UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO (CTC)
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE AUTOMAÇÃO E SISTEMAS

Luis Ramos Maia Costa

Simulação de VANTs com tilt rotores no ROS Gazebo

Florianópolis
2022

Luis Ramos Maia Costa

Simulação de VANTs com tilt rotores no ROS Gazebo

Dissertação submetida ao Programa de Pós-Graduação em Engenharia de Automação e Sistemas da Universidade Federal de Santa Catarina para a obtenção do Grau de Mestre em Engenharia de Automação e Sistemas

Orientador: Prof. Edson Roberto De Pieri, Dr.
Coorientador: Prof. Ebrahim Samer El Youssef, Dr.

Florianópolis
2022

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Costa, Luis Ramos Maia
Simulação de VANTs com tilt rotores no ROS Gazebo / Luis Ramos Maia Costa ; orientador, Edson Roberto De Pieri, coorientador, Ebrahim Samer El Youssef, 2022.
64 p.

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico, Programa de Pós-Graduação em Engenharia de Automação e Sistemas, Florianópolis, 2022.

Inclui referências.

1. Engenharia de Automação e Sistemas. 2. Simulação. 3. tilt-quadrirrotor. 4. Gazebo. 5. ROS. I. De Pieri, Edson Roberto. II. Youssef, Ebrahim Samer El. III. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Engenharia de Automação e Sistemas. IV. Título.

Luis Ramos Maia Costa

Simulação de VANTs com tilt rotores no ROS Gazebo

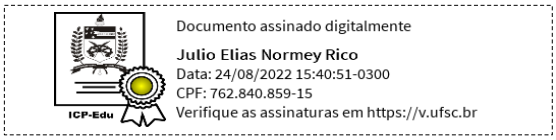
O presente trabalho em nível de Mestrado foi avaliado e aprovado, em 10 de agosto de 2022, pela banca examinadora composta pelos seguintes membros:

Prof. Bruno José Olivieri de Souza, Dr.
Tribunal Superior do Trabalho

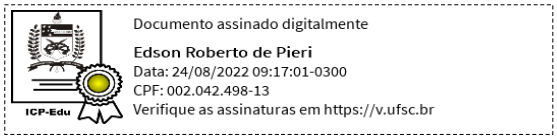
Prof. Eugenio de Bona Castelan Neto, Dr.
Universidade Federal de Santa Catarina

Prof.(a) Luciana Bolan Frigo, Dra
Universidade Federal de Santa Catarina

Certificamos que esta é a versão original e final do trabalho de conclusão que foi julgado adequado para obtenção do título de Mestre em Engenharia de Automação e Sistemas.



Coordenação do Programa de Pós-Graduação



Prof. Edson Roberto De Pieri, Dr.
Orientador(a)

Florianópolis, 2022

Este trabalho é dedicado à minha companheira e melhor amiga, Cristina, pelo apoio, pela dedicação, pela ajuda, pelo amor, pelo carinho, pela compreensão, enfim, por tudo.

AGRADECIMENTOS

Agradeço ao professor Edson, orientador deste projeto, por acreditar em meu potencial. Ao professor Ebrahim, mentor e coorientador deste projeto, por toda a ajuda, paciência, dedicação e confiança.

Ao Grupo de Robótica do Programa de Pós-Graduação em Engenharia de Automação e Sistemas da UFSC, Robotizando, especialmente ao Doutorando Julio C. Vendrichoski e à Doutoranda Thamiris L. Costa, pelas orientações e por fornecer as informações e arquivos STL do robô Lophorina.

“Se você encontrar um caminho sem obstáculos, ele provavelmente não leva a lugar nenhum.”
(Frank A. Clark)

RESUMO

Diversos trabalhos científicos e acadêmicos utilizam a simulação 3D como ferramenta para diminuir acidentes e erros, bem como promover economia no processo de desenvolvimento de protótipos. O simulador *open-source* 3D Gazebo possui integração nativa com o *framework* ROS, o que permite que o código gerado para o robô em simulação seja posteriormente utilizado no robô real. O Gazebo permite fazer a simulação física de robôs, no entanto para simular o voo de um robô aéreo é necessário a implementação de um *plugin* específico para gerar a força de empuxo. O presente trabalho apresenta um método para simulação do voo de tilt-quadrirrotores no ROS Gazebo, utilizando um *plugin* para gerar as forças de empuxo das hélices em rotação e deixando o motor de física calcular as resultantes e inércia do modelo. Esta abordagem diferencia-se de outros trabalhos que calculam as resultantes no *plugin* e aplicam no corpo do robô como um todo, o que funciona bem para o caso dos quadrirrotores mas não para o caso dos tilt-quadrirrotores onde a movimentação entre as partes do robô é importante para a simulação.

Palavras-chave: Tilt-Quadrirrotores; ROS; Gazebo.

ABSTRACT

Several scientific and academic works use 3D simulation as a tool to mitigate accidents and errors, as well as to promote savings in the prototype development process. The open-source 3D Gazebo simulator has native integration with the ROS framework, which allows the code generated for the robot in simulation to be later used in the real robot. Gazebo allows physical simulation of robots, however to simulate the flight of an aerial robot it is necessary to implement a specific plugin to generate the thrust force. The present work presents a method for simulating the flight of tilt-quadrotors in ROS Gazebo, using a plugin to generate the thrust forces of the rotating propellers and letting the physics engine calculate the resultant and inertia of the model. This approach differs from other works that calculate the results in the plugin and apply it to the robot's body as a whole, which works well for the quadrotor case but not for the tilt-quadrotor case where the movement between the robot parts is important for the simulation.

Keywords: tilt-quadrotor; ROS; Gazebo.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CTOL	Conventional Take-Off and Landing
DART	Dynamic Animation and Robotics Toolkit
XML	eXtensible Markup Language
FPV	First Person Viewer
FDM	Flight Dynamics Model
GNU	Gnu is Not Unix
JSBSim	Jon S. Berndt Simulator
LIDAR	Light Detection And Ranging
MORSE	Modular OpenRobots Simulation Engine
ODE	Open Dynamics Engine
OpenVSP	Open Vehicle Sketch Pad
PID	Proporcional, Integral e Derivativo
ROS	Robot Operational System
RViz	ROS Vizualization
STOL	Short Take-Off and Landing
SDF	Simulation Definition File
S.I.	Sistema Internacional
STL	Standard Tessellation Language
URDF	Universal Robot Description File
UFMG	Universidade Federal de Minas Gerais
UFSC	Universidade Federal de Santa Catarina
UAV	Unmanned Aerial Vehicle
VANT	Veículo Aéreo Não Tripulado
VTOL	Vertical Take-Off and Landing
YAML	YAML Ain't Markup Language
YASim	Yet Another Simulator

LISTA DE SÍMBOLOS

C_s	Coeficiente de sustentação
f_s	Força de sustentação
A	Área da asa
ρ	Densidade do ar
v	Velocidade linear
q	Pressão dinâmica
c	Corda do elemento infinitesimal
r	Distância ao eixo de rotação
w	Velocidade angular
α	Ângulo de ataque
d	Densidade do corpo
m	Massa do corpo
V	Volume do corpo
I_r	Tensor de Inércia (resultado)
i_{xx}	Componente do tensor na linha x e coluna x

SUMÁRIO

1 INTRODUÇÃO

Os veículos aéreos não tripulados (VANTs ou, em inglês, *Unmanned Aerial Vehicles*, UAVs) de asas rotativas são atraentes plataformas robóticas devido ao custo, simplicidade de uso e alta capacidade de manobra. Atualmente, os VANTs encontram ampla aplicação em fotografia aérea, proteção vegetal, agricultura de precisão, assistência a desastres, reconhecimento, apoio militar, recolhimento de amostras de solo, atuação em áreas de desastre, dentre outras. Dentre as diversas configurações de VANTs, o quadrirrotor, ou quadricóptero é muito difundido por se tratar de um modelo de VANT relativamente simples de ser implementado. O funcionamento do quadrirrotor é baseado em quatro hélices (asas rotativas) que giram em sentidos alternados, sendo o mesmo sentido nas hélices cruzadas. Esta configuração permite balancear os torques gerados pelas hélices e faz com que o quadrirrotor consiga voar e ficar estável. A partir do quadrirrotor, com a adição de atuadores que permitem a inclinação (*tilt*) dos propulsores em relação ao corpo principal, surgiu o tilt-quadrirrotor.

No processo de desenvolvimento de robôs móveis, a simulação é importante para testar tanto os componentes de software, quanto o comportamento do robô e os algoritmos de controle em diferentes ambientes, evitando o risco de danos causados por falhas (MEYER *et al.*, 2012). Construir um protótipo de um VANT pode ser muito oneroso e a possibilidade de erros durante o projeto pode elevar ainda mais este custo. Durante o uso deste equipamento também podem ocorrer acidentes que podem causar perdas materiais grandes, considerando que medidas de segurança manterão as pessoas em segurança. Em pesquisas de múltiplos VANTs (enxames, ou swarms) os custos se multiplicam somado à multiplicação dos protótipos, o que geralmente leva ao uso de um número mínimo de protótipos para provar os conceitos. Assim, o uso de simulação computacional se mostra muito mais econômico, evitando os elevados gastos de construção e reparo de equipamentos danificados, garante a segurança de pessoas durante a pesquisa e diminui o tempo necessário para iniciar e retomar a pesquisa após eventos adversos, sendo de vital importância para o desenvolvimento de pesquisas no campo de VANTs.

Na literatura temos muitas pesquisas voltadas para controle de VANTs e estas pesquisas são simuladas em ambiente computacional. Existem pesquisas que são simuladas matematicamente, principalmente em MATLAB, no entanto atualmente a comunidade científica tem valorizado simulações de ambientes em 3D que demonstram de forma mais realística a execução das tarefas.

O Gazebo é um ambiente de simulação física 3D que se conecta diretamente com o ROS (*Robot Operational System*), permitindo simular o comportamento de um ou vários robôs e depois portar a programação para um robô compatível com ROS. Por este motivo, o ambiente ROS Gazebo é amplamente utilizado em simulações de robôs, tanto aéreos, quanto terrestres.

Uma limitação do Gazebo reside no fato dele não simular as forças que mantêm um VANT planando (MEYER *et al.*, 2012), existindo então a necessidade de criar um *plugin* para simular este voo. Diversas pesquisas o fazem utilizando *plugins* próprios que recebem como entrada os parâmetros de como o robô deve voar e simulam o VANT planando através da aplicação da força resultante. Este tipo de simulação, embora de aparência realística, não

comprova a capacidade de voar do VANT: apenas gera as forças que colocam o robô na posição calculada a partir dos parâmetros de voo que foram fornecidos. Também não é possível simular o voo de um robô com características diferentes com o mesmo *plugin*, que é programado para um robô específico.

Outra forma de gerar as forças de empuxo é utilizar o *plugin* "*lift and drag*", que tem uma proposta diferente dos *plugins* mencionados anteriormente: simular as forças de empuxo e arrasto de uma superfície, de acordo com sua velocidade e aceleração. Assim, uma superfície, ao se movimentar pelo ambiente com uma determinada velocidade gera um arrasto, contrário à direção do movimento e também uma força de empuxo transversal a este movimento, o que pode dar sustentação a esta superfície, a qual pode levantar voo. Este *plugin* pode simular asas e é possível acessar uma simulação de um avião CESNA com este *plugin* utilizando apenas um comando no ROS. Este *plugin*, como será mostrado adiante, possui limitações à simulação de rotores, pois estes possuem hélices, ou asas rotativas, as quais possuem velocidades angulares e não lineares.

O presente trabalho apresenta um método alternativo para simulação física de um VANT em ROS Gazebo através da simulação da força de empuxo no eixo das hélices, de acordo com a velocidade angular destas hélices. Esta força é colocada no ambiente 3D do Gazebo e, ao interagir com as outras forças presentes no ambiente, deve permitir o voo do VANT. Este método se difere do uso de *plugins* específicos, que calculam a resultante aerodinâmica no corpo do robô como um todo, pois gera apenas a força de sustentação no eixo das hélices e deixa o cálculo de forças e inércia para o motor de física do Gazebo. Assim, se torna mais flexível, pois o *plugin* utilizado não precisa ser específico para cada simulação, sendo modificados apenas os parâmetros do *plugin* caso hajam mudanças nas hélices ou velocidade máxima dos rotores. Com este novo método também é possível simular *tilt* rotores, visto que a inclinação do eixo das hélices implicará na inclinação das forças aplicadas neles. O método proposto também se difere do uso do *plugin Lift and Drag*, utilizando parâmetros mais simples e em menor quantidade, mais adequados para hélices rotativas. O *plugin* proposto pode ainda ser utilizado em conjunto com o *plugin* "*lift and drag"* no caso de VANTs de asas fixas com propulsão por hélices.

Durante a pesquisa foram desenvolvidos:

- um *plugin* que gera a força perpendicular ao eixo e simula fisicamente em ROS Gazebo o voo de um VANT *tilt* rotor com 4 hélices;
- uma função de controle em *python* para manter o robô planando.

Este ferramental poderá ser utilizado em outros trabalhos, para simular robôs das mais diversas características e ajudar a verificar a possibilidade de voo dos mesmos.

Inicialmente o trabalho apresentará a formalização do problema com uma discussão sobre a necessidade do estudo, mostrando como o presente trabalho se diferencia dos existentes. A seguir a pesquisa é mostrada, através do detalhamento dos passos do método utilizado. Por fim, serão mostrados os resultados da simulação de voo do *tilt* rotor com o *plugin*, utilizando de um controlador em *python* para mantê-lo em uma altitude pré-determinada e serão feitas as conclusões do estudo.

1.1 OBJETIVOS

Os objetivos do trabalho estão divididos em objetivos gerais e específicos.

1.1.1 OBJETIVO GERAL

Propor um método para simulação de voo de um VANT com *tilt* rotores no ROS Gazebo com a simulação das forças de empuxo nas hélices.

1.1.2 OBJETIVOS ESPECÍFICOS

- Identificar as formas de simulação de VANTs em ROS Gazebo comuns disponíveis;
- Desenvolver um *plugin* para ROS Gazebo que gere a força de empuxo nas hélices dos VANTs;
- Reunir os dados do robô Lophorina, em desenvolvimento pelo Grupo de Robótica da UFSC;
- Desenvolver o modelo 3D do robô Lophorina no ROS Gazebo;
- Simular voo do robô Lophorina no ROS Gazebo com a simulação das forças de empuxo nas hélices;
- Definir um método de simulação de VANTs com *tilt* rotores em ROS Gazebo baseado nos resultados experimentais obtidos na simulação do voo do robô Lophorina;
- Disponibilizar códigos em forma de Framework para sustentar o método de simulação proposto.

1.2 DELIMITAÇÃO DO TRABALHO

O trabalho apresenta um método para simulação de quadrirrotores exclusivamente no ambiente 3D Gazebo, sendo utilizado para simular o robô tilt-quadrirrotor Lophorina, desenvolvido pelo grupo de robótica da UFSC. A simulação do robô apresenta a decolagem e planeio em uma altura pré determinada, utilizando-se da força de empuxo nas hélices. Como o trabalho está focado na simulação no ambiente físico em 3D, a modelagem de controle, incluindo o controle quando ocorre mudança dos ângulos de *tilt* não se encaixam no contexto deste trabalho. No entanto foi implementado um controlador PID (Controlador Proporcional, Integral e Derivativo) que fez o robô levantar voo e se manter planando em uma altura pré-determinada de 1 metro do chão. Também foi feito um teste de rotação do *tilt* em pequeno ângulo, o que produziu o efeito esperado de fazer que o robô se deslocasse para frente por algum tempo, tombando em seguida por falta de um controlador específico. Isto ocorre porque o robô é um sistema em um equilíbrio instável. Posteriormente foi feito um teste de rotação dos *tilts* em sentidos inversos de cada lado, o que também produziu o efeito esperado de fazer o robô girar em torno do seu eixo.

1.3 CONTRIBUIÇÃO CIENTÍFICA

As contribuições deste trabalho são:

1. O método simulação de um VANT com *tilt* rotores em ROS Gazebo, a partir da simulação das forças de empuxo;
2. O ferramental de apoio ao método, disponível em um *Framework* com os códigos necessários para a simulação das forças de empuxo.

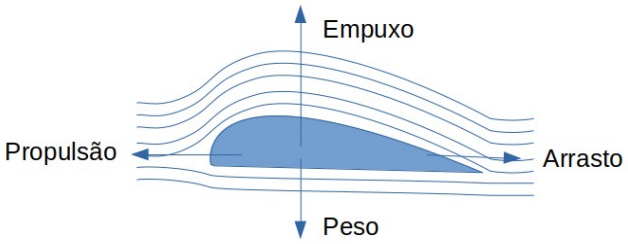
2 FUNDAMENTAÇÃO E FORMALIZAÇÃO DO PROBLEMA

Neste capítulo será feita a formalização do problema. Para podermos descrever qual problema está sendo resolvido, é necessário fundamentá-lo. Inicialmente será feita uma breve discussão de como as asas fixas e rotativas funcionam. Depois será mostrado o funcionamento dos quadrirrotores e como eles se diferenciam do tilt-quadrirrotores. São mostrados diversos tipos de tilt-quadrirrotores encontrados na literatura, mostrando a sua diversidade. Será abordada a simulação 3D e o ambiente ROS/Gazebo, que é utilizado nesta pesquisa. Depois será mostrado o *plugin* oficial do Gazebo, o *plugin Lift and Drag* e como ele é utilizado para simular asas fixas e móveis. Outros ambientes de simulação comuns encontrados na literatura também são abordados, com uma breve descrição de suas características. Por fim, é feita uma discussão sobre a necessidade do estudo, mostrando como o presente trabalho se diferencia dos existentes.

2.1 O FUNCIONAMENTO DAS ASAS

Uma aeronave consegue voar porque o perfil aerodinâmico de suas asas faz com que o ar que passa por cima das asas percorra um caminho maior que o ar que passa por baixo delas. Esta diferença de velocidades causa o efeito Bernoulli: a pressão em cima da asa é menor do que a pressão embaixo da asa, o que gera uma força resultante aerodinâmica. Esta força pode ser decomposta em duas forças: a primeira, para cima, chamada de empuxo, sustentação ou *lift* e a segunda, uma força de reação ao deslocamento da asa contra o ar, chamada de arrasto, ou *drag* (Figura 1). A força de empuxo é o que permite que a aeronave se sustente no ar. Para gerar esta força de sustentação, é necessário imprimir velocidade suficiente nas asas, através da aplicação de uma força, gerada pelos propulsores, que empurram o ar para trás, o qual reage impulsionando a aeronave para a frente. Esta força motora (também chamada de tração ou propulsão) deve gerar a velocidade suficiente para a diferença de pressão nas asas gerar a sustentação, que levanta voo ao superar o peso da aeronave. (RODRIGUES, 2014)

Figura 1: As forças envolvidas em um perfil aerodinâmico

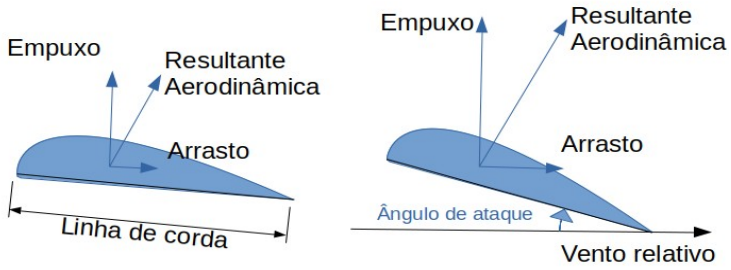


Fonte: do autor

No entanto, a força de empuxo gerada assim não é o suficiente para que o avião levante voo, sendo necessário aumentar o ângulo de ataque entre a corda da asa e o vento relativo (direção do movimento). Conforme esse ângulo vai aumentando, o empuxo também aumenta,

ao custo do aumento concomitante do arrasto (Figura 2). Quando é alcançado o ângulo de estol ou stall, o empuxo começa a diminuir, mas o arrasto continua aumentando.

Figura 2: Aumento do Empuxo e arrasto com o aumento do ângulo de ataque



Fonte: do autor

A sustentação depende da densidade do ar, do quadrado da velocidade, da viscosidade e compressibilidade do ar, da área de superfície sobre a qual o ar flui, da forma do corpo e da posição de aproximação do corpo ao fluxo de ar.

Uma maneira de lidar com dependências complexas é caracterizar a dependência por uma única variável. Para sustentação, essa variável é chamada de coeficiente de sustentação, designado Cs. Isso nos permite coletar todos os efeitos, simples e complexos, em uma única equação. A equação de sustentação afirma que a força de sustentação fs é igual ao coeficiente de sustentação Cs vezes a densidade do ar ρ vezes metade da velocidade linear da asa v ao quadrado vezes a área da asa A.

$$f_s = \frac{1}{2} C_s A \rho v^2$$

Para algumas condições de escoamento e geometrias simples e aproximações baixas, os aerodinamicistas podem determinar matematicamente o valor de Cs, mas, em geral, este parâmetro é determinado experimentalmente.

A metade da densidade vezes a velocidade ao quadrado é chamada de pressão dinâmica q. O coeficiente de sustentação expressa então a razão entre a força de sustentação e a força produzida pela pressão dinâmica vezes a área:

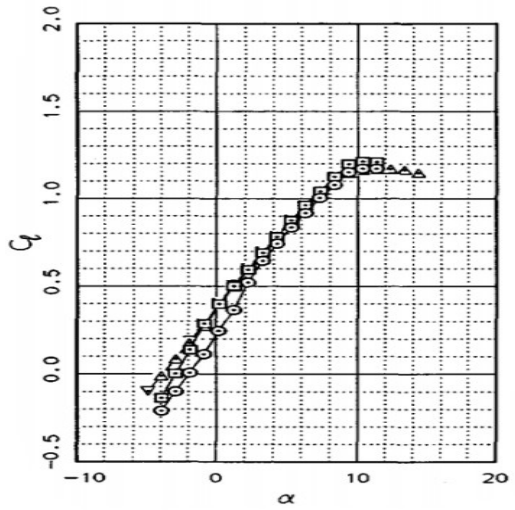
$$C_s = \frac{f_s}{q A}$$

O valor para o coeficiente de sustentação é geralmente determinado em um ambiente controlado (túnel de vento), onde se podes definir a velocidade, densidade e área e medir a

sustentação produzida. Através da divisão, chega-se a um valor para o coeficiente de sustentação. É possível então prever a sustentação que será produzida sob um conjunto diferente de velocidade, densidade (altitude) e condições de área usando a equação de sustentação.

Este valor do coeficiente de sustentação contém as dependências complexas da forma do objeto na sustentação, o que pode depender também dos efeitos da viscosidade do ar e da compressibilidade. Para velocidades baixas, os efeitos de compressibilidade são insignificantes, mas em velocidades supersônicas, a razão entre a velocidade e a velocidade do som altera o coeficiente de sustentação. Da mesma forma, a razão entre as forças inerciais e as forças viscosas, bem como o formato 3D e ângulo de ataque da asa interferem no cálculo deste coeficiente. O gráfico da medição experimental da variação do Coeficiente de Sustentação de acordo com a variação do ângulo de ataque está na Figura 3, extraída de (CYPRIANO; IMANISHI, 2014).

Figura 3: Variação do Coeficiente de Sustentação com o ângulo de ataque



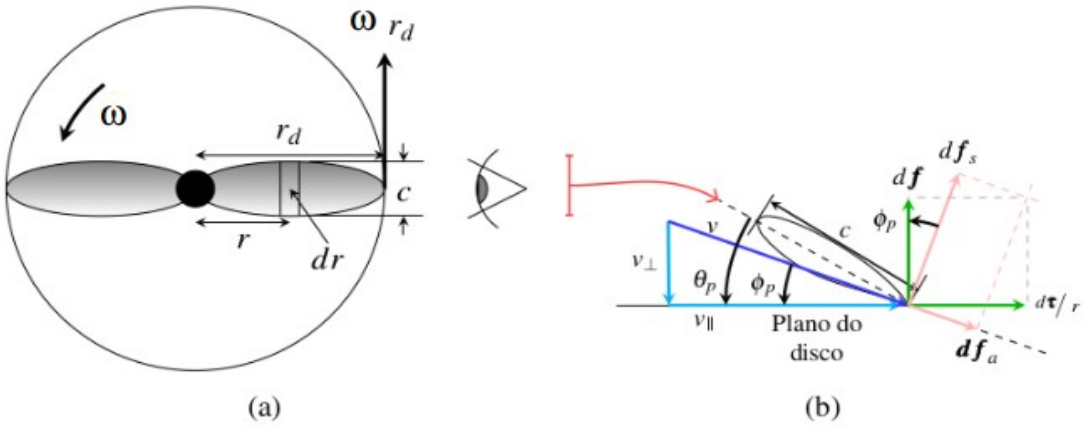
Fonte: (CYPRIANO; IMANISHI, 2014)

No caso das aeronaves de asas rotatórias, as asas são hélices que, ao girarem, se deslocam contra a coluna de ar, gerando o empuxo (Figura 4). A componente infinitesimal de força de sustentação de um elemento infinitesimal da hélice pode ser descrita como:

$$df_s = \frac{1}{2} C_s \rho v^2 c \, dr$$

Em que v é a velocidade linear tangencial do elemento infinitesimal e c é sua corda.

Figura 4: Asa rotatória



Fonte: (VENDRICHOSKI, Julio Cezar, 2017)

Pela equação percebe-se que há uma dependência da força à velocidade linear em cada parte da pá (a dependência de dfs em relação a velocidade tangencial implica na dependência em r). Como a velocidade v é igual à velocidade angular w vezes o raio:

$$df_s = \frac{1}{2} C_s \rho (\omega r)^2 c\, dr$$

$$df_s = \frac{\omega^2 \rho C_s}{2} c\, r^2\, dr$$

Assim, a força de empuxo tem variação de acordo com o raio (distância do eixo de rotação) ao quadrado multiplicado pela corda (largura da asa).

As partes da hélice mais próximas ao eixo de rotação produzem então menos força de sustentação devido ao baixo valor de r. Conforme aumenta distância do eixo de rotação o empuxo aumenta, mas próximo à extremidade das pás da hélice, a força reduz devido à diminuição da corda. Desta forma o ponto de maior força de empuxo, o centro de pressão, geralmente se localiza entre a metade da asa e a ponta.

Considerando o Coeficiente de sustentação constante em relação à velocidade angular, tem-se:

$$df_s = \frac{C_s \rho r^2 c}{2} \omega^2\, dr$$

$$f_s = \int_0^{r_d} \left(\frac{C_s \rho c\, r^2}{2} \omega^2 \right) dr$$

$$f_s = \rho\, \omega^2 \int_0^{r_d} \left(\frac{C_s c\, r^2}{2} \right) dr$$

$$f_s = K\, \omega^2$$

Assim, verifica-se ainda que a força de empuxo é proporcional à velocidade de rotação da hélice ao quadrado.

2.2 VANTS

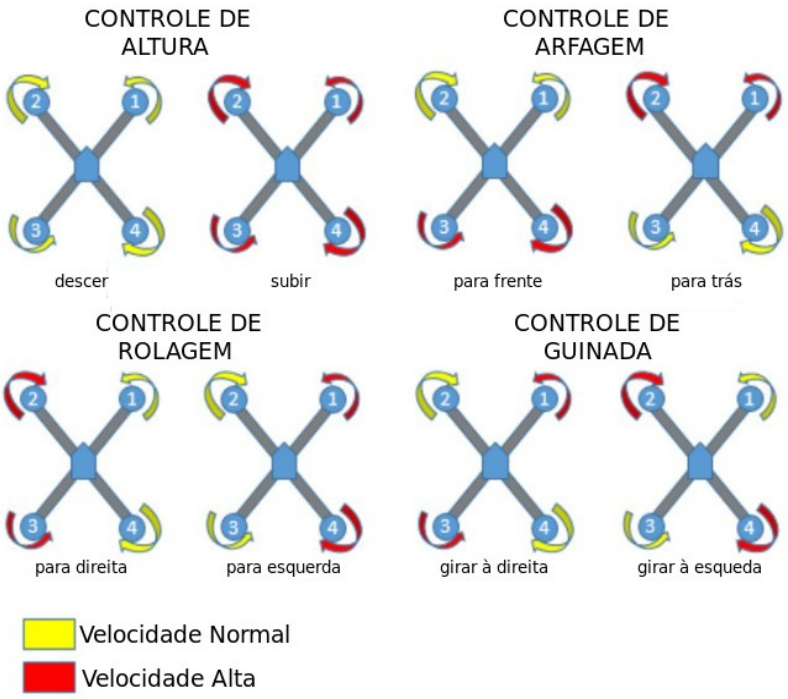
Os VANTs, como não possuem tripulação, são operados remotamente ou através de controle, geralmente computacional. Os VANTs operados por controle computacional são então considerados robôs móveis aéreos.

Os VANTs possuem grande diversidade e por isso podem ser classificados de diversas formas. Uma classificação interessante é sobre a forma de decolagem. Existem o veículos que decolam horizontalmente (CTOL, *Conventional Take-Off and Landing*, ou Decolagem e Aterrissagem Convencional e STOL, *Short Take-Off and Landing*, ou Decolagem e Aterrissagem Curta) e verticalmente (VTOL, *Vertical Take-Off and Landing*, ou Decolagem e Aterrissagem Vertical). Os VANTs também podem ser classificados em veículos de asas fixas e veículos de asas rotativas.

2.2.1 QUADRIRROTORES

Os quadrirrotores são VANTs do tipo VTOL com asas rotativas que possuem 4 rotores que giram em sentidos determinados que possibilitam ao veículo planar e se movimentar. A Figura 5, adaptada de (TUTA NAVAJAS; ROA PRADA, 2014), mostra o funcionamento básico de um quadrirrotor. Pela forma de se movimentar do quadrirrotor, pode-se perceber uma restrição deste tipo de VANT: quando ele se desloca para frente, para trás ou lateralmente, precisa se inclinar na direção do movimento. Isto ocorre porque o quadrirrotor é um sistema mecânico subatuado, ou seja, um sistema no qual a dimensão do espaço de configurações excede o espaço das entradas de controle. Como possui apenas quatro entradas de controle (a velocidade de cada uma das suas quatro hélices) e seis dimensões de configuração (as posições no espaço (x,y,z) e os ângulos de arfagem, rolagem e guinada), ele não consegue controlar independentemente todas as características de sua pose no espaço. Assim, com quatro entradas de controle, os ângulos de arfagem e rolagem estão atrelados ao movimento nos eixos x e y, ficando apenas o ângulo de guinada e o controle de altura (z) independentes.

Figura 5: Funcionamento do quadrirrotor



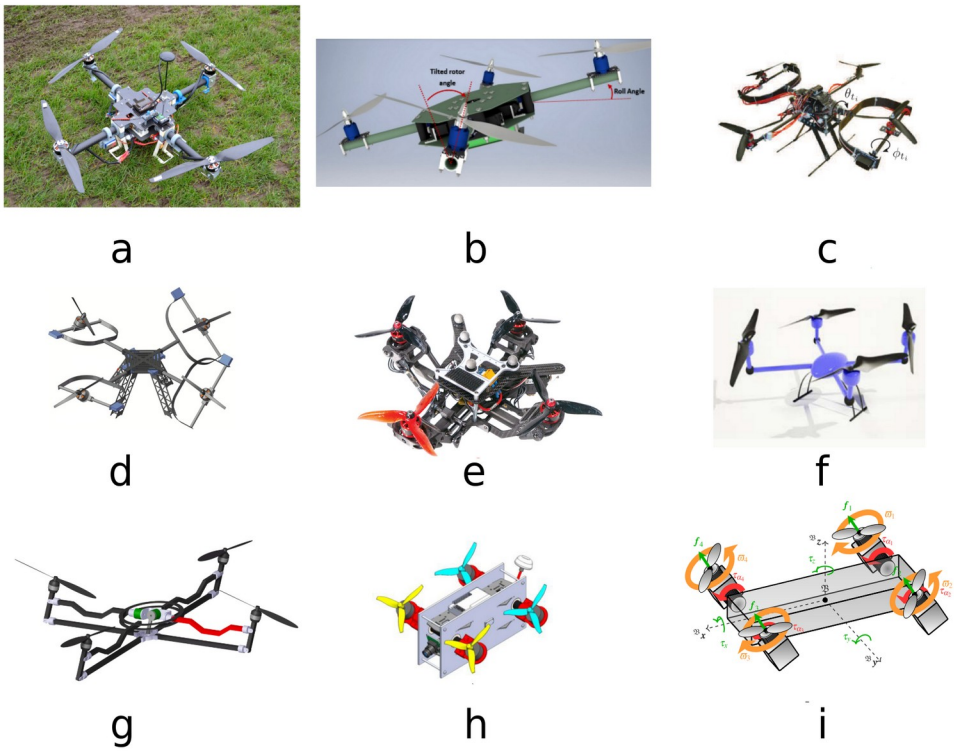
Fonte: Adaptada de (TUTA NAVAJAS; ROA PRADA, 2014)

2.2.2 TILT-QUADRIRROTORES

Os tilt-quadrirrotores ou quad-tilt-rotores permitem inclinar os propulsores, mudando o ângulo da força de empuxo gerada pelas hélices e contornando algumas restrições de movimentação do quadrirrotor, diminuindo a subatuação. Quais restrições serão contornadas pode depender da configuração adotada na construção do tilt-quadrirrotor.

A Figura 6 mostra diversos tipos de tilt-quadrirrotores encontrados na literatura. Na Figura 6(a) há um tilt-quadrirrotor, retirado de (JUNAID et al., 2018), que possui *tilt* nos quatro propulsores, o que permite rotacionar os braços da sua estrutura em formato de "+", assim como acontece em (OOSEDO et al., 2016). Na Figura 6(b) há um tilt-quadrirrotor, retirado de (NEMATI et al., 2016), que possui *tilt* nos quatro propulsores, no entanto utiliza apenas 2 *tilts* de cada vez, colocando a hélice dianteira em um nível diferente da traseira para alcançar maiores velocidades. Outros tilt-quadrirrotores possuem *tilt* apenas em dois propulsores, como é o caso de (MOUTINHO; MATEOS; CUNHA, 2015), apresentado na Figura 6(c). Este tilt-quadrirrotor, no entanto, pode orientar seus dois propulsores em dois eixos. Em (JI; MA; SAM GE, 2020), na Figura 6(d), (ZHENG et al., 2020), na Figura 6(e) e em (SENKUL; ALTUG, 2014), na Figura 6(f), os quatro propulsores possuem *tilts* com orientação em dois eixos. O tilt-quadrirrotor de (ODELGA; STEGAGNO; BULTHOFF, 2016) da Figura 6(g) pode orientar simultaneamente, através de um mecanismo, todos os propulsores em dois eixos. Outros tilt-quadrirrotores, como o (ALKAMACHI; ERCELEBI,

Figura 6: Tilt-quadrirrotores encontrados na literatura



Fonte: (a) (JUNAID et al., 2018), (b) (NEMATI et al., 2016), (c) (MOUTINHO; MATEOS; CUNHA, 2015), (d) (JI; MA; SAM GE, 2020), (e) (ZHENG et al., 2020), (f) (SENKUL; ALTUG, 2014), (g) (ODELGA; STEGAGNO; BULTHOFF, 2016), (h) (ALKAMACHI; ERCELEBI, 2017), (i) (VENDRICHOSKI, Julio C. et al., 2020)

2017) da Figura 6(h) consideraram orientar os *tilts* ao mesmo tempo em um eixo apenas, para imprimir maiores velocidades em uma direção. O tilt-quadrirrotor do grupo de robótica da UFSC (COSTA, 2019; COSTA et al., 2020; VENDRICHOSKI, Julio C. et al., 2020; VENDRICHOSKI, Julio Cezar, 2017), com seu modelo 3d aproximado apresentado na Figura 6(i), também orienta seus *tilts* em apenas um eixo, porém de forma independente.

No tilt-quadrirrotor que inclina suas hélices para frente e para trás, este movimento, além de permitir que se desloque nestes sentidos sem se inclinar, permite que ele exerça uma força maior no plano horizontal, para frente e para trás. Assim, esta aeronave alcança maiores velocidades sem a necessidade de se inclinar, motivo pelo qual existem diversos quadrirrotores de competição que utilizam esta técnica, os chamados de *drones* de corrida. Estes *drones* de corrida geralmente são operados por controle remoto com óculos de realidade virtual, chamados FPV (*first person viewer*), sendo extremamente desejável aumentar a velocidade sem inclinar o veículo pois permite a utilização de câmera fixa ao corpo sem *gimbal*[1]. Outra aplicação interessante deste tipo de veículo é o caso dos manipuladores aéreos, em que um braço robótico é acoplado a um VANT para realizar tarefas. Com um tilt-

1 O gimbal é um acessório para câmeras que pode ser utilizado em VANTs com o objetivo de estabilizar a imagem das câmeras acopladas a eles.

quadrirrotor, a força aplicada para frente e para trás é maior, possibilitando realizar tarefas que exijam mais força, como por exemplo a limpeza de vidraças em arranha-céus.

Também, ao se alterar o ângulo dos *tilts* de ambos os lados dos tilt-quadrirrotor em direções opostas, espera-se que o mesmo gire.

Todos os tilt-quadrirrotores possuem em comum a característica de alterar a orientação relativa entre seus propulsores e o corpo principal em voo. Esta característica torna importante a movimentação das partes do robô pois isso modifica a dinâmica das forças que agem sobre o robô, o que pode desestabilizar o voo. Estas interações entre as partes devem ser levadas em consideração no projeto do VANT.

O projeto de um VANT é uma tarefa complexa que depende de pesquisas em diferentes áreas: passando por desenvolvimento de modelos matemáticos, testes de simulação numérica, desenvolvimento de sistemas de controle, sistemas de computação, sistemas de sensores e atuadores. A utilização de simulações 3D pode alavancar este desenvolvimento, gerando uma camada visual na simulação enquanto evita os altos custos de prototipagem e prejuízos com testes malsucedidos.

2.3 A SIMULAÇÃO 3D

No processo de desenvolvimento de VANTs, a simulação é importante para testar o comportamento do robô, os componentes de software e algoritmos de controle em diferentes ambientes. Diversos caminhos podem ser escolhidos para realizar uma simulação.

Um caminho para simular voos de VANTs é a utilização de simulações numéricas. As simulações numéricas encontram soluções de equações diferenciais, sendo depois a validação do modelo geralmente feita através da construção de protótipos reais. Ao utilizar apenas simulações numéricas, partindo diretamente para o modelo real, é bastante difícil colocar todas as questões do ambiente no modelo matemático, o que pode acabar gerando erros e consequentes prejuízos. O objetivo de simular é chegar no protótipo físico, mas em vez de passar diretamente da simulação numérica para o mundo real, pode-se dar um passo a mais, o que traria vantagens, permitindo fazer testes de forma mais segura e econômica, ajudando a descobrir situações que não seriam previstas. A simulação 3D se apresenta como um apoio no meio do caminho, permitindo complementar os resultados iniciais obtidos com simulações numéricas, encurtando o caminho para a construção de protótipo físico funcional.

Vários ambientes de simulação estão disponíveis para desenvolvimento de simulações 3D em robótica. O Gazebo se destaca por ser um ambiente de simulação 3D de código aberto que possui uma grande comunidade de suporte e suporte nativo ao ROS (sigla para *Robot Operational System*, Sistema Operacional de Robôs).

2.3.1 O AMBIENTE ROS GAZEBO

O Gazebo é uma plataforma de simulação física em 3D. O Gazebo se comunica nativamente com ROS, sendo possível desenvolver um robô no ROS e simular seu funcionamento em 3D no Gazebo. O ROS, apesar de o nome significar sistema operacional de robôs, não é exatamente um sistema operacional: trata-se de um *framework* para

desenvolvimento de robôs. *Frameworks* são conjuntos de softwares que padronizam e agilizam o desenvolvimento de outro software, promovendo o reúso de código. Ao utilizar um *framework*, não é necessário escrever todo o código necessário para o funcionamento do software desenvolvido pois o *framework* providencia as partes fundamentais e repetitivas do código.

A partir do ROS foi desenvolvida uma nova versão, o ROS 2. A versão original costuma ser referenciada simplesmente como ROS, mas pode ser referenciada como ROS 1 para diferenciação. O ROS 2 possui um certo grau de semelhança ao ROS 1, mas as versões não são completamente compatíveis e há necessidade de alteração de código para portar de uma versão para outra. Por este motivo, a versão original ainda possui maior suporte, sendo escolhida para ser utilizada neste trabalho.

O ROS funciona como uma rede de comunicação, em que os diversos nós se comunicam passando mensagens. Assim, cada atuador em um robô pode ser um nó, assim como os elos, juntas e sensores. Os atuadores recebem mensagens para se movimentar e enviam mensagens sobre sua posição. Os elos, juntas e sensores podem enviar mensagens sobre a posição do robô, dentre outros dados. *Scripts* podem ser gerados em C e *Python* para lidar com essas mensagens e fazer o controle do robô.

As mensagens que os nós utilizam para se comunicar no ROS são organizadas em tópicos. Estes tópicos permitem comunicação entre os nós através de um sistema de publicação/inscrição. Assim, um sensor pode publicar seus dados em um determinado tópico (o nome do tópico é escolhido na configuração do sensor) e um *script*, ao escutar este tópico pode executar algum processamento e publicar em outro tópico que movimenta um atuador do robô.

Para simular e desenvolver um robô no ROS, é necessário definir o robô, escrevendo um arquivo URDF (*Universal Robot Description Format*), que é basicamente um arquivo de texto no formato XML que contém a descrição dos elos e juntas do robô, com seu formato e posição inicial. Outras características podem ser adicionadas ao URDF, como por exemplo a cor dos elos.

O ROS funciona sobre a plataforma Linux em computadores PC e também sobre o hardware de robôs reais. A estrutura do ROS é definida de maneira que, ao se escrever em um computador o URDF que define um robô e os *scripts* que o controlam, estes podem ser posteriormente transportados para o hardware real. Assim, uma vantagem de utilizar o ROS é poder escrever um código em simulação e transportar praticamente sem alterações para um robô físico compatível com o ROS. Para desenvolver uma simulação e portar para o robô, inicialmente é criada uma pasta no computador Linux para conter o projeto da simulação (chamado de pacote). Esta pasta conterá os diversos arquivos da simulação em subpastas que geralmente estão organizadas de acordo com a seguinte convenção:

- Arquivos que representam o robô (O Gazebo utiliza o arquivo SDF, *simulation definition file*, no formato XML), geralmente em uma pasta chamada URDF (*Universal Robot Description File*). URDF é o formato aceito pelo ROS, mas se torna um SDF ao adicionarmos o tensor de inércia de cada elo do robô ao arquivo XML;

- Arquivos 3D no formato Collada ou STL que são referenciados no arquivo SDF, dentro de uma pasta para os modelos, geralmente chamada *meshes*;
- Arquivos que representam o mundo onde ocorre a simulação, com os prédios, objetos, etc. (arquivo WORLD), em uma pasta world,
- Arquivos de configuração (no formato YAML), em uma pasta chamada config;
- *Scripts* que controlam o robô (em Python ou c++) e código de *plugins* (também em Python ou c++) , em uma pasta chamada src;
- *Scripts* para chamar a simulação (arquivos LAUNCH, em XML), em uma pasta chamada *launch*.

Os arquivos de *script* da pasta src podem, então, ser portados para o robô real, desde que este suporte ROS.

Para o caso da simulação computadorizada, os arquivos launch chamam o Gazebo, passando como parâmetro o world e mandando colocar o robô em algum lugar do mundo. O arquivo de definição do robô, sdf, indica as partes do robô (elos), cada uma com seu referencial em relação ao referencial pai, bem como qual é o arquivo de *mesh* que o representa e a junta que o liga ao elo pai. O robô pode possuir *scripts*, ligados a suas juntas e elos, que permitem que a simulação se comunique com o ROS, publicando nos tópicos e escutando mensagens, ou implemente alguma funcionalidade que o Gazebo não possui nativamente. Os arquivos de configuração yaml podem ser utilizados por scripts para seu funcionamento. Os *scripts* que controlam o robô podem ser chamados pelo *launch* ou por linha de comando, durante a simulação.

Ao escrever as definições de um robô no formato URDF, é possível visualizá-lo no RViz, a ferramenta de visualização 3D do ROS. O RViz mostra em 3D o arquivo URDF que configura o robô, com seus elos e juntas, bem como as posições relativas destas partes. Este visualizador pode mostrar também os dados coletados por um sensor, como por exemplo um LIDAR. Neste caso, aparece no RViz a nuvem de pontos gerada pelo sensor. Mas o RViz não simula o robô, apenas o mostra em 3D. Para que ele possa interagir com o ambiente e também sofrer as suas interações é necessário colocá-lo no ambiente de simulação Gazebo.

É interessante notar que as interações que ocorrerem no Gazebo podem modificar, por exemplo, a posição do robô, o que também será refletido no RViz, devido à forte integração entre o ROS e o Gazebo. No entanto, o RViz e o Gazebo são distintos, sendo o RViz um visualizador que mostra apenas os dados que o ROS tem acesso, enquanto o Gazebo é um simulador 3D de um ambiente que tenta representar a realidade.

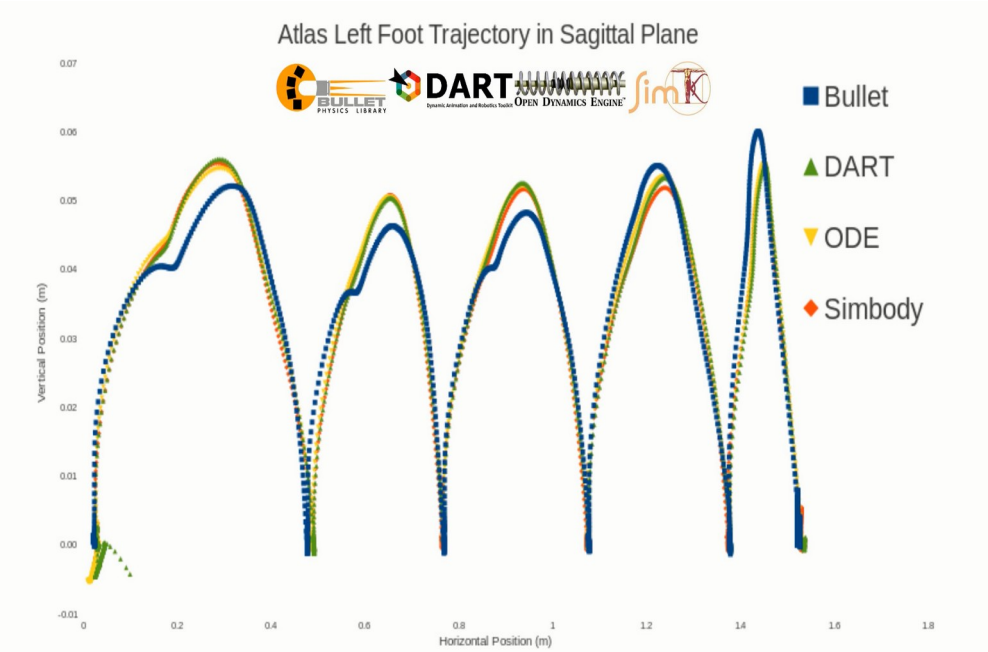
Ao colocar no RViz, por exemplo, um robô móvel equipado com um LIDAR no seu topo, será visualizado o robô e a nuvem de pontos que ele enxerga. Se o Gazebo não estiver sendo executado, não haverá nuvem de pontos nenhuma, já que apenas o robô está definido. Ao se rodar a simulação no Gazebo, havendo outros objetos no mundo simulado, o laser do LIDAR irá tocar nestes objetos. Assim, são representados no Gazebo: o robô, o laser e os objetos. No RViz aparecerá apenas o robô e a nuvem de pontos, que se moldará ao formato dos objetos, mas não haverá nenhum objeto. Isto porque o RViz apenas mostra os dados que o ROS tem conhecimento, ou seja, a nuvem de pontos, que é a interação entre o laser e os objetos. Os objetos estão no mundo da simulação (ou no mundo real, se o ROS estiver sendo executado em um hardware de um robô).

O Gazebo necessita de mais informações do que apenas o formato dos elos e juntas para poder simular o ambiente real de um robô. O desenvolvimento deste robô no Gazebo é feito através de um arquivo similar ao URDF, mas com algumas modificações que adicionam características importantes como a inércia dos elos. Este arquivo é chamado de SDF (*Simulation Description Format*)(OPEN SOURCE ROBOTICS FOUNDATION, 2014b).

O Gazebo também utiliza outro XML para descrever o mundo em que o robô está inserido, com todos os objetos que o robô pode interagir. Este arquivo que descreve o mundo real possui a extensão ".world", sendo citado como arquivo *world* ou simplesmente mundo. Neste mundo são aplicadas as leis da física através de softwares que calculam as relações entre os objetos e as forças envolvidas. Estes softwares são chamados de motores de física, ou *physics engines*.

O Gazebo suporta atualmente quatro tipos de motores de física: Open Dynamics Engine (ODE), Bullet, Dynamic Animation and Robotics Toolkit (DART) da Georgia Tech e Simbody da Stanford University. Cada uma possui qualidades diferentes, sendo que ODE e Bullet se prestam mais a simular veículos enquanto DART e Simbody são mais voltadas para simulação de braços mecânicos complicados. O motor de física padrão do Gazebo é o ODE, mas é possível modificar o motor de física utilizado por comandos dentro dos *scripts*, por configuração no arquivo world ou pela linha de comando que chama o Gazebo, podendo necessitar de instalação adicional. O vídeo em (OPEN ROBOTICS, [s. d.]) mostra a utilização dos quatro tipos diferentes de *physics engine* em um robô atlas, o que acarreta em uma pequena diferença na posição dos pés do robô durante a simulação, conforme na Figura 7, que mostra a trajetória dos pés utilizando os quatro diferentes mecanismos físicos suportados pelo Gazebo

Figura 7: Trajetórias dos pés do robô Atlas



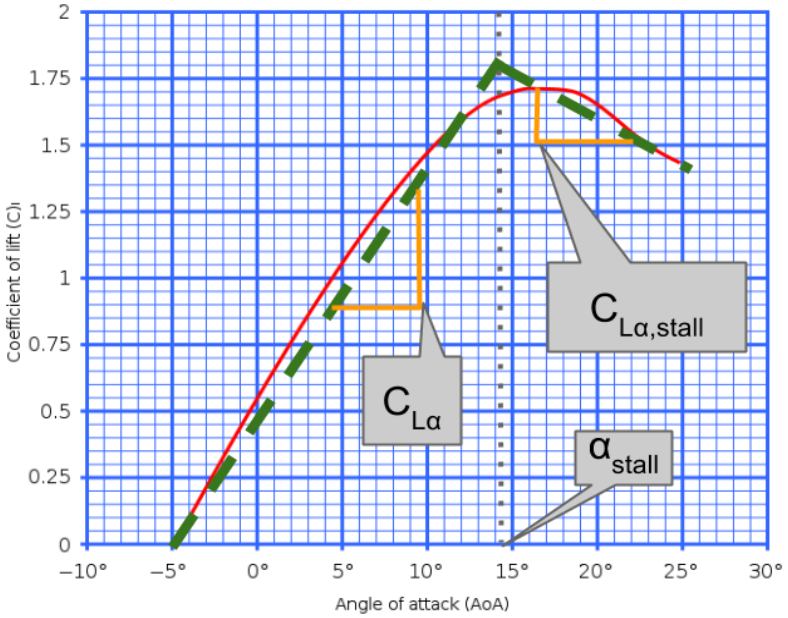
Fonte: (OPEN ROBOTICS, [s. d.])

Apesar de termos esta flexibilidade de escolha de *physics engines* no Gazebo, nenhuma delas simula a força de empuxo gerada pelas hélices em rotação sem a ajuda de um *plugin* externo (MEYER et al., 2012). Assim é necessário adicionar esta força, através de *plugins*.

2.3.1.1 O PLUGIN LIFT AND DRAG

O gazebo possui um *plugin* chamado *Lift and Drag* (OPEN SOURCE ROBOTICS FOUNDATION, 2014a) que, como o nome diz, gera as forças de empuxo (*Lift*) e de arrasto (*Drag*) de um corpo. O *plugin* utiliza os ângulos de ataque da asa para calcular o coeficiente de empuxo, fazendo aproximação por uma reta, conforme mostrado na Figura 8. Assim, ele consegue calcular a força de empuxo, utilizando vários parâmetros de entrada. São envolvidos no cálculo os ângulos de subida e descida da curva e a área da superfície.

Figura 8: Coeficiente de empuxo vs ângulo de ataque



Fonte: (OPEN SOURCE ROBOTICS FOUNDATION, 2014a)

Os parâmetros que o *plugin Lift and Drag* utiliza são:

- link_name: nome do elo que será afetado pelo grupo de propriedades de arrasto e empuxo;
- air_density: Densidade do fluido em que o modelo está suspenso;
- area: área de superfície do elo;
- a0: O α inicial ou ângulo de ataque inicial. a0 também é a interceptação em y da curva do coeficiente de elevação alfa;
- cla: A razão do coeficiente de sustentação (empuxo) e inclinação α antes do estol. Inclinação da primeira parte da curva do coeficiente de elevação alfa;
- cda: A razão do coeficiente de arrasto e inclinação alfa antes do estol;
- cp: Centro de pressão. As forças devido a sustentação e arrasto serão aplicadas aqui;

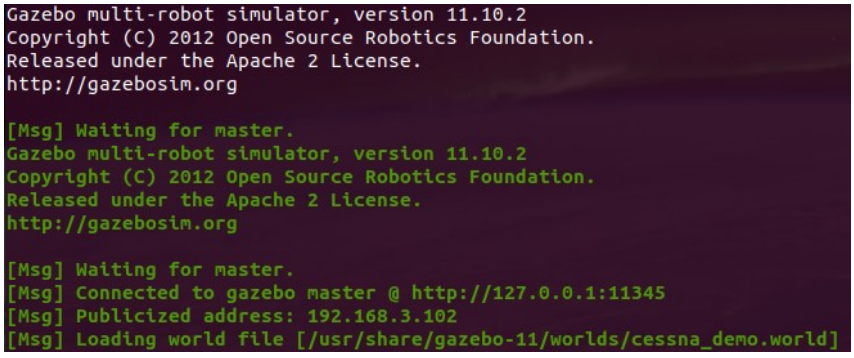
- forward: Vetor de 3 coordenadas (x,y,z) relativo ao eixo do elo, representando a direção do movimento para frente;
- upward: Vetor de 3 coordenadas (x,y,z) relativo ao eixo do elo, representando a direção de sustentação ou arrasto;
- alpha_stall: Ângulo de ataque no ponto de estol; o ângulo máximo de ataque;
- cla_stall: A razão do coeficiente de sustentação e inclinação alfa após estol. Inclinação da segunda parte da curva do coeficiente de elevação alfa;
- cda_stall: A razão entre o coeficiente de arrasto e a inclinação alfa após o estol.

O pacote do gazebo possui um exemplo de uso do *plugin Lift and Drag*, que simula as forças de arrasto e empuxo, tanto nas asas fixas quanto nas hélices de um Cessna (FOUNDATION, 2014). Esta simulação pode ser acessada com o comando:

```
gazebo --verbose worlds/cessna_demo.world
```

A saída do comando que carrega a simulação do cessna aparece na Figura 9, onde é possível ver que a pasta do arquivo *world* encontra-se em usr/share/gazebo-11/worlds/.

Figura 9: Carregando a simulação do cessna



Fonte: Do autor

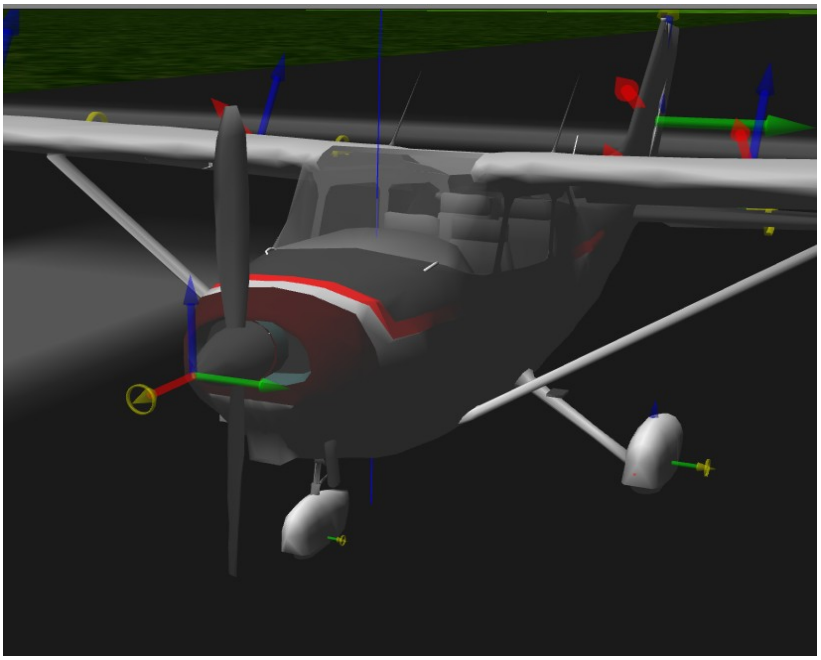
O nome do arquivo *world* é cessna_demo.world. Neste código são utilizadas várias instâncias do *plugin Lift and Drag*: uma para cada asa (direita e esquerda), uma para o profundor (*elevator*), uma para o estabilizador vertical (*rudder*) e duas para a hélice, sendo uma para a lâmina (ou pá) superior da hélice (propeller_top_blade) e outra para a lâmina inferior (propeller_bottom_blade). Pode-se ver no seguinte trecho do XML a utilização do *plugin* para as pás, com os seus parâmetros:

```
<plugin name="propeller_top_blade" filename="libLiftDragPlugin.so">
    <a0>0.4</a0>
    <cla>4.752798721</cla>
    <cda>0.6417112299</cda>
    <cma>0</cma>
    <alpha_stall>1.5</alpha_stall>
    <cla_stall>-3.85</cla_stall>
    <cda_stall>-0.9233984055</cda_stall>
    <cma_stall>0</cma_stall>
    <cp>-0.37 0 0.77</cp>
    <area>0.1</area>
```

```
        <air_density>1.2041</air_density>
        <forward>0 -1 0</forward>
        <upward>1 0 0</upward>
        <link_name>cessna_c172::propeller</link_name>
    </plugin>
    <plugin name="propeller_bottom_blade" filename="libLiftDragPlugin.so">
        <a0>0.4</a0>
        <cla>4.752798721</cla>
        <cda>0.6417112299</cda>
        <cma>0</cma>
        <alpha_stall>1.5</alpha_stall>
        <cla_stall>-3.85</cla_stall>
        <cda_stall>-0.9233984055</cda_stall>
        <cma_stall>0</cma_stall>
        <cp>-0.37 0 -0.77</cp>
        <area>0.1</area>
        <air_density>1.2041</air_density>
        <forward>0 1 0</forward>
        <upward>1 0 0</upward>
        <link_name>cessna_c172::propeller</link_name>
    </plugin>
```

Para entender melhor as direções das forças aplicadas na hélice, é necessário ver as direções dos eixos da hélice. A Figura 10 mostra os eixos da hélice do cessna: x em vermelho, y em verde e z, azul.

Figura 10: As direções dos eixos do cessna na simulação



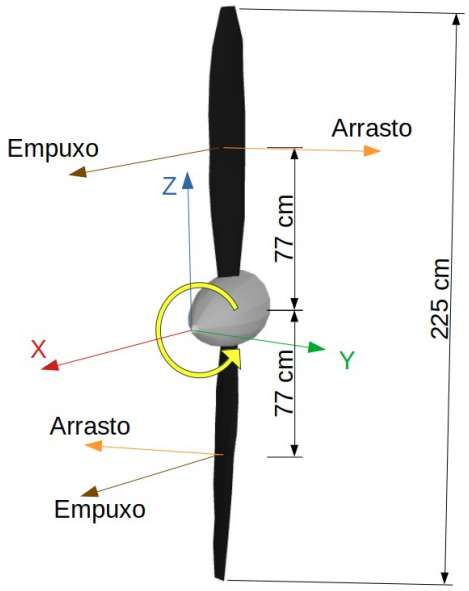
Fonte: Do autor

O parâmetro do XML upward indica um vetor que aponta qual é a direção para cima, ou seja, a direção em que será gerada a força de empuxo. Tanto para a lâmina superior, quanto para a lâmina inferior, este parâmetro deve apontar para frente do avião, ou seja, <upward>1 0 0</upward>, onde os número indicam os valores de x,y,z do vetor de direção.

No caso do vetor que indica a direção de deslocamento para frente, o vetor da lâmina inferior se deslocará na direção contrária do vetor da lâmina superior, visto que a hélice gira. O vetor da lâmina superior será <forward>0 -1 0</forward> (pois o sentido do deslocamento da lâmina superior será contrário ao eixo y, com a hélice girando no sentido horário ao vê-la de dentro da cabine) e o vetor da lâmina inferior será o contrário,<forward>0 1 0</forward>. Assim, a força de arrasto, que se contrapõe ao movimento da hélice, será gerada no sentido do eixo y para a lâmina superior e no sentido contrário no eixo inferior.

O parâmetro cp indica a posição da aplicação da força de empuxo e arrasto em relação ao eixo aplicado ao nariz do avião, que gira junto com o movimento rotatório da hélice. Para a lâmina superior estará em <cp>-0.37 0 0.77</cp> e para a lâmina inferior, em <cp>-0.37 0 -0.77</cp>. O valor negativo de x em ambas lâminas se justifica pois a origem está aplicada no nariz da hélice. Em ambas as lâminas, as forças serão aplicadas a 77 cm do eixo de rotação, no centro de pressão que está, conforme esperado, entre o meio das lâminas e a ponta, como é apresentado na Figura 11.

Figura 11: As forças de empuxo aplicadas na hélice



Fonte: Do autor

Assim, cada lâmina da hélice gerará uma força de empuxo no sentido do deslocamento do cessna e uma força de arrasto contrária ao movimento da hélice. Estas pás da hélice funcionam então como asas rotativas, gerando as forças. Conforme o giro da hélice varia, os módulos da força de empuxo e de arrasto também variam, de acordo com os outros parâmetros.

2.4 OUTROS AMBIENTES DE SIMULAÇÃO

O escopo do presente trabalho está delimitado para a simulação 3D no ambiente ROS Gazebo, no entanto outros ambientes para simulação física em 3D também são utilizados para

aviação e VANTs. A escolha do Gazebo se deu pela sua grande aceitação acadêmica, grande comunidade de suporte, integração nativa com o ROS e por ser um software livre. A Tabela 1 mostra um resumo dos ambientes de simulação mais comuns, que serão apresentados a seguir.

Tabela 1: Comparação entre os ambientes de simulação 3D

Nome	Licença	Integração com ROS	Observações
X-Plane	comercial	não	Desenvolvimento de aeronaves Simples porém limitado
FlightGear	GNU	não	Voo por modelos matemáticos
CoppeliaSim	GPL e comercial	sim	Desenvolvimento de aeronaves complexo
Provant	livre	sim	Escolha limitada de aeronaves Voo por modelos matemáticos
Morse	livre	sim	Pouco suporte Não é fiel à realidade
Gazebo	livre	sim	Grande comunidade Grande aceitação acadêmica

Fonte: Do autor

2.4.1 X-PLANE

O X-plane[2] é um simulador 3D de aeronaves realístico de licença comercial e código fechado. Trata-se de um jogo de simulação que se propõe a simular o uso de aviões da forma mais realística possível, podendo inclusive ser utilizado para treinamento de pilotos. O X-plane possui um site dedicado ao desenvolvimento[3], que permite a criação de aeronaves, cenários ou *plugins* para o programa. A criação de aeronaves é feita através de um programa adicional ao X-plane, chamado Plane Maker[4]. O plane maker é uma interface gráfica em que se pode selecionar um dos modelos já existentes de aviões do x-plane e alterar todos os parâmetros: tamanhos das asas, *flaps*, *aerons*, etc. Também é possível alterar os desenhos da fuselagem e comandos internos, o que permite criar uma aeronave completamente nova. Em (IBRAHIM JENIE et al., 2018) foi criado um quadrirrotor baseado no Parrot Bebop 2, utilizando o Plane Maker. O X-plane possui uma grande comunidade de desenvolvedores, usuários e entusiastas que compartilham informações e modelos de aeronaves através do site x-plane.org. Não foi encontrado um modelo de um quadrirrotor com *tilt*, mas os modelos existentes[5], sugerem que seja possível desenvolver um. A ferramenta não possui integração direta com o ROS, mas a integração pode ser obtida com a ferramenta XplaneROS[6].

2.4.2 FLIGHTGEAR

FlightGear é um simulador 3D de voo de código aberto desenvolvido por voluntários todo o mundo e qualquer pessoa pode contribuir. O código-fonte de todo o projeto está

2 Disponível online em https://www.x-plane.com/
3 Disponível online em https://developer.x-plane.com/
4 Disponível online em https://developer.x-plane.com/manuals/planemaker/
5 Um exemplo dos modelos existentes pode ser encontrado online em https://forums.x-plane.org/index.php?/files/file/78571-joby-s4-project/
6 Disponível online em https://github.com/castacks/xplane_ros

disponível e licenciado sob a GNU General Public License. O projeto FlightGear cria uma estrutura de simulação de voo para uso em ambientes de pesquisa ou acadêmicos. O desenvolvimento de aeronaves no FlightGear é feito através dos FDMs. O FDM é o modelo matemático que controla a física do voo dentro do simulador. A dinâmica de voo é separada do modelo físico em 3D, que apenas gera a apresentação da aeronave. É o FDM que impõe as regras de como o modelo voa. O FlightGear aceita modelos JSBSim e YASim. O FlightGear não possui integração com ROS, mas é possível ser alcançada através do desenvolvimento de um software de integração, como em (BAILON-RUIZ et al., 2017).

2.4.3 COPPELIASIM

Coppeliasim é um software de simulação 3D de código aberto, com licença mista: parte GPL e parte comercial, com uso liberado para instituições educacionais[7]. O CoppeliaSim deriva-se de um projeto antigo chamado V-REP. É possível simular aeronaves no CoppeliaSim, como pode ser visto em um vídeo[8]. O CoppeliaSim possui integração com ROS, permite programação em Java, Python, C++ e MatLab. O CoppeliaSim usa várias bibliotecas de simulação de física para realizar simulação de corpo rígido: Bullet, ODE, Vortex e Newton Game Dynamics.

2.4.4 PROVANT

O Provant é um projeto colaborativo criado em 2012 que envolve a Universidade Federal de Minas Gerais (UFMG) e a Universidade Federal de Santa Catarina (UFSC). O objetivo deste projeto é criar conhecimento relacionado ao projeto e controle de VANTs. Este projeto gerou um produto, o Simulador de código aberto Provant[9], apresentado em (LARA et al., 2017). Este simulador apresenta uma interface gráfica para escolha das aeronaves a serem simuladas e a simulação ocorre no Gazebo. O Provant possui integração com ROS. O objetivo deste simulador é fornecer uma interface que facilite testes de estratégias de controle. O Provant utiliza um modelo matemático para cálculo das resultantes aerodinâmicas na simulação. Em (LARA et al., 2017) é apresentado um link para um vídeo de demonstração do trabalho[10] onde é possível ver que as hélices não giram, sendo os parâmetros da inércia das hélices calculados internamente pelo *plugin* do simulador.

2.4.5 MORSE

MORSE (Modular OpenRobots Simulation Engine) é um simulador robótico acadêmico, de código aberto, baseado no Blender Game Engine e no motor Bullet Physics. Possui integração com o ROS. O Morse é um projeto relativamente novo, possuindo uma comunidade menor e pouca documentação de suporte. A representação física do Morse ainda não é considerada fiel à realidade pelos próprios desenvolvedores, apesar de baseada em Bullet.

---

7 As informações de licença do CppeliaSim estão disponíveis juntamente com as orientações para compilação do Coppeliasim em https://www.coppeliarobotics.com/helpFiles/en/compilingCoppeliaSim.htm

8 Disponível online em https://www.youtube.com/watch?v=YRAI7-MICbY

9 Disponível online em https://github.com/Guiraffo/ProVANT-Simulator

10 Disponível online em https://youtu.be/k7MykhlrkAk

2.4.6 OPENVSP

OpenVSP é uma ferramenta 3D, originalmente desenvolvida pela NASA, que permite modelar a geometria de aeronaves e auxilia na análise aerodinâmica e estrutural dos modelos. OpenVSP não é um simulador 3D, mas sim uma ferramenta muito útil na modelagem das aeronaves.

2.5 CONCLUSÃO SOBRE A NECESSIDADE DO ESTUDO

O ambiente Gazebo é o ambiente *open source* mais aceito pela comunidade científica e possui integração nativa com ROS, no entanto não simula as forças de empuxo nativamente, necessitando de *plugins* adicionais.

O *Lift Drag Plugin* simula a força de empuxo e de arrasto de um corpo se deslocando horizontalmente de forma linear com um ângulo de ataque e áreas de superfície definidos. No entanto, no caso de asas rotatórias, a velocidade linear de cada parte infinitesimal da asa varia de acordo com a distância do eixo de rotação.

Para usar o *plugin Lift and Drag* é necessário então conhecer as curvas do coeficiente de empuxo vs ângulo de ataque de cada uma das pás da hélice. Estes dados são obtidos experimentalmente em túneis de vento, uma ferramenta cara e pouco acessível. Também não está claro como o *plugin* faz os cálculos de asas rotativas, que possuem velocidades diferentes de acordo com a distância do eixo de rotação, o que influencia no cálculo do Cs, visto que a velocidade linear não é constante. O uso deste *plugin* também exige que seja executado um *plugin* para cada pá de hélice, aumentando a complexidade dos cálculos para hélices com mais pás. O aumento de complexidade dos cálculos causa grande impacto na simulação 3D no Gazebo, principalmente para hélices menores, que possuem pequenos valores de inércia e giram em alta velocidade. Este é o caso das asas rotativas dos quadrirrotores e tilt-quadrirrotores.

Como o *plugin Lift and Drag* é de difícil utilização para simular asas rotativas, nos trabalhos que fazem simulações de quadrirrotores não costuma ser utilizado, sendo programado um *plugin* específico para a simulação. Porém, geralmente o quadrirrotor inteiro é simulado como um bloco e as forças de empuxo e os torques são calculados para o robô e aplicados no centro de massa. Fazer isto obriga a manter as hélices fixas pois ao colocar qualquer corpo dentro do Gazebo, há necessidade de se fornecer o tensor de inércia deste corpo e, caso elas se movam, serão gerados torques adicionais aos já calculados no *plugin*. Assim, em muitas simulações as hélices não giram, como em (LARA et al., 2017) sendo inclusive muitas vezes representadas como um disco, como é o caso de (MEYER et al., 2012) , que utiliza o modelo representado na Figura 12.

Figura 12: Quadrirrotor de Meyer et al.

Fonte: (MEYER *et al.*, 2012)

Para toda a simulação há necessidade de se fazer escolhas de simplificação e as escolhas que envolvem a simulação de quadrirrotores não são necessariamente adequadas à simulação de *tilt* quadrirrotores. Utilizar hélices fixas ou substituí-las por discos não é adequado para a simulação de tilt-quadrirrotores. Isto porque o movimento entre as partes dos tilt-quadrirrotores importa na simulação e a mudança da posição dos *tilts* implica em uma mudança na direção entre os torques das hélices. O cálculo das resultantes no robô, além de complexo é específico para cada configuração de aeronave e, ao modificar qualquer parte do robô, o cálculo precisa ser refeito. A utilização de um *plugin* genérico que gere as forças da hélice se torna então mais interessante.

O presente trabalho apresenta um método para simulação do voo de tilt-quadrirrotores no ROS Gazebo, utilizando um *plugin* para gerar as forças de empuxo das hélices em rotação, mas permitindo que o motor de física do Gazebo faça os cálculos de torques e inércia, de forma a facilitar aproximações da simulação à realidade. As forças de empuxo serão aplicadas no eixo de rotação e calculadas a partir de uma função da velocidade de rotação, ao invés dos parâmetros da asa, como é feito pelo *plugin Lift and Drag*. Ao mudar o enfoque dos fatores geradores da força de sustentação para a resultante desta força, a complexidade da simulação e da coleta de dados sobre as hélices diminui, dispensando a utilização de túnel de vento para coletar os parâmetros da hélice. São medidas apenas as forças de empuxo geradas de acordo com o aumento da velocidade, assim como foi feito em (CYPRIANO; IMANISHI, 2014), ou com outros métodos, como apresentado em (BERTOLDO et al., 2018). Além disso, ao utilizar um *plugin* que gera apenas a força de empuxo, deixando torques e inércia da hélice serem calculados pela sua rotação no Gazebo, este *plugin* se torna genérico o suficiente para ser utilizado em diversas configurações de robôs, não sendo necessário um *plugin* diferente para cada robô simulado. Apenas os parâmetros do empuxo gerado pela hélice são passados ao *plugin*, sendo estes parâmetros a única modificação necessária, caso as hélices utilizadas sejam trocadas, o que dispensa a troca ou alteração do *plugin*. Desta forma, este *plugin* poderá inclusive ser utilizado complementarmente ao *plugin Lift and Drag* para simular VANTs de asas fixas com propulsão de hélices.

3 MÉTODO DE SIMULAÇÃO DE VANTS

Neste capítulo serão detalhados os 9 passos sequenciais do método de simulação de VANTs com *tilt* rotores em ROS Gazebo proposto por esta pesquisa, a saber:

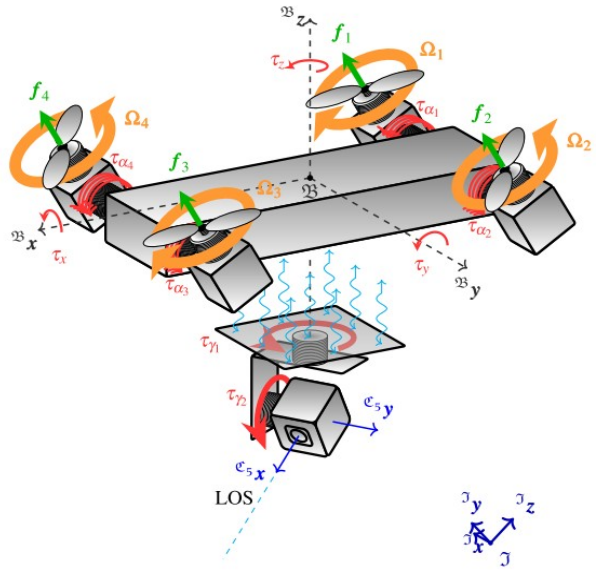
1. Escolher os componentes reais do robô: quais motores, hélices, baterias e placas serão utilizados. Deve ser feito um levantamento das dimensões, pesos e todas as características dos componentes. No caso das hélices, é importante saber o empuxo que é gerado em relação à rotação;
2. Montar o modelo 3D do robô, de forma mais realística possível, em STL, com a representação dos componentes reais que serão utilizados no robô;
3. Dividir o STL em partes com massa suficientemente homogênea, levando em consideração quais serão as partes móveis do robô;
4. Montar o arquivo URDF, colocando os eixos dos STLs nos lugares corretos;
5. Calcular os tensores de inércia de cada parte do robô e transformar o arquivo URDF em SDF;
6. Colocar o SDF no gazebo;
7. Criar as transmissões, interfaces e atuadores para que o robô no Gazebo possa reagir às mensagens;
8. Colocar o *plugin* de empuxo nos atuadores;
9. Escrever o *script* de controle do robô.

3.1 ESCOLHA DOS COMPONENTES REAIS DO ROBÔ

O Grupo de Robótica do Programa de Pós-Graduação em Engenharia de Automação e Sistemas da UFSC desenvolveu o robô *tilt* quadrirrotor utilizado neste trabalho. Este robô foi modelado matematicamente em diversos trabalhos anteriores e há um protótipo sendo construído, mas não havia uma simulação 3D antes do presente trabalho.

Em (VENDRICHOSKI, Julio Cezar, 2017) foram apresentados 2 robôs com *tilt* rotores, sendo um com duas e outro com quatro hélices, conforme na Figura 13. Em (COSTA, 2019) foi desenvolvido um controle robusto para o *tilt* quadrirrotor, ainda sem nome, sendo chamado apenas de VANT-VTOL, devido suas características de decolagem vertical. Em (VENDRICHOSKI, Julio C. et al., 2020) foram apresentadas as vantagens do uso deste mecanismo de *tilt* em relação ao robô convencional, mas o nome do robô quadrirrotor com *tilt* nas quatro hélices não foi mencionado.

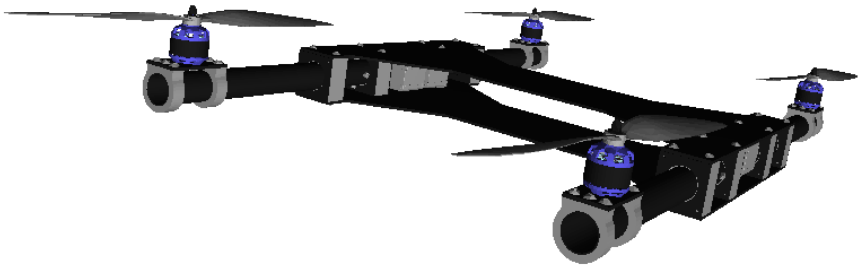
Figura 13: Robô quadrirrotor com 4 hélices com *tilt*



Fonte: (VENDRICHOSKI, Julio Cezar, 2017)

No presente trabalho é utilizada uma versão mais realística deste robô, sem *gimbal*, apresentada na Figura 14, que será batizado de Lophorina. O robô tem como base este arquivo STL cedido por Venrichoski para a realização deste trabalho. O robô é um tilt-quadrirrotor que, além dos propulsores, possui quatro motores adicionais localizados no corpo principal. Estes motores permitem girar em ângulos definidos, independentemente e para ambos os sentidos os propulsores, junto com os braços que os sustentam.

Figura 14: O robô Lophorina



Fonte: Do Autor, a partir de modelo fornecido pelo Grupo de Robótica do Programa de Pós-Graduação em Engenharia de Automação e Sistemas da UFSC

Também foram fornecidas informações estimadas de que o corpo principal do robô possui a massa de 1,5 kg e que cada propulsor possui a massa de 0,15 kg. A massa das hélices foi estimada em 8 g conforme o peso de hélices comerciais encontradas no mercado.

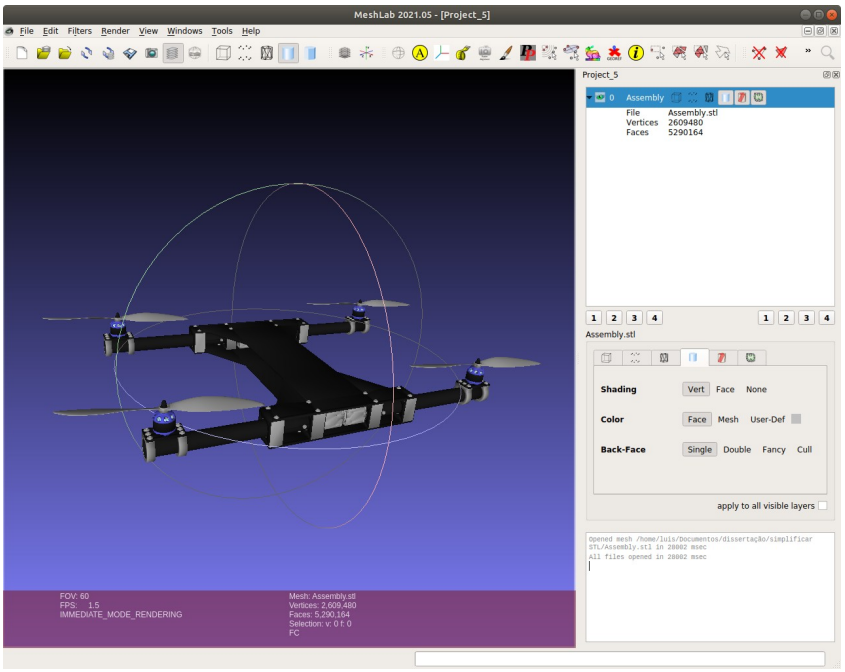
3.2 MONTAGEM DO MODELO 3D DO ROBÔ

Para montar o modelo 3D do robô é necessário dividir o arquivo 3D do robô completo em arquivos menores contendo as partes móveis. Assim, o arquivo STL do corpo completo do robô foi dividido em 9 partes, a saber: o corpo do robô, os quatro motores das hélices com seus respectivos braços que giram em *tilt* e as hélices. Desta forma, com estas partes unidas por juntas rotativas e os tensores de inércia de cada parte, o gazebo pode simular os torques envolvidos.

3.2.1 SIMPLIFICAÇÃO DO STL

Antes de dividir o arquivo, caso ele seja muito grande, é necessário diminuir seu tamanho para diminuir o custo computacional de exibição. Na Figura 15 é exibido o arquivo original aberto no MeshLab[11], que foi simplificado com percentual de redução 0.1 (10%), conforme mostrado na Figura 16. O arquivo STL cedido tinha quase 265 MB, e após esta simplificação ficou com apenas 26 MB.

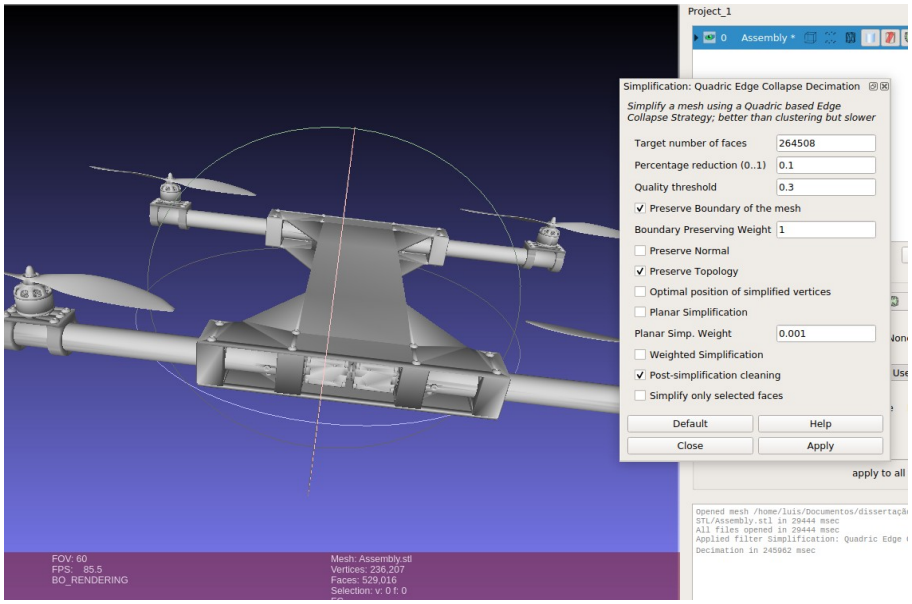
Figura 15: Arquivo original do Lophorina no MeshLab



Fonte: Do Autor

11 O MeshLab é um software de código aberto para processamento de arquivos 3D que fornece um conjunto de ferramentas para editar, limpar, curar, inspecionar, renderizar e converter esses arquivos.

Figura 16: Simplificação do modelo original para 10%

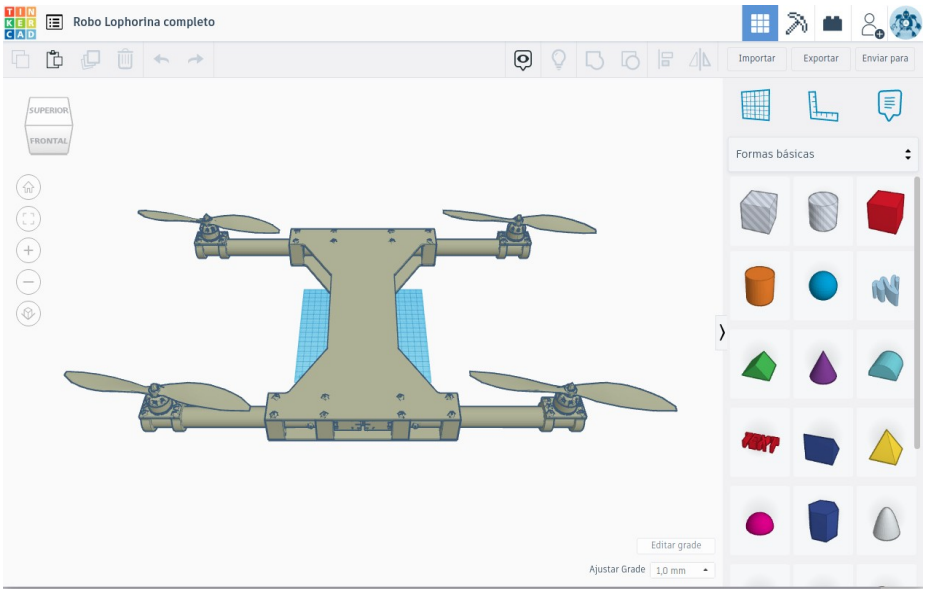


Fonte: Do Autor

3.3 DIVISÃO DO STL EM PARTES

Para dividir o STL pode ser utilizado o programa Tinkercad[12] através da inserção de cubos de corte. O STL do robô lophorina completo aberto no Tinkercad está na Figura 17.

Figura 17: O robô lophorina completo no Tinkercad

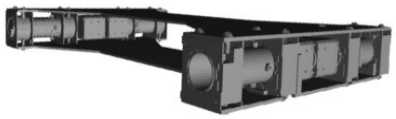


Fonte: Do Autor

12 Tinkercad é um programa de modelagem tridimensional online gratuito, desenvolvido pela empresa AutoDesk, que funciona diretamente no navegador da web, e é conhecido por sua simplicidade e facilidade de uso, sendo amplamente utilizado para gerar modelos para impressão 3D.

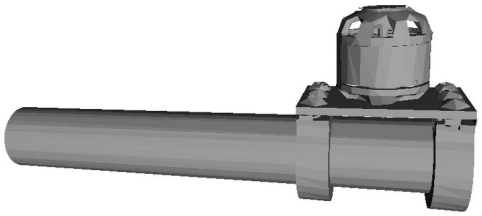
O STL resultante do corpo do robô está na Figura 18, os seus quatro braços com propulsores são representados pelo mesmo STL que está na Figura 19, as hélices que giram no sentido horário são representadas pelo STL da Figura 20 e as hélices anti-horárias pelo STL da Figura 21.

Figura 18: Corpo do robô



Fonte: Do Autor

Figura 19: *B*raços com propulsores



Fonte: Do Autor

Figura 20: Hélice cw

Fonte: Do Autor

Figura 21: Hélice ccw

Fonte: Do Autor

3.4 MONTAGEM DO ARQUIVO URDF

Os STLs das partes móveis do robô formam a parte visual da simulação física em 3D. O arquivo SDF é um arquivo XML que contém a descrição dos elos e juntas do robô, sendo então um arquivo-texto, estruturado em forma de marcadores, ou *tags*. O XML deve sempre começar com uma *tag* que indica que se trata de um XML e contém a versão utilizada:

```
<?xml version="1.0"?>
```

Para descrever um robô, é utilizada a *tag* robot:

```
<robot name="lophorina">
</robot>
```

O robô é definido dentro desta *tag*. Cada elo (*link*) e junta (joint) deve estar dentro desta *tag*. O robô deve estar estruturado como uma árvore, em que cada elo é unido a outro elo superior por uma junta, com exceção do elo raiz, que não possui uma junta. Por ser organizado em forma de árvores, com apenas um elo raiz, o uso de URDF não é considerado bom para descrever robôs paralelos. O elo raiz não terá partes visíveis, nem terá nenhum volume, sendo apenas uma organização lógica.

```
<link name="base_link">
</link>
```

O próximo elo será o corpo do robô.

```
<link name="robot_body">
</link>
```

Entre os elos devem haver juntas, que mantém a estrutura de árvore do arquivo XML. Entre o elo raiz e o corpo do robô então uma junta fixa. Esta junta especificará qual é o elo pai e o filho.

```
<joint name="base_link_to_robot_body" type="fixed">
   <parent link="base_link"/>
   <child link="robot_body"/>
</joint>
```

O elo do corpo do robô, diferentemente do elo raiz, deve ser visível, utilizando o STL do corpo do robô gerado anteriormente. O ROS utiliza a referência aos arquivos do tipo “package://”, que permite que se referencie a arquivos por pacote, mesmo sem saber onde estão na estrutura de pastas do computador. Os STLs utilizados, como foi visto anteriormente, têm as dimensões em milímetros e o Gazebo trabalha no S.I., em metros. Assim, dentro da definição do corpo do robô, deve ser colocada uma escala 1000 vezes menor. A origem é em qual ponto deve ser começado a desenhar a origem do STL do corpo do robô. Ela deve ser especificada para que o Gazebo saiba posicionar a parte visual do elo, visto que a organização de juntas e elos é apenas uma estrutura lógica. A origem é uma especificação de pose do robô, ou seja, posição e orientação. Como, neste caso, não está sendo especificada a posição (xyz), apenas a orientação (rpy), ela será considerada como [x, y, z] = [0, 0, 0].

```
<visual>
  <geometry>
    <mesh filename="package://lophorina/meshes/robot_body.stl" scale="0.001 0.001
0.001"/>
```

```
    </geometry>
    <origin rpy="0 0 1.57079632679"/>
  </visual>
```

Para o ROS calcular as colisões entre os elos do robô, ele não pode simplesmente utilizar o arquivo *mesh* (STL) fornecido pois este arquivo é muito complexo, o que geraria um processamento muito grande. Assim, definimos a área de colisão como uma figura geométrica mais simples. No caso do robô móvel, os movimentos dos elos não permitem que haja colisão entre as partes do robô. Então é possível utilizar um paralelogramo único para colisão de todo robô com o ambiente. Este paralelogramo deve ter um tamanho que possa conter todo o STL do robô. Assim, o corpo do robô possuirá esta área de colisão e nenhum outro elo terá o *tag* de colisão.

```
  <collision>
    <geometry>
      <box size="0.5 0.8 0.1"/>
    </geometry>
  </collision>
```

O próximo elo na construção do robô será o braço com o propulsor frontal da direita. Ele possuirá apenas a parte visual, visto que não fará colisões.

```
  <link name="motor_front_right">
    <visual>
      <geometry>
        <mesh filename="package://lophorina/meshes/tilt_motor.stl" scale="0.001 0.001 0.001"/>
      </geometry>
      <origin rpy="0 0 1.57079632679" xyz="0 0 0"/>
    </visual>
  </link>
```

A junta que ligará este braço ao corpo do robô será de revolução, então é necessário especificar a velocidade desta junta, bem como o esforço que ela suporta e o ângulo máximo e mínimo, em radianos. Também é especificado o vetor do eixo de rotação desta junta.

```
  <joint name="robot_body_to_motor_front_right" type="revolute">
    <limit effort="1000.0" velocity="0.5" lower="-0.78539816339" upper="0.78539816339"/>
    <parent link="robot_body"/>
    <child link="motor_front_right"/>
    <axis xyz="0 1 0"/>
    <origin xyz="0.1825 0.126 0.025"/>
  </joint>
```

O elo da hélice do braço dianteiro direito também não terá colisão:

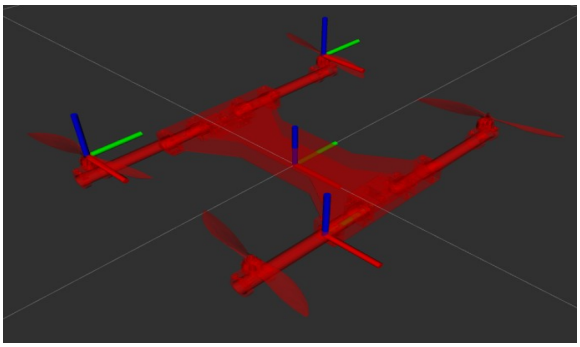
```
  <link name="front_right_propeller">
    <visual>
      <geometry>
        <mesh filename="package://lophorina/meshes/cw_propeller.stl" scale="0.001 0.001 0.001"/>
      </geometry>
      <origin xyz="0 0 0"/>
    </visual>
  </link>
```

A junta da hélice é do tipo contínua, ou seja, não tem um ângulo máximo e mínio, apenas gira livremente para um sentido ou outro.

```
<joint name="motor_front_right_to_front_right_propeller" type="continuous">
  <parent link="motor_front_right"/>
  <child link="front_right_propeller"/>
  <axis xyz="0 0 1"/>
  <origin xyz="0 0.138 0.05"/>
</joint>
```

Ao continuar este processo para os outros braços do robô, obtém-se o arquivo URDF completo do robô, que apresenta sua visualização no RViz[13] na Figura 22, evidenciando a posição das origens de algumas peças, através da apresentação dos eixos.

Figura 22: Visualização do arquivo URDF no RViz



Fonte: Do Autor

3.5 CÁLCULO DOS TENSORES DE INÉRCIA

O Gazebo necessita do tensor de inércia das partes do robô para poder fazer a simulação física. O cálculo pode ser feito pelo MeshLab, que gera um tensor de inércia a partir do arquivo STL. O tensor de inércia é uma função linear que traz a generalização do momento de inércia de um corpo com rotação em relação aos três eixos, passando pela origem definida no STL. Os termos do tensor de inércia dependem da massa e da distância ao eixo de rotação, elevada ao quadrado. Como a massa não é fornecida no MeshLab para cálculo da inércia, o programa considera que o arquivo *mesh* possui densidade 1 e utiliza o volume como massa:

$$d = \frac{m}{V}$$

$$m = d \, . \, V$$

como d = 1:

$$m = V$$

13 RViz (sigla para ROS *Visualization*) é uma ferramenta de visualização 3D que acompanha o ROS e permite mostrar graficamente os robôs, sensores e objetos da simulação que o ROS tem acesso.

Os fatores do tensor de inércia são funções da massa m, assim para obter o tensor de inércia do objeto, é necessário multiplicar o valor da massa real do objeto pelo resultado do tensor obtido no MeshLab e dividir pelo valor do volume que foi calculado.

Para calcular o tensor de inércia, o MeshLab considera que a peça representada pelo STL possui massa uniforme e coloca os eixos de rotação passando pelo centro de massa do objeto. Assim, não é possível colocar todo o STL de um robô no MeshLab e calcular a inércia diretamente. É necessário dividi-lo nas partes móveis e em partes de materiais suficientemente uniformes e calcular separadamente a inércia. Para o robô Lophorina, as partes móveis foram consideradas como feitas todas de material de mesma densidade, no entanto caso houvesse alguma parte de densidade muito diferente, seria necessário dividir esta parte, calcular a inércia e adicionar separadamente ao modelo com juntas fixas.

Para calcular o tensor de inércia, o MeshLab necessita calcular o volume do objeto representado no STL. Arquivos 3D são formados por malhas (*meshes*), que são uma junção de faces triangulares. Para calcular o volume é necessário que o *mesh* seja fechado, sem nenhuma abertura entre as faces, de forma a não ter nenhum "vazamento de volume", o que é chamado de "*watertight*". Existem softwares para corrigir essas falhas através de um processo automatizado chamado de curar o objeto. No entanto, ao encontrar uma abertura, o software pode errar este cálculo e juntar uma face com a face errada. Por isso a correção só funciona se as aberturas entre as faces forem suficientemente pequenas. Além disso, quanto maior o STL, mais complexa a solução dos vazamentos e maior a possibilidade de erros no cálculo.

Então existem duas formas de abordar o problema de vazamento de volume: simplificar os STLs para depois consertar as aberturas com um software ou então fazer um STL que utiliza formas geométricas simples que representem as partes do robô para fins de cálculo da inércia.

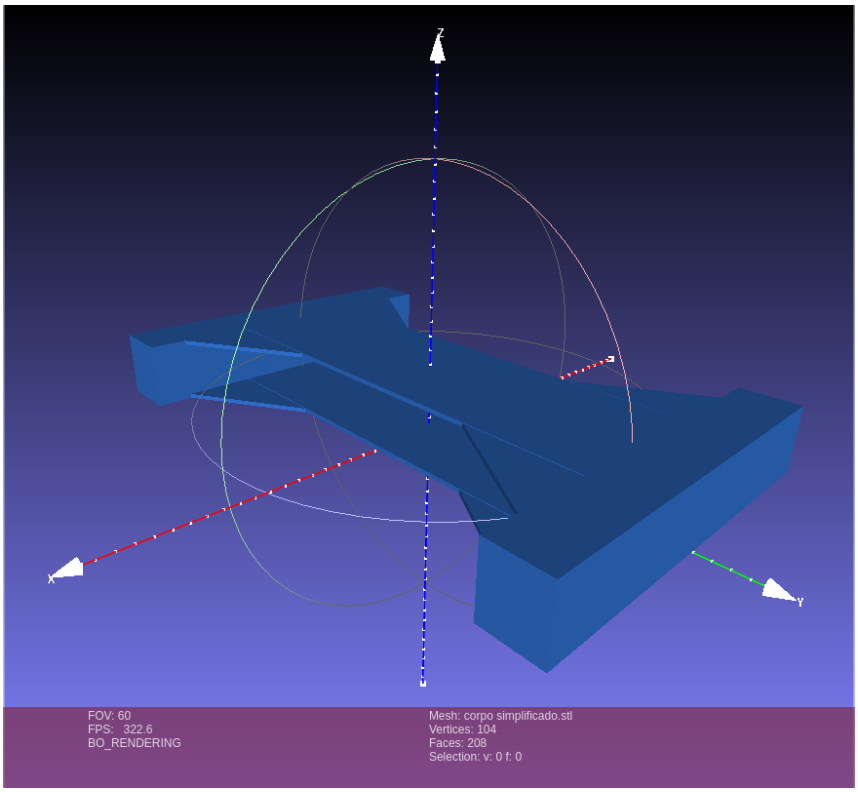
Para resolver o problema de vazamento de volume no Lophorina, foram utilizadas as duas formas. As hélices, por serem pequenas, após a simplificação do STL, foram curadas no Meshlab. O corpo do robô e o braço, por serem arquivos maiores e mais complexos, necessitaram da criação, no Tinkercad, de versões simplificadas para fins de cálculo dos tensores de inércia. A Figura 23 mostra o corpo do robô simplificado para cálculo do tensor de inércia.

O resultado do MeshLab para o cálculo do tensor de inércia do corpo do robô é:

```
Mesh Bounding Box Size 252.000000 400.000000 46.000000
Mesh Bounding Box Diag 474.994751
Mesh Bounding Box min -126.000000 -200.000000 2.000000
Mesh Bounding Box max 126.000000 200.000000 48.000000
Mesh Surface Area is 265922.093750
Mesh Total Len of 312 Edges is 20217.511719 Avg Len 64.799721
Mesh Total Len of 312 Edges is 20217.511719 Avg Len 64.799721 (including faux edges))
Thin shell (faces) barycenter: 0.047894 0.319325 25.045809
Vertices barycenter 0.167111 -0.115385 25.307692
Mesh Volume is 1232795.125000
Center of Mass is 0.014166 0.796665 25.105858
Inertia Tensor is :
| 34480574464.000000 13872.077148 1849.271606 |
| 13872.077148 5602350080.000000 -1860285.625000 |
| 1849.271606 -1860285.625000 39523184640.000000 |
```

Principal axes are :
| -0.000000 1.000000 0.000055 |
| 1.000000 0.000000 -0.000000 |
| -0.000000 0.000055 -1.000000 |
axis momenta are :
| 5602349568.000000 34480574464.000000 39523184640.000000 |
Applied filter Compute Geometric Measures in 19 msec

Figura 23: Corpo do robô simplificado



Fonte: Do Autor

Logo abaixo do texto “Inertia Tensor is :” estão os valores do tensor de inércia. Os tamanhos nos arquivos 3D geralmente são apresentados como escalares e as unidades de medida ficam a cargo do usuário saber quais são. Isto ocorre porque diversos formatos de arquivo não possuem a unidade de medida. *Bounding Box* é um paralelepípedo que envolve todo o desenho 3D, tocando nos pontos extremos de sua superfície. Pelo tamanho da *Bounding Box* é possível inferir que este STL está em mm pois o robô não possui 400 metros, nem 400 cm de comprimento. Assim, o tamanho de 400 mm para o corpo do robô é o aceitável. O volume é 1232795,125 mm^3. O centro de massa, como esperado está no centro do objeto, aproximadamente no meio da altura do mesmo.

O tensor de inércia fornecido não é ainda o tensor de inércia necessário para montar modelo 3D do robô pois o Gazebo trabalha com o Sistema Internacional de medidas (S.I.), os dados estão em milímetros e o tensor original considera densidade unitária, fazendo a massa ser igual ao volume. Para resolver o problema da densidade unitária, é necessário dividir os

termos do tensor de inércia pelo volume calculado e posteriormente multiplicar pela massa real. No entanto, tanto o volume quanto os termos do tensor estão na escala errada. Como o STL está em milímetros e o S.I. utiliza metros, considera-se que cada dimensão do STL está multiplicada por uma escala s = 1000. Quanto ao volume, cada dimensão está multiplicada pela escala s, logo é fácil perceber que é necessário dividir o volume por s^3 para chegar ao S.I. A massa pode ser considerada uma função do volume (volume multiplicado pela densidade), que varia de acordo com o cubo das dimensões. Assim, a massa varia de acordo com s^3. Sendo a inércia uma função da massa e do quadrado das distâncias ao eixo, ela variará de acordo com s5. Logo, os resultados do tensor de inércia devem ser divididos por s5 para chegar ao S.I. Assim, é calculado o tensor de inércia no octave-online.net[14]:

```
octave:1> v = 1232795.125
v = 1.2328e+06

octave:2> s = 1000
s = 1000

octave:3> m = 1.5
m = 1.5000

octave:4> Io = [ 34480574464.000000 13872.077148 1849.271606 ; 13872.077148
5602350080.000000 -1860285.625000 ; 1849.271606 -1860285.625000 39523184640.000000 ]
Io =

  3.4481e+10  1.3872e+04  1.8493e+03
  1.3872e+04  5.6024e+09 -1.8603e+06
  1.8493e+03 -1.8603e+06  3.9523e+10


octave:5> Ir = (( Io / (s ^ 5) ) / (v / s ^ 3) ) * m
Ir =

  4.1954e-02  1.6879e-08  2.2501e-09
  1.6879e-08  6.8166e-03 -2.2635e-06
  2.2501e-09 -2.2635e-06  4.8090e-02
```

Onde v é o volume dado, s é o fator de escala do STL, m é a massa real, Io é o Tensor de inércia original e Ir é o tensor de inércia resultante. Estes dados estão prontos para serem utilizados no arquivo SDF do Gazebo.

O braço com o propulsor simplificado está na Figura 24 . Embora o Meshlab considere o conjunto com a mesma densidade, levando em consideração que o braço é oco, o cilindro que representa o propulsor é a peça mais volumosa do conjunto, contendo a maior parte da inércia.

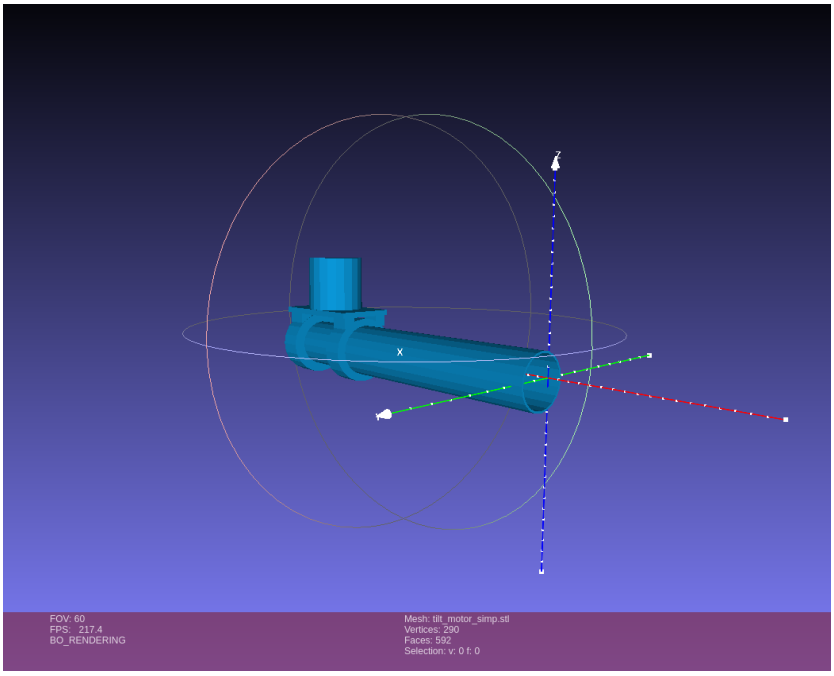
O resultado do cálculo do tensor de inércia para o braço é:

```
Mesh Bounding Box Size 163.000000 36.000000 68.000000
Mesh Bounding Box Diag 180.247055
```

14 octave-online.net é um site que permite utilizar o GNU Octave diretamente no navegador da web. GNU Octave é uma linguagem computacional, desenvolvida para computação matemática, similar ao MATLAB, mas sendo software livre de código aberto.

Mesh Bounding Box min -5.000000 -6.000000 -17.000000
Mesh Bounding Box max 158.000000 30.000000 51.000000
Mesh Surface Area is 35146.195312
Mesh Total Len of 888 Edges is 18732.412109 Avg Len 21.095058
Mesh Total Len of 888 Edges is 18732.412109 Avg Len 21.095058 (including faux edges))
Thin shell (faces) barycenter: 89.996338 11.704096 5.789940
Vertices barycenter 113.801521 11.834483 8.291229
Mesh Volume is 43863.851562
Center of Mass is 116.074806 12.213773 18.455067
Inertia Tensor is :
| 19066046.000000 -538219.437500 -12948187.000000 |
| -538219.437500 79352400.000000 -162024.359375 |
| -12948187.000000 -162024.359375 67166472.000000 |
Principal axes are :
| -0.969626 -0.008834 -0.244432 |
| -0.244465 0.002862 0.969654 |
| 0.007867 -0.999957 0.004934 |
axis momenta are :
| 15797046.000000 70430424.000000 79357432.000000 |
Applied filter Compute Geometric Measures in 21 msec

Figura 24: Braço com propulsor simplificado



Fonte: Do Autor

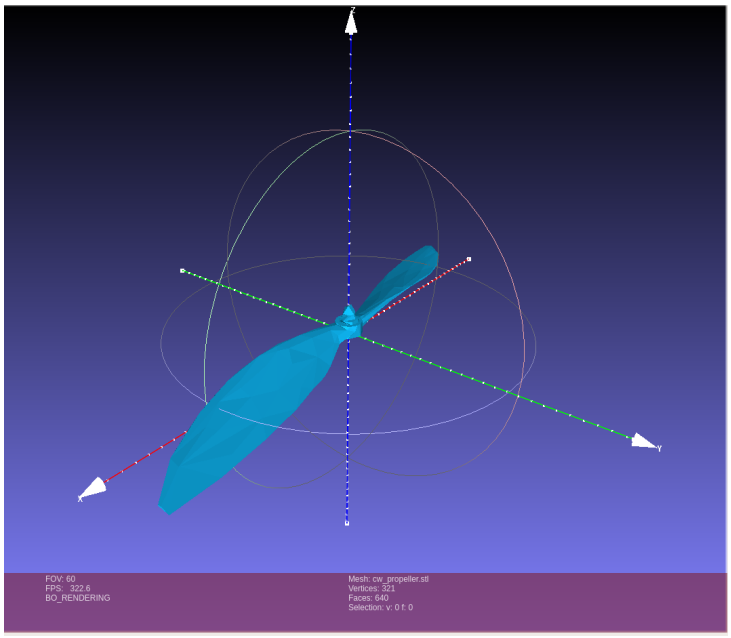
Este STL também está em mm. O centro de massa, como esperado, está muito mais próximo do propulsor que do resto do corpo. O cálculo do tensor de inércia é feito no octave:

octave:1> v = 43863.851562
v = 4.3864e+04
octave:2> s = 1000

```
s = 1000
octave:3> m = 0.15
m = 0.1500
octave:4> Io = [19066046.000000 -538219.437500 -12948187.000000 ; -538219.437500
79352400.000000 -162024.359375 ; -12948187.000000 -162024.359375 67166472.000000]
Io =

  1.9066e+07  -5.3822e+05  -1.2948e+07
 -5.3822e+05   7.9352e+07  -1.6202e+05
 -1.2948e+07  -1.6202e+05   6.7166e+07

octave:5> Ir = (( Io / (s ^ 5) ) / (v / s ^ 3) ) * m
Ir =

  6.5200e-05  -1.8405e-06  -4.4279e-05
 -1.8405e-06   2.7136e-04  -5.5407e-07
 -4.4279e-05  -5.5407e-07   2.2969e-04
```

A hélice com rotação no sentido horário é apresentada na Figura 25.

Figura 25: Hélice com rotação em sentido horário (cw)



Fonte: Do Autor

O cálculo da inércia da hélice que gira no sentido horário é:

```
Mesh Bounding Box Size 246.481995 60.458000 15.089001
Mesh Bounding Box Diag 254.236542
Mesh Bounding Box min -123.000000 -30.000000 -0.001000
Mesh Bounding Box max 123.482002 30.458000 15.088000
Mesh Surface Area is 11346.156250
Mesh Total Len of 849 Edges is 6280.970215 Avg Len 7.398080
```

Mesh Total Len of 849 Edges is 6280.970215 Avg Len 7.398080 (including faux edges))
Thin shell (faces) barycenter: 0.023345 0.176751 5.170054
Vertices barycenter -0.156198 0.309955 4.736077
Mesh Volume is 6402.151855
Center of Mass is -0.763048 -0.015906 5.621359
Inertia Tensor is :
| 806945.625000 -3977168.500000 -12147.658203 |
| -3977168.500000 27074700.000000 -1484.647095 |
| -12147.658203 -1484.647095 27827462.000000 |
Principal axes are :
| -0.989212 -0.146491 -0.000443 |
| -0.146490 0.989210 -0.001898 |
| -0.000716 0.001813 0.999998 |
axis momenta are :
| 217967.843750 27663672.000000 27827468.000000 |
Applied filter Compute Geometric Measures in 38 msec

Colocando no Octave:

```
octave:1> v = 6402.151855
v = 6402.2
octave:2> s = 1000
s = 1000
octave:3> m = 0.008
m = 8.0000e-03
octave:4>  Io  =  [806945.625000  -3977168.500000  -12147.658203  ;  -3977168.500000
27074700.000000 -1484.647095 ; -12147.658203 -1484.647095 27827462.000000]
Io =

   8.0695e+05  -3.9772e+06  -1.2148e+04
  -3.9772e+06   2.7075e+07  -1.4846e+03
  -1.2148e+04  -1.4846e+03   2.7827e+07

octave:5>  Ir = (( Io / (s ^ 5) ) / (v / s ^ 3) ) * m
Ir =

   1.0083e-06  -4.9698e-06  -1.5179e-08
  -4.9698e-06   3.3832e-05  -1.8552e-09
  -1.5179e-08  -1.8552e-09   3.4773e-05
```

Para a hélice que gira no sentido anti-horário, o tensor resultante é:

Mesh Bounding Box Size 246.481995 60.458000 15.089001
Mesh Bounding Box Diag 254.236542
Mesh Bounding Box min -123.482002 -30.000000 -0.001000
Mesh Bounding Box max 123.000000 30.458000 15.088000
Mesh Surface Area is 11346.142578
Mesh Total Len of 861 Edges is 6330.386230 Avg Len 7.352365
Mesh Total Len of 861 Edges is 6330.386230 Avg Len 7.352365 (including faux edges))
Thin shell (faces) barycenter: -0.458642 0.281244 5.170054
Vertices barycenter -0.773191 0.191361 4.670477
Mesh Volume is 6402.094238
Center of Mass is -1.245094 0.473913 5.621384
Inertia Tensor is :
| 806944.312500 3977169.000000 -12148.998047 |
| 3977169.000000 27074702.000000 1484.506226 |
| -12148.998047 1484.506226 27827464.000000 |

```
Principal axes are :
| -0.989212 0.146491 -0.000443 |
| -0.146490 -0.989210 -0.001900 |
| 0.000717 0.001815 -0.999998 |
axis momenta are :
| 217966.421875 27663672.000000 27827470.000000 |
```

Colocando no octave:

```
 octave:6> v = 6402.094238
v = 6402.1
octave:7> Io = [806944.312500 3977169.000000 -12148.998047 ; 3977169.000000
27074702.000000 1484.506226 ; -12148.998047 1484.506226 27827464.000000]
Io =

  8.0694e+05   3.9772e+06  -1.2149e+04
  3.9772e+06   2.7075e+07   1.4845e+03
 -1.2149e+04   1.4845e+03   2.7827e+07

octave:8>  Ir = (( Io / (s ^ 5) ) / (v / s ^ 3) ) * m
Ir =

  1.0084e-06   4.9698e-06  -1.5181e-08
  4.9698e-06   3.3832e-05   1.8550e-09
 -1.5181e-08   1.8550e-09   3.4773e-05
```

Chega-se aos tensores de inércia do robô. Estes tensores estão organizados da forma:

$$I_r = \begin{bmatrix} i_{xx} & i_{xy} & i_{xz} \\ i_{yx} & i_{yy} & i_{yz} \\ i_{zx} & i_{zy} & i_{zz} \end{bmatrix}$$

Estes dados serão utilizados na montagem do arquivo SDF do robô para fazer a simulação no Gazebo.

3.5.1 TRANSFORMAÇÃO DO ARQUIVO URDF EM SDF

O arquivo SDF é uma extensão do arquivo URDF, contendo os tensores de inércia de cada elo, como no modelo:

```
<inertial>
   <origin xyz="0 0 1" rpy="0 0 0"/>
   <mass value="0.15"/>
   <inertia
   ixx="1.0" ixy="0.0" ixz="0.0"
   iyy="1.0" iyz="0.0"
   izz="1.0"/>
 </inertial>
```

Os valores da inércia foram obtidos no passo anterior. Não são necessários os valores de iyx, izx e izy pois o tensor é simétrico.

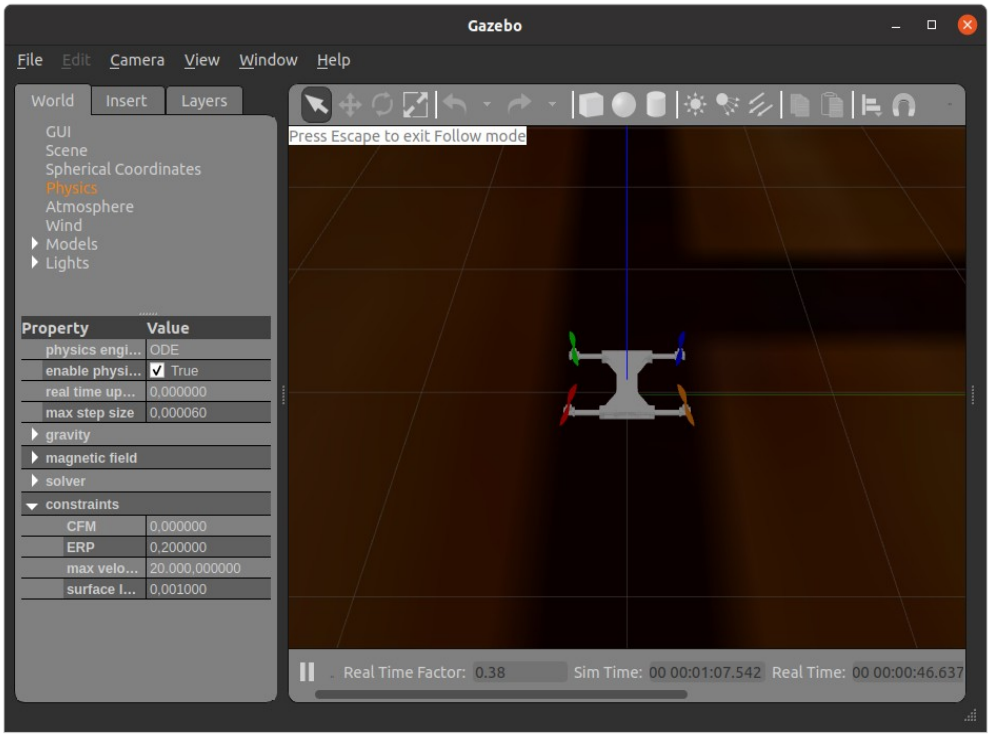
3.6 COLOCAR O ARQUIVO SDF NO GAZEBO

Para colocar o arquivo SDF no Gazebo podem ser necessários ajustes no ambiente de simulação.

Por padrão o Gazebo não permite fazer rotações em altas velocidades, necessárias para a simulação das hélices. Para resolver este problema é preciso colocar uma configuração no arquivo ".world". É necessário modificar o valor do parâmetro contact_max_correcting_vel no XML para que a simulação permita velocidades maiores de rotação.

Os motores de física do Gazebo necessitam também de adequações para simular inércia pequena, de corpos pequenos. Um corpo com inércia zero sofre uma resultante infinita se aplicada qualquer força sobre ele. Assim, as juntas do robô podem se romper se ele tiver inércia muito pequena, próxima de zero. Os motores de física truncam os tensores de inércia para dar performance à simulação, obrigando a modificar a taxa da velocidade da simulação em relação ao tempo real para que o motor de física não ignore os valores pequenos. Caso isso não seja corrigido, o robô vibra até romper as juntas e os elos, que perdem seus referenciais, são levados para o ponto de origem do mundo simulado (world).

Figura 26: A simulação do lophorina no Gazebo 11



Fonte: Do Autor

O ajuste da velocidade da simulação estará correto quando os objetos de menor massa e inércia (no caso as hélices) pararem de vibrar ou se moverem sozinhos. Isto é feito ajustando os valores de max_step_size, real_time_factor e real_time_update_rate no arquivo world. O

parâmetro real_time_update_rate pode ser zero para que o Gazebo encontre o valor ideal. No real_time_factor o valor 1 faz com que o Gazebo tente fazer a velocidade da simulação se aproximar da velocidade real. O valor do max_step_size deve ser diminuído até que as hélices fiquem paradas na simulação. Isto porque as hélices são os objetos com os menores valores no tensor de inércia. Para o caso deste trabalho, o valor máximo de 0.00006 permitiu que as hélices ficassem paradas e alcançando uma taxa de tempo real de 0,38 (a simulação rodando a 38% do tempo real), como mostrado na Figura 26.

Para alcançar uma velocidade aceitável de simulação sem problemas de inércia pode ser necessário mudar o motor de física, que por padrão é o ODE. O DART e o Bullet possuem maior desempenho, a custa de simplificações. O motor de física Simbody é o mais adequado por fazer menos simplificações do que o ODE. Por este motivo é o mais pesado, mas também é o mais utilizado para simulações biomecânicas, que necessitam de maior precisão. Neste trabalho foi utilizado o ODE pois foi possível alcançar uma simulação com velocidade aceitável sem precisar mudar o motor de física.

3.7 TRANSMISSÕES, INTERFACES E ATUADORES

Para que o robô possa se mexer é necessário definir transmissões, que são as responsáveis por movimentar os eixos do robô. Uma transmissão liga um atuador a uma junta, através de um tipo de interface. Para os *tilts* a interface delas é do tipo PositionJointInterface, que permite a escolha da posição da junta pela definição do ângulo:

```
<joint name="base_link_to_motor_front_right" type="revolute">
  <limit effort="1000.0" velocity="0.5" lower="-0.78539816339" upper="0.78539816339"/>
  <parent link="base_link"/>
  <child link="motor_front_right"/>
  <axis xyz="0 1 0"/>
  <origin xyz="0.1825 0.126 0.025"/>
 </joint>

 <transmission name="base_link_to_motor_front_right_trans">
  <type>transmission_interface/SimpleTransmission</type>
  <actuator name="$base_link_to_motor_front_right_motor">
   <mechanicalReduction>1</mechanicalReduction>
  </actuator>
  <joint name="base_link_to_motor_front_right">
   <hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
  </joint>
 </transmission>
```

Para as hélices, deve ser feita uma transmissão com interface do tipo VelocityJointInterface, como no exemplo:

```
<joint name="motor_front_right_to_front_right_propeller" type="continuous">
  <limit effort="100" velocity="10000"/>
  <parent link="motor_front_right"/>
  <child link="front_right_propeller"/>
  <axis xyz="0 0 1"/>
  <origin xyz="0 0.138 0.05"/>
 </joint>

 <transmission name="motor_front_right_to_front_right_propeller_trans">
```

```
  <type>transmission_interface/SimpleTransmission</type>
  <actuator name="$motor_front_right_to_front_right_propeller_motor">
    <mechanicalReduction>1</mechanicalReduction>
  </actuator>
  <joint name="motor_front_right_to_front_right_propeller">
    <hardwareInterface>hardware_interface/VelocityJointInterface</hardwareInterface>
  </joint>
</transmission>
```

3.8 O PLUGIN DA FORÇA DE EMPUXO

Neste trabalho foi desenvolvido um *plugin* que aplica no eixo de rotação da hélice uma força perpendicular ao plano de rotação da mesma, de acordo com a velocidade angular de rotação desta hélice. Esta força pode ser considerada proporcional ao quadrado da velocidade de rotação da hélice se assumido que a distorção das pás pode ser ignorada quando em alta frequência de rotação (VALAVANIS; VACHTSEVANOS, 2015).

Para o propósito deste trabalho, serão então geradas funções de segundo grau a partir de valores experimentais de força obtidos de (CYPRIANO; IMANISHI, 2014). Nesta pesquisa, foi utilizada uma hélice comum de um *drone* comercialmente vendido, um dos modelos utilizados pelo Aero da Universidade Federal do Espírito Santo, com dimensões de 12,25” de diâmetro e 3,75” de passo, do fabricante APC Propellers. Esta hélice foi acoplada a um motor que se apoiava por uma haste à bancada de trabalho. O motor então girou em diversas velocidades diferentes e foi medida a força de tração na haste que apoiava o conjunto, a qual corresponde à força de empuxo gerada pela hélice. Esta força medida então foi comparada com a força de tração que foi fornecida pelo fabricante, apresentando um erro percentual. O resultado deste trabalho gerou uma planilha com as forças medidas, apresentada na Tabela 2.

Tabela 2: Tração Medida e Tração Fornecida pelo Fabricante

ω (RPM)	ω (rad/s)	T Fabricante (N)	T Medida (N)	Erro (%)
1000	104,719755	0,2001	0,2197	9,8
2000	209,43951	0,7962	0,8787	10,36
3000	314,159265	1,7962	1,9771	10,07
4000	418,87902	3,1849	3,5148	10,36
5000	523,598775	4,9731	5,4919	10,43
6000	628,31853	7,1705	7,9084	10,29
7000	733,038285	9,7816	10,7641	10,04
8000	837,75804	12,8153	14,0593	9,71

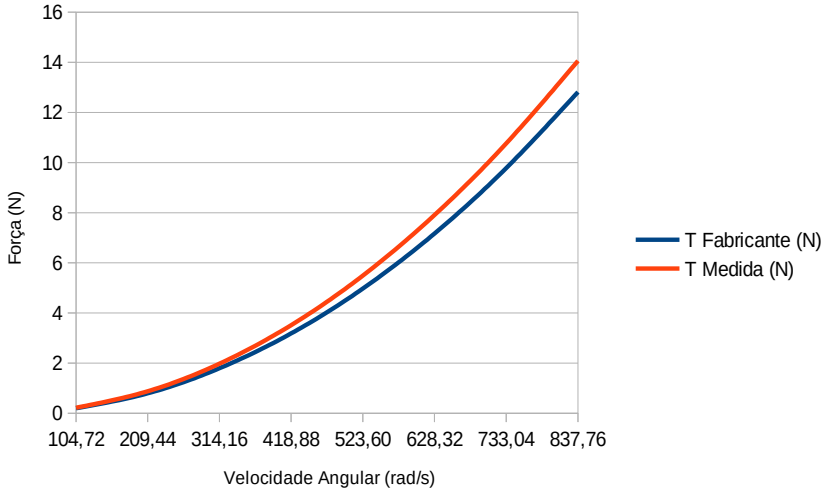
Fonte: Adaptada de (CYPRIANO; IMANISHI, 2014)

Com base nos dados da Tabela 2, foi gerado o gráfico apresentado na Figura 27, utilizando-se os valores das velocidades angulares em rad/s, que é o padrão do Gazebo. Como

a força de empuxo é proporcional ao quadrado da velocidade angular, foi feita a aproximação destas curvas pelo método dos mínimos quadrados para uma função de segundo grau da forma y = c x2:

$$y = cx^2$$
$$f_s = K\,\omega^2$$

Figura 27: Gráfico das T*raç*ões medidas versus RPM.



Fonte: Do autor

O somatório de todas as forças de sustentação medidas então é igual ao somatório de todas as constantes vezes as velocidades angulares ao quadrado, então, para as forças medidas experimentalmente:

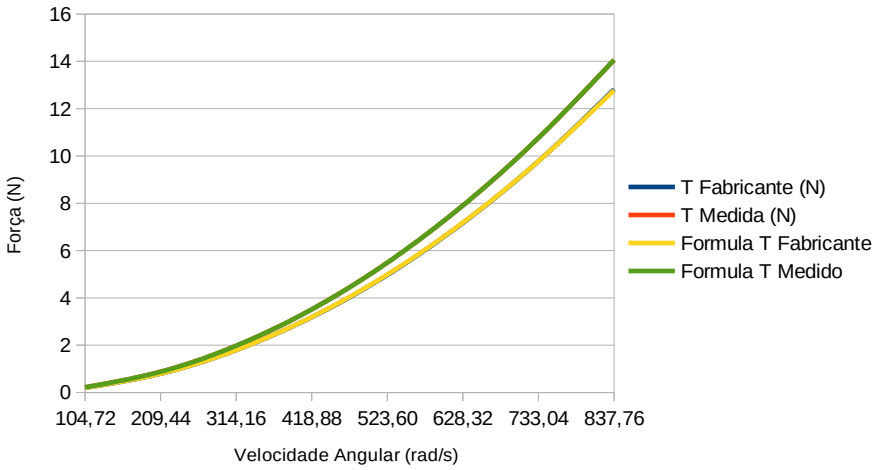
$$\sum f_s = \sum K\,\omega^2$$
$$44{,}814 = K \sum \omega^2$$
$$2237110{,}32580105\,K = 44{,}814$$
$$K = 0{,}00002003209$$

Tendo os valores dos coeficientes calculados para a força medida pelo fabricante e a força medida experimentalmente, foi traçado o gráfico das curvas, que sobrepõe o gráfico de forças original, conforme apresentado na Figura 28. Para o objetivo do presente trabalho foram utilizados os valores das forças medidas experimentalmente.

Foi então gerado um *plugin,* que deve ser aplicado em cada hélice do robô, e recebe como parâmetros:

- O valor de K (que foi calculado para uma hélice específica, mas pode ser ajustado de acordo com a hélice utilizada);
- A força máxima que o motor poderá gerar;
- A direção de rotação da hélice (horária ou anti-horária).

Figura 28: Gráfico das *Tração*es medidas e curvas aproximadas versus RPM



Fonte: Do autor

A força de arrasto nas hélices foi suprimida no *plugin* pois o arrasto na hélice apenas diminui a velocidade de rotação desta hélices e, na simulação proposta, considera-se a rotação efetiva da hélice para gerar a força, não sendo objetivo da simulação calcular a perda de potência do motor devido ao arrasto.

Com o *plugin* aplicado às quatro hélices do robô, a simulação consegue mostrar os efeitos das mudanças de posição dos *tilts* do robô independentemente.

3.9 O SCRIPT DE CONTROLE DO ROBÔ

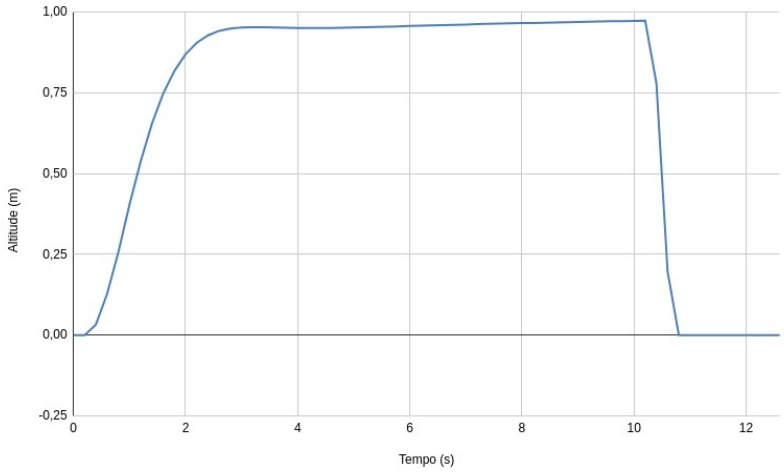
Para utilizar o *plugin* desenvolvido, deve-se publicar mensagens com os valores de referência para as velocidades de rotação das hélices. Estas publicações são feitas por um *script* controlador no tópico do *plugin* ros_gazebo_controller que lê estes valores de referência e aplica a velocidade nas hélices, realizando um controle PID. Conforme a hélice recebe velocidade angular, o *plugin* lê esta velocidade pelo ROS e gera a força correspondente no eixo da hélice, simulando a força exercida no mundo real. Quando as forças de empuxo excedem as forças do peso do VANT, o veículo começa a subir.

O controlador em *python* lê a altitude do *drone* e comanda o aumento ou redução da velocidade de rotação das hélices. Este *script python* implementa um controlador PID que, simulando um sensor de posição, escuta as poses dos elos no tópico gazebo/link_states e depois publica nos tópicos que controlam as hélices e os *tilts* do robô.

4 RESULTADOS EXPERIMENTAIS

Para que possa ser simulado o voo do robô foi necessário implementar um controlador que permita fazer o robô levantar voo e ao menos se manter planando em uma altura pré-determinada. Foi então desenvolvido um controlador PID que permitia que o robô se mantivesse a uma altura de 1 metro do chão. O controlador lê a posição do robô no eixo z e retorna a rotação que as hélices devem ter. O gráfico da altitude do robô em relação ao tempo para um voo de planeio a 1 metro do chão, seguido de um pouso, está na Figura 29.

Figura 29: Gráfico de altitude de um voo de planeio seguido de pouso

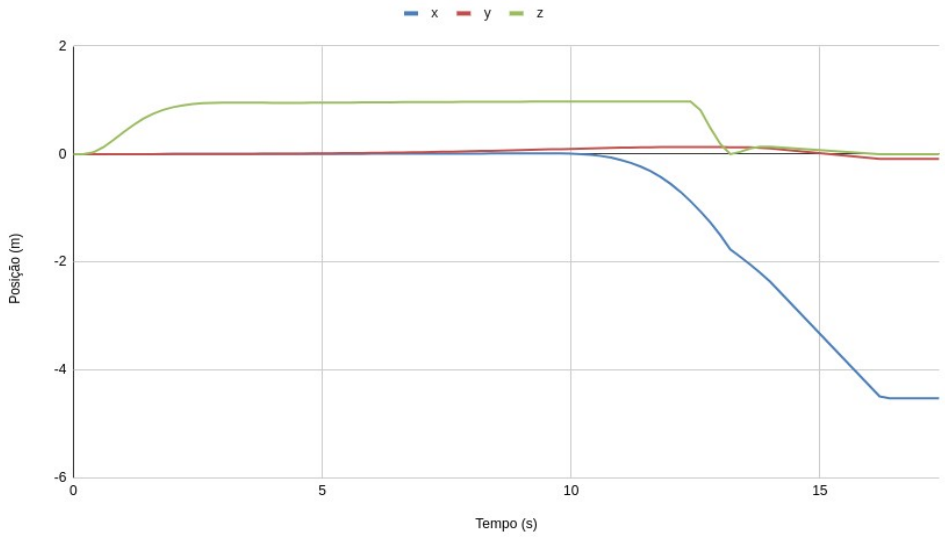


Fonte: Do Autor

Para verificar a movimentação do robô foi feita uma simulação com uma sequência de manobras[15], obtendo-se os gráfico da Figura 30 e Figura 31 Inicialmente, o robô está em repouso na origem e é comandado que planeie a 1 metro do chão. Depois de aguardar 5 segundos para que o robô alcance a altura desejada, o programa que controla o robô começa a modificar as rotações indicadas pela função do controlador PID para que o robô realize algumas manobras, utilizando a mecânica exemplificada na Figura 8. Com uma adição e remoção de 100 rad/s nas hélices cruzadas, foi feita a rotação do robô em torno do eixo z (*yaw*, ou guinada) para a direita entre t= 5s e t= 7s e para a esquerda, entre t= 7s e t= 9s. A partir de t=9s, com adição de 1 rad/s nas hélices traseiras e remoção de 1 rad/s nas hélices da frente, o robô andou para frente (no eixo x, indo na direção negativa), se inclinando no eixo y (*pitch*, ou arfagem). Em t=12 segundos, foi feito o pouso do robô, que continuou se deslocando para frente, por inércia.

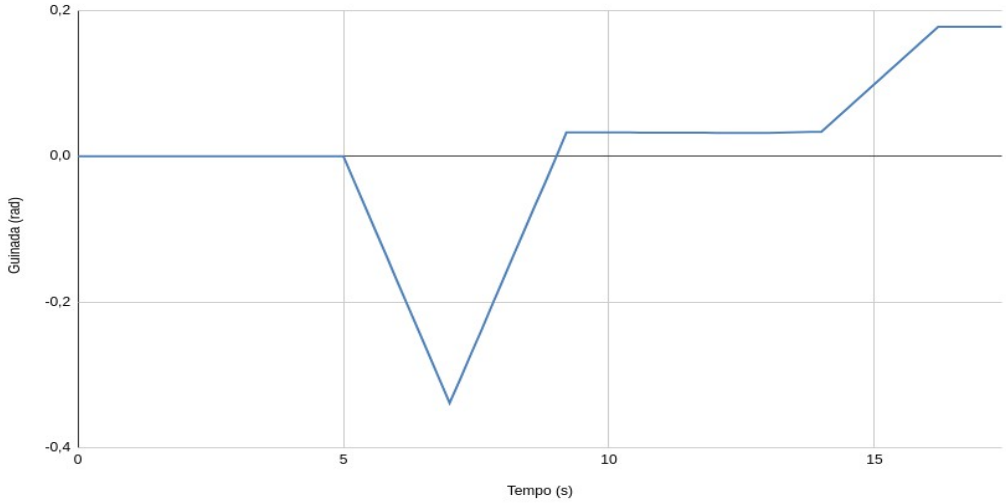
15Vídeo do teste disponível em https://youtu.be/NoptDkeeqQs

Figura 30: Gráfico da posição do robô durante manobras



Fonte: Do Autor

Figura 31: Gráfico da guinada do robô durante a manobra



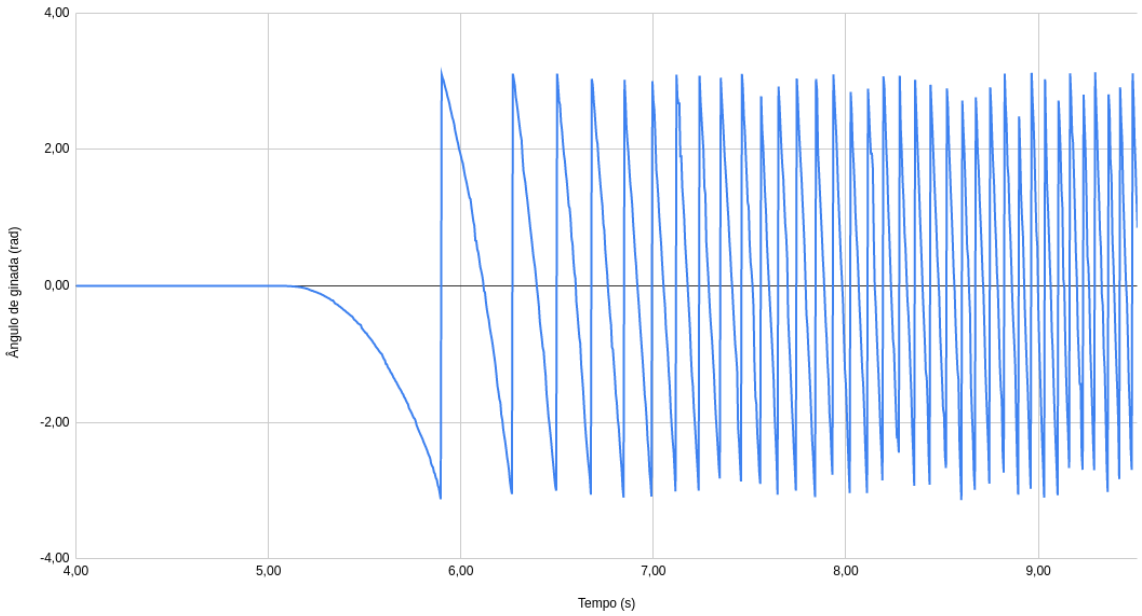
Fonte: Do Autor

Com a simulação em funcionamento e o controlador PID mantendo o robô planando, foi feito também um teste de rotação do *tilt* em pequeno ângulo ($\pi/36$ rad, ou, aproximadamente, 0,08727 rad), o que produziu o efeito esperado de fazer que o robô se deslocasse para frente por algum tempo, tombando em seguida por falta de um controlador específico[16]. Isto ocorre porque o robô é um sistema em um equilíbrio instável.

16Vídeo disponível em https://youtu.be/wS2huZBAi4w

Em um outro experimento, para testar o comportamento do VANT como til quadrirrotor, com o robô planando a 1 m do chão, novamente foi dado o comando para inclinar os *tilts* em um pequeno ângulo (π/36 rad), mas dessa vez os *tilts* da direita foram inclinados para trás e os *tilts* da esquerda foram inclinados para frente, o que fez o robô girar em guinada, no sentido horário, como esperado[17]. Esta rotação se difere da guinada do primeiro experimento pois não foi gerada por diferença de rotação entre as hélices, e sim por inclinação dos *tilts*. O gráfico do ângulo de guinada (em radianos) em relação ao tempo está na Figura 32. Conforme o robô gira no sentido horário, o ângulo de guinada diminui até alcançar -π e continua diminuindo, contando a partir de π. Conforme vai acelerando, o tempo para realizar a volta completa vai diminuindo.

Figura 32: Gráfico do Ângulo de Guinada



Fonte: Do Autor

4.1 O FRAMEWORK

Para simplificar o uso dométodo, foi criado um *framework* a partir dos códigos gerados, composto da estrutura básica de pastas da simulação, contendo:

- O arquivo URDF desenvolvido para o robô Lophorina
- O arquivo SDF desenvolvido para o robô Lophorina
- Os arquivos de *launch*
- Os arquivos de configuração das transmissões e atuadores
- O arquivo *World* da simulação
- O *plugin* de empuxo
- Os *scripts* de controle

17Vídeo disponível em https://youtu.be/FBq38P3SSUk

Estes arquivos podem ser modificados, de acordo com o método proposto, para gerar a simulação de qualquer outro robô quadrirrotor, ou até mesmo com mais hélices, com ou sem *tilt*.

Os arquivos deste *framework* estão disponíveis no Github[18].

18 Disponível em https://github.com/lrmcbr/lophorina

5 CONCLUSÃO

O uso de um *plugin* que gere as forças de empuxo no eixo de rotação das hélices permitindo que o motor de física do Gazebo faça os cálculos de inércia é vantajoso pois:

- O *plugin* se torna suficientemente genérico para que possa ser utilizado em diversos modelos de aeronave, incluindo tilt-quadrirrotores, sem necessidade de modificações no *plugin*, apenas especificação dos parâmetros da hélice na chamada do *plugin*.
- A simulação se torna mais flexível, não necessitando de alterações no cálculo de resultantes para modificar características da aeronave, como modificar ângulos de *tilt* e adicionar um braço robótico ao robô e pegar objetos pesados com ele.

Um ponto interessante desta abordagem é que nesta simulação não são inseridos os parâmetros de voo calculados para o VANT, apenas as características físicas dos elos, a força de rotação dos propulsores (juntas) e a força gerada pela hélice de acordo com sua velocidade angular. O voo é uma consequência deste sistema estar inserido em um ambiente que simula as forças.

Por outro lado, esta abordagem da simulação exige maior poder computacional que a simulação com o cálculo da resultante embutido, sendo necessário diminuir a velocidade da simulação e modificar parâmetros para evitar simplificações no motor de física que influenciem no resultado ou utilizar hardwares mais poderosos.

São trabalhos futuros:

- Implementar um controle mais completo do robô, para permitir navegação e aplicações mais reais.
- Testar esta abordagem de implementação com outros ambientes de simulação 3D existentes, o que envolve um estudo similar.

REFERÊNCIAS

ALKAMACHI, A.; ERCELEBI, E. Modelling and control of H-shaped racing quadcopter with tilting propellers. Facta Universitatis, Series: Mechanical Engineering, [s. l.], v. 15, n. 2, p. 201–215, 2017. Available at: https://doi.org/10.22190/FUME170203005A

BAILON-RUIZ, R. et al. System simulation of a fleet of drones to probe cumulus clouds. 2017 International Conference on Unmanned Aircraft Systems, ICUAS 2017, [s. l.], p. 375–382, 2017. Available at: https://doi.org/10.1109/ICUAS.2017.7991448

BERTOLDO, G. E. B. et al. Obtenção Experimental dos Coeficientes de Empuxo e de Torque para Modelagem Matemática de Quadrirrotores. Proceeding Series of the Brazilian Society of Computational and Applied Mathematics, [s. l.], v. 6, n. 2, p. 010168-1,010168-2, 2018.

COSTA, T. L. Modelagem e controle de um veículo aérero não tripulado com rotores inclináveis e câmera orientável. [s. l.], 2019.

COSTA, T. L. et al. Modeling and Control of an Unmanned Aerial Vehicle with Tilt Rotors Equipped with a Camera. [s. l.], p. 1–6, 2020. Available at: https://doi.org/10.1109/icar46387.2019.8981605

CYPRIANO, M. A.; IMANISHI, R. M. M. Estudo das forças geradas por um hélice. 1–47 f. 2014. - UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO, [s. l.], 2014.

FOUNDATION, O. S. R. Aerodynamics. [S. l.], 2014. Available at: https://gazebosim.org/tutorials?tut=aerodynamics&cat=physics. Acesso em: 27 mar. 2022.

IBRAHIM JENIE, Y. et al. Quadrotor Uav Simulation Modeling Using X-Plane Simulation Software. International Journal of Mechanical and Production Engineering, [s. l.], n. 6, p. 2321–2071, 2018. Available at: http://iraj.in

JI, R.; MA, J.; SAM GE, S. Modeling and Control of a Tilting Quadcopter. IEEE Transactions on Aerospace and Electronic Systems, [s. l.], v. 56, n. 4, p. 2823–2834, 2020. Available at: https://doi.org/10.1109/TAES.2019.2955525

JUNAID, A. Bin et al. Design and implementation of a dual-axis tilting quadcopter. Robotics, [s. l.], v. 7, n. 4, p. 1–20, 2018. Available at: https://doi.org/10.3390/robotics7040065

LARA, A. et al. Desenvolvimento De Um Ambiente De Simulação De Vants Tilt-Rotor Para Testes De Estrategias De Controle. XIII Simposio Brasileiro de Automaçao Inteligente, [s. l.], p. 2135–2141, 2017.

MEYER, J. et al. Comprehensive simulation of quadrotor UAVs using ROS and Gazebo. In: , 2012. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). [S. l.: s. n.], 2012. Available at: https://doi.org/10.1007/978-3-642-34327-8_36

MOUTINHO, A.; MATEOS, E.; CUNHA, F. The tilt-quadrotor: Concept, modeling and identification. Proceedings - 2015 IEEE International Conference on Autonomous Robot Systems and Competitions, ICARSC 2015, [s. l.], v. 1, p. 156–161, 2015. Available at: https://doi.org/10.1109/ICARSC.2015.38

NEMATI, A. et al. Design, fabrication and control of a tilt rotor quadcopter. ASME 2016 Dynamic Systems and Control Conference, DSCC 2016, [s. l.], v. 2, n. October, 2016. Available at: https://doi.org/10.1115/DSCC2016-9929

ODELGA, M.; STEGAGNO, P.; BULTHOFF, H. H. A fully actuated quadrotor UAV with a propeller tilting mechanism: Modeling and control. IEEE/ASME International Conference on Advanced Intelligent Mechatronics, AIM, [s. l.], v. 2016-Septe, p. 306–311, 2016. Available at: https://doi.org/10.1109/AIM.2016.7576784

OOSEDO, A. et al. Large attitude change flight of a quad tilt rotor unmanned aerial vehicle. Advanced Robotics, [s. l.], v. 30, n. 5, p. 326–337, 2016. Available at: https://doi.org/10.1080/01691864.2015.1134344

OPEN ROBOTICS. Gazebo Multi-Physics Engine Support. [S. l.], [s. d.].

OPEN SOURCE ROBOTICS FOUNDATION. Aerodynamics. [S. l.], 2014a. Available at: https://gazebosim.org/tutorials?tut=aerodynamics&cat=physics.

OPEN SOURCE ROBOTICS FOUNDATION. Tutorial: Using a URDF in Gazebo. [S. l.], 2014b. Available at: https://classic.gazebosim.org/tutorials?tut=ros_urdf.

RODRIGUES, L. E. M. J. Fundamentos da Engenharia Aeronáutica com Aplicações ao Projeto SAE-AeroDesign: Aerodinâmica e Desempenho. [S. l.: s. n.], 2014. v. 1E-book.

SENKUL, F.; ALTUG, E. Adaptive control of a tilt-roll rotor quadrotor UAV. 2014 International Conference on Unmanned Aircraft Systems, ICUAS 2014 - Conference Proceedings, [s. l.], p. 1132–1137, 2014. Available at: https://doi.org/10.1109/ICUAS.2014.6842367

TUTA NAVAJAS, G. H.; ROA PRADA, S. Building your own quadrotor: A mechatronics system design case study. 2014 3rd International Congress of Engineering Mechatronics and Automation, CIIMA 2014 - Conference Proceedings, [s. l.], n. 2, p. 1–5, 2014. Available at: https://doi.org/10.1109/CIIMA.2014.6983444

VALAVANIS, K. P.; VACHTSEVANOS, G. J. Handbook of unmanned aerial vehicles. [S. l.: s. n.], 2015. Available at: https://doi.org/10.1007/978-90-481-9707-1

VENDRICHOSKI, Julio C. et al. Mathematical modeling and control of a quadrotor aerial vehicle with tiltrotors aimed for interaction tasks. [s. l.], p. 161–166, 2020. Available at: https://doi.org/10.1109/icar46387.2019.8981636

VENDRICHOSKI, Julio Cezar. MODELAGEM E CONTROLE DE UM VEÍCULO AÉREO HÍBRIDO COM ROTORES INCLINÁVEIS E CÂMERA ORIENTÁVEL. 157 f. 2017. - Universidade Federal de Santa Catarina, [s. l.], 2017.

ZHENG, P. et al. TiltDrone: A Fully-Actuated Tilting Quadrotor Platform. IEEE Robotics and Automation Letters, [s. l.], v. 5, n. 4, p. 6845–6852, 2020. Available at: https://doi.org/10.1109/LRA.2020.3010460