DESENVOLVIMENTO DE SOFTWARE PARA CÁLCULO DE DESEMPENHO DE AERONAVES VOLTADO AO AERODESIGN[1]

DEVELOPMENT OF AN AIRCRAFT PERFORMANCE CALCULATION SOFTWARE FOR AERODESIGN

Luiz Felipe Camargo Souza[2]

RESUMO

No aerodesign, a análise de desempenho da aeronave é uma etapa essencial da metodologia de projeto, pois define parâmetros que afetam diretamente características físicas da aeronave e implicam nas análises de estabilidade, cargas e estruturas, além de auxiliarem na tomada de decisões sobre mudanças no projeto. No contexto da Equipe Nisus Aerodesign, tal etapa é realizada por meio da implementação de métodos numéricos na linguagem de programação Python e é necessário garantir a atualização das entradas utilizadas e fornecimento adequado de informações aos demais setores, conforme o andamento do projeto. O objetivo deste trabalho é propor um software que implemente os métodos de cálculo do setor de desempenho da Equipe Nisus Aerodesign, para garantir a utilização dos parâmetros mais recentes, obtenção e transmissão de dados apropriada, e simplificação do esquema de trabalho utilizando programação orientada a objetos e padrões de projeto da linguagem Python.

Palavras-chave: Software, aerodesign, desempenho, padrões de projeto.

1 INTRODUÇÃO

Um dos desafios enfrentados durante o projeto de aeronaves é a definição de seu comportamento em voo, dadas as interações das forças que nela atuam. Segundo Yechout (2003), a sustentação, o arrasto, o peso e a tração são as quatro forças primárias que atuam durante o voo e a complexidade do processo de cálculo dos parâmetros de desempenho reside na determinação da forma como interagem e afetam o movimento.

Para determinar esses parâmetros, faz-se a utilização de métodos analíticos e numéricos envolvendo as equações do movimento, cuja determinação adequada só pode ser alcançada a partir do entendimento das forças atuantes durante o voo (Yechout, 2003). Torna- se coerente, neste escopo, a implementação destes métodos em algoritmos utilizando uma linguagem de programação, a fim de otimizar o processo de cálculo.

No contexto da Equipe Nisus Aerodesign, do Centro Tecnológico de Joinville (CTJ),da Universidade Federal de Santa Catarina (UFSC), a implementação dos métodos de cálculo dos parâmetros de desempenho é feita em Python 3.12 (Python Software Foundation, 2024)., já que é uma linguagem de sintaxe simples e de fácil utilização.

Um problema enfrentado pelo setor de desempenho da equipe, contudo, apresenta-se na organização dos algoritmos de cálculo dos parâmetros do setor. Devido às mudanças no projeto da aeronave durante cada etapa ao longo do ano, alterações nos códigos precisam ser feitas e novos membros precisam utilizar e entender o funcionamento das versões mais recentes rapidamente, exigindo que haja uma organização clara das versões existentes e uma boa

[1] Trabalho de Conclusão de Curso apresentado como requisito parcial para obtenção do grau de bacharel no Curso de Bacharelado em Ciência e Tecnologia, Centro Tecnológico de Joinville (CTJ), Universidade Federal de Santa Catarina (UFSC), sob orientação da Dra. Tatiana Renata Garcia.

[2] Graduando do Bacharelado em Ciência e Tecnologia – E-mail: lufelcamargo@gmail.com

documentação dos códigos.

O objetivo deste trabalho foi a implementação de um software que reúna os algoritmos necessários para o cálculo dos parâmetros de desempenho e relacione suas execuções de maneira adequada, promovendo a interação com o usuário por meio de uma interface gráfica do utilizador (GUI/UI), encapsulando os processos necessários para o funcionamento do conjunto total do software e simplificando o processo de análise.

Buscou-se uma interação simples com o usuário, bastando apenas obter as entradas necessárias, que poderão ser alteradas sem a necessidade de interagir diretamente com o código, e os resultados serão obtidos automaticamente. Tal facilidade foi atingida por meio da implementação utilizando programação orientada a objetos (POO) e seus padrões de projeto, que permitem a alteração dos dados de maneira simplificada, já que esses levam o software a uma arquitetura de baixa complexidade (Gamma *et al.*, 2009).

2 DESENVOLVIMENTO

Guedes (2009) divide o Processo Unificado (PU) de desenvolvimento de software em quatro partes principais: a concepção, a elaboração, a construção e transição. Na concepção, são exploradas características que versam sobre a viabilidade do sistema, sendo uma delas o caso de negócio do projeto, no qual se avalia a sua finalidade e a ordem de grandeza que pode alcançar (Larman, 2007). Na elaboração, são feitas análises de requisitos e o projeto do software, na construção ocorre a implementação e teste e a transição engloba a implantação do sistema.

A finalidade do sistema desenvolvido neste trabalho é ser uma ferramenta capaz de executar todos os processos de cálculo e análise do setor de desempenho da Equipe Nisus Aerodesign. A partir dessa definição, listou-se os requisitos funcionais (RF), responsáveis por descrever as funcionalidades do sistema, requisitos não funcionais (RNF), que determinam as condições e restrições do anterior e regras de negócio (RN), que devem ser seguidas ao executar uma funcionalidade (Guedes, 2009).

Pressman (2002, apud Pereira, 2011) destaca que um software de qualidade tem requisitos funcionais estabelecidos de forma clara, além de possuir padrões de desenvolvimento bem documentados. Isto é, as fases de concepção e elaboração devem ser bem trabalhadas para que o produto atenda às necessidades definidas. Além disso, as fases de elaboração e construção ocorrem iterativamente, ou seja, os requisitos implementados devem ser testados para que o funcionamento seja garantido (Guedes, 2009).

2.1 LISTAGEM DE REQUISITOS

Após a definição da finalidade do projeto, foi feita a listagem dos RFs, RNFs e RNs, seguindo o PU. Os Quadros 1, 2 e 3 mostram os requisitos determinados na fase de elaboração do sistema.

Quadro 1 – Requisitos funcionais do sistema.

(continua)

Requisito	Descrição
RF1	O programa deve realizar regressão polinomial
RF2	O programa deve exibir gráficos
RF3	O programa deve salvar as características de uma aeronave em um arquivo de texto
RF4	O programa deve fazer a leitura de arquivos de texto
RF5	O programa deve exibir erros

(conclusão)

Requisito	Descrição
RF6	Antes da leitura de arquivos de texto, o programa deve especificar o formato do arquivo
RF7	O programa deve permitir a alteração das entradas iniciais
RF8	O programa deve ser capaz de comparar diferentes configurações de aeronave
RF9	O programa deve possuir diferentes abas para cada etapa de análise
RF10	Caso um recurso precise que uma rotina seja executada anteriormente a seu uso, este deve avisar o usuário da necessidade

Fonte: Autor (2024).

Quadro 2 – Requisitos não funcionais do sistema.

Requisito	Descrição
RNF1	Na leitura de arquivos de texto, o programa deve permitir ao usuário alterar os parâmetros utilizados
RNF2	O programa deve possuir uma paleta de cores escuras
RNF3	O programa deve funcionar nos sistemas operacionais Windows 10/11 e Ubuntu 20.04 ou superior
RNF4	O programa deve possuir uma aba de tutorial no primeiro acesso
RNF5	O programa deve exibir erros

Fonte: Autor (2024).

Quadro 3 – Regras de negócio do sistema.

Requisito	Descrição
RN1	As entradas iniciais devem ser obtidas através de uma janela de pop-up
RN2	As ferramentas do software devem ser acessadas através de um menu na parte superior esquerda
RN3	Mensagens de erro devem ser exibidas na mesma janela onde o usuário está interagindo
RN4	As características que podem ser comparadas são relacionadas às fases de voo e MTOW da aeronave.
RN5	O cálculo do MTOW deve permitir a alteração dos valores de busca do método da bisseção utilizado para obtenção do dado

Fonte: Autor (2024).

Seguindo os conceitos do PU, buscou-se realizar a implementação de tais requisitos por meio da programação orientada a objetos, dispondo-se de padrões de projeto e ferramentas da Linguagem de Modelagem Unificada (UML), utilizada para modelar sistemas de forma precisa, completa, concisa e sem ambiguidades (Pereira, 2011).

2.2 PADRÕES DE PROJETO

Segundo Kristensen e Osterbye (1994), a programação conceitual é a modelagem dos processos de um sistema feita a partir de abstrações e fenômenos expressos em uma linguagem de programação que os suporte. Tal suporte a abstrações é chamado de orientação a objetos, na qual se utiliza de recursos de uma linguagem de programação para encapsular dados e elementos procedurais e criar a noção de herança de classes (Bergin e Greenfield, 1988). O Python é uma linguagem de programação orientada a objetos, ou seja, contempla recursos de abstração e encapsulamento, além de possibilitar a implementação de um framework durante o

desenvolvimento.

De acordo com Gamma *et al.* (2009), uma das formas de avaliar a qualidade de um sistema orientado a objetos é observando o nível de cuidado tomado com as colaborações entre os objetos que compõem o todo. Os padrões de projeto para o desenvolvimento de software foram elaborados a partir desse conceito, propondo soluções reutilizáveis para problemas conhecidos, de tal modo que se obtenha uma otimização da estrutura geral de uma arquitetura orientada a objetos, tornando-a de fácil compreensão e visualização.

O projeto de soluções reutilizáveis é de alta complexidade e requer um grau elevado de compreensão do problema enfrentado (Gamma *et al.*, 2009). Os padrões de projeto atuam nesse ponto, sendo soluções concebidas por projetistas experientes que implementam adequadamente a relação entre objetos e fornecem alternativas para a resolução de problemas recorrentes (Gamma *et al.*, 2009). Compreende-se, dessa forma, que os padrões de projeto atuam de uma forma que permite a reutilização de uma mesma implementação para problemas similares, diferentemente de funções, que atuam resolvendo problemas específicos, privando-se do reuso.

O número de padrões de projeto existentes é elevado e se pode separá-los em três grupos: padrões de criação, padrões estruturais e padrões comportamentais. Cada padrão atua em uma determinada parte de um sistema e a utilização de múltiplos padrões em um único projeto é essencial para que estre seja elaborado e funcione de maneira otimizada (Gamma *et al.*, 2009). As características de cada grupo são:

- Segundo Gamma *et al.* (2009), os padrões de criação abstraem o processo de instanciação de classes no sistema, o que colabora para torná-lo independente da forma com a qual seus objetos são criados, compostos e representados. Seu uso é importante quando o sistema passa a depender mais da composição de objetos do que da herança de classes, definindo-se uma série de comportamentos fundamentais que, quando compostos, formam comportamentos mais complexos (Gamma *et al.*, 2009).

- Os padrões estruturais se preocupam com a forma utilizada para compor os objetos e formar estruturas maiores (Gamma et al., 2009). Padrões estruturais de classes utilizam a herança para compor interfaces ou implementações, enquanto padrões estruturais de objetos trabalham na composição de objetos para a obtenção de novas funcionalidades, dando dinamicidade ao sistema pelo fato da composição poder ser alterada em tempo de execução (Gamma et al., 2009).

- Para os padrões comportamentais, a forma com a qual os objetos são criados ou compostos deixa de ser o foco e dá lugar a como a comunicação entre estes é estabelecida. Segundo Gamma *et al.* (2009), esse fluxo de comunicação é complexo em tempo de execução, por isso os padrões comportamentais afastam o foco deste fluxo de controle e permite que o desenvolvedor se preocupe somente com a forma que os objetos se interconectarão.

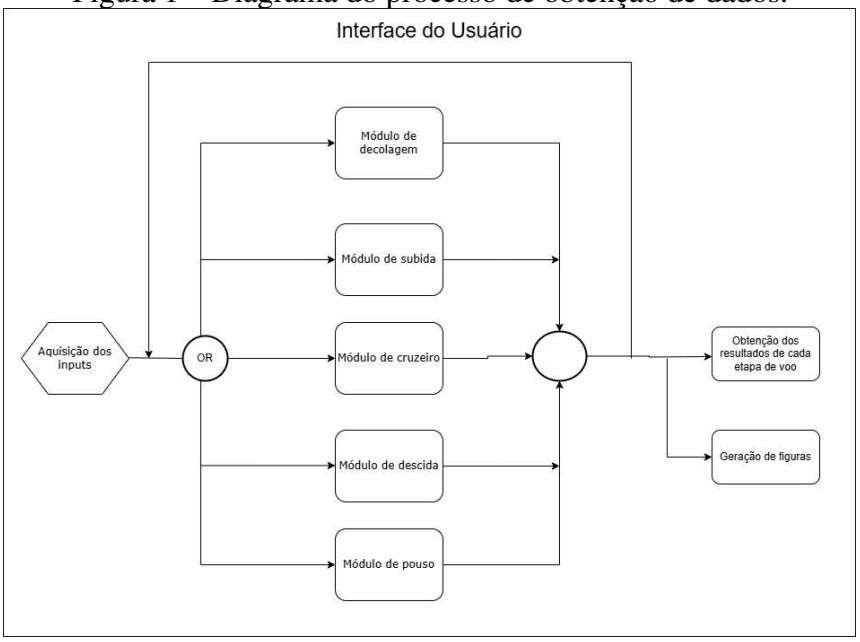
2.3 VISÃO GERAL DO SISTEMA

O funcionamento do software desenvolvido neste trabalho busca facilitar ao máximo a obtenção dos dados de desempenho da aeronave em análise, sem que o usuário precise editar as implementações dos métodos utilizados, de tal forma que todo o processo de cálculo é ocultado pela interface de utilização. O funcionamento é dividido entre os módulos de decolagem, subida, cruzeiro, descida e pouso, os quais representam as etapas de voo. A Figura 1 mostra o

fluxograma de funcionamento do software.

Percebe-se que, após a obtenção das entradas, o usuário pode escolher qual módulo pretende utilizar, sendo este de execução única. Após o término dos cálculos, os resultados obtidos serão mostrados e poderão ser armazenados, assim como gráficos relacionando variáveis pertinentes para cada etapa de voo e a análise de uma outra etapa de voo poderá ser realizada, reiniciando o processo.

Figura 1 – Diagrama do processo de obtenção de dados.



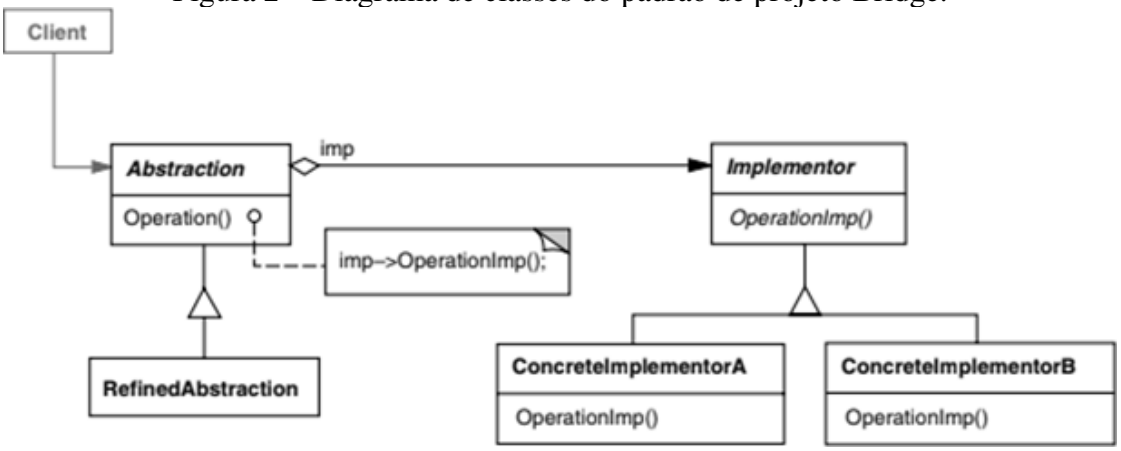
Fonte: Autor (2023).

A construção de todos os módulos foi realizada utilizando POO e padrões de projeto, afim de otimizar a estrutura do projeto e garantir o funcionamento do sistema. Os padrões utilizados foram o Bridge e Singleton, para a construção dos módulos de cálculos (decolagem, subida, cruzeiro, descida e pouso) e de bibliotecas (aerodinâmica, aeronave e método da bisseção), respectivamente, aproveitando-se da própria estrutura da linguagem Python para consolidar a relação entre classes.

2.4 BRIDGE

O padrão de projeto Bridge é um padrão estrutural utilizado para permitir diversas implementações para uma abstração (Gamma *et al.*, 2009). Utiliza-se uma classe abstrata para definir uma interface e demais classes concretas que farão a implementação de maneiras diferentes utilizando a herança de classes, impedindo o vínculo permanente da abstração e da implementação, sendo possível selecionar a segunda durante a execução (Gamma *et al.*, 2009). A Figura 2 mostra a estrutura do padrão Bridge.

Figura 2 – Diagrama de classes do padrão de projeto Bridge.



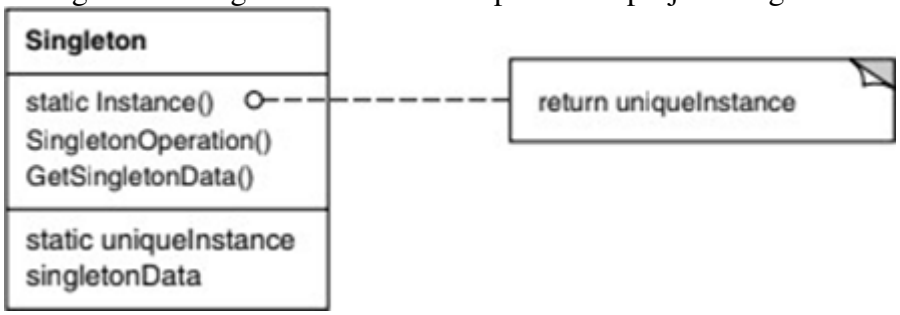
Fonte: Gamma *et al.* (2009, p. 153).

Percebe-se que a interface de abstração é definida pela classe Abstraction e a classe Implementor define a interface de implementação para ConcreteImplementorA e ConcreteImplementorB, que efetivamente implementam o método OperationImp() para casos particulares.

2.5 SINGLETON

Segundo Krzysztof (2008), o padrão de projeto Singleton é utilizado para garantir que uma classe tenha apenas uma instância, deixando-a a em um contexto global, de tal forma que seu acesso seja possível por diversos módulos de um sistema. A Figura 3 mostra a estrutura do padrão.

Figura 3 – Diagrama de classes do padrão de projeto Singleton.



Fonte: Gamma *et al.* (2009, p. 130).

A estrutura do Singleton é simples, sendo o pilar principal a garantia de um ponto global de acesso a classe. No caso da linguagem Python, a definição de biblioteca padrão da linguagem utiliza este padrão, facilitando a utilização dele no projeto.

2.6 ESTRUTURA DOS MÓDULOS

Dadas as particularidades da análise de cada etapa de voo, os módulos do software foram construídos de maneiras distintas, dadas as necessidades de cada caso. Para os módulos

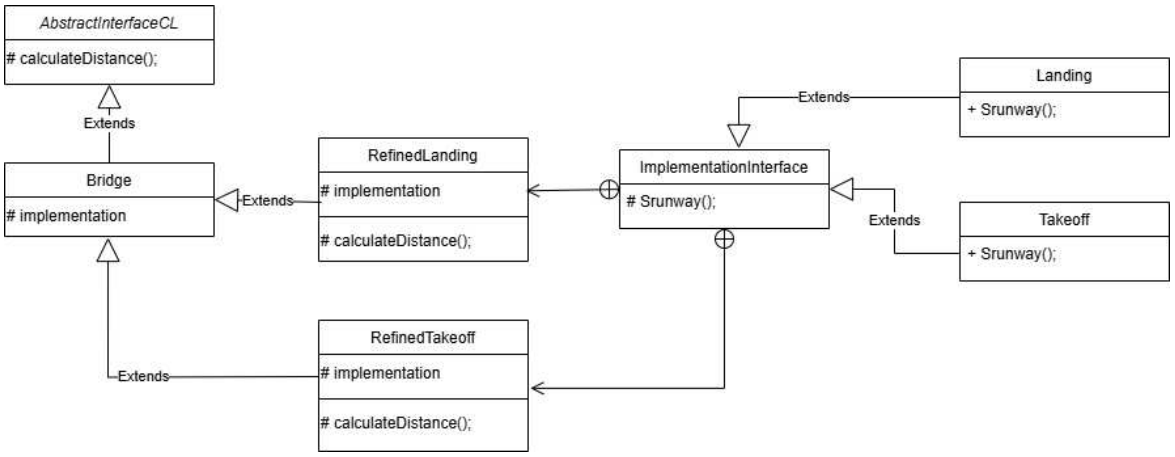
que compartilham características, a estrutura geral foi mantida similar, a fim de deixar o código padronizado e facilitar o entendimento do código para futuras adições e melhorias.

2.6.1 Módulos de decolagem e pouso

As etapas de decolagem e pouso de uma bateria de voo do aerodesign se sustentam na distância que o avião precisa percorrer para decolar e pousar, respectivamente, sendo essa calculada por meio de um método numérico de integração das equações do movimento. Sabendo que o dado de interesse é uma distância e que o método utilizado para cada caso apresenta pequenas diferenças, optou-se por utilizar o padrão Bridge na construção dos módulos.

A separação dos módulos se torna, dessa forma, mais clara e inteligível, já que a variação da implementação de cada caso fica clara a partir da particularização do método definido na interface. Além disso, evita-se redundâncias no código quanto à definição das mesmas variáveis em cada um dos módulos, otimizando todo o processo. A Figura 4 mostra a estrutura dos dois módulos e suas relações utilizando o Bridge.

Figura 4 – Diagrama dos módulos de decolagem e pouso do projeto.



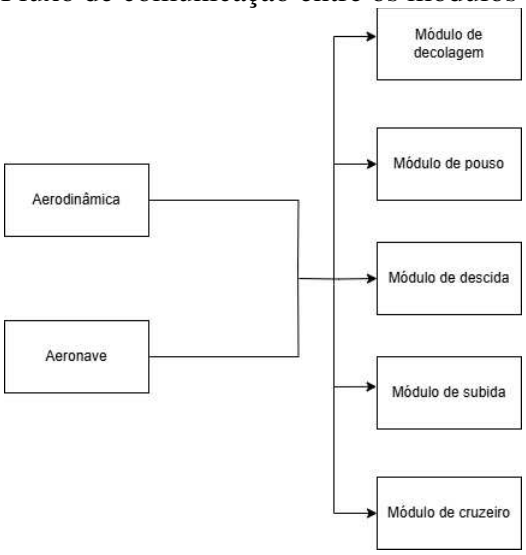
Fonte: Autor (2023).

As classes Landing e Takeoff implementam o cálculo da distância em cada caso e herdam a classe ImplementationInterface, na qual a interface de implementação é definida. Esta é uma classe interna de RefinedLanding e RefinedTakeoff, que se comunicam por meio da herança com a interface abstrata AbstractInterfaceCL por meio da classe Bridge. Os demais cálculos dos dois módulos são realizados por meio de funções que implementam métodos da mecânica do voo.

2.6.2 Módulos de aerodinâmica e aeronave

Para que os cálculos dos parâmetros de desempenho possam ser realizados, é necessário que parâmetros de aerodinâmica sejam definidos previamente, pois algumas equações utilizadas dependem destes. Da mesma forma, parâmetros da própria aeronave precisam ser definidos no começo do processo. Mostrou-se necessário a definição de classes exclusivas para ambos, utilizando o padrão Singleton da própria linguagem Python por meio da definição de bibliotecas. A Figura 5 mostra a comunicação entre os módulos utilizando o Singleton.

Figura 5 – Fluxo de comunicação entre os módulos do projeto.



Fonte: Autor (2023).

Percebe-se que todos os módulos de cálculo (representando cada uma das fases de voo) têm acesso ao módulo de criação da aeronave, cujos valores de variáveis serão definidos assim que o usuário acessar o software, e ao módulo de aerodinâmica, que depende das saídas de cálculos do setor.

Por fim, a estrutura geral do sistema desenvolvido é ilustrada no Anexo A, Figura 14, contendo o diagrama de classes que relaciona seus componentes. Por meio deste, é possível compreender as relações entre cada parte do software e de que forma cada uma delas tem acesso aos recursos implementados.

3 DESENVOLVIMENTO DO SISTEMA

Johnson (1997) traz duas definições para frameworks, a primeira sendo um design reutilizável de uma parte ou de todo um sistema, representado por classes abstratas e a maneira como suas instâncias interagem, a segunda é um esqueleto de uma aplicação que pode ser customizado por um desenvolvedor. Dentro desses conceitos, ressalta-se a interação entre classes abstratas e a possibilidade de customização de um modelo de aplicação por um desenvolvedor, presentes na linguagem Python, já que é orientada a objetos. Utilizou-se, a partir disso, os frameworks Kivy e KivyMD para a construção das telas, visando otimizar o processo de desenvolvimento.

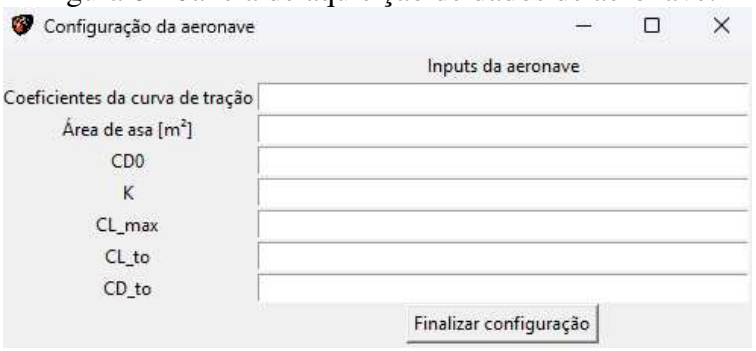
De acordo com a documentação do Kivy (2024), o framework é uma ferramenta de código aberto focada no desenvolvimento e distribuição de aplicações multiplataforma com uma UI. Sua escolha partiu da fase de elaboração do projeto, visando atender ao RNF3 a partir de uma única ferramenta. No mais, o reaproveitamento de padrões e de código são princípios fundamentais deste trabalho, portanto, optou-se pela utilização do KivyMD como padrão de designs para o Kivy.

O KivyMD trabalha a partir da utilização de widgets, os elementos que compõem a tela, como botões, barras de navegação e imagens, que implementam o Material Design a partir de objetos (KivyMD, 2024). Este é um sistema criado pela empresa Google para promover o desenvolvimento de aplicações de alto nível para Android, iOS, Flutter e a Web (Google, 2024). Cada widget, dessa forma, faz a utilização de um padrão já estabelecido, garantindo um design já validado de UI.

Houve, ainda, a necessidade de utilização do framework Kaki para o gerenciamento das telas. O KivyMD por si só não consegue fazer a atualização de telas de uma aplicação em tempo de execução, no instante que o script tem uma nova versão. Esse é um recurso importante para o desenvolvimento, já que o teste de funcionamento da implementação depende da visualização dos Widgets dispostos em tela. O Kaki realiza tal gerenciamento de telas utilizando a implementação do padrão de projeto Factory e o objeto ScreenManager do KivyMD.

Por fim, janelas de aquisição de dados a partir de entradas do usuário foram construídas utilizando o módulo Tkinter, a biblioteca de janelas padrão do Python. A ferramenta está disponível em sistemas Unix, MacOS e Windows, contemplando o requisito de a aplicação ser multiplataforma. Além disso, sua escolha se baseou na facilidade de implementar funções de aquisição de dados com o módulo e a necessidade de uma interface simples nas janelas de interação, simplificando a comunicação com os módulos e widgets internos. A Figura 6 mostra um exemplo de janela de aquisição de dados, as entradas obtidas são descritas no Anexo B, Quadro 5.

Figura 6 – Janela de aquisição de dados de aeronave.



Fonte: Autor (2024).

A tela em questão é a janela de configuração de aeronaves, na qual o usuário passa parâmetros de aerodinâmica que serão utilizados nas funções do objeto da aeronave. Destaca-se a simplicidade da interface, composta apenas de texto com pouca formatação, campos de entrada de dados e fundo em tonalidades de cor similares e um único botão, responsável por confirmar as entradas, chamar a função de criação da aeronave e fechar a janela. A Figura 7 mostra o algoritmo que representa o funcionamento e comunicação dessa janela com o resto da aplicação.

Figura 7 – Comunicação entre módulos durante a aquisição de entradas.

```
def AquisicaoEntradas():
        cria objeto da janela de aquisição de entradas
        abre janela de aquisição de entradas

        se botão de submissão de entradas for clicado:
                fechar janela de aquisição entradas
                obter os valores preenchidos em cada campo da janela
                converter os valores obtidos de string para float
                acessar o módulo da aeronave
                atribuir os valores obtidos aos atributos respectivos do objeto aeronave
                executar os cálculos de decolagem
                executar os cálculos de subida
                executar os cálculos de cruzeiro
                executar os cálculos de descida
                executar os cálculos de pouso
                atualizar as strings de resultados
```

Fonte: Autor (2024).

Como mostrado, a comunicação com o widget principal do software é realizada por meio de funções que recebem a instância do objeto App, que guarda as informações de todas as variáveis necessárias e tem acesso aos módulos da Figura 5 (no tópico 2.6.2). O acesso ao objeto da aeronave é garantido, podendo definir os atributos obtidos do usuário no momento de criação de uma nova instância e permitindo o armazenamento do último estado desta no fechamento do programa. Este é essencial, já que o fluxo de chamadas de funções é alterado pela existência ou inexistência do arquivo contendo o último estado da aeronave.

3.1 ESTRUTURA DO PROJETO KIVYMD

O projeto KivyMD se divide em duas partes principais, código Python, contendo as definições de widgets na forma de classes e funções do sistema, e código Kv language (Kvlang), linguagem desenvolvida pelo Kivy para implementação da árvore de widgets de maneira declarativa. Esta permite a implementação da árvore de widgets, isto é, a forma como estão organizados e se relacionam dentro de uma mesma tela (KivyMD, 2024). Determinam, além disso, os call-backs, alterações geradas pela utilização de alguma função do software, permitindo a separação da lógica de implementação do processo de design da UI.

A Kvlang garante, dessa forma, clareza na separação de cada componente do projeto, seja ele responsável por implementar um fluxo de execução ou definir a organização de uma tela do software. O processo de manutenção e atualização do sistema se torna facilitado, já que é possível trabalhar separadamente no desenvolvimento da lógica de implementação e UI, dada sua separação. Permite-se, portanto, liberdade de trabalho e alteração dos recursos já implementados sem a necessidade de grandes alterações no projeto.

3.2 GERENCIAMENTO DE TELAS COM KAKI

Dado que software possui mais de uma tela e cada uma precisa ser atualizada em tempo de execução, mostrando os resultados mais atualizados para cada aeronave de referência para os cálculos, houve a necessidade de utilizar um gerenciador de telas. O próprio Kivy oferece uma solução com seu widget ScreenManager, porém, utilizou-se o módulo Kaki para contornar o problema. Este também implementa o hotreload, mecanismo de atualização de tela do programa enquanto este é desenvolvido, permitindo a visualização das atualizações em tempo real.

Figura 8 – Especificações do ScreenManager do Kaki

```
DEBUG = 1

KV_FILES = {
    os.path.join(os.getcwd(), 'aircraft_performance/screens/screenmanager.kv'),
    os.path.join(os.getcwd(), 'aircraft_performance/screens/mainscreen.kv'),
    os.path.join(os.getcwd(), 'aircraft_performance/screens/takeoffscreen.kv'),
    os.path.join(os.getcwd(), 'aircraft_performance/screens/cruizescreen.kv'),
    os.path.join(os.getcwd(), 'aircraft_performance/screens/landingscreen.kv'),
}

CLASSES = {
    'MainScreenManager': 'aircraft_performance.screens.screenmanager',
    'MainScreen':        'aircraft_performance.screens.mainscreen',
    'TakeoffScreen':     'aircraft_performance.screens.takeoffscreen',
    'CruizeScreen':      'aircraft_performance.screens.cruizescreen',
    'LandingScreen':     'aircraft_performance.screens.landingscreen',
}

AUTORELOADER_PATHS = [
    ('.', {'recursive': True})
]

def build_app(self):
    return Factory.MainScreenManager()
```

Fonte: Autor (2024).

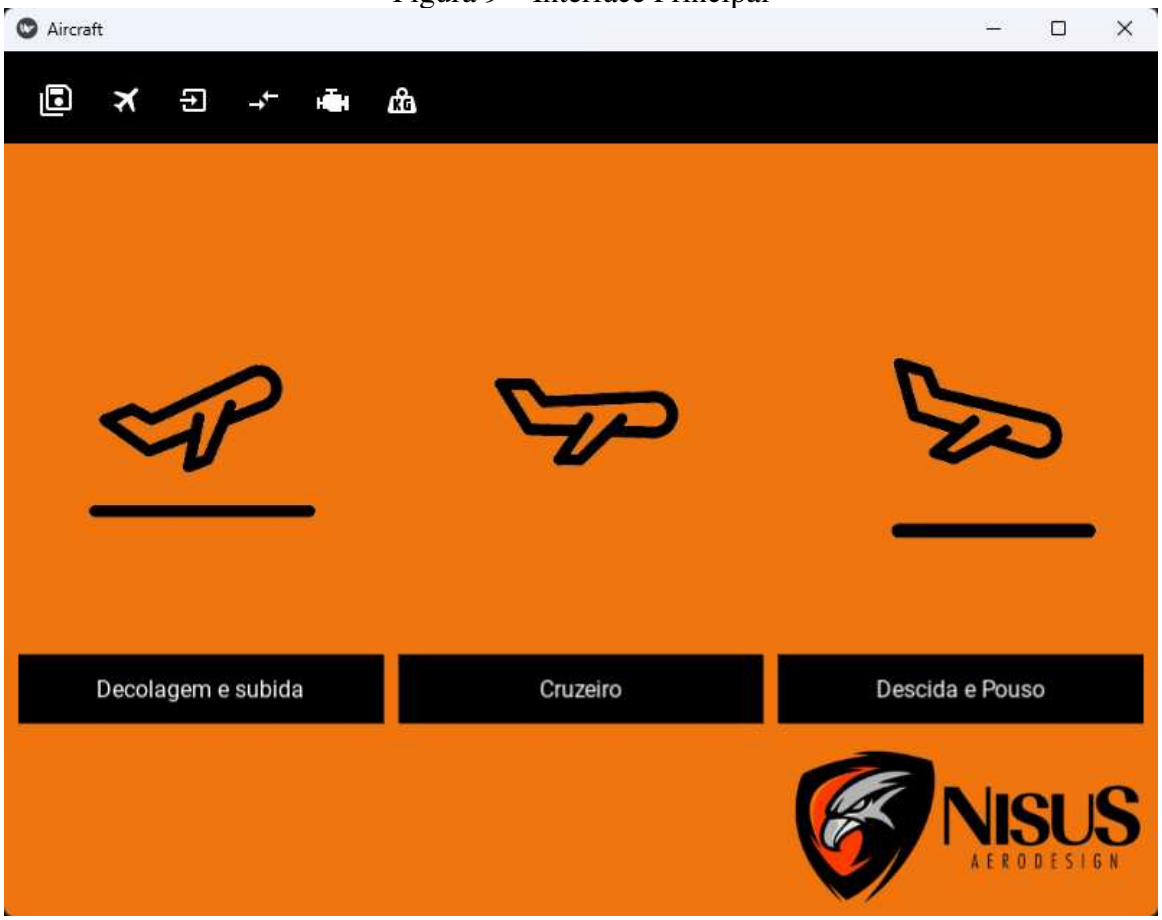
A implementação do Kaki é feita por meio do seu método Builder, que constrói o App, objeto contendo todas as especificações do software, incluindo os arquivos Python e Kvlang. É feita a declaração dos arquivos Kvlang e das classes (widgets de telas) que devem ser atualizados, como mostra a Figura 8.

Percebe-se a utilização do padrão de projeto Factory na função build_app(), reforçando a utilização dos mesmos no projeto. A atualização de múltiplas telas durante o desenvolvimento do software, dessa forma, é implementada, assim como seu gerenciamento, para garantir os resultados mais recentes dos cálculos realizados internamente pelo sistema.

3.3 INTERFACE PRINCIPAL

A interface principal é o local de acesso às funcionalidades da aplicação pelo usuário, a partir de botões. Por meio dela, pode-se navegar por telas e utilizar de recursos como salvamento e abertura de uma aeronave, já que encapsula toda a comunicação entre módulos e telas utilizando o Kivy e KivyMD. A Figura 9 mostra a organização dos widgets na interface principal.

Figura 9 – Interface Principal



Fonte: Autor (2024).

Destaca-se os ícones do Material Design, utilizados na barra de navegação superior, e a simplicidade proporcionada pelos frameworks utilizados. Cada um dos botões da região central (Decolagem, Subida, Cruzeiro, Descida e Pouso) levam para janelas específicas, nas quais são executados os métodos necessários para cálculos dos parâmetros de cada uma das fases de voo. Além disso, a implementação conta com responsividade, ou seja, computadores com monitores

de resoluções diferentes terão a UI redimensionada para proporções adequadas para sua tela, facilitando a utilização do software.

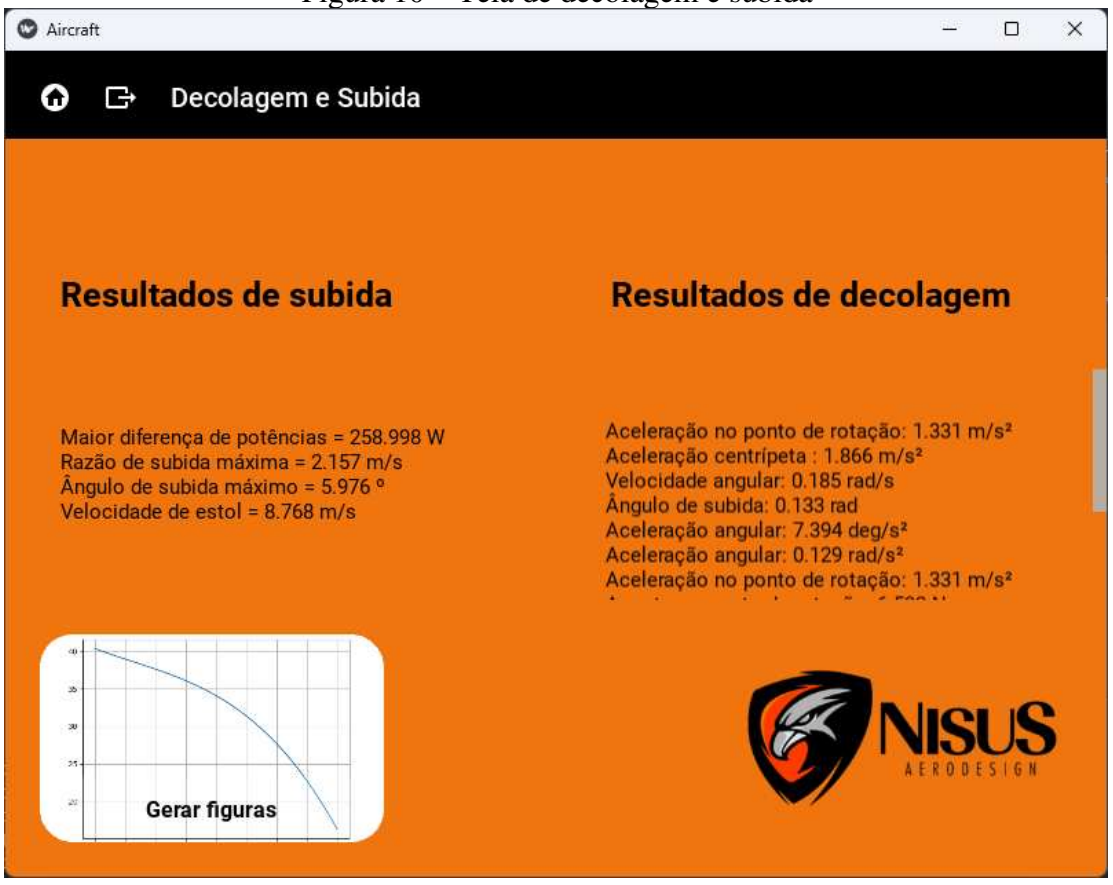
Nota-se que, os RF3, RF4, RF7, RF8 e RF9, podem ser acessados já na interface principal, tanto pelos recursos da barra de navegação quanto pelos botões centrais. Na barra de navegação estão dispostos os botões de salvamento de dados, abertura da janela de configuração, importação de dados e comparação de aeronaves, respectivamente, contemplando a leitura e escrita de arquivos e comparação de análises. Já os demais botões, implementam a divisão de telas para as análises de cada fase de voo.

A função de cada botão, se já não descrita na própria UI, pode ser visualizada colocando o cursor do mouse acima dele. A primeira utilização do software fica, dessa forma, intuitiva, já que todas as funcionalidades principais podem ser acessadas da tela inicial e as funções de cada widget são descritas para o usuário durante a utilização do software. Essa robustez é fruto do PU e dos requisitos definidos no início do projeto, que garantiram o uso adequado dos frameworks escolhidos.

3.4 TELAS SECUNDÁRIAS

As telas acessíveis pela interface principal do software são responsáveis por apresentar os resultados de cálculos realizados internamente, para cada uma das fases de voo. Ao acessar cada uma, é possível exportá-los (resultados de todas as fases de voo) em um arquivo csv, gerar gráficos por meio da Matplotlib ou retornar à tela inicial. As Figuras 10, 11 e 12 mostram cada uma das telas secundárias.

Figura 10 – Tela de decolagem e subida



Fonte: Autor (2024).

Figura 11 – Tela de cruzeiro

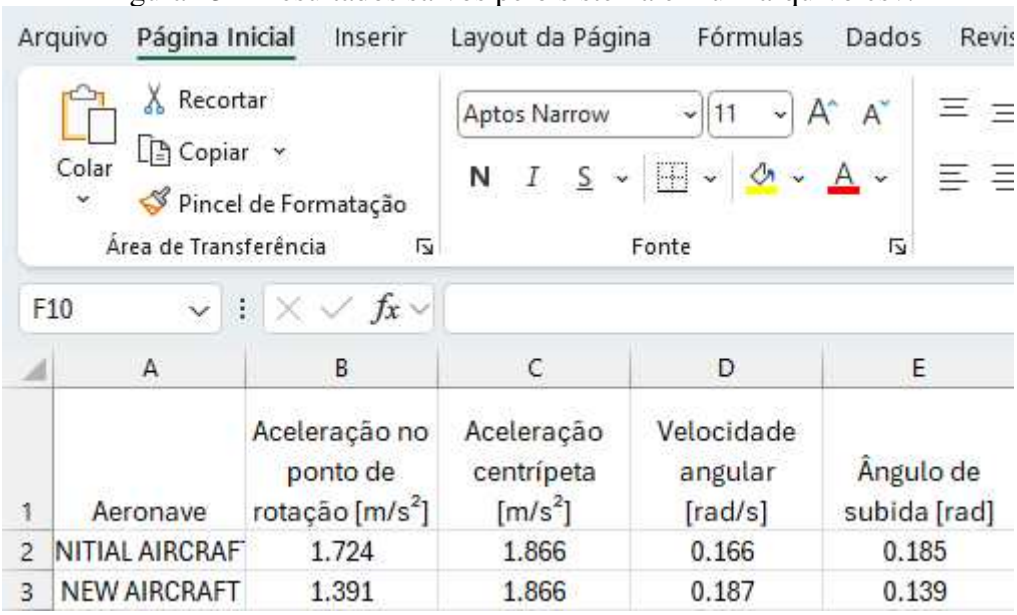
Fonte: Autor (2024).

Figura 12 – Tela de descida e pouso

Fonte: Autor (2024).

Os resultados podem ser somente visualizados, já que são dispostos em tela, ou armazenados em arquivos de texto para futuro processamento por demais setores da Equipe, como mostra a Figura 13.

Figura 13 – Resultados salvos pelo sistema em um arquivo csv.



	A	B	C	D	E
1	Aeronave	Aceleração no ponto de rotação [m/s²]	Aceleração centrípeta [m/s²]	Velocidade angular [rad/s]	Ângulo de subida [rad]
2	NITIAL AIRCRAF	1.724	1.866	0.166	0.185
3	NEW AIRCRAFT	1.391	1.866	0.187	0.139

Fonte: Autor (2024).

Os parâmetros mostrados são uma parte dos resultados calculados para cada etapa de voo na comparação de aeronaves pelo sistema, salvando os dados em um arquivo csv, aberto utilizando o Microsoft Excel, para facilitar estudos sobre os dados, armazenamento e transmissão destes aos demais setores da equipe.

Os gráficos gerados para os parâmetros pertinentes também podem ser visualizados no Anexo C, Figuras 15 – 18. Estas são utilizadas nas discussões do relatório de projeto entregue à comissão organizadora da competição, compondo parte da nota final da equipe.

4 CONSIDERAÇÕES FINAIS

Ao longo deste trabalho, foi desenvolvido um software em Python para o cálculo de parâmetros de desempenho das aeronaves projetadas pela Equipe Nisus Aerodesign, fazendo-se a utilização de padrões de projeto para sua construção. A modelagem do sistema foi realizada a partir da definição de requisitos funcionais, não funcionais e regras de negócio, dos quais foram definidos os recursos que dariam suporte à implementação.

As telas foram desenvolvidas por meio do Kivy e KivyMD e, para janelas simples, utilizou-se o Tkinter. Os dois primeiros fazem uso do Material Design, uma linguagem de design livre da empresa Google amplamente utilizada para a construção de interfaces gráficas. Esses recursos permitiram a construção de janelas de fácil utilização, já que cada tela conta com poucos elementos e os itens possuem identificação ou descrição claras.

As funcionalidades implementadas buscaram englobar os processos executados ao longo do desenvolvimento da aeronave, incluindo a aquisição, processamento e salvamento de dados. Conforme as mudanças de características físicas do avião projetado ocorrem, é necessário o reprocessamento dos parâmetros de Desempenho, portanto, implementou-se uma janela de aquisição de entradas necessárias para o desenvolvimento dos cálculos, que são feitos de maneira automática após a confirmação desses e, posteriormente, um único botão cuida do salvamento dos resultados obtidos.

Alcançou-se, dessa forma, uma otimização, tanto de tempo quanto de quantidade de trabalho das etapas de cálculo realizadas ao longo do projeto, permitindo que seja feita a melhoria de processos pelo setor de Desempenho ou o desenvolvimento de novas metodologias. Além disso, houve a flexibilização dos prazos de treinamento de novos membros para entendimento dos métodos encapsulados pelo software desenvolvido e posterior manutenção deste.

Quanto ao atendimento aos requisitos mostrados na Figura 1, o RF1 é atendido pela funcionalidade de obtenção da curva de tração de motores elétricos, na qual é feita uma regressão polinomial a partir de dados de arquivo de texto e obtido uma curva de terceiro grau relacionando Tração disponível, velocidade e altitude de voo. O RF2 foi parcialmente atendido, já que as telas de cada fase de voo possuem a opção de gerar gráficos avaliados pelo setor, porém, nem todos os gráficos presentes no relatório de projeto de Desempenho são gerados, dadas as variações dos parâmetros das aeronaves de anos diferentes de acordo com regulamento, o que dificulta a garantia de boa visualização de qualquer plotagem, sendo necessária a inclusão de uma ferramenta de edição de gráficos para ser totalmente atendido.

Os RF3 e RF4 foram atendidos, já que é permitida tanto a leitura quanto o salvamento de dados em arquivos de texto nas extensões .txt e .csv. O RF5 foi parcialmente atendido, já que as mensagens de erro mostradas são somente do próprio Python, quando há algum problema no funcionamento do sistema, tornando a RN3 também parcialmente atingida. É interessante que mensagens de erro sejam desenvolvidas para instruir o usuário caso utilize recursos de forma inapropriada, como formatações inadequadas de arquivos de texto ou formatos incorretos de entradas para atendimento pleno do requisito, o que envolve um trabalho de entendimento do fluxo de processos e melhoria da experiência do usuário.

O RF6 também foi atendido parcialmente, já que o software pede a especificação de quais colunas utilizar para a leitura de entradas dos arquivos de texto, mas não restringe totalmente o formato do arquivo. Os RF7, RNF1 e RN1 foram totalmente atendidos, já que a alteração das entradas pode ser feita a qualquer momento e estes são coletados por meio de janelas de pop-up. O RF8 foi parcialmente atendido, já que a implementação permite somente a comparação entre dados de duas aeronaves e a implementação deve ser alterada para permitir números maiores de aeronaves comparadas.

O RF9 foi atendido plenamente, pois há a separação de abas para cada etapa de análise. O RF10 não foi atendido, já que não há nenhuma etapa que necessita de rotinas executadas previamente para o funcionamento correto do software, não sendo necessária sua implementação. É importante ressaltar, entretanto, que os parâmetros utilizados estejam corretos, já que variáveis calculadas em uma etapa de análise podem ser utilizadas em outra etapa e, caso não sejam consistentes, o funcionamento do sistema não será afetado, mas os resultados estarão incorretos.

O RNF2 foi completamente atendido, já que a paleta de cores é laranja escuro/preto em sua maioria, buscando o conforto ocular durante a utilização do software. O RNF3 foi plenamente atendido, já que os recursos utilizados durante o desenvolvimento funcionam tanto no sistema operacional Windows quanto Ubuntu e testes foram realizados em diferentes computadores para garanti-lo. O RNF4 não foi atendido, já que não foi desenvolvida uma aba de tutorial para o sistema, cuja inclusão será necessária pelos que fizerem a manutenção dele.

A RN2 foi plenamente atingida, já que as funcionalidades do software são acessadas por meio de uma barra de navegação na parte superior esquerda de cada interface. A RN4 também foi plenamente atingida, sendo possível a comparação de parâmetros não só das etapas especificadas, mas de todas as etapas de voo analisadas. Por fim, a RN5 foi plenamente atingida, sendo permitida a alteração dos parâmetros de busca citados a qualquer momento durante a utilização, refazendo-se o cálculo.

Como resultado geral, o software desenvolvido neste trabalho contempla as

necessidades de implementação deste no contexto da Equipe Nisus Aerodesign para a simplificação dos processos do setor de Desempenho. É importante ressaltar que os membros devem ter noção dos métodos encapsulados pelo sistema, para que o conhecimento não se perca com o passar do tempo e a qualidade do relatório de projeto seja mantida. No mais, treinamentos sobre a utilização do software e sua construção devem ser elaborados, assim como um guia/documentação para consulta, já que o código fonte já é comentado.

O sistema desenvolvido abre caminho para a melhoria do projeto por meio do desenvolvimento de novas metodologias a partir da simplificação dos processos já existentes. Sua manutenção e atualização é de extrema importância para o projeto, que está em constante mudança de acordo com o regulamento de competição e a permanência da sua utilização é diretamente dependente de ambas. A ferramenta deve ser aproveitada de forma consciente, de modo que não se crie uma dependência total dela e, em caso de problemas no seu funcionamento, as análises de Desempenho não possam ser realizadas, comprometendo o andamento do projeto.

O Quadro 4 sintetiza o atendimento aos requisitos estipulados para o trabalho desenvolvido, especificando sua situação após a o término do desenvolvimento da primeira versão.

Quadro 4 – Atendimento aos requisitos do projeto

Requisito	Situação
RF1	Totalmente atendido
RF2	Parcialmente atendido
RF3	Totalmente atendido
RF4	Totalmente atendido
RF5	Parcialmente atendido
RF6	Parcialmente atendido
RF7	Totalmente atendido
RF8	Parcialmente atendido
RF9	Totalmente atendido
RF10	Não atendido (não necessário ao projeto)
RNF1	Totalmente atendido
RNF2	Totalmente atendido
RNF3	Totalmente atendido
RNF4	Não atendido
RN1	Totalmente atendida
RN2	Totalmente atendida
RN3	Parcialmente atendida
RN4	Totalmente atendida
RN5	Totalmente atendida

Fonte: Autor (2024).

O quadro deve ser utilizado em futuros desenvolvimentos e durante a manutenção do software desenvolvido neste trabalho, buscando sempre manter o atendimento aos requisitos definidos no início do desenvolvimento ou fazendo as alterações necessárias, de acordo com as necessidades do projeto da Equipe Nisus AeroDesign.

REFERÊNCIAS

BERGIN, Joseph; GREENFIELD, Stuart. What does Modular-2 need to fully support object oriented programming?. **ACM Sigplan Notices**, v. 23, n. 3, p. 73-82, 1988.

GAMMA, Erich. Padrões de projetos: soluções reutilizáveis. Bookman editora, 2009.

GOOGLE. **Material Design Documentation**. Disponível em: https://material.io/design. Acesso em: 2 out. 2024.

GUEDES, Gilleanes TA. **UML 2-Uma abordagem prática**. Novatec Editora, 2018.

JOHNSON, Ralph E. Frameworks=(components+ patterns). **Communications of the ACM**, v. 40, n. 10, p. 39-42, 1997.

KIVY. **Kivy Documentation**. Disponível em: https://kivy.org/doc/stable/. Acesso em: 2 out. 2024

KIVYMD. **KivyMD Documentation**. Disponível em: https://kivymd.readthedocs.io/. Acesso em: 2 out. 2024.

KRISTENSEN, Bent Bruun; ÖSTERBYE, Kasper. Conceptual modeling and programming languages. **ACM Sigplan Notices**, v. 29, n. 9, p. 81-90, 1994.

KRZYSZTOF, Stencel; WEGRZYNOWICZ, Patrycja. Implementation variants of the singleton design pattern. In: **OTM Confederated International Conferences" On the Move to Meaningful Internet Systems**. 2008.

LARMAN, Craig. **Utilizando UML e padrões**. Bookman Editora, 2007.

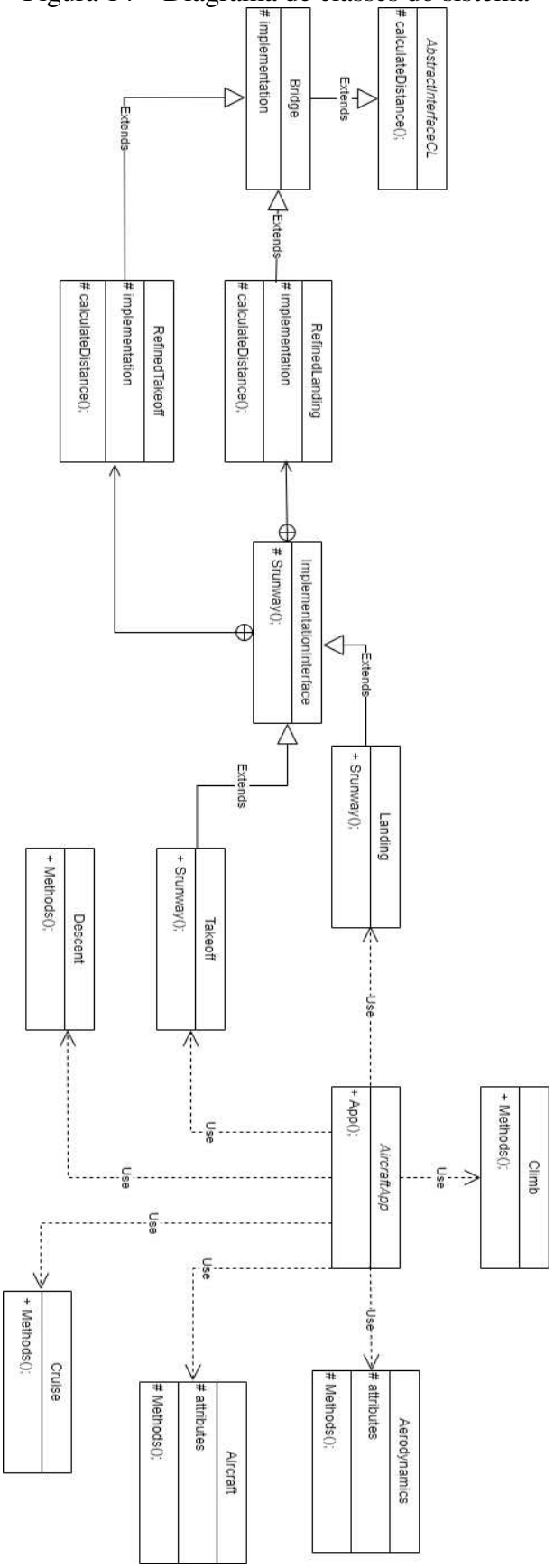
PYTHON SOFTWARE FOUNDATION. **Python 3.12 Documentation**. Disponível em: https://docs.python.org/3/. Acesso em: 2 out. 2024.

PYTHON SOFTWARE FOUNDATION. **Tkinter – Python Interface to Tcl/Tk**. Disponível em: https://docs.python.org/3/library/tkinter.html. Acesso em: 2 out. 2024.

YECHOUT, Thomas R. **Introduction to aircraft flight mechanics**. Aiaa, 2003.

ANEXO A – DIAGRAMA DE CLASSES DO SISTEMA

Figura 14 – Diagrama de classes do sistema



Fonte: Autor (2024).

ANEXO B – ENTRADAS INICIAIS OBTIDAS PELO SISTEMA

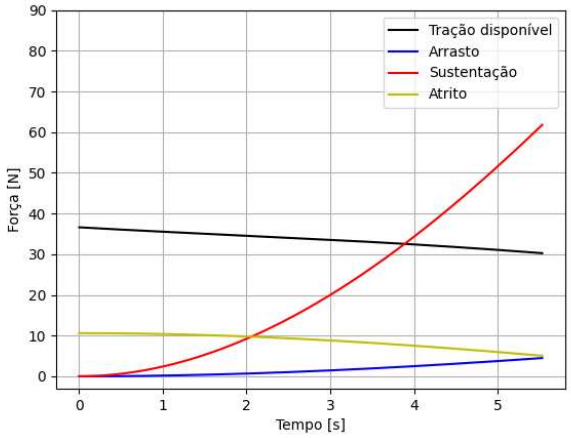
Quadro 5 – Entradas iniciais do sistema

Sigla	Descrição
-	Coeficientes da curva de tração do grupo motopropulsor, relacionando a tração disponível com a altitude e velocidade operacional
S	Área de asa da aeronave
CD0	Coeficiente de arrasto considerando o efeito solo
k	Termo constante da polar de arrasto, que relaciona os coeficientes de arrasto e de sustentação, relacionado ao fator de Oswald
CL_max	Coeficiente de sustentação máximo
CL_to	Coeficiente de sustentação durante a decolagem
CD_to	Coeficiente de arrasto durante a decolagem

Fonte: Autor (2024).

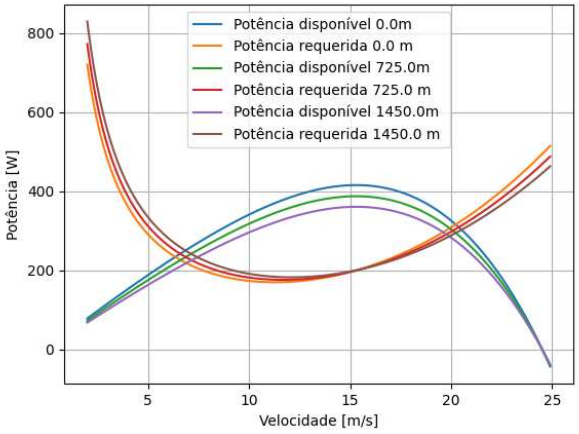
ANEXO C – FIGURAS GERADAS PELO SISTEMA

Figura 15 – Evolução das forças na aeronave durante a decolagem



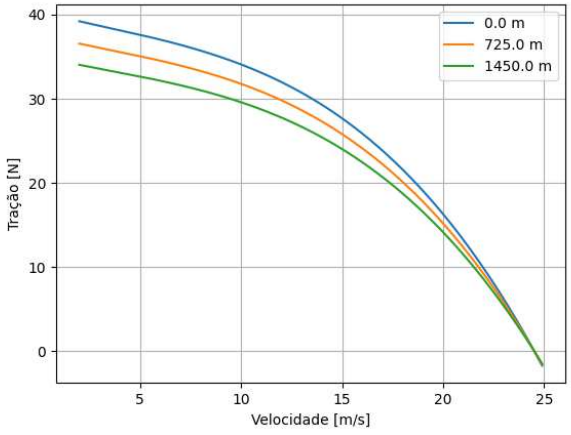
Fonte: Autor (2024).

Figura 16 – Potências disponível e requerida variando a altitude durante cruzeiro



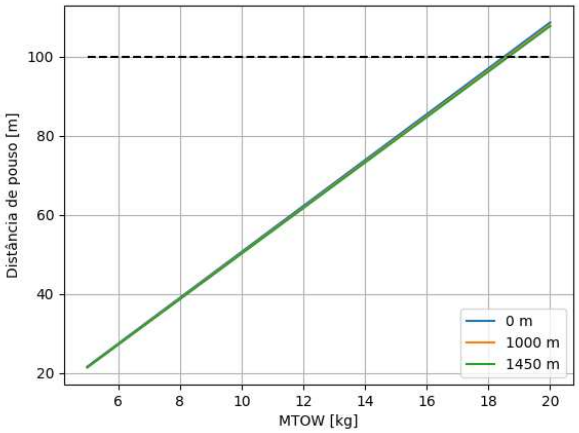
Fonte: Autor (2024).

Figura 17 – Curvas de tração para diferentes altitudes



Fonte: Autor (2024).

Figura 18 – Distância de pouso x MTOW



Fonte: Autor (2024).

AGRADECIMENTOS

Agradeço primeiramente aos meus pais, Selma e Silvino, que me deram suporte incondicional durante toda a graduação, impulsionando meu progresso durante as fases mais difíceis e permitindo que chegasse até este ponto. Obrigado por serem meus pilares e fonte de inspiração para seguir em frente.

Às Irmãs Azuis, em especial Dorly, Elizane, Iandra e Mara, por todo o suporte que deram à minha família para garantir minha estabilidade em uma cidade distante em momentos de dificuldade.

À Equipe Nisus AeroDesign, onde pude desenvolver boa parte das habilidades que garantiram a realização deste trabalho, vivi meus melhores momentos da graduação e construí grandes vínculos.

À irmã que ganhei da graduação, Laura Paulino, que me acompanhou durante toda esta jornada como um braço direito e ao irmão que ganhei da vida, Bruno Puzi, de quem primeiro recebi e sigo recebendo apoio. Aos grandes amigos Arthur Gouveia, Beatriz Faga, Danilo Machado, Gabrielle Zolet, Gabriel Costa, Hans Schulz, Irisson Lima, Laura Lasta, Lorena Vicente, Pedro Carvalho, Pedro Dallabrida e Widmark Kauê, que me apoiaram, tornaram a difícil rotina de estudos mais leve e toleraram as inúmeras e frequentes queixas sobre a vida acadêmica.

Ao time de Data Science das Lojas Quero-Quero, que me ajudou a desenvolver competências necessárias para este trabalho e para a vida profissional. Em especial, Angelo Carmignani, Gabriel Pastorello, Gabrieli Desiderio e Matheus Funck, pela paciência e atenção dedicados.

À minha orientadora Prof. Dra. Tatiana Renata Garcia, por ter aceitado a orientar este trabalho, e pela atenção e colaboração com seu desenvolvimento, assim como por compreender as dificuldades enfrentadas e direcionar soluções.

A todos os demais que, de alguma forma, contribuíram para o desenvolvimento deste trabalho.