo nao esta armado:%s\n ARME O veiculo!\n"%vehicle.
    armed)
else:
vehicle.simple_takeoff(target_altitude)
# ESPERANDO O VEICULO CHEGAR A ALTITUDE DESEJADA
while True:
print(" Altitude atual: ", vehicle.location.global_relative_frame
    .alt)
SalvaTrajVEI(vehicle)
# Break and return from function just below target altitude.
```

```
if vehicle.location.global_relative_frame.alt >= target_altitude
   * 0.99:
print("veiculo chegou na ALTITUDE desejada!")
SalvaTrajVEI(vehicle)
break
time.sleep(1)
"""
# -----------------------------------------------------------
# Funcao para criar e enviar uma mensagem LOCAL_POSITION_NED
# -----------------------------------------------------------
"""
def send_local_position_ned_messageFrameCoord7(vehicle, x, y, z,
   vx, vy, vz):
msg = vehicle.message_factory.
   set_position_target_local_ned_encode(
0,          # time_boot_ms (nao utilizado)
0, 0,       # target_system, target_component (nao utilizado)
7,          # coordinate_frame: link https://ardupilot.org/dev/
   docs/copter-commands-in-guided-mode.html
3520,       # tipo de orientacao (bitmask) USA-SE:3520->Pos,Vel
x, y, z,    # posicao (NED)
vx, vy, vz, # velocidade (NED)
0, 0, 0,    # aceleracao, forca
0, 0)       # yaw, yaw_rate (nao utilizado)
try:
vehicle.send_mavlink(msg)
vehicle.flush()
current_x = vehicle.location.local_frame.north
current_y = vehicle.location.local_frame.east
current_z = vehicle.location.local_frame.down
print('   Posicao Desejada:\n   xd=%f, yd=%f, zd=%f\n' %(x, y, z)
   )
print('   Posicao veiculo :\n   x=%f, y=%f, z=%f\n' %(current_x,
   current_y, current_z))
except Exception as e:
print('Erro ao enviar comando:', str(e))
"""
# -----------------------------------------------------------
# Funcao para criar e enviar uma mensagem LOCAL_POSITION_NED
# -----------------------------------------------------------
"""
```

```
def send_local_position_ned_messageFrameCoord1(vehicle, x, y, z,
    vx, vy, vz):
msg = vehicle.message_factory.
    set_position_target_local_ned_encode(
0,          # time_boot_ms (nao utilizado)
0, 0,       # target_system, target_component (nao utilizado)
1,          # coordinate_frame: link https://ardupilot.org/dev/
    docs/copter-commands-in-guided-mode.html
3520,       # tipo de orientacao (bitmask) USA-SE:3520->Pos,Vel
x, y, z,    # posicao (NED)
vx, vy, vz, # velocidade (NED)
0, 0, 0,    # aceleracao, forca
0, 0)       # yaw, yaw_rate (nao utilizado)
try:
vehicle.send_mavlink(msg)
vehicle.flush()
current_x = vehicle.location.local_frame.north
current_y = vehicle.location.local_frame.east
current_z = vehicle.location.local_frame.down
print('   Posicao Desejada:\n   xd=%f, yd=%f, zd=%f\n' %(x, y, z)
    )
print('   Posicao veiculo :\n   x=%f, y=%f, z=%f\n' %(current_x,
    current_y, current_z))
except Exception as e:
print('Erro ao enviar comando:', str(e))
"""
# ------------------------------------------------------------
# Endereco de conexao com o veiculo
# ------------------------------------------------------------
"""
connection_string = 'tcp:127.0.0.1:5763'
"""
# ------------------------------------------------------------
# Conectar ao veiculo
# ------------------------------------------------------------
"""
vehicle = connect_to_vehicle(connection_string)
"""
# ------------------------------------------------------------
# AJUSTE/VERIFICAcaO DOS PARAMETROS DO CONTROLADOR IBSMC POSIcaO
    E ATTITUDE
# ------------------------------------------------------------
```

```
"""
Read_IBSMC_Param(vehicle, "pos")
Read_IBSMC_Param(vehicle, "att")
# ------------------------------------------------------------
# PARAMETROS DO IBSMC
PASSO_INT  = PASSO_INT
LAMBDA = LAMBDA
# ------------------------------------------------------------
# PARAMETROS DO IBSMC POSIcaO
RPO_KP      = np.array([RPO_KP_1,RPO_KP_2,RPO_KP_3])
RPO_GAMA_P  = np.array([RPO_GAMA_P1,RPO_GAMA_P2,RPO_GAMA_P3])
RPO_GGAMA_P = np.array([RPO_GGAMA_P1,RPO_GGAMA_P2,RPO_GGAMA_P3])
EPISON_POS  = np.array([EPISON1_POS,EPISON2_POS,EPISON3_POS])
# ------------------------------------------------------------
# PARAMETROS DO IBSMC ATTITUDE
RAT_KA      = np.array([RAT_KA_1,RAT_KA_2,RAT_KA_3])
RAT_GAMA_A  = np.array([RAT_GAMA_A1,RAT_GAMA_A2,RAT_GAMA_A3])
RAT_GGAMA_A = np.array([RAT_GGAMA_A1,RAT_GGAMA_A2,RAT_GGAMA_A3])
EPISON_ATT  = np.array([EPISON1_ATT,EPISON2_ATT,EPISON3_ATT])
# ------------------------------------------------------------
Send_Param_Valor(vehicle,LAMBDA,PASSO_INT,RPO_KP,RPO_GAMA_P,
   RPO_GGAMA_P,EPISON_POS,RPO_DELTA,RAT_KA,RAT_GAMA_A,RAT_GGAMA_A,
   EPISON_ATT,RAT_DELTA)
"""
# ------------------------------------------------------------
# ------------------------------------------------------------
"""
if vehicle is not None:
inicio_simulacao = time.time()
# ------------------------------------------------------------
# MUDAR PARA O FLIGHT MODE GUIDED
vehicle.mode = VehicleMode("GUIDED")
while True:
if vehicle.mode == VehicleMode("GUIDED"):
break
# ARMAR OS MOTORES
arm_or_disarm(vehicle, "arm")
# ------------------------------------------------------------
# DEFININDO A ALTITUDE DESEJADA EM METROS
target_altitude = 10
# COMEcA A DECOLAGEM
```

```
print("\nINICIANDO OS PROCEDIMENTOS DA DECOLAGEM!\nAltura
   desejada:%f\n" %target_altitude)
takeoff(vehicle, target_altitude)
# ------------------------------------------------------
# COLETANDO PONTO INICIAL DO TRACKING
xStartTrack = vehicle.location.local_frame.north
yStartTrack = vehicle.location.local_frame.east
zStartTrack = vehicle.location.local_frame.down
print('POSICAO INICIAL:\n   xi=%f, yi=%f, zi=%f\n' %(xStartTrack,
    yStartTrack, zStartTrack))
# ------------------------------------------------------
# MUDAR PARA O FLIGHT MODE ELICOID
vehicle.mode = VehicleMode("ELICOID")
# ------------------------------------------------------
tempo = 0
tempo_simulacao = 1e4*PASSO_INT
while tempo_simulacao >= tempo:
print("DURAcaO: %f" %tempo)
# ENVIANDO A MENSSAGEM PARA O veiculo
# target = TrajHelicoidalTempoIBSMCFrameCoord7(vehicle,
   xStartTrack, yStartTrack, zStartTrack, tempo)
# send_local_position_ned_messageFrameCoord7(vehicle, target[0],
   target[1], target[2], target[3], target[4], target[5])
# ENVIANDO A MENSSAGEM PARA O veiculo
target = TrajHelicoidalTempoIBSMCFrameCoord1(xStartTrack,
   yStartTrack, zStartTrack,tempo)
send_local_position_ned_messageFrameCoord1(vehicle, target[0],
   target[1], target[2], target[3], target[4], target[5])
SalvaTrajVEI(vehicle)
SalvaTrajDES(vehicle,target)
time.sleep(PASSO_INT)
tempo += PASSO_INT
# ------------------------------------------------------
# COLETANDO PONTO FINAL DO TRACKING
xEndTrack = vehicle.location.local_frame.north
yEndTrack = vehicle.location.local_frame.east
zEndTrack = vehicle.location.local_frame.down
"""
# ------------------------------------------------------
# ------------------------------------------------------
"""
# MUDANDO O FLIGHT MODE
```

```
vehicle.mode = VehicleMode("LAND")
# ESPERANDO O VEICULO CHEGAR A ALTITUDE DESEJADA
while True:
print("POUSANDO, Altitude=%f" %vehicle.location.
   global_relative_frame.alt)
# Break and return from function just below target altitude.
if vehicle.location.global_relative_frame.alt <= 0.03:
print("veiculo chegou ao solo!")
break
time.sleep(1)
# -----------------------------------------------------------
# MUDANDO O FLIGHT MODE
vehicle.mode = VehicleMode("STABILIZE")
time.sleep(5)
# Desarmamento do veiculo
arm_or_disarm(vehicle, False)
# Encerrar conexao com o veiculo
vehicle.close()
```