



CURSO SUPERIOR DE ENGENHARIA DE COMPUTAÇÃO

Fabiano Dutra Filereno

**Controle e Automação de Vant com Visão
Computacional para Sensoriamento
Populacional**

**Porto Alegre-RS
2020**

Fabiano Dutra Filereno

Controle e Automação de Vant com Visão Computacional para Sensoriamento Populacional

Trabalho apresentado para o Curso de Engenharia do Computação, de Centro Universitário Uniftec como parte dos requisitos para avaliação da unidade curricular de TCC.

Centro Universitário Uniftec

Orientador: prof. Dr. Leonardo Albernaz Amaral

Porto Alegre-RS

2020

Fabiano Dutra Filereno

Controle e Automação de Vant com Visão Computacional para Sensoriamento Populacional

Trabalho apresentado para o Curso de Engenharia do Computação, de Centro Universitário Uniftec como parte dos requisitos para avaliação da unidade curricular de TCC.

prof. Dr. Leonardo Albernaz Amaral
Orientador

Professor
Convidado 1

Professor
Convidado 2

Porto Alegre-RS
2020

Listas de ilustrações

Figura 1 – Metodologia utilizada	5
Figura 2 – Exemplo de Redes Neurais	8
Figura 3 – Diferença entre IA, ML e DL	9
Figura 4 – Rede Neural	10
Figura 5 – Rede Neural Convolucional	11
Figura 6 – Pixel em Visão Computacional	12
Figura 7 – Exemplo Simples de Treinamento de Rede Neural com Método de Trigêmeos	15
Figura 8 – Exemplo de Comparação de Vetores 128-d	16
Figura 9 – Extração de um Vetor 128-d com Reconhecimento Facial	17
Figura 10 – Conexão MAVLink da Raspberry Pi com Pixhawk	21
Figura 11 – Exemplo de Funcionamento do SITL	23
Figura 12 – Fluxo de Operação SITL, MAVProxy, Ardupilot e MAVLink	24
Figura 13 – VANT do Tipo Quadricóptero	25
Figura 14 – Frame de um VANT Quadricóptero	26
Figura 15 – Pixhawk 1	27
Figura 16 – Diagrama Eletrônico dos Conectores da Pixhawk 1	27
Figura 17 – Motor sem Escova	29
Figura 18 – Partes do Motor	29
Figura 19 – Controlador Eletrônico de Corrente utilizado em VANTS	30
Figura 20 – Diagrama Eletrônico de um Controlador Eletrônico de Corrente	30
Figura 21 – Módulo de GPS	31
Figura 22 – Hélice	32
Figura 23 – Bateria de Polímero de Lítio	32
Figura 24 – Esquema Básico de Montagem de um VANT do tipo Quadricóptero	33
Figura 25 – Teoria de Sustentação de uma Hélice	34
Figura 26 – Diagrama de Esforços cortantes na Estrutura do Quadricóptero	35
Figura 27 – Diagrama de Momento Fletor sobre a Estrutura do Quadricóptero	36
Figura 28 – Sentido de Movimento dos Motores e Inercias	37
Figura 29 – Movimentos de um Quadricóptero	38
Figura 30 – Movimento Horizontal " <i>Roll</i> "	39
Figura 31 – Movimentos Lineares de um Quadricóptero	40
Figura 32 – Plano Cartesiano na Superfície Esférica	41
Figura 33 – NED	43
Figura 34 – Simulação real para abstrata	45
Figura 35 – Arquitetura Básica do Sistema de Simulação	46

Figura 36 – Diagrama de Funcionamento do Sistema	48
Figura 37 – Diagrama UML Total do Sistema	49
Figura 38 – Diagrama UML do Sistema de Reconhecimento Facial	50
Figura 39 – Diagrama UML do Sistema de Controle	51
Figura 40 – Sistema de Coordenadas do OpenCV	53
Figura 41 – Conversão do Sistema de Coordenadas OpenCV para Coordenadas Cartesianas	54
Figura 42 – Sistema de Coordenadas Cartesianas em Visão Computacional	55
Figura 43 – Projeção do Sistema de Coordenadas OpenCV para Cartesianas	56
Figura 44 – Regiões de Interesse	57
Figura 45 – Gráfico que Representa o Gradiente de Velocidade	60
Figura 46 – Gráfico que Representa o Gradiente de Velocidade para X e Y	61
Figura 47 – Raspberry Pi 3 B	64
Figura 48 – Dispositivo de Captura	65
Figura 49 – Teste de Reconhecimento Facial (2 Pessoas)	67
Figura 50 – Teste de Reconhecimento Facial (1 Pessoa)	68
Figura 51 – Simulação de Nenhum Deslocamento	69
Figura 52 – Simulação de Deslocamento para o Norte	70
Figura 53 – Simulação de Deslocamento para o Sul	70
Figura 54 – Simulação de Deslocamento para o Leste	71
Figura 55 – Simulação de Deslocamento para o Nordeste	71
Figura 56 – Simulação de Deslocamento para o Noroeste	72
Figura 57 – Primeira Simulação de Velocidade para o Norte	73
Figura 58 – Segunda Simulação de Velocidade para o Norte	73
Figura 59 – Terceira Simulação de Velocidade para o Norte	74
Figura 60 – Simulação de Velocidade para o Sul	74

Lista de abreviaturas e siglas

VANT	Veiculo Aéreo não Tripulado
IA	Inteligencia Artificial
MA	Machine Learning
DL	Deep Learning
BSD	Berkeley Software Distribution
MIT	Massachusetts Institute of Technology
CUDA	Compute Unified Device Architecture
GPU	Graphics Processing Unit
GPGPU	General Purpose Graphics Processing Unit
SIMD	Single Instruction, Multiple Data
API	Application Programming Interface
CNN	Convolucional Neural Network
NN	Neural Network
ResNET	Residual Neural Network
XML	Extensible Markup Language
GCS	Ground Control Station
UAV	Unmanned Aerial Vehicle
GUI	Graphical User Interface
POSIX	Portable Operating System Interface
SITL	Software in the Loop
UDP	User Datagram Protocol
TCP	Transmission Control Protocol
NASA	National Aeronautics and Space Administration

ROS	Robot Operating System
MEMs	Micro-Electro-Mechanical Systems
GPS	Global Positioning System
FPU	Unidade de Ponto Flutuante
MEAS	Managed Ethernet Access Service
ESD	Electrostatic Discharge
UART	Universal Asynchronous Receiver Transmitter
RSSI	Received Signal Strength Indication
PPM	Pulse Position Modulation
I2C	Inter-Integrated Circuit
SPI	Serial Peripheral Interface
CAN	Controller Area Network
ADC	Analog-to-Digital Converters
ESC	Electronic Speed Controllers
LíPo	Lítio Polimero
SSD	Unidade de Estado Sólida
SATA	Serial Advanced Technology Attachment
GPIO	General Purpose Input/Output

List of symbols

α	Letra Grega Alpha
τ	Letra Grega minúscula tau
Θ	Letra Grega maiúscula Teta
λ	Letra Grega minúscula lambda
ψ	Letra Grega minúscula psi
ϕ	Letra Grega minúscula phi

Sumário

1	INTRODUÇÃO	1
1.1	Objetivo Geral	2
1.2	Objetivos Específicos	3
1.3	Resultados Esperados	3
1.4	Visão Geral da Metodologia	3
1.5	Justificativa	5
2	FUNDAMENTAÇÃO TEÓRICA	6
2.1	Inteligência Artificial	6
2.1.1	Um pouco sobre a Historia	6
2.1.2	Machine Learnig	6
2.1.3	Deep Learnig	7
2.1.4	Visão Computacional	10
2.1.5	Redes Neurais Profundas (Deep Learning)	11
2.2	Tecnologias Open Source para Tratamento de Imagens	12
2.2.1	OpenCV	12
2.2.2	Biblioteca Dlib	13
2.2.3	Biblioteca Face Recognition	13
2.2.4	Nvidia Cuda	13
2.2.5	Reconhecimento Facial	14
2.3	Tecnologias de Linguagem de Programação para Cálculos de Álgebra e Geometria	18
2.3.1	Biblioteca NumPy	18
2.3.2	Biblioteca SciPy	18
2.4	Tecnologias Open Source de Desenvolvimento para Veículos Aéreos não Tripulados	18
2.4.1	Firmwares para VANT	18
2.4.1.1	Ardupilot	18
2.4.2	Ferramentas de Desenvolvimento para VANT	19
2.4.2.1	API de Desenvolvimento Dronekit para ArduPilot	19
2.4.3	Protocolo de Comunicação MAVLink	19
2.4.4	Proxy de Protocolo MAVLink para VANT (MAVProxy)	21
2.4.5	Simulador de VANT SITL	22
2.4.6	Simulador de Robótica Gazebo	24
2.5	Tecnologias e a Anatomia de Veículos Aéreos não Tripulados	24

2.5.1	Frame	25
2.5.2	Controladora de Voo	26
2.5.3	Motor sem Escova	29
2.5.4	Controlador Eletrônico de Corrente (ESC)	30
2.5.5	Módulo de GPS	31
2.5.6	Hélice	31
2.5.7	Bateria de Polímero de Lítio (LíPo)	32
2.5.8	Esquema de Montagem Básica de um Quadricóptero	33
2.6	Teoria de funcionamento e dinâmica de voo de um quadricóptero	34
2.7	Sistema de Orientação de um VANT	40
3	METODOLOGIA	44
3.1	Métodos	46
3.2	Protótipo	47
3.2.1	Fluxogramas dos Protótipos e Funcionamento do Sistema	47
3.2.1.1	Digramas UML da Codificação e Lógica do Software	48
3.2.2	Teoria do Sistema de Controle de Robótica para o VANT	52
3.2.2.1	Conversão do Sistema de Coordenadas OpenCV para Cartesiano	52
3.2.2.2	Regiões de Interesse	56
3.2.2.3	Coeficiente de Aceleração	58
3.2.2.4	Movimentação do VANT	58
3.3	Treinamento da rede Neural	61
3.4	Controles do VANT	62
3.5	Componentes Físicos	62
3.5.1	Computador Utilizado nas Simulações	62
3.5.2	Computador Complementar	63
3.5.3	Dispositivo de Captura	65
3.6	Sistemas Operacionais	65
3.7	Computador Complementar vs Computador Utilizado nas Simulações (Benchmarking)	66
4	ANALISE DOS RESULTADOS E CONSIDERAÇÕES FINAIS	67
4.1	Resultados	67
4.2	Considerações Finais	74
	REFERÊNCIAS	76

APÊNDICES 79

APÊNDICE A – CÓDIGO FONTE DA VISÃO COMPUTACIONAL 80

APÊNDICE B – CÓDIGO FONTE DOS CONTROLES DO VANT 88

1 Introdução

A segurança pública hoje é um tema muito noticiado em veículos de mídia, segundo estudos hoje no Brasil aproximadamente 10% dos crimes cometidos são solucionados ([BRUM, 2018](#)) e isso se dá principalmente devido ao fato das polícias terem um baixo efetivo não dando conta de cobrirem certas áreas. Outro fator muito importante é a falta de investimento em tecnologias o que não acontece em países de primeiro mundo que possuem uma alta taxa de soluções em casos criminais pelo fato destes países terem um alto investimento tecnológico na área de segurança. Um bom exemplo seria a China que possui hoje o melhor sistema de vigilância existente, composto de um complexo produto de IA utilizando visão computacional, o que faz com que eles possuam uma taxa de crimes mais baixa que a de países da Europa e equivalente ao de países como a Suíça ([ZMOGINSKI, 2019](#)).

No Brasil, os crimes são solucionados, pelo fato da polícia geralmente adquirir provas através de alguma imagem feita por uma câmera de terceiro, ou seja, equipamentos privados que foram instalados em residências ou empresas [16]. Contudo, percebemos que no Brasil existe uma grande falta de investimentos em pesquisa focada em novas tecnologias que poderiam suprir o mercado e principalmente a área de segurança.

Hoje existem tecnologias com distribuição livre que podem ser utilizadas no mercado Brasileiro na área de segurança. Um exemplo são as ferramentas de visão computacional como o OpenCV, ferramenta muito poderosa que é utilizada em vários países com foco principalmente em reconhecimento facial, identificação de padrões de comportamento, detecção de objetos, entre outras várias funcionalidades, todas utilizando aprendizagem de máquina (Machine Learning) [17].

Com investimento em pesquisa e desenvolvimento, a tecnologia de visão computacional pode ser aplicada como ferramenta para suprir a demanda de tecnologias na área de segurança pública no Brasil.

Alguns países que utilizam detecção facial já estão desenvolvendo sistemas integrados desta tecnologia com a utilização de um veículo aéreo não tripulado (VANT) ([ZMOGINSKI, 2019](#)) [2]. Com um VANT é possível cobrir uma grande área utilizando o espaço aéreo e em casos de desaparecimentos, este equipamento poderá acessar áreas de difícil acesso. Um bom exemplo é o caso do rompimento da barragem de Brumadinho onde ocorreu de vários corpos não serem localizados pelo fato da lama dificultar as buscas, e uma empresa estrangeira veio com uma solução utilizando VANTS que possibilitaram a busca em certos locais, esses equipamentos possuíam câmeras com tecnologias que possibilitavam identificar um corpo em meio ou dentro da lama [18].

No site da Ardupilot (ARDUPILOT.ORG, 2020)[3], já existe um projeto em desenvolvimento para conectar e configurar um computador complementar, o Nvidia Jetson¹, Google Coral Dev Board² e Raspberry Pi³ a uma controladora de voo, essa conexão utiliza um protocolo chamado MAVLink e possui uma documentação bem desenvolvida que explica detalhadamente como controlar um VANT utilizando a ferramenta Dronekit, e a partir dele é possível prototipar uma grande diversidade de projetos unindo ferramentas e tecnologias diversas ao VANT.

Com isso surgiu a ideia do estudo de integrar via protocolo de comunicação (MAVLink) à Raspberry Pi com a controladora de voo do VANT, embarcar um sistema Linux com ferramentas de visão na Raspberry Pi, e assim possibilitar o desenvolvimento de um sistema com IA focado em visão computacional que possibilite o desenvolvimento de controles direcionais que atuem no VANT, sendo assim possível controlá-lo através de um sistema que tenha uma câmera como sensor de orientação e direcionamento.

1.1 Objetivo Geral

O objetivo geral deste trabalho é desenvolver o protótipo de um VANT controlado por um sistema embarcado com visão computacional, sistema que será implementado em um computador complementar (Raspberry Pi) o qual se comunicará com a controladora de voo por protocolo de comunicação.

A ideia é desenvolver um sistema de robótica que atue através de controle e automação em um VANT, ele tem o objetivo de controlar, movimentar o VANT usando visão computacional.

Será implementado um algoritmo de visão computacional para reconhecimento facial que foi pré-treinado utilizando CNN (convolucional neural network), ele identificará faces que constam no conjunto de dados treinado pela rede neural.

Um novo sistema de coordenadas será construído baseando-se no sistema de coordenadas do OpenCV, o resultado vai gerar um plano de coordenadas do tipo cartesianas.

As componentes X e Y adquiridas pelo sistema cartesiano serão enviadas para a controladora de voo fazendo com que o VANT se desloque de acordo com a movimentação percebida pelo sistema de visão computacional.

A validação será através da abstração do hardware por um conjunto de ferramentas de software que simulam o funcionamento do VANT.

¹<<https://www.nvidia.com/pt-br/autonomous-machines/embedded-systems/jetson-nano/>>

²<<https://coral.ai/products/dev-board>>

³<<https://www.raspberrypi.org/products/raspberry-pi-3-model-b/>>

1.2 Objetivos Específicos

- Desenvolver um algoritmo de visão computacional para reconhecimento facial;
- Desenvolver um sistema de controle e automação para VANT baseando-se em Visão Computacional;
- Comparar o desempenho do algoritmo de visão computacional entre o computador utilizado nas simulações e o computador complementar (Raspberry Pi);

1.3 Resultados Esperados

- Reconhecimento de faces inseridas no conjunto de dados pré-treinado;
- Distinguir entre faces inseridas no conjunto de dados e as que não foram inseridas;
- Desenvolver um sistema que possibilite guiar o VANT;
- Demonstrar que é possível guiar o VANT em direções estipuladas pelo sistema de visão computacional;
- Realizar testes e adquirir dados para determinar se é viável ou não utilizar visão computacional para definir um sistema de robótica para um VANT;

1.4 Visão Geral da Metodologia

A ideia surgiu devido o interesse em integrar um veículo aéreo não tripulado (VANT) com algum dispositivo para realização de alguma tarefa, e após buscar trabalhos de pesquisa, protótipos experimentais, bibliografias sobre o assunto e sites especializados em tecnologias aplicadas a VANTs, descobriu-se que é possível conectar, uma controladora de voo ([DRONECODE, 2020](#)) [7] a um computador complementar (Raspberry Pi)([RASPERRYPI.ORG, 2020a](#))[8], e enviar comandos para o VANT, criando um sistema de controle e automação usando visão computacional ([ARDUPILOT.ORG, 2020](#))[3]. Em seguida também se descobriu que é possível implementar a ferramenta OpenCV na Raspberry Pi fazendo a integração entre tecnologias de visão computacional e a tecnologia de um VANT.

Como o foco principal do trabalho é o controle de um VANT utilizando visão computacional, primeiramente se realizou uma pesquisa para descobrir se existe um computador complementar que pudesse comportar um software de visão computacional e que também tivesse capacidade computacional de compilar e rodar o algoritmo, e com processamento o suficiente para processar a tecnologia de visão computacional.

A pesquisa também abordou como comunicar o computador complementar com a controladora do VANT e se existia a possibilidade de enviar comandos para a controladora de voo.

Foi preciso buscar na internet muito material principalmente tutorias para aprender e dominar o funcionamento das ferramentas que seriam utilizadas no desenvolvimento, por exemplo o OpenCV, e as ferramentas de desenvolvimento para VANTs. Essas tecnologias são muito vastas e possuem muitas funcionalidades e módulos, foi preciso testar quais seriam as melhores maneiras de utiliza-las no projeto, e quais módulos utilizar, logo foi gasto muito tempo simulando e testando esses módulos para adequar quais seriam utilizados e serviriam na implementação.

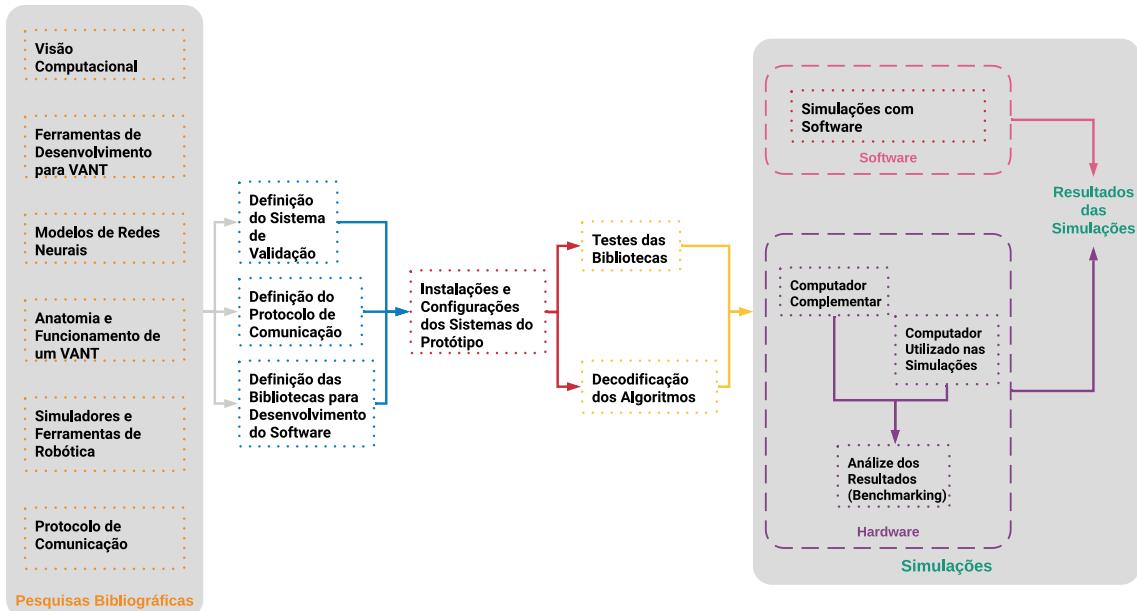
Uma boa parte do tempo de pesquisa do trabalho foi buscar bibliografias sobre IA e Deep Learning, entender sobre redes neurais e visão computacional para o entendimento de como funcionam essas tecnologias.

A metodologia utilizada pode ser definida em 3 etapas que são exemplificadas de maneira procedural e ilustrada na figura 1.

Elas foram definidas da seguinte forma:

- Uma primeira etapa de pesquisa bibliográfica buscando definir quais materiais encontrados seriam utilizados como referência para prosseguir, e através deles verificar se era possível chegar a um resultado satisfatório e se o protótipo teria uma aplicação prática e justificável.
- Depois das pesquisas a etapa que definiu quais bibliotecas, ferramentas, linguagem de programação, protocolos e qual seria a maneira de simulação e validação seria utilizada, assim como as instalações e configurações dos sistemas para posterior codificação dos algoritmos e testes de bibliotecas e códigos.
- A última foi para realizar simulações de funcionamento e comportamento para se chegar em resultados.

Figura 1 – Metodologia utilizada



Fonte: do autor.

1.5 Justificativa

Devido ao aumento da criminalidade e dos altos índices de violência no cotidiano das pessoas, surgiu a cultura do medo e o sentimento de insegurança. Tais fatores demandaram algumas mudanças nos serviços de segurança patrimonial bem como, nas formas de monitoramento. Nesse sentido, tornou-se necessário expandir as formas de controle, seja por meio de câmeras de vigilância ou monitoramento eletrônico (SOUZA et al., 2016)[4]. No entanto, os equipamentos atuais como as câmeras utilizadas na segurança já não são mais eficientes e possuem tecnologias ultrapassadas.

Na cidade do Rio de Janeiro existe um sistema de câmeras de alta tecnologia com capacidade para realizar reconhecimento facial. Este sistema foi implantado para testes através de uma parceria entre a OI e a Huawei (SILVA, 2019)[5], sendo que as câmeras e a tecnologia de reconhecimento facial embarcados são de propriedade da Chinesa Huawei e possuem alto custo monetário, e alguns estados Brasileiros passam por uma crise financeira, porem precisam investir em segurança (SANTOS, 2018a)[6].

Este contexto de insegurança pública e falta de investimento nacional em tecnologias de segurança é que originou a motivação para a proposta deste trabalho, ou seja, a necessidade de investir em tecnologias de baixo ou médio custo, que possam ser aplicados para melhorar o desempenho do sistema de segurança no Brasil, visando a vigilância, através do sensoriamento utilizando câmeras e hardware de custo acessível e softwares livres.

2 Fundamentação Teórica

2.1 Inteligência Artificial

2.1.1 Um pouco sobre a Historia

O termo inteligência artificial foi expressa pela primeira vez por Alan Turing em 1950, ano no qual ele lançou um artigo falando sobre o seu “jogo da imitação”, hoje conhecido como “Teste de Turing”, já na época ele reconhece vários desafios a serem vencidos, inclusive com uma frase emblemática “as maquinas podem pensar” [19].

Acredito que, em cerca de 50 anos, será possível programar computadores, com uma capacidade de memória de cerca de 10^9 (TURING, 2009) tradução nossa.

O que é inteligência artificial? Este assunto tem intrigado várias áreas de estudo como a biologia, psicologia, filosofia, talvez uma maneira simples de definir é dizer que, a inteligência artificial ou “IA” é um campo de pesquisa da tecnologia que relaciona conceitos da fisiologia humana, alguns mais simples como os próprios sentidos, um exemplo é a visão e outros conceitos mais focados na capacidade do ser humano em conseguir raciocinar para tomar certas atitudes na solução de problemas complexos, por exemplo no caso da visão como conseguimos distinguir um objeto de outros, um individuo de outro, etc. A inteligência artificial nada mais é do que a tentativa de imitar o comportamento humano em máquinas programas [20].

O conceito aprendizagem de máquina é muito importante em IA pois ele agrupa os conceitos de treinamento que é; aprender a interpretar entradas que vem em conjuntos finitos ou infinitos, classificá-los e assim gerar uma resposta ou como é mais conhecido uma saída de dados.

Também existe o conceito de aprendizado por hábito que é a; capacidade de um computador realizar alguma função logica (A) e armazená-la como dado para posteriormente utilizá-la como referência para poder reagir a uma função semelhante a função (A), isso é conhecido em inteligência artificial como uma rede neural [20].

2.1.2 Machine Learnig

Machine Learning ou aprendizado de máquina é uma subárea da inteligência artificial, ela surgiu por volta dos anos 1980 junto com a impulsão de novos computadores quando eles começaram a evoluir em termos de hardware e processamento [21]. Em aprendizado de máquina, computadores são programados para aprender e evoluir com

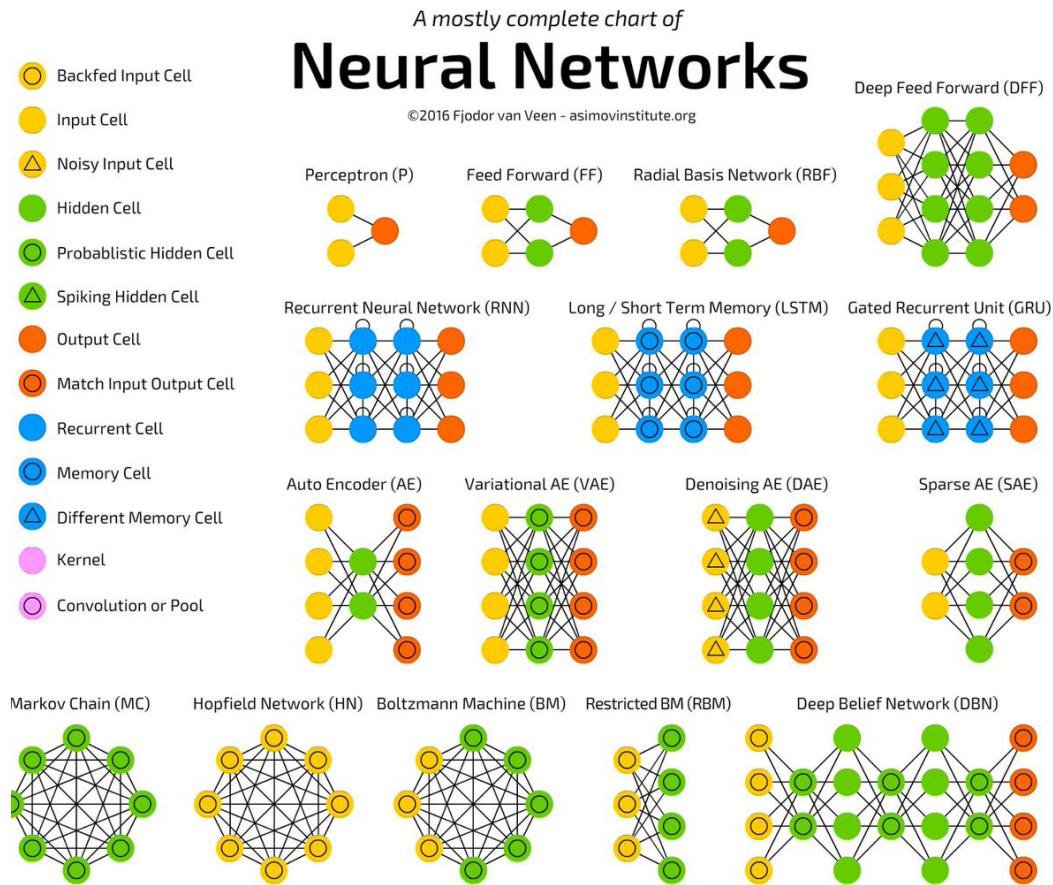
processos anteriores, para isso eles utilizam inferência ou indução, assim eles obtêm conclusões genéricas se baseando em exemplos armazenados nos processos anteriores. Assim o algoritmo aprende a induzir uma função que será um problema a ser resolvido, a partir de experiências anteriores que são chamados de dados [21]. Exemplos bem sucedidos:

- Reconhecimento de palavras;
- Predição de taxa de cura de pacientes;
- Detecção de fraudes em cartões de crédito;
- Veículos autônomos;
- Jogos de xadrez autônomo;
- Diagnóstico de câncer por análise de dados por expressão genética;

2.1.3 Deep Learning

Deep Learning ou aprendizado aprofundado é uma subárea do aprendizado de máquina é um conceito que utiliza redes neurais artificiais para que a máquina consiga não só resolver problemas, mas também aprender novas maneiras de resolvê-los [22]. É um conceito que estuda uma maneira de imitar o funcionamento dos neurônios humanos e contextualizá-los em modelos matemáticos, utilizando modelagem cognitiva a partir da mente humana. Esse estudo deu origem ao conceito de redes neurais artificiais e quando falamos de Deep Learning estamos nos referindo diretamente a redes neurais [22]. A figura 2 mostra os vários modelos de redes neurais desde o primeiro modelo que seria um perceptron que é basicamente um ramo de uma rede (foi o primeiro modelo), até conceitos mais profundos de redes neurais [22].

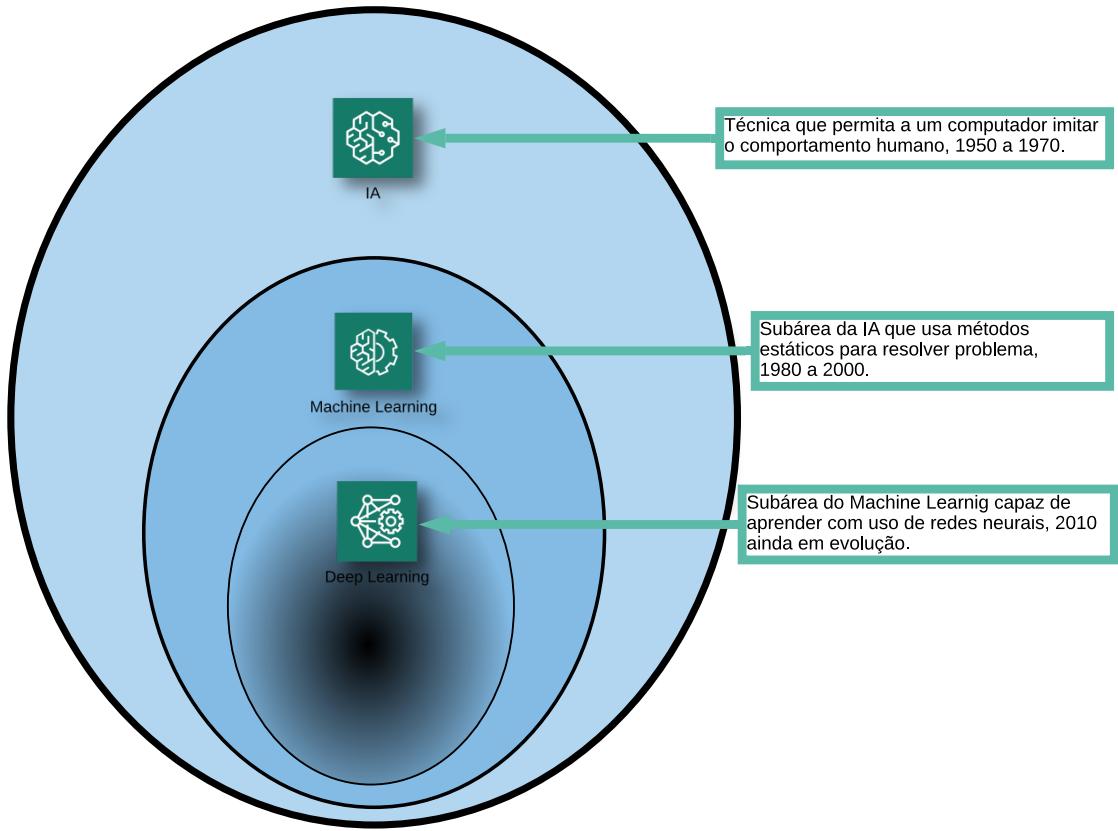
Figura 2 – Exemplo de Redes Neurais



Fonte: GitBook, ([SANTOS, 2018b](#)).

A figura 3 mostra um exemplo da diferença entre Inteligência Artificial, Machine Learning e Deep Learning, e quando aproximadamente elas começaram a ser abordadas e implementadas.

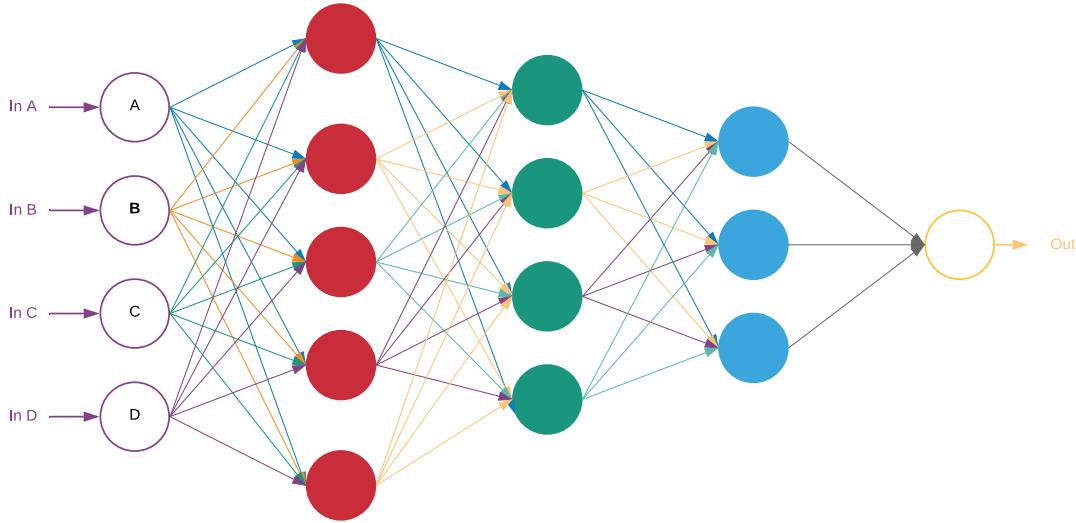
Figura 3 – Diferença entre IA, ML e DL



Fonte: do autor.

Uma rede neural nada mais é do que uma maneira de imitar o que ocorre no cérebro humana quando realizamos tarefas e assim ativamos neurônios que por sua vez ativam outros, e assim realizando uma reação em cadeia que gera o que chamamos de pensamento. A figura 4 é um simples exemplo de uma rede neural em máquinas aonde existem entradas, que ativam neurônios, que por sua vez se ligam com todos os outros neurônios e assim sucessivamente até chegarem em apenas uma saída [20].

Figura 4 – Rede Neural



Fonte: do autor com base em Oracle. ([TIERNEY ORACLE GROUNDBREAKER AMBASSADOR, 2018](#)).

2.1.4 Visão Computacional

Visão computacional é um campo da inteligência artificial que treina computadores para interpretar e entender o mundo visual. Através do uso de imagens digitais, câmeras e vídeos junto a modelos de Deep Learning, as máquinas podem identificar e classificar objetos corretamente, e então, reagir ao que elas veem. [10].

Os estudos com visão computacional beiram a época do início da internet, sim isso é verdade já naquela época se estudava este campo, porém tudo era mais difícil, devido à falta de tecnologias para suportar o processamento de grande volume de dados, faltava o volume de dados hoje conhecido como (Big Data) e algoritmos de processamento paralelo [9]. Nessa década vários fatores impulsionam o uso e estudo de visão computacional, tecnologias moveis saturaram o mundo com vídeos e fotos o (Big data), o poder computacional cresceu abruptamente e tornou-se mais barato, e novos algoritmos com redes neurais convolucionais aproveitam melhor a capacidade do hardware e do software [10].

Existem muitas tecnologias de visão computacional, hoje no mercado é comum a utilização de algumas tecnologias distintas, tais como:

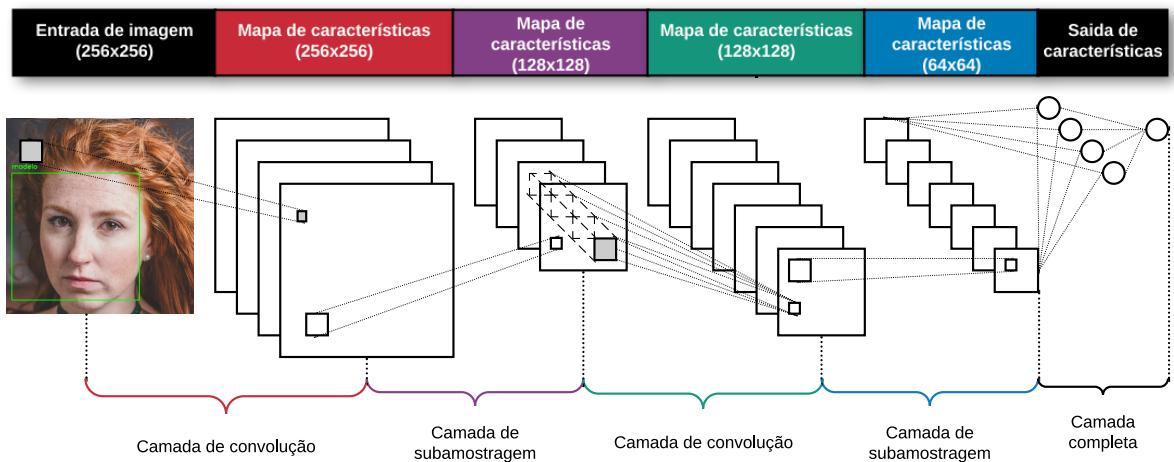
- **Segmentação de imagem:** Examina uma fração de uma imagem.
- **Detecção de objetos:** Detecta um ou mais objetos dentro de um campo de imagem.
- **Reconhecimento facial:** Detecção avançada de objetos que pode distinguir indivíduos.

- **Deslocamento dinâmico de objetos:** Detecta a reorganização de pixels dentro de um campo de imagem.

2.1.5 Redes Neurais Profundas (Deep Learning)

Aprendizagem Profunda ou Deep Learning, é uma subárea da Aprendizagem de Máquina, que emprega algoritmos para processar dados e imitar o processamento feito pelo cérebro humano [9]. Deep Learning nada mais é do que uma evolução das redes neurais, através da utilização de algoritmos de aprendizagem e camadas de aprendizagem cria-se modelos capazes de reconhecer padrões. Existem várias arquiteturas de redes neurais, uma é a rede neural convolucional profunda demonstrada na figura 5, este tipo trabalha muito bem com entrada de dados multidimensional ou espacial que é o caso do tratamento de imagens.

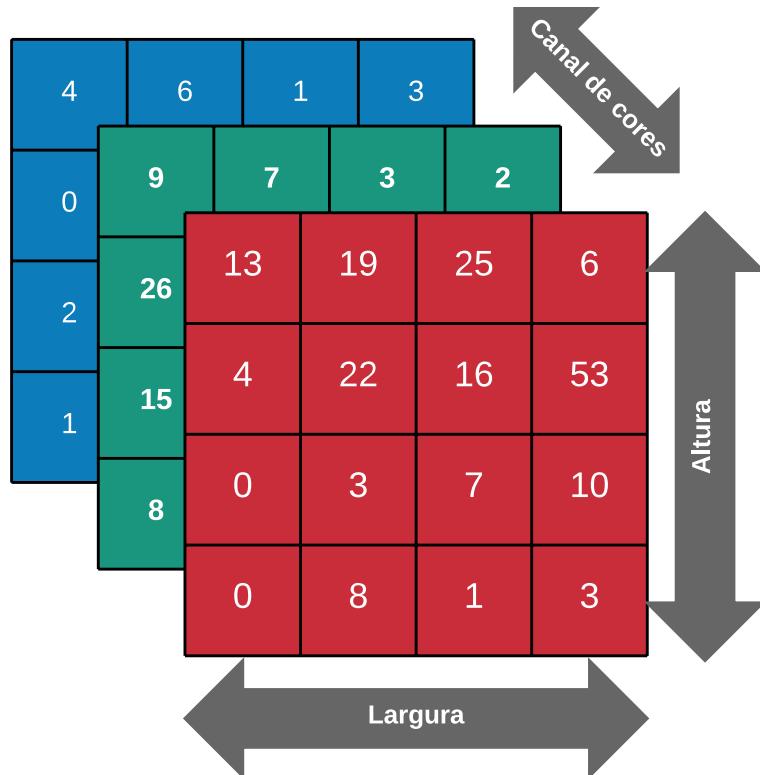
Figura 5 – Rede Neural Convolucional



Fonte: do autor com base em Lambda3, ([LONGARINI, 2019](#))

Em visão computacional as entradas são matrizes tridimensionais contendo altura, largura e profundidade, o que determina o padrão ou canal de cor do pixel, a figura 6 demonstra esse padrão.

Figura 6 – Pixel em Visão Computacional



Fonte: do autor.

Uma rede neural profunda necessita de um processo denominado de "treinamento" e para isso precisa-se de uma grande quantidade de dados (imagens), e através de uma abordagem supervisionada com as imagens devidamente marcadas como por exemplo sendo o rosto que buscamos, assim realizamos o treinamento. Então teremos um modelo que receberá imagens não marcadas e ele deverá ser capaz de classificá-las.

2.2 Tecnologias Open Source para Tratamento de Imagens

2.2.1 OpenCV

O OpenCV é uma biblioteca de código aberto licenciada por BSD que inclui várias centenas de algoritmos de visão computacional. Foi iniciada pela Intel em 1999, é uma biblioteca multiplataforma que concentra seu foco no processamento de imagens em tempo real, inclui implementações sem patentes e possui todos os algoritmos mais recentes de visão computacional. Em 2008, a Willow Garage assumiu o suporte e o OpenCV 2.3.1 foi lançado, possui uma interface de programação para C++, Python e Android.

O OpenCV possui uma estrutura modular, o que significa que o pacote inclui várias bibliotecas compartilhadas ou estáticas [12].

2.2.2 Biblioteca Dlib

O Dlib é um kit de ferramentas de linguagem de programação com licença *Boost Software License* iniciado em 2002 construído em C++ mas extensível para outras linguagens como Python, ela é utilizada largamente na indústria e academicamente para desenvolver sistemas complexos e que envolvam computação de alto desempenho. Hoje em dia ela é desenvolvida para lidar com tráfego de redes, threads, interfaces gráficas, estrutura de dados, e principalmente em aprendizado de máquina, processamento de imagens, mineração de dados e otimização numérica [27].

2.2.3 Biblioteca Face Recognition

É uma ferramenta de reconhecimento facial que possui simples utilização sobre a licença *MIT*, construída utilizando a biblioteca de aprendizado profundo de última geração Dlib, elas juntas chegam a uma precisão de detecções que chega até 99,38% segundo o site *Labeled Faces in the Wild*¹. É uma ferramenta muito simples de usar por ser baseada em chamadas de funções simples de identificar em sua documentação, porém é uma das melhores ou talvez a melhor ferramenta de detecção facial hoje no mundo. Com ela é possível não só detectar rostos, mas identificar partes da face como por exemplo; o nariz, boca, olhos, e com algumas outras integrações pode-se até detectar expressões faciais [28].

2.2.4 Nvidia Cuda

No início da computação os computadores sempre operaram com apenas um núcleo programável com códigos executados sequencialmente, porém a demanda por mais desempenho fez com que empresas empregassem esforços para criar tecnologias para melhorar o desempenho, e isso exigiu ainda mais do limite físico do silício. Para contornar isso foram criados os mecanismos de *pipeline*, *threads* e outros, uma alternativa foi adicionar processadores em paralelo e mais atualmente criaram processadores e gpus multicores com vários núcleos em um único chip. Porem foi necessário alterar as técnicas dos códigos que rodavam unicamente em paralelo para aproveitar esses novos recursos. Em 1999 a empresa americana NVIDIA ao perceber essa demanda por aumento no desempenho em um tipo de processamento específico, lançou sua primeira GPU a GeForce 256, e logo em 2000 criou as GPUs de uso geral (GPGPU). Inicialmente as GPUs tinham um processamento fixo focada em processamento de gráficos em três dimensões, mas nos últimos anos elas tem melhorado seu hardware focando no aspecto programável, e o resultado disso é uma grande capacidade de processamento focado em aritmética e não só em gráficos. Hoje as GPUs têm seu processamento focado em gráficos da classe SIMD, são desenvolvidas especificamente para cálculos de pontos flutuantes, usados em renderização de imagens, suas principais

¹<<http://vis-www.cs.umass.edu/lfw/>>

características são; massiva capacidade de processamento paralelo, total programabilidade e grande desempenho em cálculos que possua grande volume de dados. Para dominar esse conteúdo programático era necessário um grande conhecimento do desenvolvedor em várias APIs e da arquitetura das placas gráficas além da representação através de coordenadas de vértices, texturas e shaders, aumentando drasticamente o conteúdo programático a ser absorvido.

Foi ai que foi desenvolvida a ferramenta NVIDIA CUDA em 2006, ela permite ao desenvolvedor uma facilidade ao desenvolver utilizando recursos da GPU sem a necessidade de conhecer a multiplicidade de novos componentes de programação, e utilizando todo o poder de processamento das GPUs, ela também suporta várias linguagens de programação tendo como principais o C++ e o Python. A biblioteca Dlib utiliza os recursos de processamento do CUDA para melhorar drasticamente o desempenho em processamento e tratamento de imagens, e no treinamento de redes neurais que necessitam de cálculos que possuem grande volume e dados [29].

2.2.5 Reconhecimento Facial

Se reparar percebera que hoje em dia algumas redes sociais conseguem identificar seus amigos nas fotos automaticamente, ele marca com um retângulo os rostos das pessoas parece magica mas não é, essa técnica é que se chama reconhecimento facial, é uma tecnologia muito recente e incrível, por exemplo o FaceBook pode chegar a um acerto de 98% de preciso no reconhecimento (GEITGEY, 2016).

Apenas reconhecer amigos é muito fácil é uma utilização muito simples para essa incrível técnica, porem no desenvolvimento desse trabalho daremos uma utilização realmente útil para essa tecnologia. As técnicas de remoção de revestimentos faciais baseados em aprendizado profundo que utilizaremos, são altamente precisas e capazes de serem utilizadas em tempo real. Sera empregado o método de aprendizado métrico profundo "*deep metric learning*" que trabalho com imagens estáticas ou então fluxo de video, a biblioteca de reconhecimento facial (Dlib) gera um vetor de saída conhecido como 128-d, ou seja, ele gera um vetor de tamanho 128 que quantificam a face contendo valores numéricos reais ou pesos como é mais conhecido por quem tem conhecimento na área.

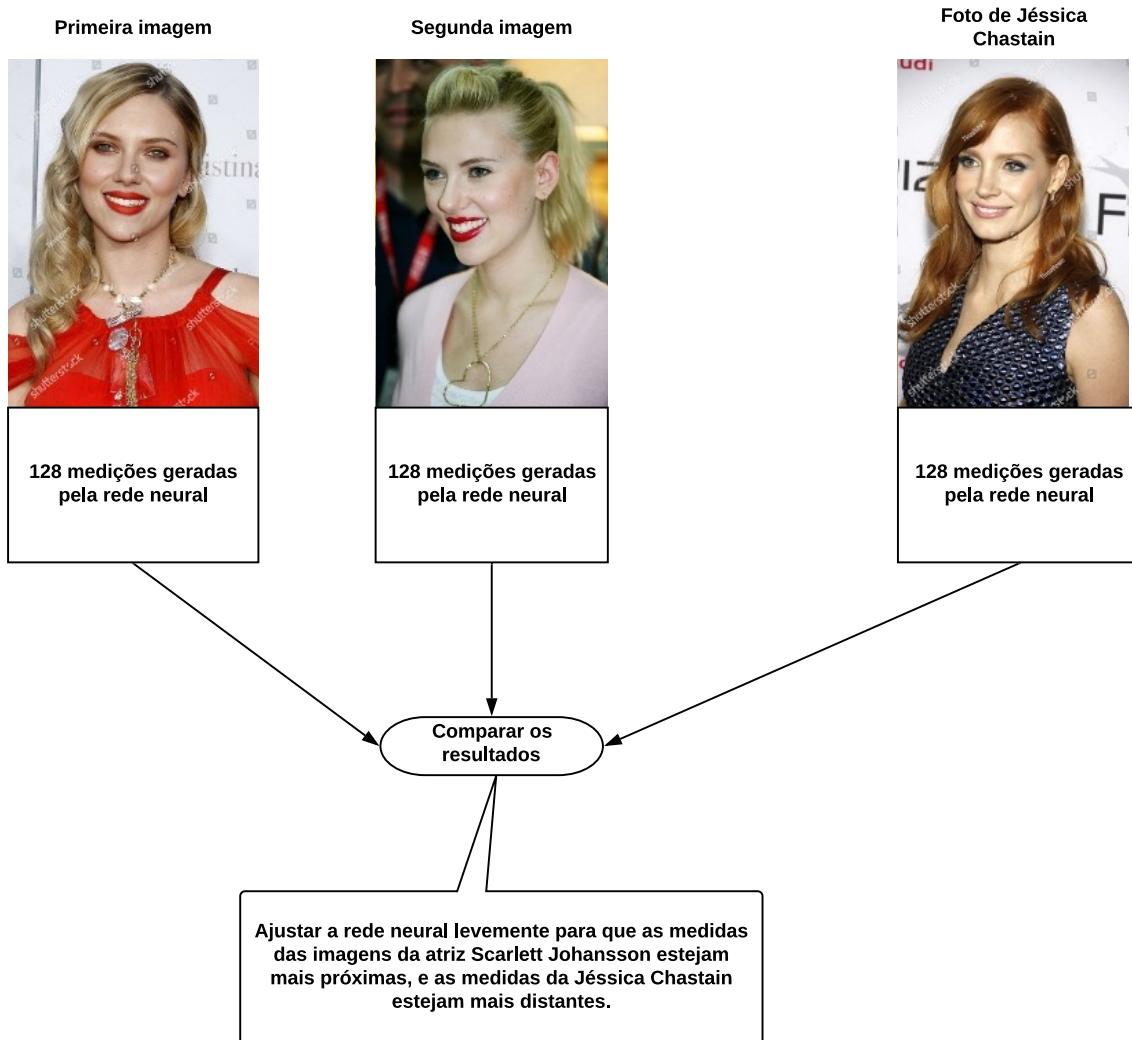
São chamados de pesos porque são eles que indicaram que uma imagem de entrada corresponde a uma pessoas adicionada no banco de imagens da rede neural, e ao comparar esses pesos ira gerar um resultado negativa (não é a pessoa) ou positivo (é a pessoa).

Para treinar uma rede neural utilizando essas técnicas é utilizado o modelo denominado de trigêmeos, para a técnica de reconhecimento facial utilizando aprendizado métrico profundo, e envolvera uma etapa de treinamento de trigêmeos.

Essa técnica consiste em três imagens de entrada, aonde ao menos duas das três

são da mesma pessoa, o (NN) gera um vetor de 128-d para cada imagem de rosto, e logo para as duas imagens de mesmo rosto, ajustamos os pesos da rede neural para tornar o vetor o mais próximo possível via métrica de distância ([ROSEBROCK, 2018](#)),

Figura 7 – Exemplo Simples de Treinamento de Rede Neural com Método de Trigêmeos



Fonte: do autor com base em ([GEITGEY, 2016](#)).

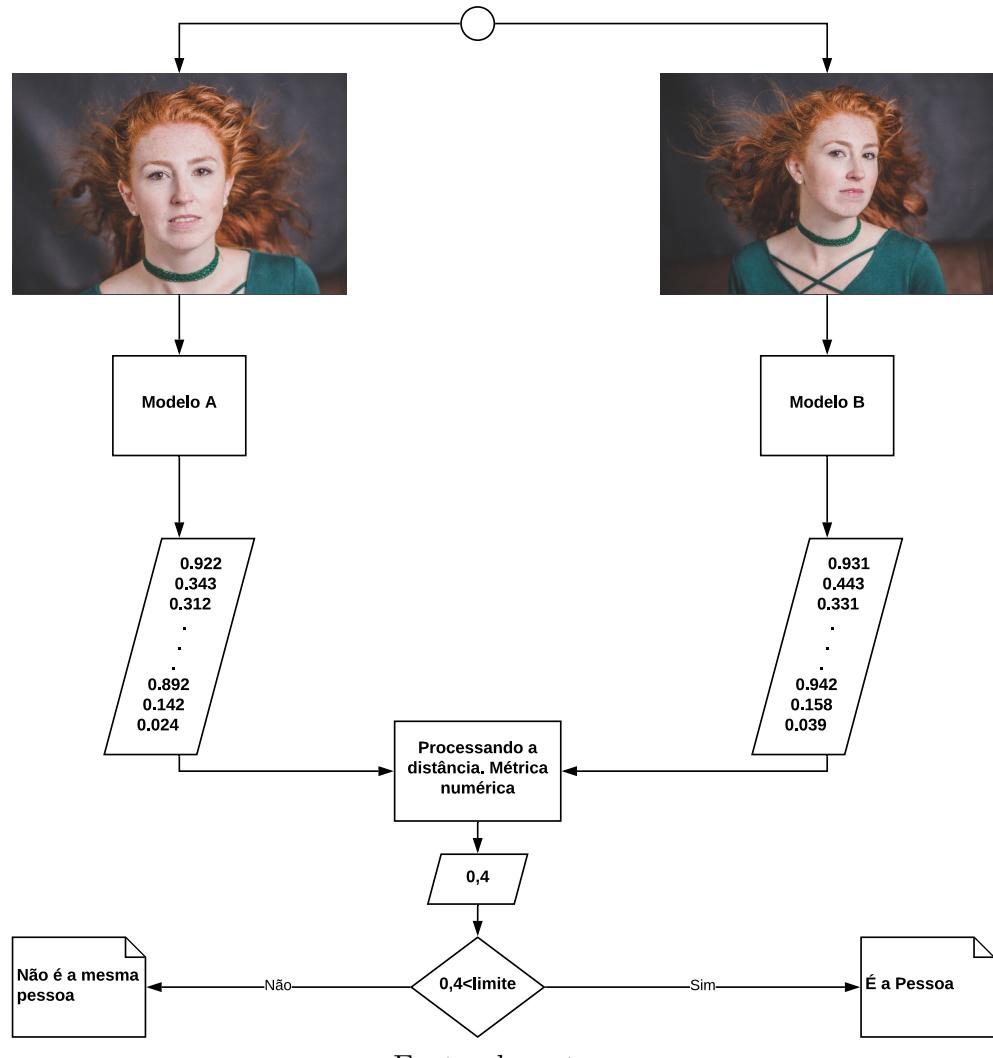
Na ilustração 7 esta uma representação que mostra uma simulação muito simples de como funciona o método com trigêmeos, duas das três faces são da atriz Scarlet Johansson que sera alvo no reconhecimento facial, e uma foto da atriz Jéssica Chastain que estará no nosso conjunto de dados. Então vamos considerar as faces, logo a ideia geral é que ajustaremos os pesos de nossa rede neural, para que as medias de 128-d das duas fotos da atriz Scarlet Johansson estejam o mais próximas o possível e mais distante da foto da Jéssica Chastain ([GEITGEY, 2016](#)).

A técnica utilizada com a biblioteca (Dlib) e (Face Recognition) é baseada no (ResNet-34) do artigo “*Deep Residual Learning for Image Recognition*” ([HE et al., 2015](#)),

mas com menos camadas e a metade do numero de filtros.

A rede (Dlib) foi criada e treinada por Davis King em um conjunto de aproximadamente 3 milhões de imagens, ela pode chegar a um acerto comparado a outras técnicas de ponta de aproximadamente 99,38% de precisão, e para concluir ela é uma rede (ResNet) com 29 camadas de convolução (KING, 2017).

Figura 8 – Exemplo de Comparaçao de Vetores 128-d



Fonte: do autor.

A figura 8 é uma representação simples de como se chega a um resultado positivo no reconhecimento facial, como podemos ver os modelos pré-treinados já com seus respectivos pesos de vetores 128-d são comparados, e se perceber os seus valores numéricos são bem próximos, perceba que os valores da direita e esquerda são bem próximos e que o limite é 0,4, logo são comparados e como no exemplo da figura 8 se seu resultado for inferior ao limite estipulado, isso indica que são imagens de uma mesma pessoa, gerando uma saída positiva ($Y=1$) (GEITGEY, 2016).

Na figura 9 foi realizada uma simulação utilizando o método de codificação de uma face com reconhecimento facial para extração de um vetor 128-d, e ajustamos o algoritmo para mostrar em tela os pesos que foram encontrados. Ele gera exatamente 128 valores numéricos reais.

Figura 9 – Extração de um Vetor 128-d com Reconhecimento Facial



Fonte: do autor.

2.3 Tecnologias de Linguagem de Programação para Cálculos de Álgebra e Geometria

2.3.1 Biblioteca NumPy

A biblioteca NumPy é hoje uma das principais ferramentas da linguagem Python utilizadas para cálculos numéricos e estruturas de dados, entre seus recursos ela possui cálculo de matrizes muito rápido e eficiente, funções que manipulam elementos da matriz e entre matrizes, possui ferramentas para leitura e gravação de dados de matrizes em disco, operações de álgebra linear, transformada de Fourier e uma confiável geração de números aleatórios [30].

O NumPy é uma biblioteca altamente otimizada para operações numéricas, ele suporta o OpenCV e todas as estruturas de matriz do OpenCV são convertidas para matrizes NumPy de e para ela. Portanto, quaisquer que sejam as operações que você possa realizar no NumPy, você pode combiná-lo com o OpenCV, aumentando o número de possibilidades. Além disso, várias outras bibliotecas como SciPy, que suporta NumPy podem ser combinadas [15].

2.3.2 Biblioteca SciPy

A biblioteca SciPy juntamente com a NumPy se tornam uma ferramenta extremamente madura, avançada e confiável para processamento de aplicações científicas. Há união dela com a NumPy estende muito além os recursos, e entre os recursos estão; Integração e derivações numéricas, decomposição de matrizes, solucionador de matrizes esparsas, e um grande conjunto de funções para trabalhar com cálculos que envolvem probabilidade e estatísticas.

2.4 Tecnologias Open Source de Desenvolvimento para Veículos Aéreos não Tripulados

2.4.1 Firmwares para VANT

2.4.1.1 Ardupilot

Segundo a site do desenvolvedor, o Ardupilot é o software de piloto automático de código aberto mais avançado, completo e com mais recursos disponíveis, foi desenvolvido ao longo de mais de 5 anos por uma equipe de diversos engenheiros profissionais e cientistas da computação. É o único software de piloto automático capaz de controlar qualquer sistema de veículo imaginável, de aviões convencionais, multirotores e helicópteros, barcos

e até submarinos. E agora sendo expandido para oferecer suporte a novos tipos de veículos emergentes, como quadriciclos e helicópteros compostos.

Instalado em mais de um milhão de veículos em todo o mundo e com suas ferramentas avançadas de registro de dados, análise e simulação, o ArduPilot é o *firmware* de piloto automático mais testado e comprovado. A base de código-fonte aberto significa que está evoluindo rapidamente, sempre na vanguarda do desenvolvimento da tecnologia. Com muitos fornecedores de periféricos criando interfaces, os usuários se beneficiam de um amplo ecossistema de sensores, computadores complementares e sistemas de comunicação.

O pacote de software é instalado em aeronaves de muitas empresas como; 3DR, jDrones, PrecisionHawk, AgEagle e Kespry, e também é usado para testes e desenvolvimento por várias grandes instituições e corporações, como NASA, Intel, Boeing, além de inúmeras faculdades e universidades em todo o mundo [14].

2.4.2 Ferramentas de Desenvolvimento para VANT

2.4.2.1 API de Desenvolvimento Dronekit para ArduPilot

O Dronekit é uma API feita para desenvolvedores criarem aplicativos que se comunicam com veículos (VANTS) através do protocolo MAVLink. É uma implementação em linguagem python para o controle de drones através de linguagem de programação, fornece acesso programático as informações de telemetria, estado e parâmetros a veículos conectados, permite o gerenciamento de missões e também o controle direto dos movimentos e operações do veículo.

O uso e direcionado diretamente a computadores complementares como Raspberry Pi², Google Coral Dev Board³ e Nvidia Jetson⁴ para implementar funções adicionais que a controladora de VANT não suporta, entre elas podemos citar a visão computacional e a modelagem 3D. Ele suporta Linux, Mac e Windows é disponibilizado sobre a licença de código aberto Apache 2.0.

2.4.3 Protocolo de Comunicação MAVLink

É possível criar uma ponte de comunicação entre o computador complementar (Raspberry Pi) e a controladora de voo (Pixhawk), e com ele pode-se capturar pacotes da controladora de voo como por exemplo; registro de telemetria, altitude, velocidade do veículo. Assim como se consegui capturar informações da controladora de voo, também é possível o contrário enviar comandos de voo para a controladora, e todas essas informações trafegam através do protocolo de comunicação MAVLink.

²<<https://www.raspberrypi.org/products/raspberry-pi-3-model-b/>>

³<<https://coral.ai/products/dev-board>>

⁴<<https://www.nvidia.com/pt-br/autonomous-machines/embedded-systems/jetson-nano/>>

O MAVLink é um protocolo de mensagens muito leve para comunicação em VANT (e entre componentes de VANT) [13].

O MAVLink segue um moderno padrão de design de publicação-assinatura e ponto a ponto híbrido, e os fluxos de dados são enviados/publicados como tópicos, enquanto os sub-protocolo de configuração como o protocolo de missão ou o parâmetro são ponto a ponto com retransmissão [13].

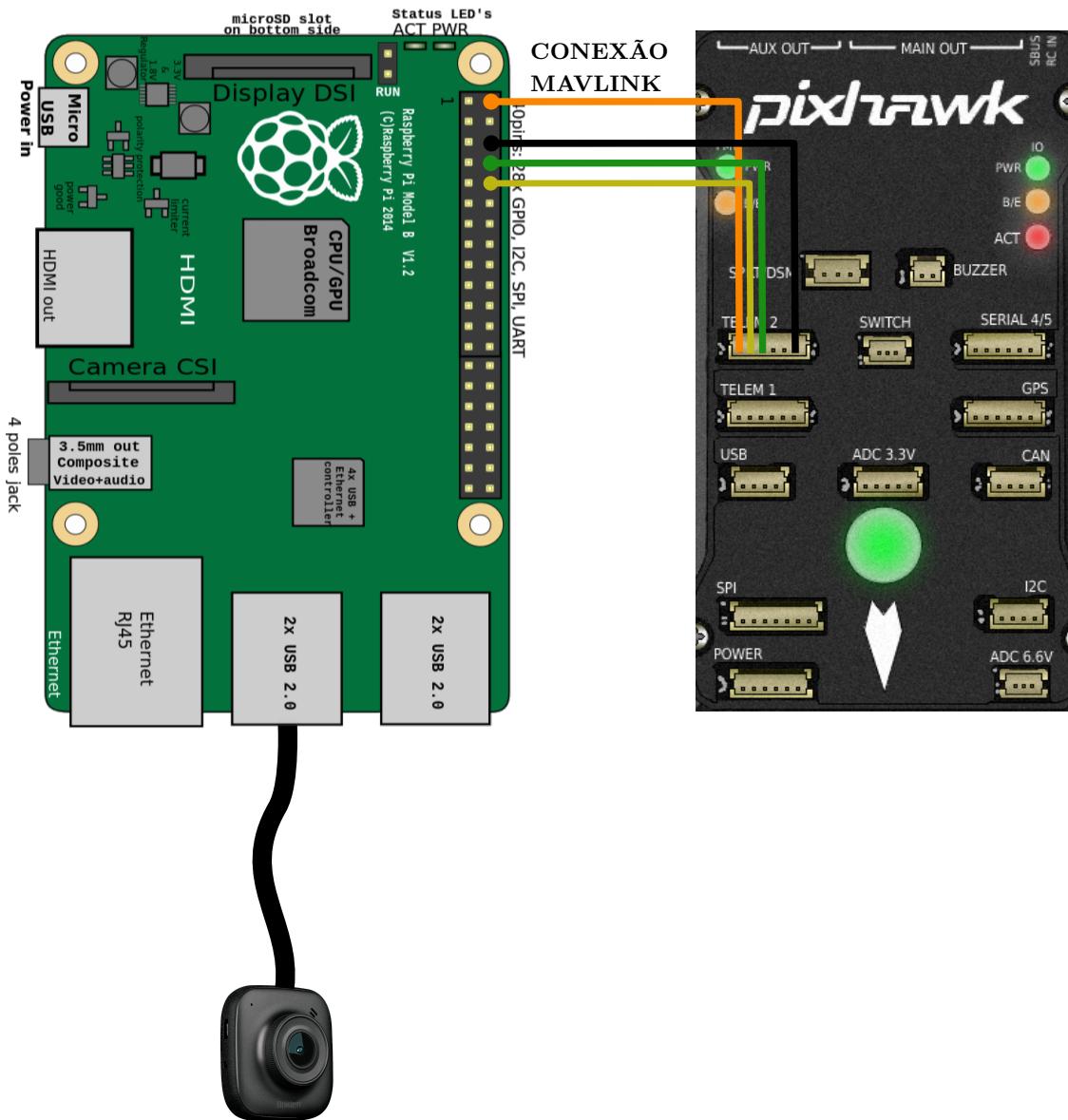
As mensagens são definidas nos arquivos XML, cada arquivo XML define o conjunto de mensagens suportado por um sistema MAVLink específico, também conhecido como "dialeto". O conjunto de mensagens de referência implementado pela maioria das estações de controle de solo e pilotos automáticos é definido em common.xml (a maioria dos dialetos é construída sobre essa definição) [13]. A cadeia de ferramentas MAVLink usa as definições de mensagens XML para gerar bibliotecas MAVLink para cada uma das linguagens de programação suportadas. VANTS, estações de controle de solo e outros sistemas MAVLink usam as bibliotecas geradas para se comunicar. Eles geralmente são licenciados pelo MIT e, portanto, podem ser usados sem limites em qualquer aplicativo de código fechado sem publicar o código-fonte do aplicativo de código fechado [13].

Características do MAVLink:

Muito eficiente; o MAVLink 1 possui apenas 8 bytes de sobrecarga por pacote, incluindo sinal de início e detecção de queda de pacote. O MAVLink 2 possui apenas 14 bytes de sobrecarga (mas é um protocolo muito mais seguro e extensível). Como o MAVLink não requer nenhum enquadramento adicional, é muito adequado para aplicativos com largura de banda de comunicação muito limitada.

Muito confiável; MAVLink é usado desde 2009 para se comunicar entre vários veículos, estações terrestres (e outros nós) em canais de comunicação variados e desafiadores (alta latência / ruído). Ele fornece métodos para detectar quedas de pacotes, corrupção e autenticação de pacotes. Suporta muitas linguagens de programação, executando em vários microcontroladores e sistemas operacionais (incluindo ARM7, ATMega, dsPic, STM32 e Windows, Linux, MacOS, Android e iOS). Permite até 255 conexões simultâneas na rede (veículos, estações terrestres). Permite comunicações externas e internas (por exemplo, entre um GCS e um VANT, e entre a controladora de voo e a câmera do VANT habilitada para MAVLink). A figura 10 demonstra o processo de comunicação e ligação física do MAVLink entre a Raspberry Pi e a Pixhawk.

Figura 10 – Conexão MAVLink da Raspberry Pi com Pixhawk



Fonte: Ardupilot Documents ([ARDUPILOT](#), 2019)

2.4.4 Proxy de Protocolo MAVLink para VANT (MAVProxy)

O MAVProxy é um software de proxy para comunicação entre estação terrestre (GCS) ground station software e VANT, ele é totalmente funcional para o controle de um VANT. Ele foi desenvolvido pela CanberraUAV, é minimalista, portátil e extensível para qualquer *firmware* de VANT que suporte o protocolo MAVlink, visto no índice 2.4.4, um exemplo é o firmware ArduPilot que será abordado mais à frente.

Ele foi desenvolvido para suportar computação complementar e vários datalinks, é hoje a ferramenta mais versátil para o controle e desenvolvimento de software baseados em VANTS, principalmente aqueles direcionados ao *firmware* Ardupilot. Muitos recursos

existentes em outras ferramentas GCS tem sua origem no MAVProxy [25]. O MAVProxy é liberado sobre a licença GNU (general public license) v3 ou superior.

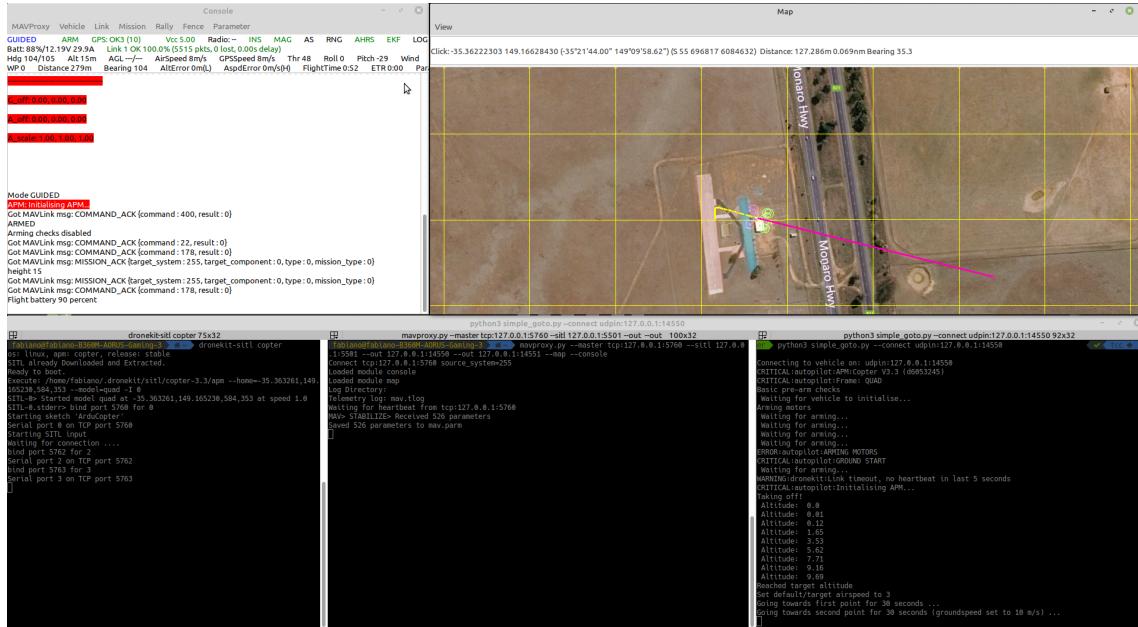
Recursos:

- É baseado em linha de comando muito focada em Linux. Existem plugins que fornecem uma GUI (graphical user interface).
- Poder ser executado em localhost ou em rede em vários computadores.
- Abrange as normas POSIX (portable operating system interface), ou roda em qualquer sistema operacional que tenha chamadas na linguagem python.
- Por ser muito leve roda muito bem em computadores com processamento limitado como o Raspberry Pi.
- Tab-completion of commands.
- Possui módulos carregáveis para várias tarefas como mapas, console e joysticks.

2.4.5 Simulador de VANT SITL

O simulador SITL (software in the loop) é uma ferramenta gráfica, no caso ele possui uma GUI (graphical user interface) que implementa a simulação de funcionamento total de um VANT, ele permite utilizar um plane (avião), copter (drone) ou um rover (veículo terrestre), e isso sem a necessidade de nenhum hardware, resumindo ele permite testar o comportamento do código desenvolvido com o Dronekit em um simulador de drone assim evitando possíveis imprevistos que poderiam causar algum dano material ao seu equipamento físico. A figura 11 mostra um exemplo do simulador rodando em um ambiente Linux, ele demonstra um drone executando um script em python que se chama "*simple_goto.py*", ou seja o script manda para o console do VANT duas coordenadas para a qual ele deve viajar e logo em seguida retornar para a home (ponto de partida).

Figura 11 – Exemplo de Funcionamento do SITL



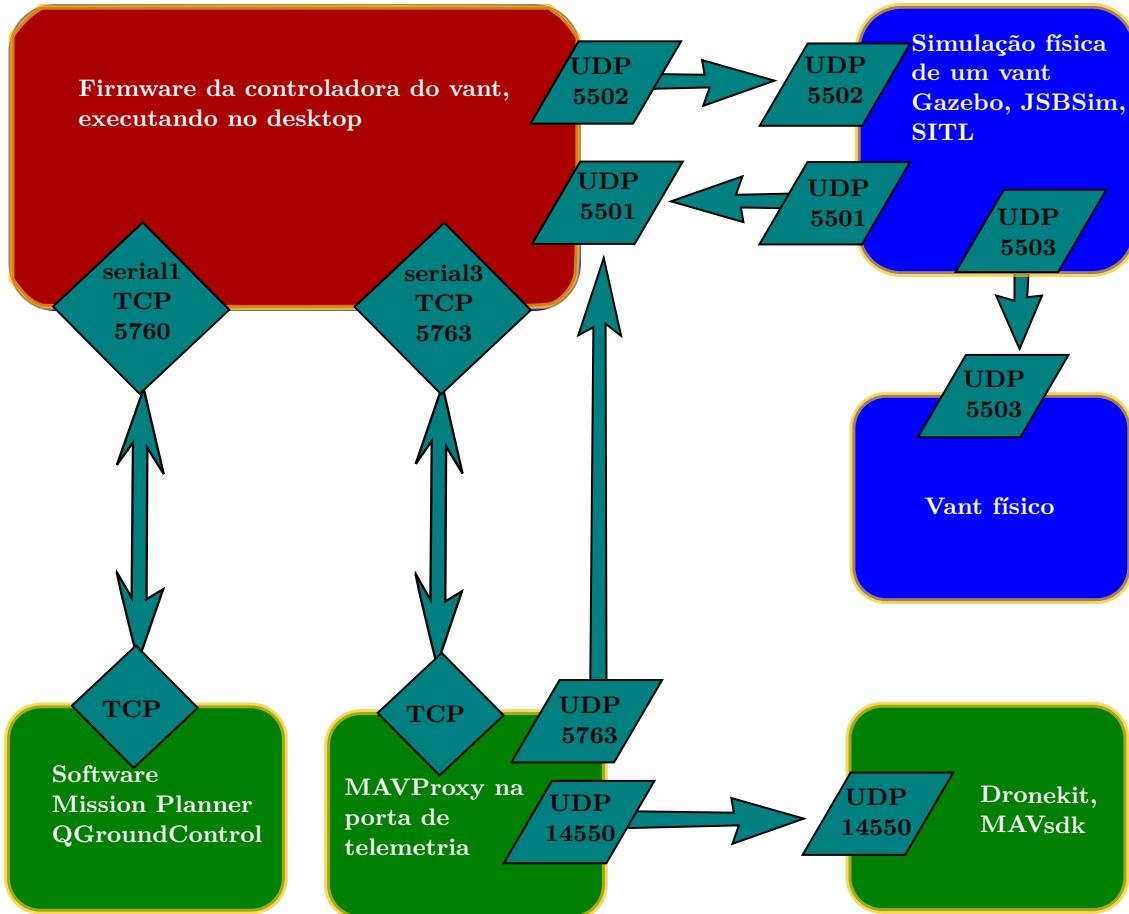
Fonte: do autor.

O SITL permite executar o *firmware* ArduPilot diretamente no seu computador (nootbook, ou desktop), suporta várias plataformas entre elas Linux, Mac e Windows. Com a junção do SITL e o ArduPilot se tem uma ampla gama de simuladores de veículos embutidos. Por exemplo:

- Drone (multi-rotor)
- Aeronave de asa fixa
- Veículo terrestre
- Veículo subaquático
- Gimbal para suporte de câmera filmadora
- E uma ampla variedade de sensores como, Lidars e sensores óticos

Com eles se torna mais fácil o desenvolvimento de aplicativos para drone, isso porque dispõe de uma gama de ferramentas como, analisadores estáticos, depuradores interativos e análise dinâmica. A simulação é uma maneira rápida, fácil e segura de testar códigos implementados para o ArduPilot antes de tentar usar (voar) no mundo real. [26]. A figura 12 mostra o fluxo de funcionamento das intercomunicações entre os módulos para criar simulações de funcionamento de um VANT.

Figura 12 – Fluxo de Operação SITL, MAVProxy, Ardupilot e MAVLink



Fonte: do autor com base em Ardupilot Documents, ([ARDUPILOT](#), 2019)

2.4.6 Simulador de Robótica Gazebo

O Gazebo é um simulador de código aberto utilizado principalmente em robótica experimental, ele é um simulador muito bem projetado sendo assim tornando possível testar rapidamente algoritmos, projetar robôs, treinar sistema de IA usando cenários extremamente realísticos. O Gazebo possui um poderoso e robusto mecanismo de simulação física, ou seja, simula vários fenômenos físicos como; aceleração da gravidade, temperatura, pressão, luminosidade, e terrenos, também dispõe de gráficos de alta qualidade e uma interface gráfica e programática implementada em C++ [31].

2.5 Tecnologias e a Anatomia de Veículos Aéreos não Tripulados

Segundo a site do Ardupilot um VANT do tipo quadricóptero é um veículo aéreo mecanicamente simples, cujo movimento é controlado pela aceleração ou desaceleração de várias unidades de rotor (motor/hélice).

Os quadricópteros são aerodinamicamente instáveis e exigem absolutamente uma controlador de voo para uma elevação estável. Como resultado, eles são sistemas "*Fly by Wire*" e se o computador não estiver funcionando, você não voara. O controlador de voo combina dados de pequenos giroscópios MEMs, acelerômetros (iguais aos encontrados em smartphones) para manter uma estimativa precisa de sua orientação e posição [14].

Figura 13 – VANT do Tipo Quadricóptero



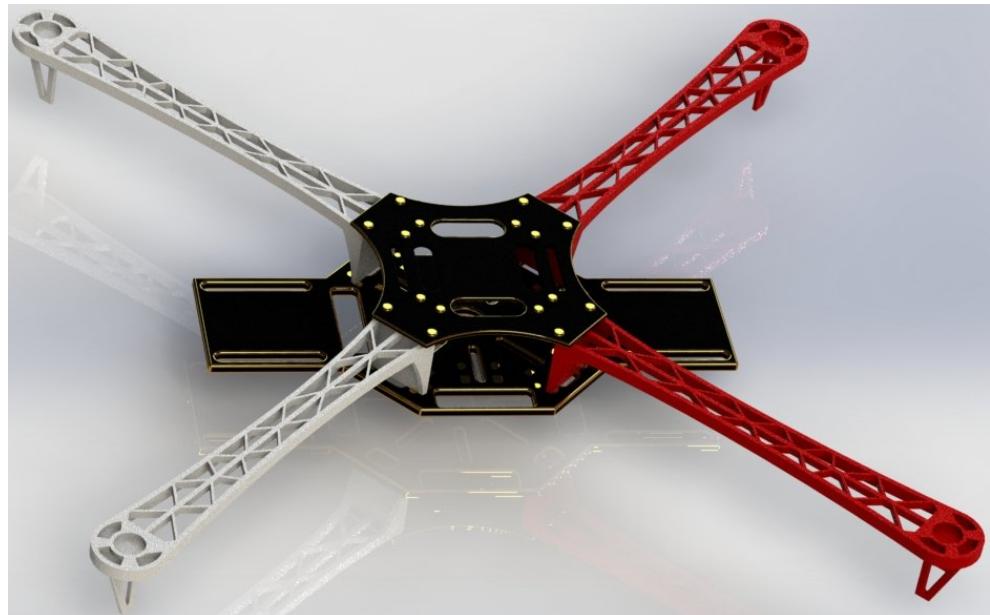
Fonte: do autor

O VANT do tipo quadricóptero exemplificado na figura 13, poderia ser utilizado em uma simulação (teste) físico do protótipo.

2.5.1 Frame

O frame é basicamente uma armação que pode ser de plástico, fibra de carbono, entre outros materiais leves, possui alguns modelos como quadricóptero (quatro hélices), hexacóptero (seis hélices) octocóptero (oito hélices) [3]. A figura 14 mostra um frame de um quadricóptero.

Figura 14 – Frame de um VANT Quadricóptero



Fonte: do autor

2.5.2 Controladora de Voo

Uma controladora de voo é um *hardware*, é basicamente um equipamento físico composto por um circuito integrado o qual possui sensores como barômetro, gps, giroscópio e acelerômetros. Eles captam informações externas como a pressão do ar, posição geográfica, posição de equilíbrio em relação a gravidade (nívelamento), convertem essas atividades em sinais digitais e dessa forma a controladora juntamente com o firmware (Ardupilot), conseguem interpretar esses sinais e utilizá-los para que se consiga manter o equipamento em equilíbrio no ar, tornando ele capaz de voar e manter sua estabilidade ([DRONECODE, 2020](#))^[7].

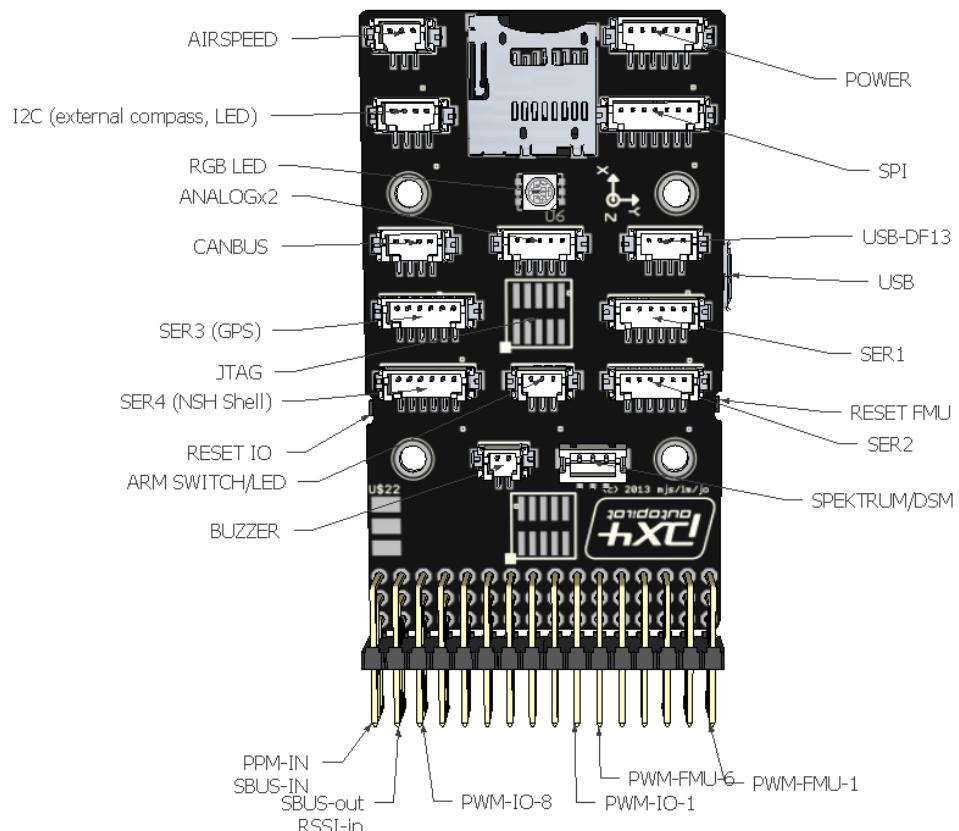
As controladoras de voo ou piloto automático são utilizadas para desenvolver equipamentos de controle remoto como um quadricóptero, veículos terrestres, aviões e submersíveis aquáticos. Originalmente foram fabricadas e vendido pela 3DR, hoje é possível adquirir cópias chinesas com custos bem mais em conta, devido ao projeto ser open source [3]. As figuras 15 e 16 mostram a Pixhawk 1 uma das controladoras mais utilizadas.

Figura 15 – Pixhawk 1



Fonte: PX4 Documentation, ([PX4.IO, 2020](#))

Figura 16 – Diagrama Eletrônico dos Conectores da Pixhawk 1



Fonte: Ardupilot Documents, ([ARDUPILOT](#), 2019)

- Processador:
 - Núcleo ARM Cortex M4 de 32 bits com FPU
 - 168 Mhz / 256 KB de RAM / 2 MB Flash
 - Co-processador à prova de falhas de 32 bits
- Sensores:
 - MPU6000 como principal acelerometro e giroscópio
 - Giroscópio ST Micro de 16 bits
 - Acelerômetro / bússola ST Micro de 14 bits (magnetômetro)
 - Barômetro MEAS
- Alimentação:
 - Controlador de diodo ideal com failover automático
 - Todas as saídas periféricas protegidas contra sobrecorrente, todas as entradas protegidas contra ESD
- Interfaces:
 - 5x portas seriais UART, 1 com capacidade de alta potência, 2 com controle de fluxo HW
 - Entrada de Satélite Spektrum DSM / DSM2 / DSM-X
 - Entrada Futaba S.BUS (saída ainda não implementada)
 - Sinal de soma PPM
 - Entrada RSSI (PWM ou tensão)
 - I2C, SPI, 2x CAN, USB
 - Entradas ADC de 3.3V e 6.6V
- Dimensões:
 - Peso 38g
 - Largura 50 mm (2,0 pol.)
 - Altura 15,5 mm (0,6 pol.)
 - Comprimento 81,5 mm (3,2 pol.)

2.5.3 Motor sem Escova

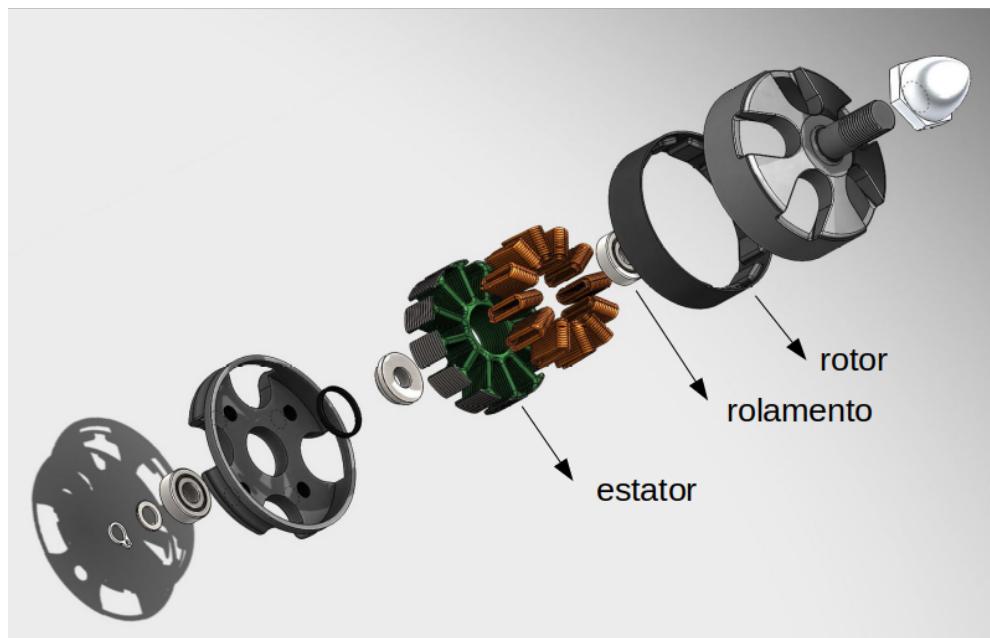
São motores de corrente elétrica continua de baixa tensão e sem escova, são síncronos e para funcionar necessitam de um (drive) ou inversor de frequência [24]. A figura 17 mostra um motor e na ilustração 18 as partes internas.

Figura 17 – Motor sem Escova



Fonte: Ardupilot Documents, ([ARDUPILOT](#), 2019)

Figura 18 – Partes do Motor



Fonte: do autor

2.5.4 Controlador Eletrônico de Corrente (ESC)

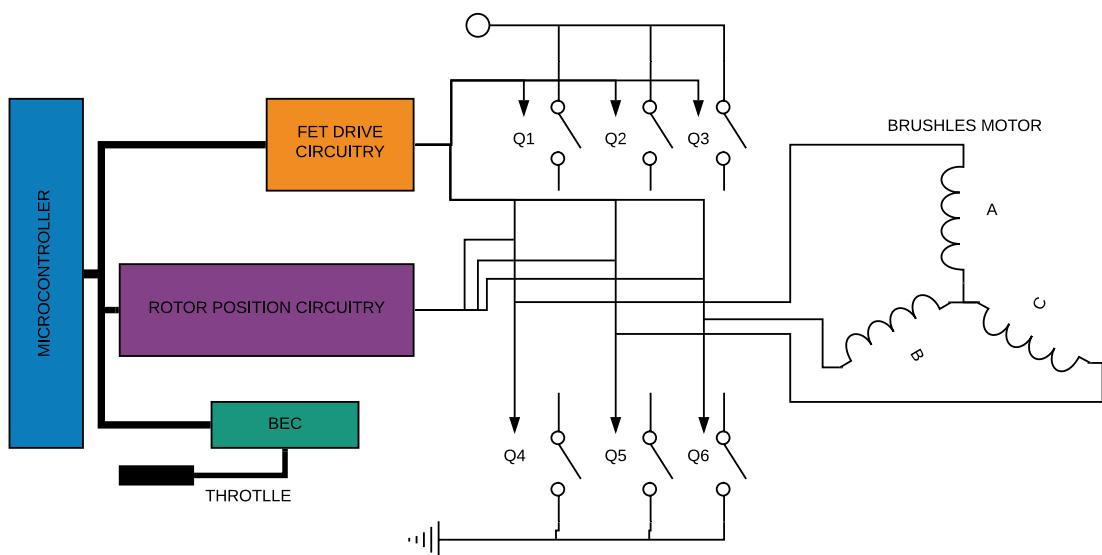
Círcuito eletrônico que controla a velocidade de rotação dos motores. A figura 19 mostra um (ESC) muito utilizado em quadricópteros comuns e na ilustração 20 o diagrama eletrônico.

Figura 19 – Controlador Eletrônico de Corrente utilizado em VANTs



Fonte: Ardupilot Documents, ([ARDUPILOT](#), 2019)

Figura 20 – Diagrama Eletrônico de um Controlador Eletrônico de Corrente



Fonte: do autor com base em ([TEFAY et al., 2011](#))

2.5.5 Módulo de GPS

Módulo de navegação GPS + Compass, é utilizado para a orientação do drone na superfície terrestre. A figura 21 mostra um modulo de GPS.

Figura 21 – Modulo de GPS

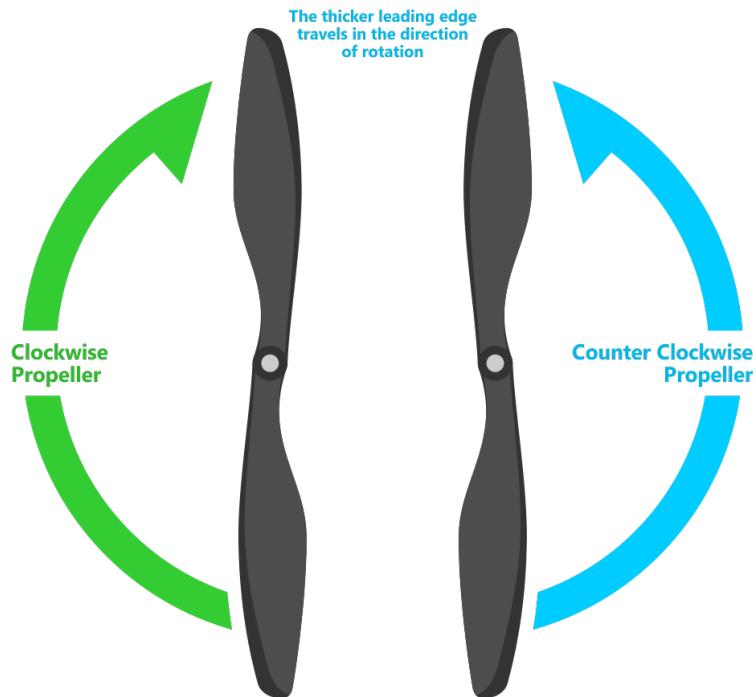


Fonte: Ardupilot Documents, ([ARDUPILOT, 2019](#))

2.5.6 Hélice

É um instrumento de propulsão ou tração, geralmente se acopla a um motor, ao girar empurra o que está na sua volta (ar ou água), assim converte energia rotacional em translacional, empurrando o objeto a que está engatada. Funcionam como asas e obedecem aos princípios de Bernoulli e a terceira lei de Newton. Podem ser fabricadas de resinas, plásticos, fibra de carbono entre outros materiais. A figura 22 mostra um exemplo de uma hélice.

Figura 22 – Hélice



Fonte: Ardupilot Documents, ([ARDUPILOT](#), 2019)

2.5.7 Bateria de Polímero de Lítio (LíPo)

As baterias de polímero de lítio, são baterias que contêm sais de lítio num polímero sólido como o óxido de polietileno em vez de solvente, isso as torna adaptáveis ou moldáveis a diferentes formatos. Cada célula tem tensão nominal de 3,7V, sendo que utilizam elas em série chegando a mais de uma especificação de alimentação, exemplo: 2s (duas células), 3s (três células). A figura 23 mostra uma bateria do tipo LíPo 30c.

Figura 23 – Bateria de Polímero de Lítio

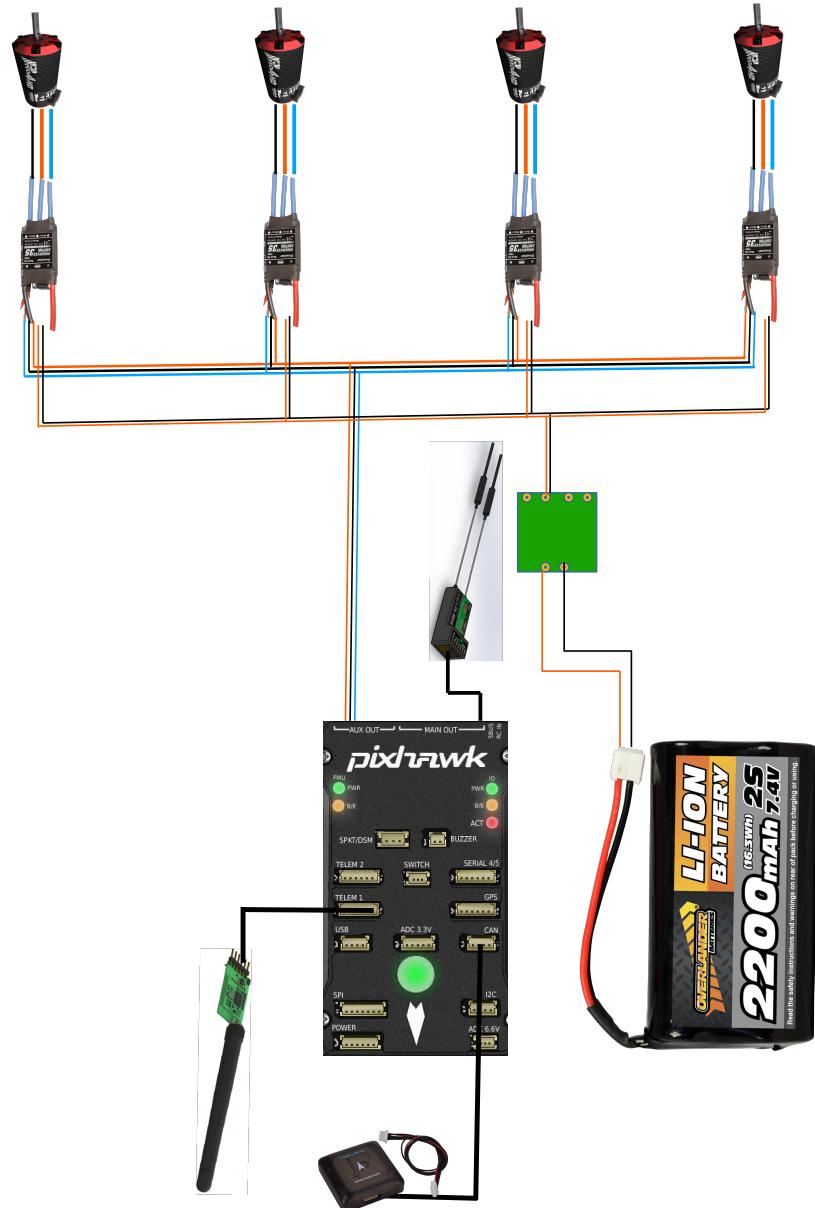


Fonte: Ardupilot Documents, ([ARDUPILOT](#), 2019)

2.5.8 Esquema de Montagem Básica de um Quadricóptero

A ilustração 24 é uma representação das conexões de montagem de um VANT do tipo quadricóptero mais básica possível.

Figura 24 – Esquema Básico de Montagem de um VANT do tipo Quadricóptero



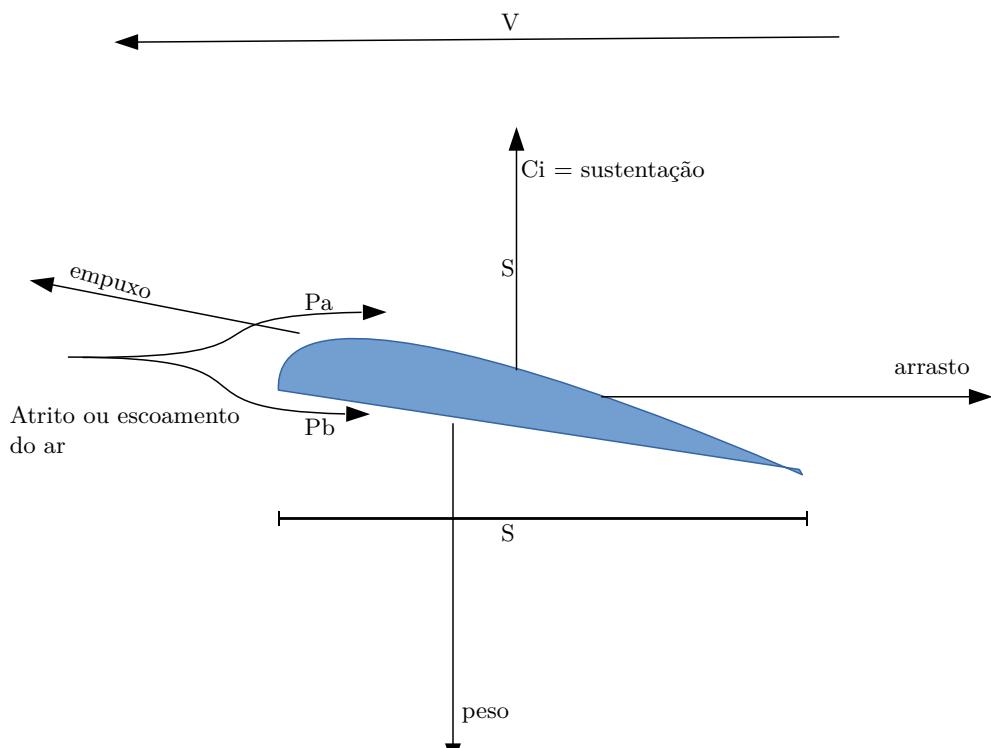
Fonte: do autor

2.6 Teoria de funcionamento e dinâmica de voo de um quadricóptero

Para iniciar a explicação deste tópico poderíamos começar falando sobre alguma das leis de Newton ou demonstrar através de cálculos que toda a ação possui uma reação de mesma intensidade em diferentes direções, porém a ideia é dar uma breve introdução das reações físicas que fazem com que um VANT do tipo quadricóptero voe.

E para iniciar é preciso entender como uma estrutura do tipo hélice age para que consiga elevar e sustentar o VANT, e para isso será ilustrada a figura 25 que mostra uma representação gráfica dessas reações.

Figura 25 – Teoria de Sustentação de uma Hélice



Fonte: do autor

Como pode se ver existem quatro reações principais, o empuxo, sustentação, peso e o arrasto, na figura 25 Pb e Pa representam a pressão aplicada na hélice, na parte de baixo da hélice Pb e na parte de cima Pa, essa pressão é exercida pelo deslocamento do ar que passa pela estrutura, basicamente a sustentação acontece pela diferença de pressão entre Pa e Pb, e o design da estrutura que faz com que a velocidade do ar embaixo da estrutura seja maior do que em cima.

- C_i = coeficiente de sustentação;

- P = densidade do ar;
- S = área da superfície da asa;
- V = velocidade do ar;
- L = força de sustentação;

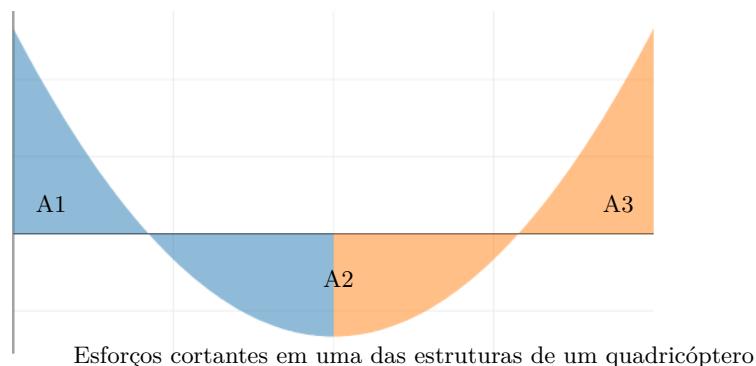
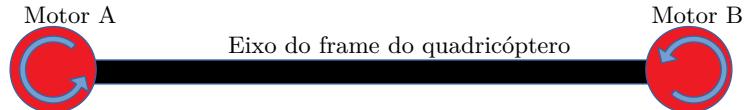
A equação 2.1 prova está sustentação:

$$L = Ci \left(\frac{p}{2} \right) S v^2 \quad (2.1)$$

Existem outras reações como o arrasto⁵, porem para este projeto é importante saber quais reações criam sustentação e fazem o veículo ganhar altitude vencendo a força da gravidade, essa reação é o empuxo⁶(CYPRIANO, 2014).

Agora vamos dar uma olhada nas reações aplicadas na estrutura do quadricóptero, como elas reagem, por exemplo o esforço cortante e o momento fletor⁷. E para expressar as reações a figura 26 e 27 mostra elas graficamente.

Figura 26 – Diagrama de Esforços cortantes na Estrutura do Quadricóptero



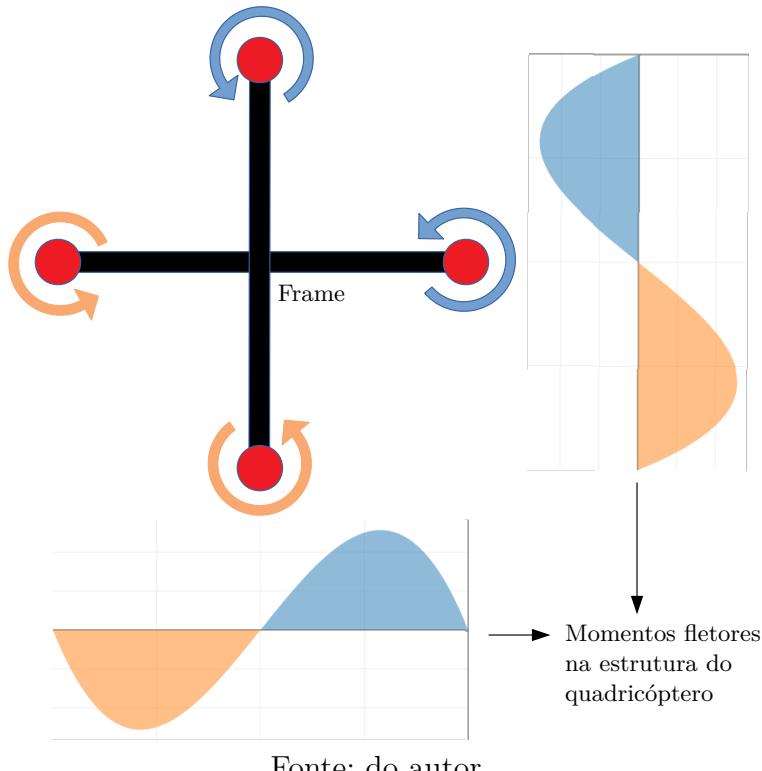
Fonte: do autor

⁵Reação que puxa a estrutura na direção posterior ou contraria a qual ela quer se deslocar

⁶força exercida pelas hélices para baixo que empurram o veiculo para cima

⁷Reações que a rotação do motor aplica sobre a estrutura do VANT

Figura 27 – Diagrama de Momento Fletor sobre a Estrutura do Quadricóptero



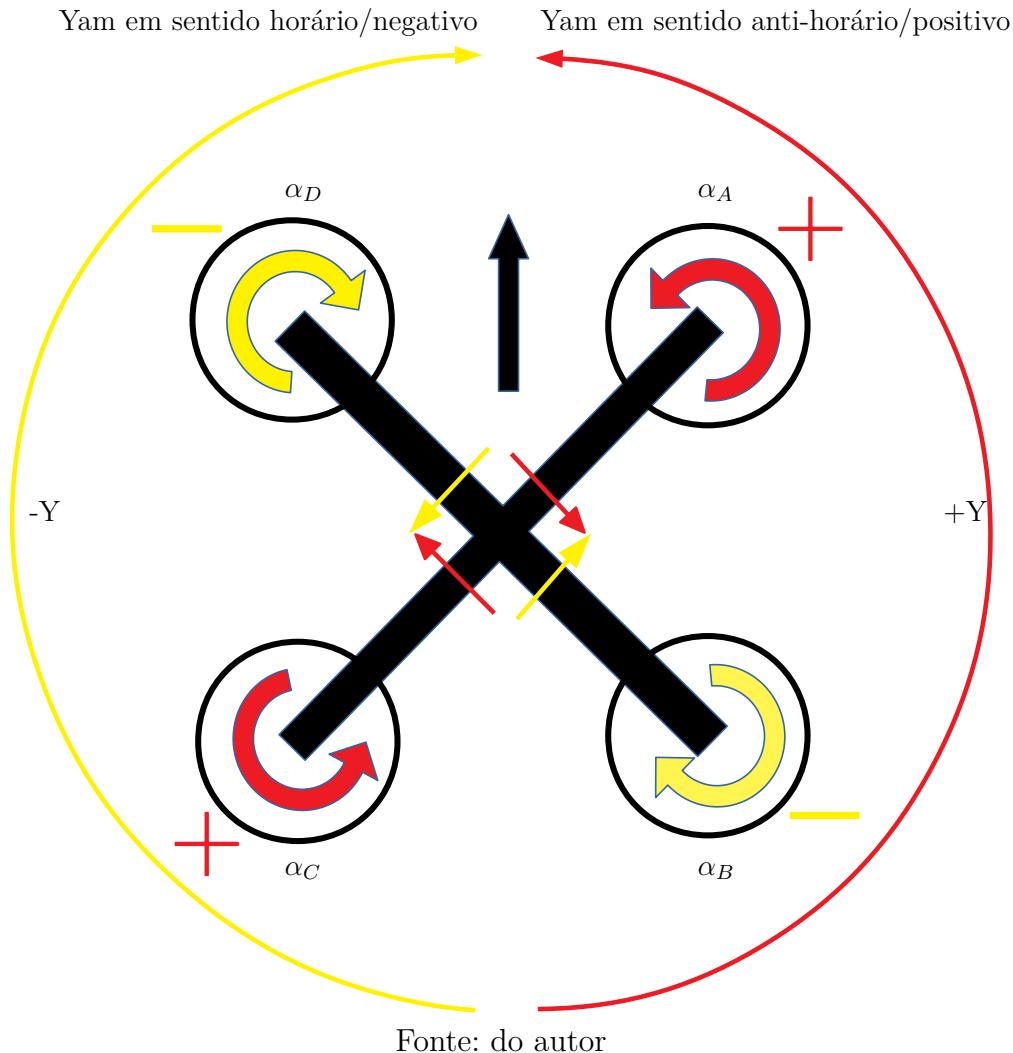
Fonte: do autor

Na figura 26 as reações que a rotação do motor geram na estrutura no frame dois pontos de esforço cortante, logo é possível observar que elas se cancelam, no caso a área de $A_1=A_2$ e $A_3=A_4$ o que faz com que elas se cancelam e com isso o equipamento não rotacione continuamente em um sentido. Já na figura 27 os gráficos de momento fletor demonstram que todas as forças no frame do quadricóptero acabam sempre se anulando. No ponto central do frame do quadricóptero o momento angular é zero dessa forma ele não rotaciona ([BURGGRÄF ALEJANDRO RICARDO PÉREZ MARTÍNEZ, 2019](#)).

No plano vertical um quadricóptero pode subir, descer e pairar, com isso é possível deduzir que para subir serão aplicadas forças nos motores de maneira que crie uma força ascendente que supere seu peso, e logo para pairar essas forças precisam se igualar ao máximo possível, mas tendo em mente que se igualarem é fisicamente impossível devido as varias reações da física que são exercidas no quadricóptero, logo para descer ele deve diminuir a velocidade de rotação dos motores de maneira que ascenda lentamente devido ao seu peso exercer empuxo gravitacional.

Na prática isso é simples os rotores giram jogando ar para baixo o ar empurra o equipamento para cima, e quem faz com que ele fique nivelado é a controladora de voo que simultaneamente com os sensores iniciais controlam a inclinação aplicando mais potência para cada motor independente de maneira que ele fique nivelado com a gravidade da terra, sim isso acontece muito rápido e freneticamente não é perceptível a olho nu ([MENG et al.,](#)

Figura 28 – Sentido de Movimento dos Motores e Inercias

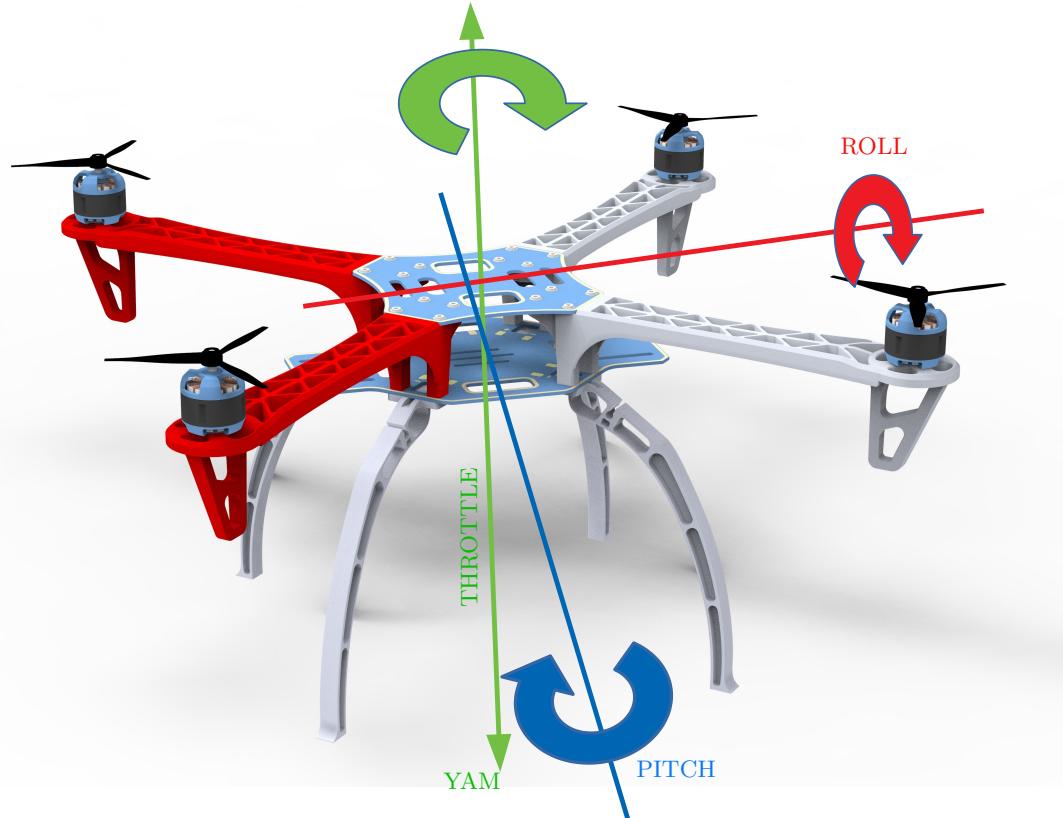


2018).

Até aqui já sabemos como o quadricóptero sobe, desce, e paira, também sabemos o porque dele não girar em apenas um sentido, agora sera abordada a dinâmica do quadricóptero, ação que faz do quadricóptero um equipamento incrível, tendo capacidade de se movimentar rapidamente e em direções variadas, para quem não sabe é possível realizar uma manobra de "loop" ou "flip", é a capacidade de dar um giro muito rápido virando de ponta cabeça e voltando rapidamente a posição inicial.

A figura 29 mostra todos os movimentos e seus respectivos nomes assim como são conhecidos, o "yam" é o giro que o quadricóptero da em cima do seu próprio eixo pode ser em sentido horário e anti-horário, "roll" é uma inclinação lateral em relação a frente fazendo com que ele vá para a direita ou esquerda e logo o "pitch" é a inclinação que faz com que ele se desloque na direção frontal ou traseira.

Figura 29 – Movimentos de um Quadricóptero



Fonte: do autor

O VANT de quatro hélices possui basicamente dois movimentos que estimulam deslocamento, um deles é a guinada que faz com que ele se desloque horizontalmente dentro do plano cartesiano, já o movimento de giro ele rotaciona mudando a sua posição frontal e traseira respectivamente, muda o sentido de orientação da frente do equipamento como por exemplo de norte para leste e o sensor que rege esse movimento é a bussola do modulo de GPS. O movimento de giro acontece alterando a rotação dos motores, dois ao mesmo tempo no caso, um exemplo é como rotacionar ele no sentido horário. Perceba que na figura 28 α_D e α_B são negativos e α_A e α_C positivos, logo se atribuirmos a α_D e α_A um valor numérico 2 e a α_C e α_B o valor 2 igualmente chegaremos a um resultado zero o que indica que o VANT esta pairando sem girar.

A soma dos torques dos motores gera a equação 2.2 que rege o movimento de Yam.

$$(k)(\alpha_D + \alpha_A - \alpha_C - \alpha_B) = Yd^2 \left(\frac{\theta Y}{dt^2} \right) \quad (2.2)$$

Vamos realizar outro exemplo de movimento, vamos supor que por exemplo queira girar em sentido horário, segundo a figura 28 e a equação 2.2 atribuiremos para α_C e α_A o valor 3 e a α_D e α_B o valor 1, então chegaremos novamente a zero o que indica que o

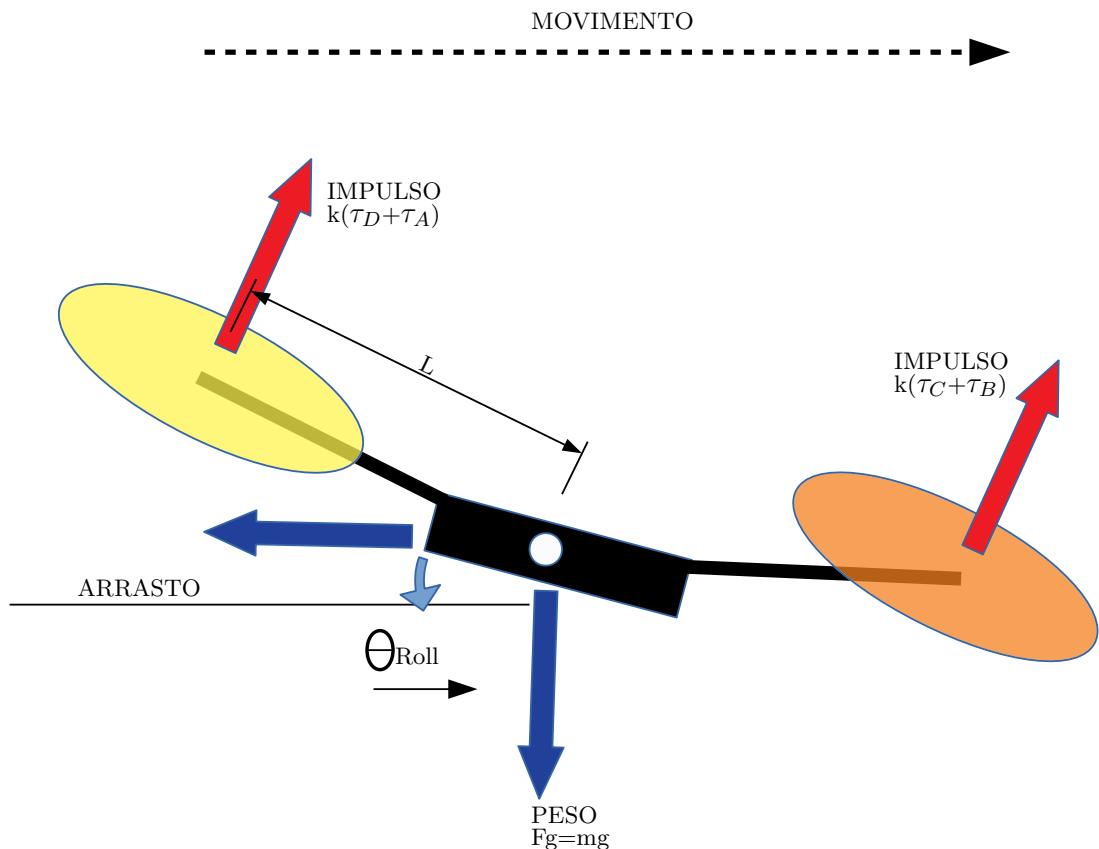
VANT esta estabilizado, porem a maior propulsão nos motores α_C e α_A faz com que ele gire no sentido horário ou seu Y sera negativo como a ilustração 28 mostra ([BURGGRÄF ALEJANDRO RICARDO PÉREZ MARTÍNEZ, 2019](#)).

A mesma equação apenas com uma pequena alteração pode ser aplicada para criar o movimento horizontal conhecido com "roll".

A soma dos momentos no centro de massa gera a equação que rege "roll".

$$Lk(k)(\alpha_D + \alpha_A - \alpha_C - \alpha_B) = Rd^2 \left(\frac{\theta R}{dt^2} \right) \quad (2.3)$$

Figura 30 – Movimento Horizontal "Roll"

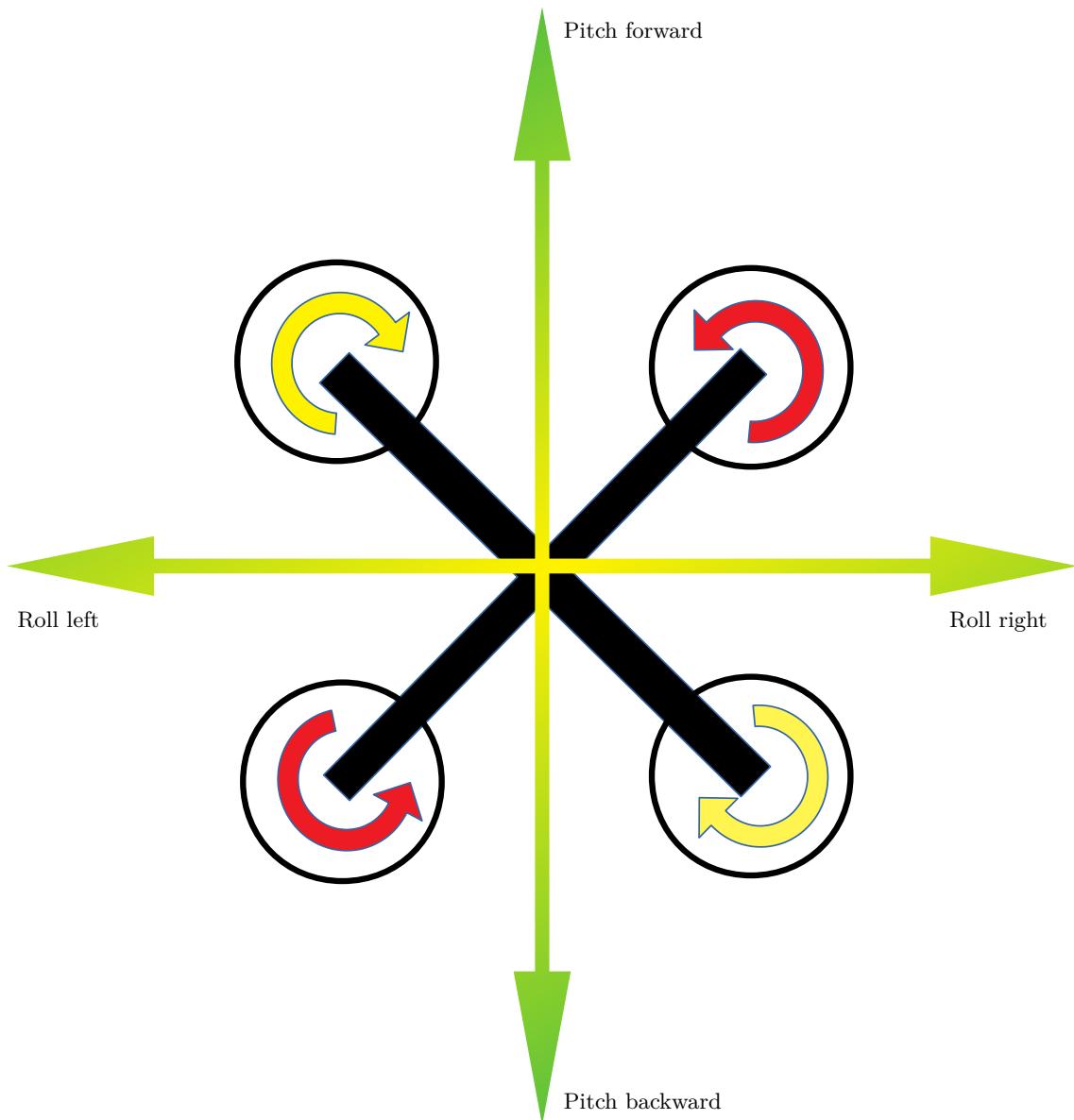


Fonte: do autor

A figura 30 ilustra uma vista lateral do movimento horizontal "roll" e forças que o regem, perceba que o VANT esta inclinado mas como se consegue coloca-lo nesta posição? O aumento do impulso nos motores α_C e α_B e diminuição nos motores α_D e α_A . O impulso total continua igual zero ou seu momento angular resulta em zero, porem a inclinação faz com que ele vá na direção em que esta inclinado, e a equação 2.3 e quem define esse

movimento horizontal. E para concluir a ilustração 31 denomina todos os movimentos horizontais de um quadricóptero (MSARDONINI, 2015).

Figura 31 – Movimentos Lineares de um Quadricóptero



Fonte: do autor

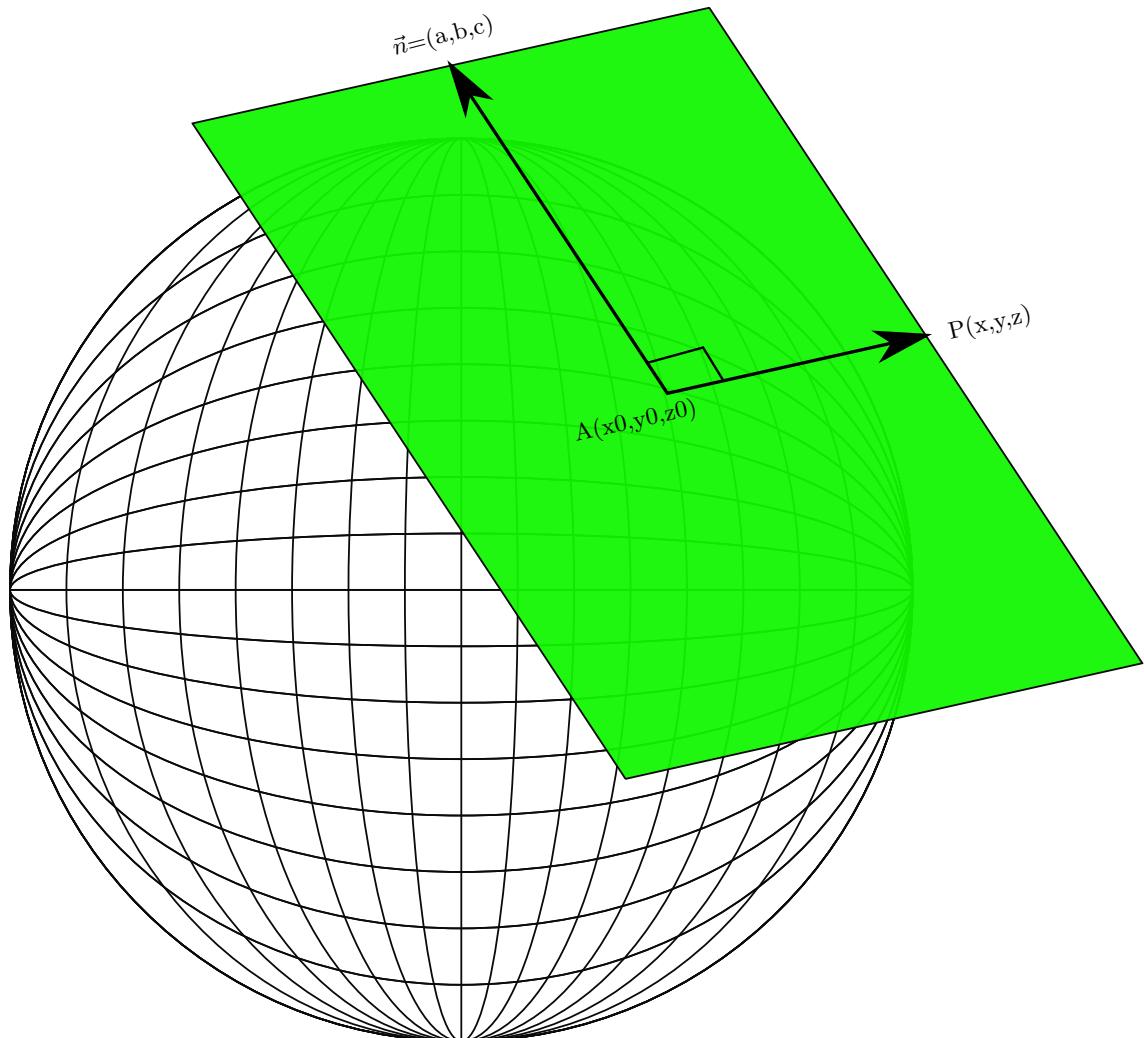
2.7 Sistema de Orientação de um VANT

Para orientar-se e se deslocar na superfície terrestre o VANT se orienta pelo sistema NED (North, East ,Down) que se baseia nos "cálculos de coordenadas do plano tangencial", o qual equaciona como um plano se desloca por uma superfície esférica, sendo regido e se parecendo com a equação 2.4. Isso é um conteúdo de geometria analítica e não é de difícil

entendimento sendo encontrado em qualquer material de álgebra básica e para melhor entendimento a ilustração 32 foi adicionada.

$$PT = a(x - x_0) + b(y - y_0) + c(z - z_0) \quad (2.4)$$

Figura 32 – Plano Cartesiano na Superfície Esférica



Fonte: do autor

Como foi abordado o VANT se orientar usando o sistema de coordenadas NED aonde N é o eixo Norte e Leste D é um eixo que aponta para baixo do VANT chegando próximo ao centroide da terra, são vetores de orientação usados na aviação aérea e cibernética marinha. Ele é usado comumente em sistemas GPS para movimentar-se ao redor do globo e são tangentes as linhas das coordenadas geográficas. A ilustração 33 mostra graficamente todas as referencias entre o frame de um VANT com o sistema de coordenadas:

- A tangente Leste-Oeste em paralelo com os paralelos;

- A tangente Norte-Sul em paralelo com os meridianos;

Também é possível observar na ilustração 33 como funciona o sistema de orientação do Dronekit e do firmware:

- O movimento pitch esta relacionado com o angulo de inclinação ϕ aonde uma rotação ante-horário o movimenta para frente tendo X negativo e uma rotação horário movimenta para traz tendo X positivo;
- O movimento roll esta relacionado com o angulo de inclinação λ aonde uma rotação ante-horário o movimenta para esquerda tendo Y negativo e uma rotação horário movimenta para direita tendo Y positivo;
- O movimento de rotação yaw esta relacionado com ψ , também pode ser chamado de azimute porque o giro do VANT em yaw ocorre ao longo do azimute da terra;

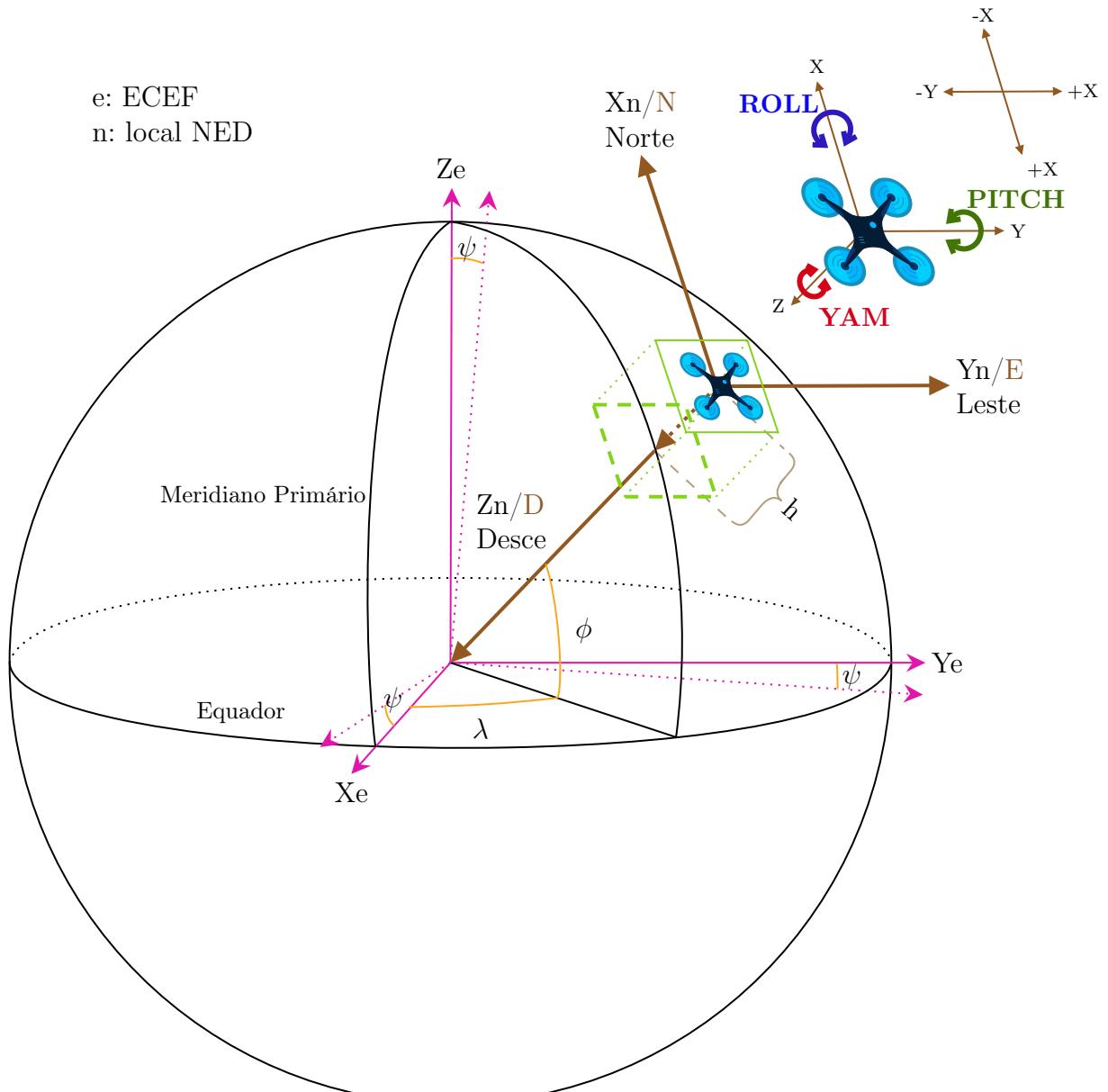
Existe uma relação entre o sistema NED e ECEF, aonde ECEF é o sistema padrão de coordenadas geodésicas e a relação de $n(X,Y,Z)$ com $e(X,Y,Z)$ é uma matriz de rotação do tipo 2.5.

$$R = \begin{pmatrix} -\sin(\phi) \cos(\lambda) & -\sin(\lambda) & -\cos(\phi) \cos(\lambda) \\ -\sin(\phi) \sin(\lambda) & \cos(\lambda) & -\cos(\phi) \sin(\lambda) \\ \cos(\phi) & 0 & -\sin(\phi) \end{pmatrix} \quad (2.5)$$

E a equação 2.6 relaciona ECEF com NED, aonde P é o ponto de intersecção do plano com a superfície da terra ou o centro do frame do VANT.

$$NED = R(ECEF - P) \quad (2.6)$$

Figura 33 – NED



Fonte: do autor com base em ([MUNGUÍA, 2014](#)), ([KISSAI; SMITH, 2019](#)), ([JSBSIM, 2020](#)).

Em NED "N"equivale ao eixo X (Norte ou Sul) direção frontal do VANT, "E"equivale ao eixo Y (Leste ou Oeste) direção lateral do VANT e "D"equivale ao eixo Z ou para baixo (aponta para o centro da terra), isso porque para uma aeronave o que interessa esta abaixo dela, e tem seu termino bem próximo ao centro da terra, h é a altura do VANT em relação ao solo ([KISSAI; SMITH, 2019](#)).

3 Metodologia

Segundo ([RAMPAZZO, 2005](#)) metodologia vem do grego (*methodos + logia*) ou "estudo do método", e método do grego (*methodos*) (*methà + odon*) quer dizer "o caminho para chegar", logo metodologia é o estudo de um conjunto de etapas dispostas em ordem para o estudo de uma determinada ciência e para alcançar um objetivo científico.

Neste capítulo, e etapa do estudo serão explanadas as simulações que foram realizadas para chegar a uma conclusão sobre o funcionamento do protótipo.

Foi determinado que os testes e simulação para validar o protótipo seria feito em módulos delimitados em funções específicas do protótipo, isso porque devido a algumas especificações como; tempo, custo verificou-se que não seria possível realizar um teste total do protótipo. Porem é devidamente valido e viável realizar testes modulares provando que a maior parte dos resultados esperados funciona.

O funcionamento total seria por exemplo um teste de campo utilizando um VANT real em um ambiente real enfrentando condições reais de temperatura, pressão, vento, luminosidade entre várias outras condições adversas que poderiam afetar o funcionamento. Porem depois de toda a pesquisa descobriu-se que testes de funcionamento que envolvam robótica ou mais especificamente um VANT, são realizados primeiramente com simulações utilizando softwares e ferramentas desenvolvidos para esse fim, logo optou-se por validar o protótipo com o uso destas ferramentas.

Devido a alguns objetivos que especificam a utilização de um computador complementar (Raspberry Pi) que sera acoplado ao VANT e sera responsável pela parte de visão computacional, foi realizada uma simulação do tipo *benchmark* rodando o código fonte de visão computacional comparando os resultados com o do computador utilizado para rodar os simuladores.

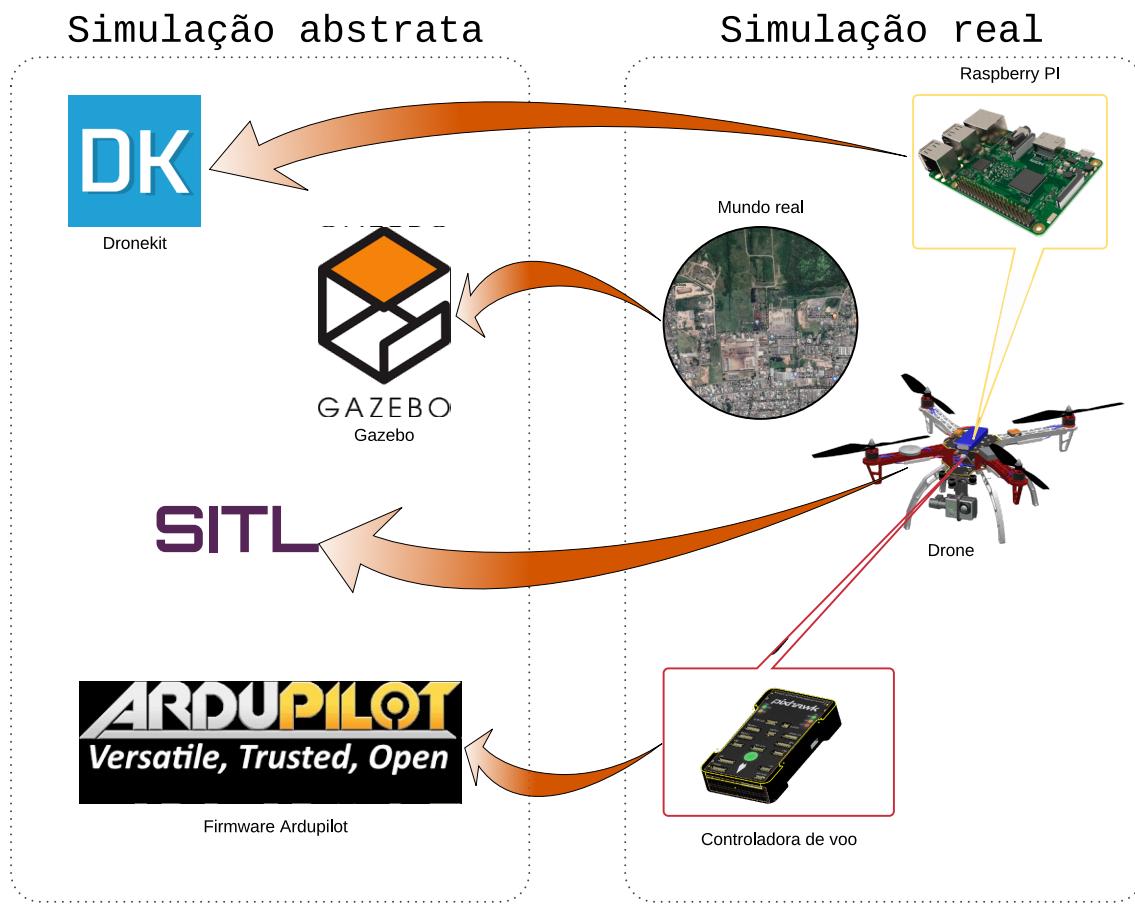
De maneira a auxiliar na compreensão de como funcionara o sistema a ilustração [34](#) representa como a simulação real é projetada para o sistema de simulação abstrato, perceba que existe uma divisão entre cada ferramenta. Iniciando de cima para baixo nos temos:

- O computador complementar (Raspberry Pi) que comportou a ferramenta de programação para VANTS Dronekit é abstraída pelo próprio Dronekit API que funcionara juntamente com o software de visão computacional.
- O ambiente físico, gravidade, vento, terreno e luminosidade no caso o local aonde o VANT seria pilotado, é abstraído pelo software Gazebo.

- O equipamento de voo quadricóptero é simulado pelo simulador de VANTs SITL que vem embutido no firmware Ardupilot.
- A controladora de voo Pixhawk que executa o firmware Ardupilot é simulada pelo próprio, que já possui embutido um sistema que possibilita testes e simulações através de software.

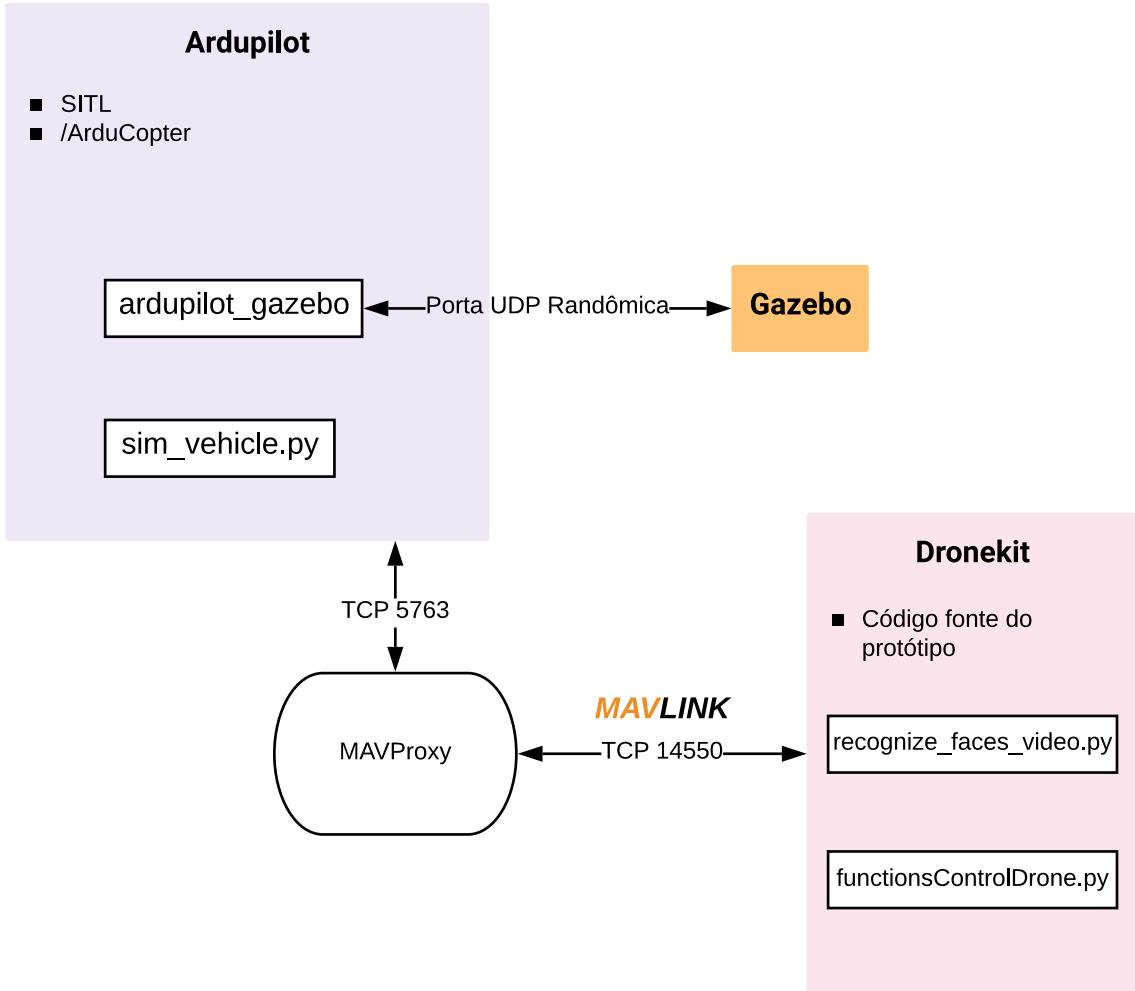
A arquitetura do sistema de simulação pode ser observado na ilustração 35 ficando mais claro como os módulos dos simuladores se integram.

Figura 34 – Simulação real para abstrata



Fonte: do autor.

Figura 35 – Arquitetura Básica do Sistema de Simulação



Fonte: do autor.

3.1 Métodos

Para uma melhor compreensão do funcionamento do sistema foram desenvolvidos diagramas, fluxogramas e gráficos que explicam modularmente cada processo. Simulações com software foram utilizadas para testar o funcionamento integrado de alguns desses módulos.

Uma simulação benchmark de hardware também foi realizado para comparação e validação do protótipo.

Os módulos foram subdivididos, um deles será o de visão computacional que validará o reconhecimento facial, esse módulo vai validar as funções de distinção entre um humano inserido no banco de busca e outros que não foram, e um denominado de controle e automação que de acordo com o tema do trabalho é o mais importante por determinar toda parte de controle do VANT.

3.2 Protótipo

O protótipo consistem em um sistema que determinara se é possível utilizar visão computacional, ou mais especificamente o método de reconhecimento facial, para controlar um veiculo do tipo VANT, ou seja desenvolver um sistema de controle e automação para essa finalidade. Primeiramente seria utilizado partes físicas de um VANT do tipo quadricóptero acompanhando uma controladora de voo (Pixhawk) com o *firmware* Ardupilot, e a parte de visão computacional estaria em um computador complementar (RaspBeryy Pi), os dois seriam interligados utilizando protocolo MAVLink. Porem se optou por utilizar um conjunto de ferramentas de simulação para realizar os testes funcionais.

Essas ferramentas de simulação, simulam todo o hardware que seria utilizado no caso de simulações reais físicas.

Como todas as ferramentas são implementas em linguagem python, a codificação do algoritmo também foi desenvolvida em python para uma melhor integração das ferramentas, módulos e bibliotecas.

Foi determinado que o sistema se divide em duas partes; visão computacional responsável pela etapa de reconhecimento facial. E o sistema de controle e automação; responsável por toda parte que implementa o sistema de controla e automação do VANT.

No diagrama 36 é possível observar mas detalhadamente as duas partes aonde cada circulo representa uma etapa da logica de processo do algoritmo, e a intercessão entre os círculos representam a ação que determina a mudança de estado entre processos.

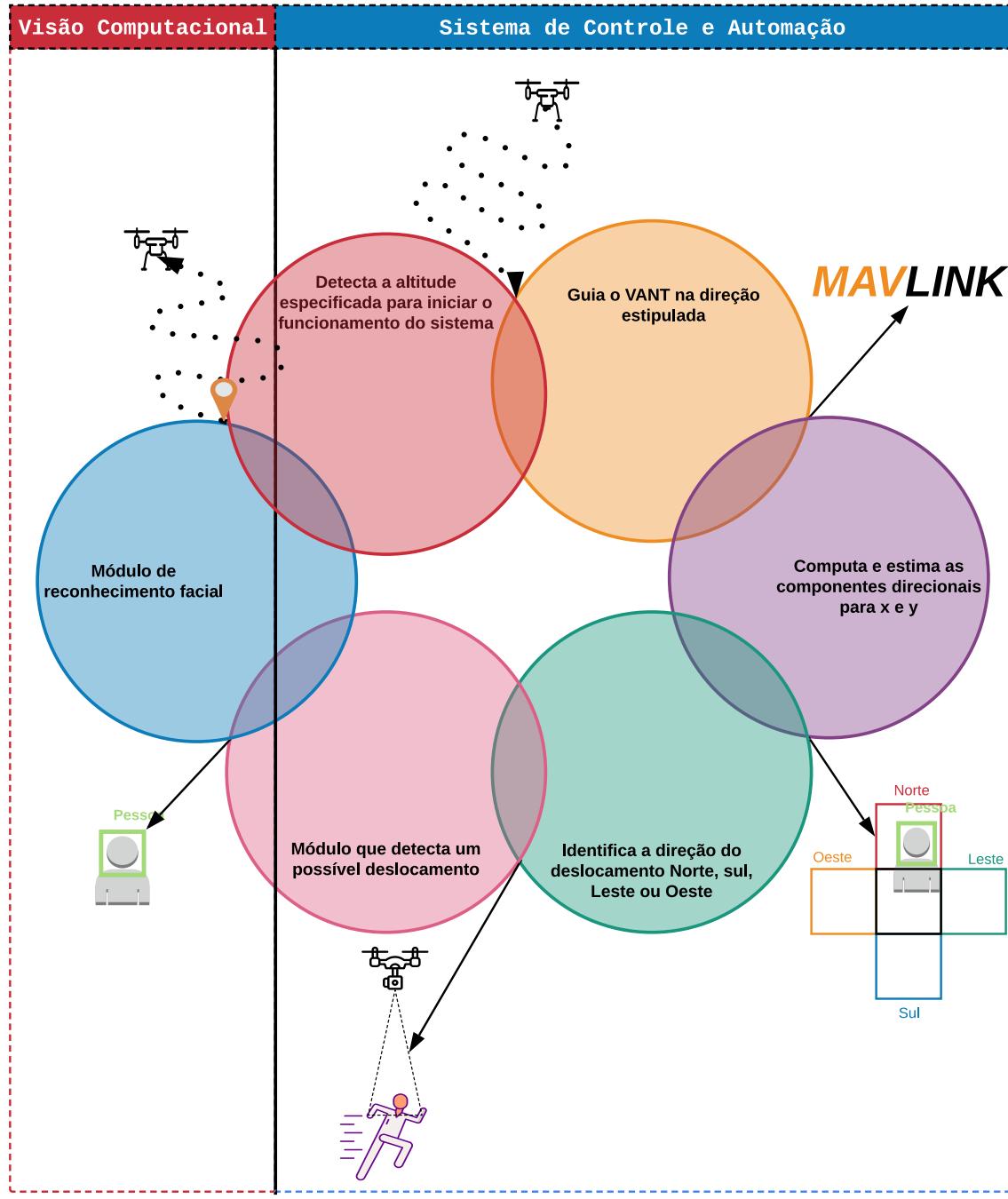
O inicio do sistema poderia ser definido de varias maneiras, porem foi determinado que ao atingir uma altitude especificada daria inicio ao sistema iniciando o módulo de reconhecimento facial, quando uma face for reconhecida o algoritmo detecta se ouve uma mudança de posição que represente um deslocamento justificável, caso seja verdadeiro o deslocamento, é identificado para qual região de interesse se deslocou, logo se o deslocamento for em direção a alguma das regiões (Norte, Sul Leste, Oeste), é enviado para o VANT a direção em que ele deve se movimentar, porem caso o deslocamento seja na diagonal, o sistema computa os valores das componentes de velocidade X e Y enviando para o VANT.

3.2.1 Fluxogramas dos Protótipos e Funcionamento do Sistema

Como ja foi abordado antes a ilustração 36 representa o sistema em uma visão macro e como um todo dividida em dois blocos (controle e visão computacional) e dentro deles quais etapas fazem parte, assim como as ações que acarretam em mudança de etapa. E para explanarmos mais detalhadamente o funcionamento do sistema, três diagramas UML representaram em visão midi como funciona cada parte, primeiro o digrama UML 37 mostra o sistema em um todo, o 38 as etapas de visão computacional e o 39 a parte de

controle.

Figura 36 – Diagrama de Funcionamento do Sistema



Fonte: do autor.

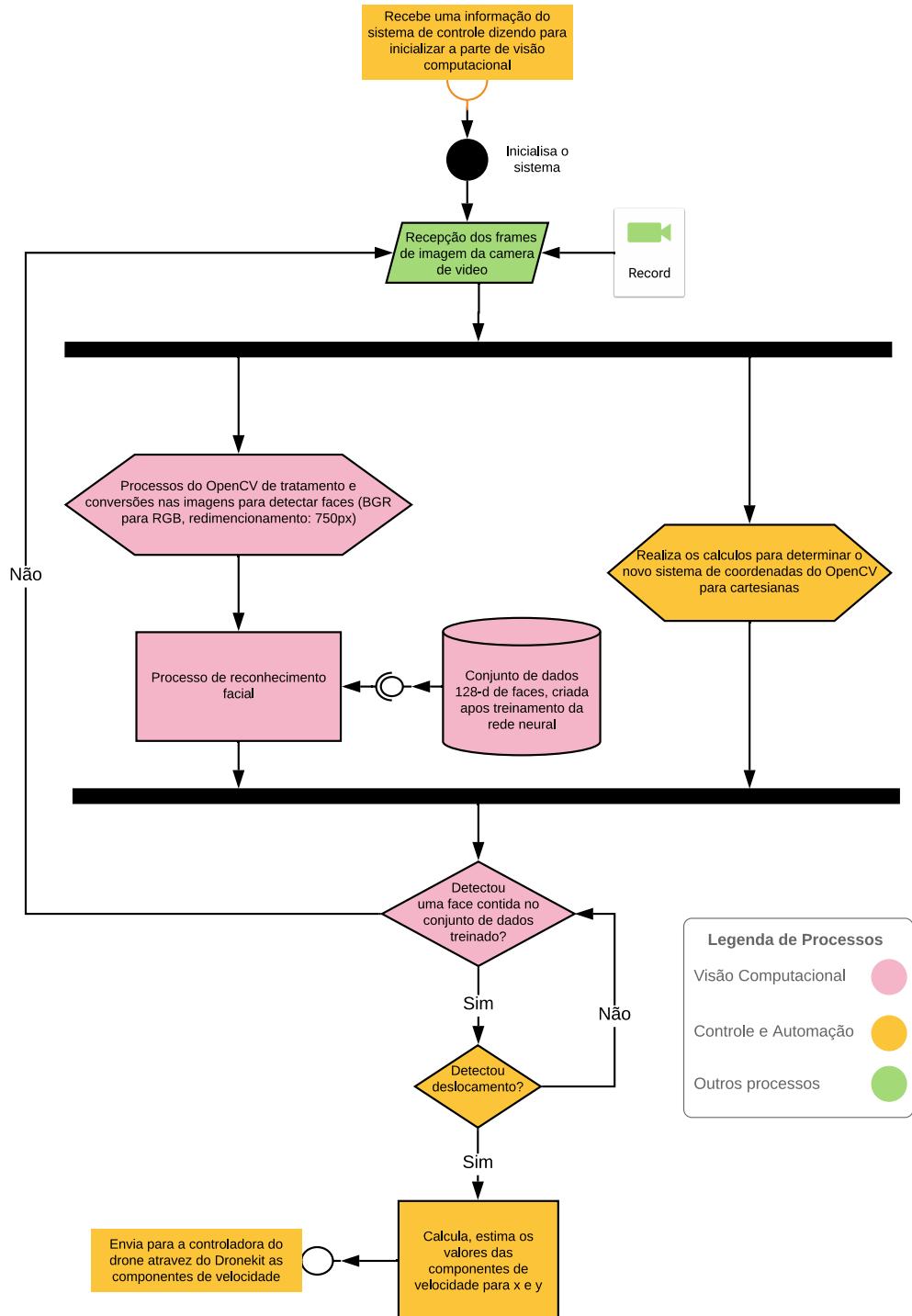
3.2.1.1 Diagramas UML da Codificação e Lógica do Software

No diagrama 37 cada elemento UML mostra a qual parte do sistema ele se adequa através da legenda dividida em cores, salmão para visão computacional e amarelo para controle. Cada elemento contem a definição da lógica que ele executa dentro do software.

A ilustração 38 defini as partes e funções lógicas das etapas que conferem o algoritmo de reconhecimento facial do protótipo.

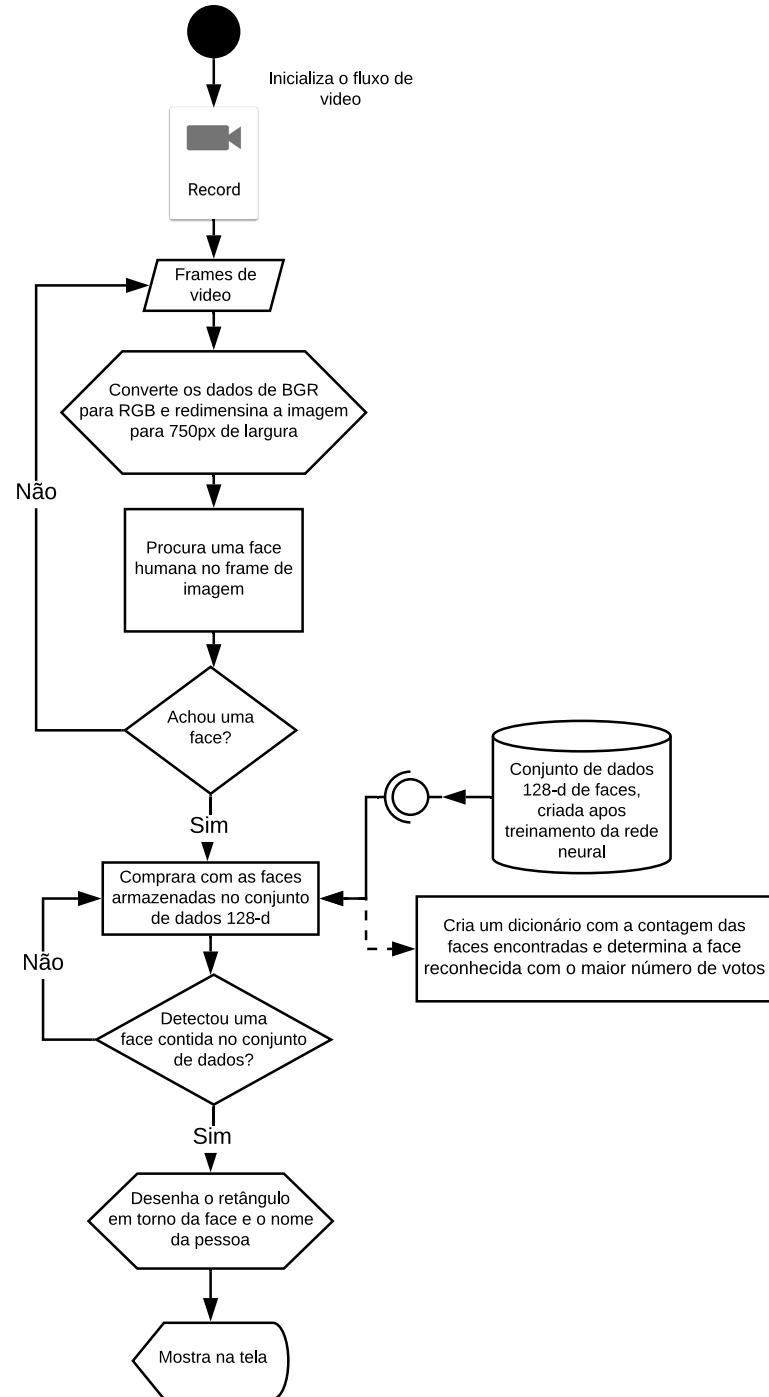
E na ilustração 39 a definição das partes dentro do algoritmo que representam o sistema de controle do VANT.

Figura 37 – Diagrama UML Total do Sistema



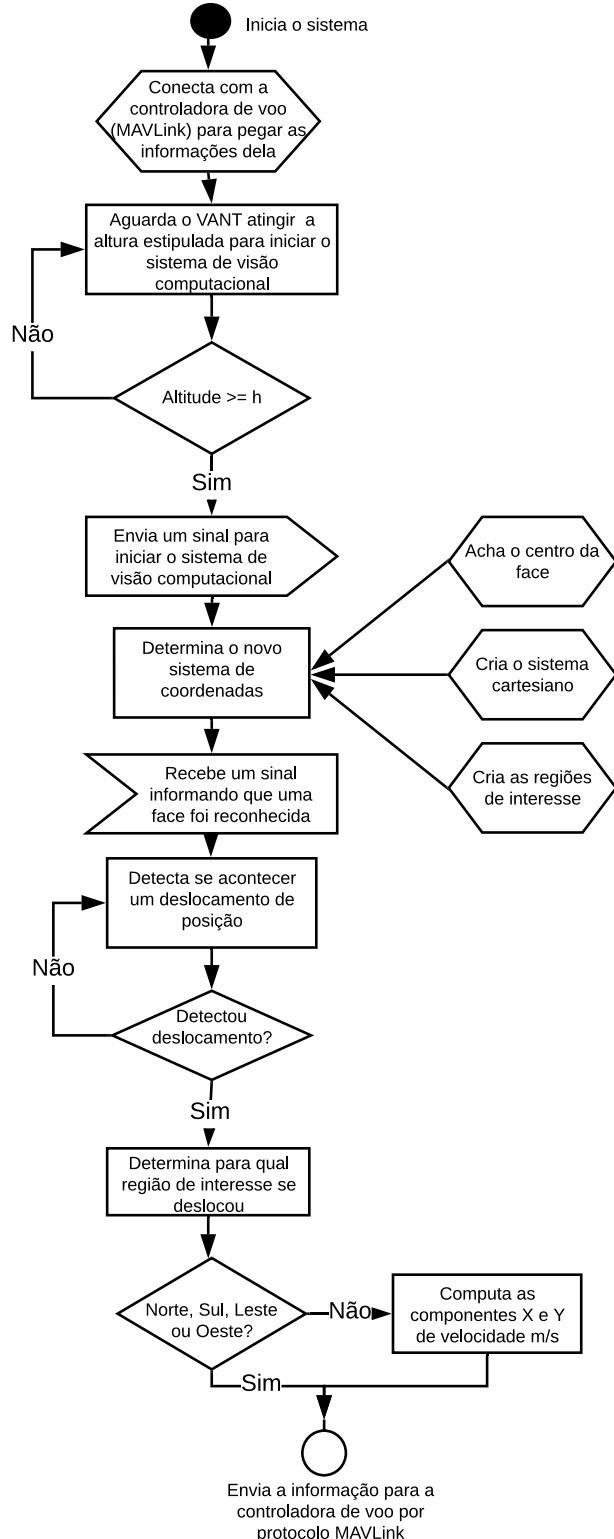
Fonte: do autor.

Figura 38 – Diagrama UML do Sistema de Reconhecimento Facial



Fonte: do autor.

Figura 39 – Diagrama UML do Sistema de Controle



Fonte: do autor.

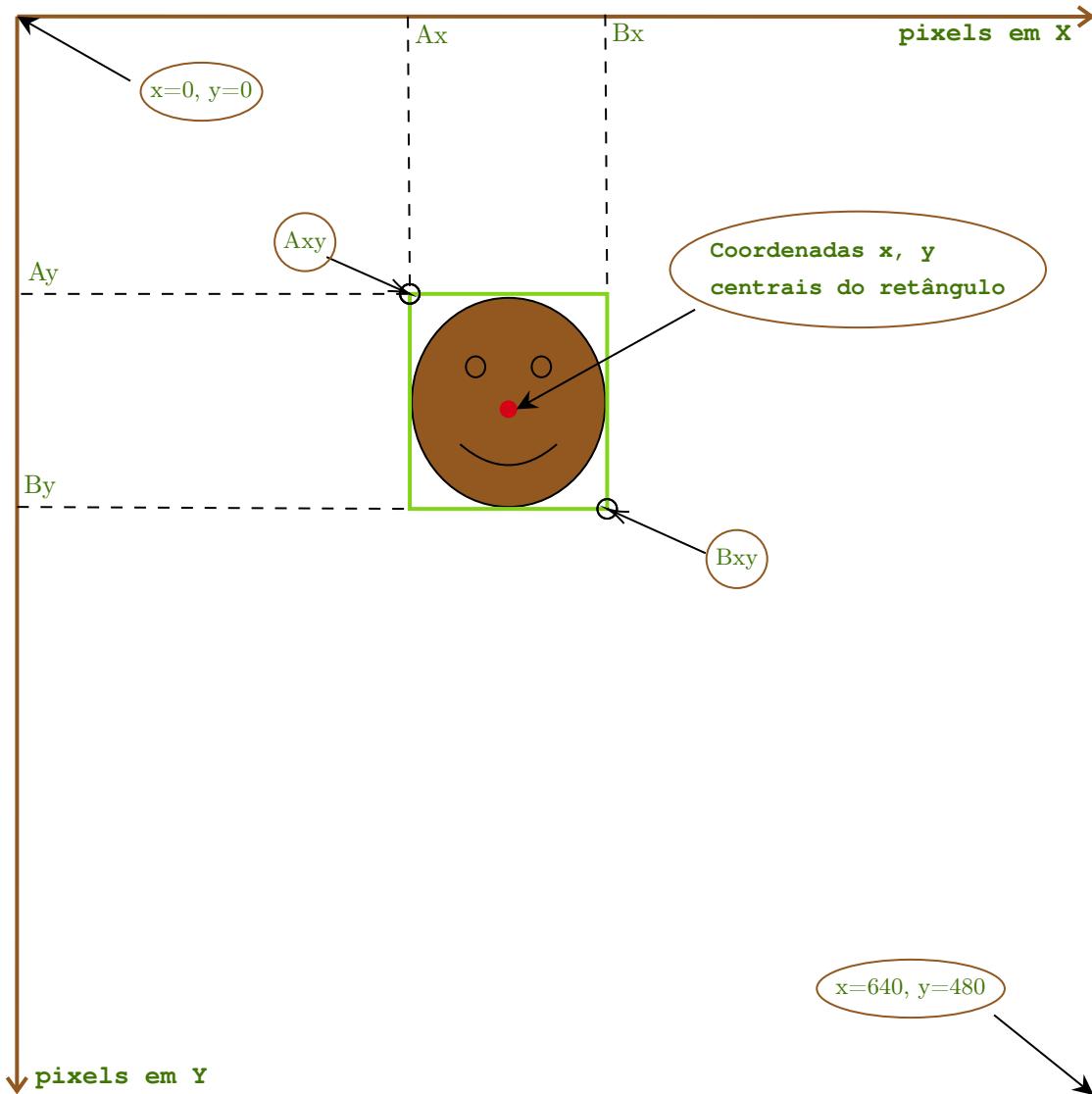
3.2.2 Teoria do Sistema de Controle de Robótica para o VANT

Foi necessariíssimo desenvolver um novo sistema de coordenadas baseando-se em um sistema cartesiano. Para isso foram realizados alguns cálculos em cima do sistema de coordenadas do OpenCV, para obter um protótipo de sistema do tipo cartesiano que gera-se valores para os eixos X e Y de forma balanceada

3.2.2.1 Conversão do Sistema de Coordenadas OpenCV para Cartesiano

Uma das primeiras etapas da implementação foi modificar o sistema de coordenadas de pixels no qual o OpenCV se baseia, para tornar conveniente sua utilização no sistema que detecta em qual direção o VANT tem que se movimentar. A ilustração 40 representa como o OpenCV trabalha com coordenadas de pixels, perceba que o ponto de origem das coordenadas verticais e Horizontais se situam na parte superior esquerda, logo não é possível identificar o sentido direcional em que o ponto na cor vermelha no centro da face esta se movendo. Para fins de orientação se estipulou que as linhas de pixels na horizontal pertencem a X e os na vertical a Y, e como é possível identificar com uma ferramenta do OpenCV o centro do retângulo desenhado em torno da face através de A_{xy} e B_{xy} , então se encontrou as coordenadas X e Y do ponto vermelho central.

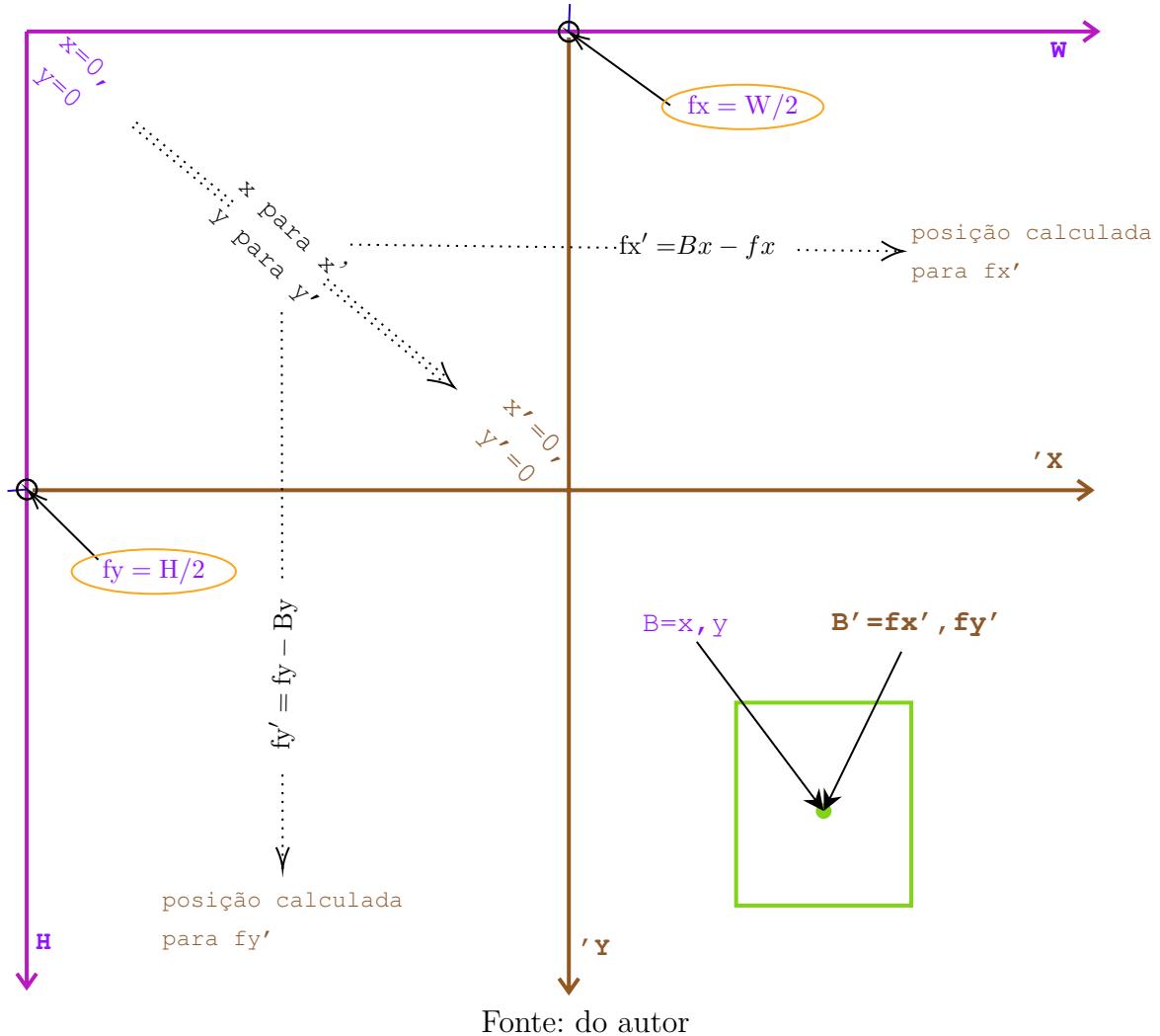
Figura 40 – Sistema de Coordenadas do OpenCV



Fonte: do autor

Para que o sistema funcione-se as coordenadas X=0 e Y=0 foram transportadas para um ponto central da tela, portanto foram realizados cálculos para determinar dois eixos um na horizontal e outro na vertical e sua intersecção ou ponto médio é o epicentro do novo sistema de coordenadas. Na ilustração 41 as duas retas na cor marrom representam o novo sistema, assim foi possível equacionar um novo ponto central para o retângulo que é desenhado em torno da face, esse novo ponto é dado como B' e ele recebe os valores calculados para fx' e fy'.

Figura 41 – Conversão do Sistema de Coordenadas OpenCV para Coordenadas Cartesianas



O objetivo de obter um sistema de coordenadas que pudesse ser utilizado para orientar o direcionamento do VANT foi alcançado e ele pode ser observado na ilustração 42, esse sistema de coordenadas é denominado de cartesiano e possui quatro quadrantes distintos e bem definidos, é o mesmo utilizado para localização geográfica na superfície terrestre. Da mesma forma que aeronaves conseguem se orientar na superfície terrestre usando coordenadas cartesianas, também é possível guiar um VANT para chegar ao objetivo do protótipo.

Com o novo sistema de coordenadas foi possível obter valores para X e Y com uma maior precisão (0.005m/s a 5m/s), assim como os quatro quadrantes conseguem indicar para qual direção a aeronave tem que se deslocar. Esses valores de velocidade posteriormente serão enviados para a controladora de voo do VANT.

A interpretação das informações da ilustração 41 pode ser feita através das seguintes sintaxes matemáticas;

- Sintaxe do quadrante I:

$$B' \in (\text{Quadrante I}) \mid \left(Bx - \frac{W}{2} > 0 \right) \wedge \left(\frac{H}{2} - By > 0 \right) \quad (3.1)$$

- Sintaxe do quadrante II:

$$B' \in (\text{Quadrante II}) \mid \left(Bx - \frac{W}{2} < 0 \right) \wedge \left(\frac{H}{2} - By > 0 \right) \quad (3.2)$$

- Sintaxe do quadrante III:

$$B' \in (\text{Quadrante III}) \mid \left(Bx - \frac{W}{2} < 0 \right) \wedge \left(\frac{H}{2} - By < 0 \right) \quad (3.3)$$

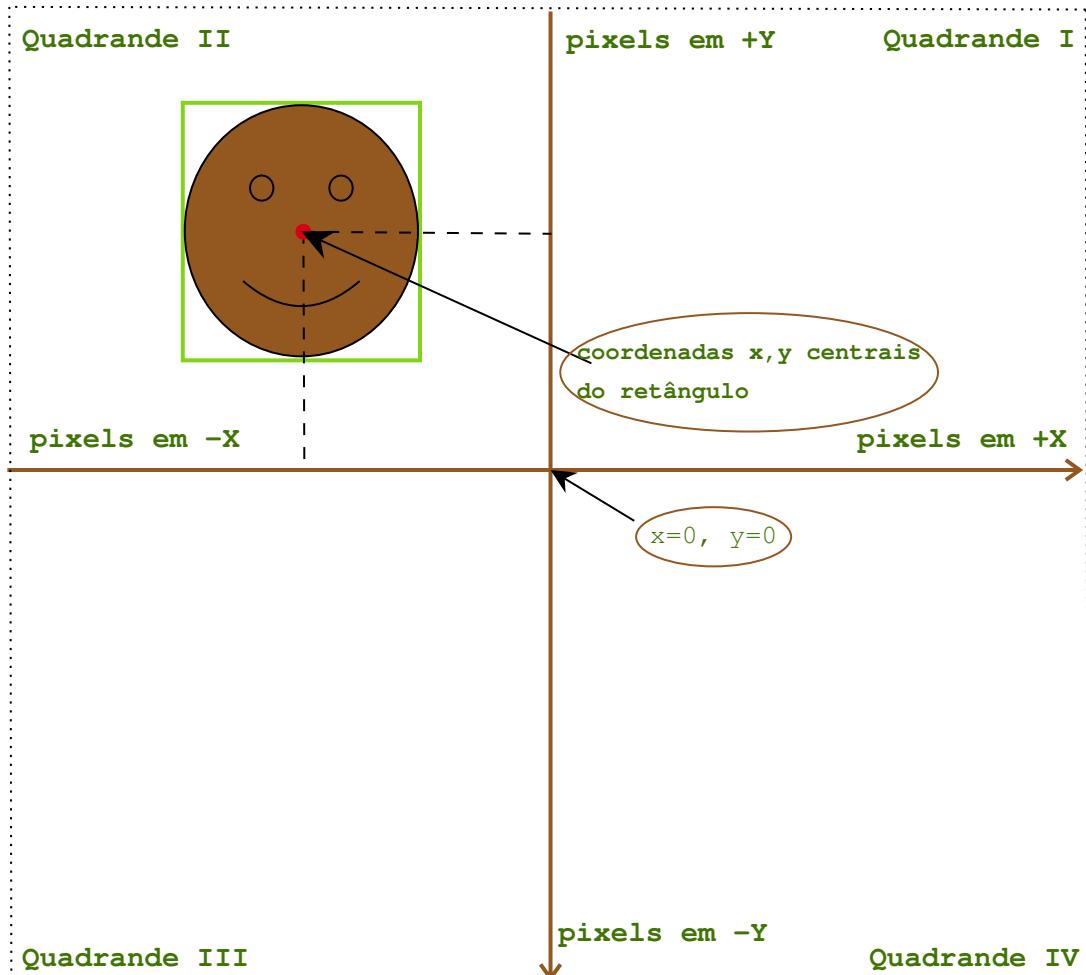
- Sintaxe do quadrante IV:

$$B' \in (\text{Quadrante IV}) \mid \left(Bx - \frac{W}{2} > 0 \right) \wedge \left(\frac{H}{2} - By < 0 \right) \quad (3.4)$$

- Sintaxe quando não pertence a nenhum quadrante:

$$B' \notin (\text{Quadrante I} \vee \text{II} \vee \text{III} \vee \text{IV}) \mid \left(Bx - \frac{W}{2} = 0 \right) \vee \left(\frac{H}{2} - By = 0 \right) \quad (3.5)$$

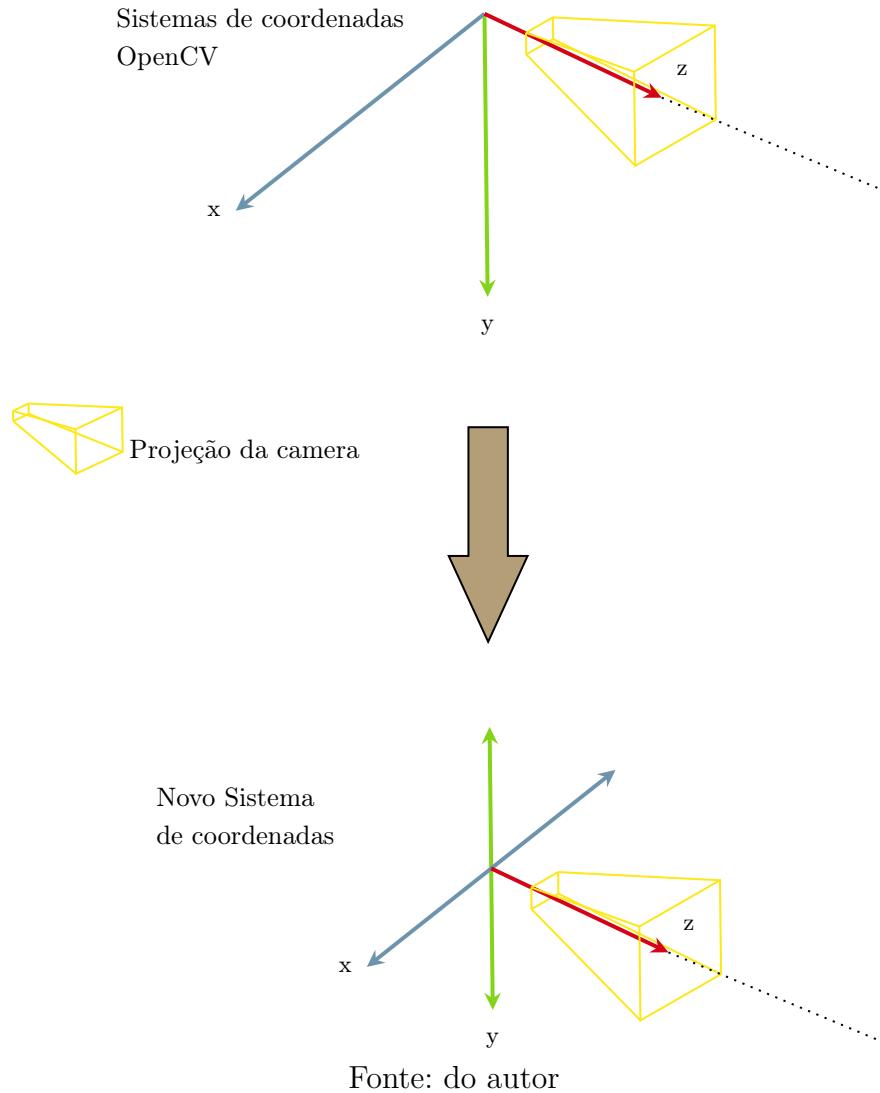
Figura 42 – Sistema de Coordenadas Cartesianas em Visão Computacional



Fonte: do autor

Para concluir essa parte a ilustração 43 representa a projeção em 3D do sistema de coordenadas original que o OpenCV utiliza e como fica apos a transformação. O OpenCV possui o eixo Z (profundidade), ele é utilizado para calcular a posição em um sistema 3D ou adquirir a velocidade de deslocamento, porem essa etapa não sera abordada mas pode ser um bom tema para trabalhos futuros ou aperfeiçoamento do sistema.

Figura 43 – Projeção do Sistema de Coordenadas OpenCV para Cartesianas



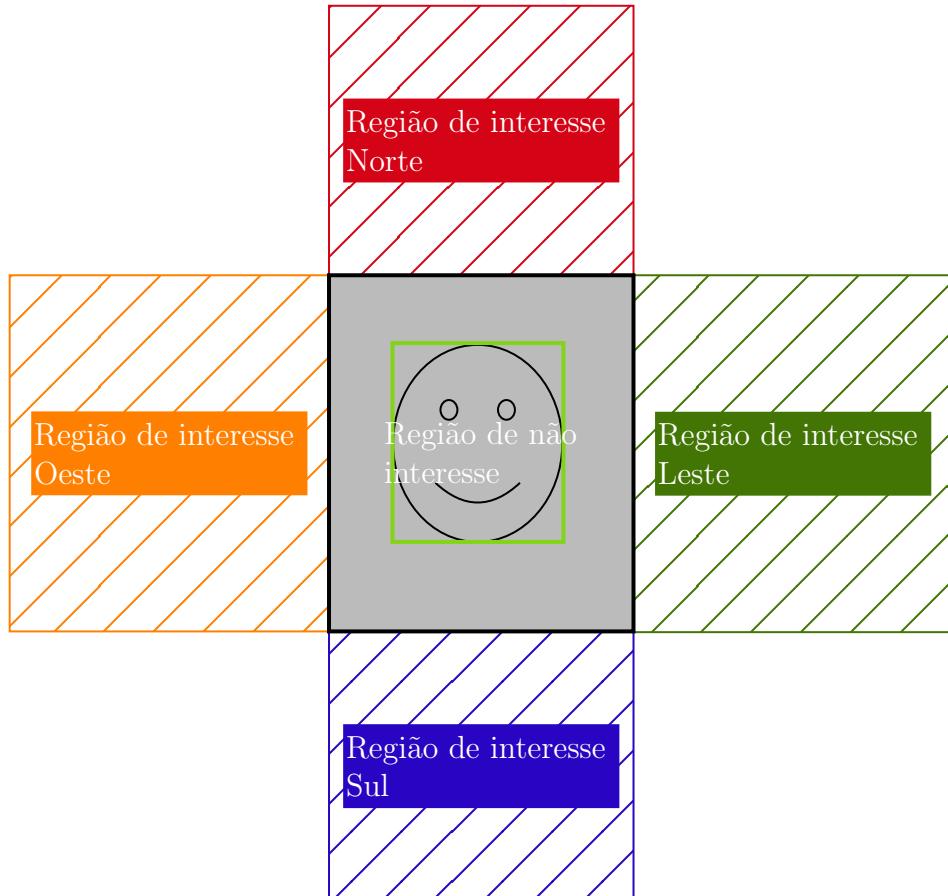
Fonte: do autor

3.2.2.2 Regiões de Interesse

Essa etapa foi fundamental para a implementação e funcionamento do protótipo, a ideia é simples foram delimitadas regiões de interesse utilizando o OpenCV, a ilustração 44 mostra todas as cinco áreas. No centro da tela se situa a região de não interesse e como o nome sugere dentro dessa região ou retângulo não existe reação do sistema de estreamento, porem caso a pessoa se move e saia de dentro o sistema detecta em qual direção esta se deslocando. A simplicidade vem do fato de o objetivo ou a lógica principal ser sempre

manter o alvo (retângulo verde criado pelo OpenCV em torno da face humana) no dentro da região de não interesse.

Figura 44 – Regiões de Interesse



Fonte: do autor

As dimensões das regiões de interesse são dinâmicas ou seja elas variam de tamanho conforme a dimensão do retângulo de detecção facial do OpenCV. Essa técnica foi determinada para tratar a distância em que a detecção facial é feita, assim se o alvo da detecção estiver em uma distância maior as regiões são menores e mais perto maiores, dessa forma as reações do VANT se adequaram com a distância.

A teoria de dimensionamento dinâmico pode ser explanada da seguinte forma, observando as ilustrações 40 e 41, primeiro se achou a diferença entre o centro do retângulo da face B até as bordas do retângulo Ax e Ay, depois foi dimensionado a partir do centro da tela a região de não interesse central somando ou subtraindo o resultado a fx e fy. A partir daí foi possível dimensionar as regiões de interesse Norte, Sul, Leste, Oeste, Nordeste, Noroeste, Sudeste e Sudoeste.

Porem a ideia das regiões de interesse não trata todas as possibilidades, como uma movimentação na diagonal, e dessa forma foi necessário desenvolver outras etapas.

3.2.2.3 Coeficiente de Aceleração

Foi determinado, calculado um valor para o coeficiente de aceleração do VANT, observando a ilustração 41 foi pré-determinado um valor de velocidade máxima de 5m/s, esse valor foi dividido pela média da soma de f_x e f_y , então se multiplicou esse valor pela posição central da caixa delimitadora (retângulo) da face (f_x' , f_y').

3.2.2.4 Movimentação do VANT

Para demonstrar como o VANT interage e interpreta os dados que recebe do sistemas de visão computacional e atua para se orientar, foram criados dois modelos de gráfico do tipo vetor gradiente, aonde o tom de cor mais forte representa o valor máximo e consequentemente o tom mais claro o mínimo valor. Foi estipulado que o valor máximo seria de $\approx 5/m_s$, observando que segundo o site Ardupilot.org-docs¹ um VANT do tipo quadricóptero atinge de 10m/s $\approx 13m/s$ no máximo, antes de se tornar incapaz de manter a altitude e a velocidade horizontal.

O primeiro gráfico ilustrado pela figura 45 tem como objetivo interpretar como o sistema de visão computacional atua junto com o Dronekit. Se o gráfico 45 for sobreposto pela tela do OpenCV a parte aonde o gradiente é mais fraco fica no centro, ou seja, local aonde o software de visão computacional definiu como região de não interesse (não existe velocidade ou é insignificante), ja nas bordas a coloração é mais forte, isso significa que quando a pessoa que esta sendo monitorada pelo software estiver se aproximando das bordas da interface de captura da câmera, a velocidade aumenta gradualmente, tendo como objetivo sempre manter o alvo no centro da tela, logo conforme o alvo estiver se aproximando do centro a velocidade gradualmente vai diminuindo até parar assim que chegar ao centro e estiver na região de não interesse.

As escalas nos eixos X e Y contem os possíveis valores de velocidade em m/s que podem ser enviados para o VANT, como ja foi abordada no referencial teórico é utilizado o sistema de orientação por coordenadas NED e por regra nesse sistema de coordenadas o X fica no eixo das ordenadas e Y no eixo das abcissas.

No gráfico 46 foram adicionadas duas cores para distinguir as velocidades em X e Y, os VANTS representam graficamente algumas posições dentro do sistema de coordenadas com seus respectivos valores de velocidade nos vetores X e Y assim como qual direção eles tomaram ao receber os valores de velocidade. A cor verde representa o gradiente de velocidade no eixo X e a cor vermelha no eixo Y e igualmente como no gráfico 45 nas bordas tem sua maior intensidade de velocidade e no centro não existe ou é insignificante as forças de velocidade. Nos eixos diagonais a X e Y pontos (A,B,C,D) as cores se sobrepõem, isso significa que os valores de velocidade serão iguais ou próximos para X e Y nesse ponto.

¹<<https://ardupilot.org/copter/docs/auto-mode.html>>

Analizando todos os pontos no gráfico 46 aonde estão os VANTs:

- VANT 1: Nesse ponto $x=-0.9\text{ms}$ e $y=1.9\text{m/s}$, e de acordo com a regra 3.2 pertence ao quadrante II, tem direção Noroeste tendendo a se aproximar mais do eixo X por ter uma maior velocidade na componente X.
- VANT 2: Nesse ponto $x=5\text{ms}$ e $y=0\text{m/s}$, e de acordo com a regra 3.5 não pertence a nenhum quadrante por ter uma das componentes de velocidade com valor nulo, porem podemos dizer que tem direção Norte de acordo com a componente x.
- VANT 3: Nesse ponto $x=-3.95\text{ms}$ e $y=-6.5\text{m/s}$, e de acordo com a regra 3.3 pertence ao quadrante III, tem direção Sudoeste e tendendo a se aproximar mais do eixo Y por ter uma maior velocidade na componente Y.
- VANT 4: Nesse ponto $x=1.5\text{ms}$ e $y=3.8\text{m/s}$, e de acordo com a regra 3.1 pertence ao quadrante I, tem direção Nordeste tendendo a se aproximar mais do eixo Y por ter uma maior velocidade na componente Y.
- VANT 5: Nesse ponto $x=-3.6\text{ms}$ e $y=3.75\text{m/s}$, e de acordo com a regra 3.4 pertence ao quadrante IV, tem direção Sudeste e nesse caso tem diferença de valores nas componentes X e Y insignificante não tendendo a se direcionar a nenhum dos eixos.

Figura 45 – Gráfico que Representa o Gradiente de Velocidade

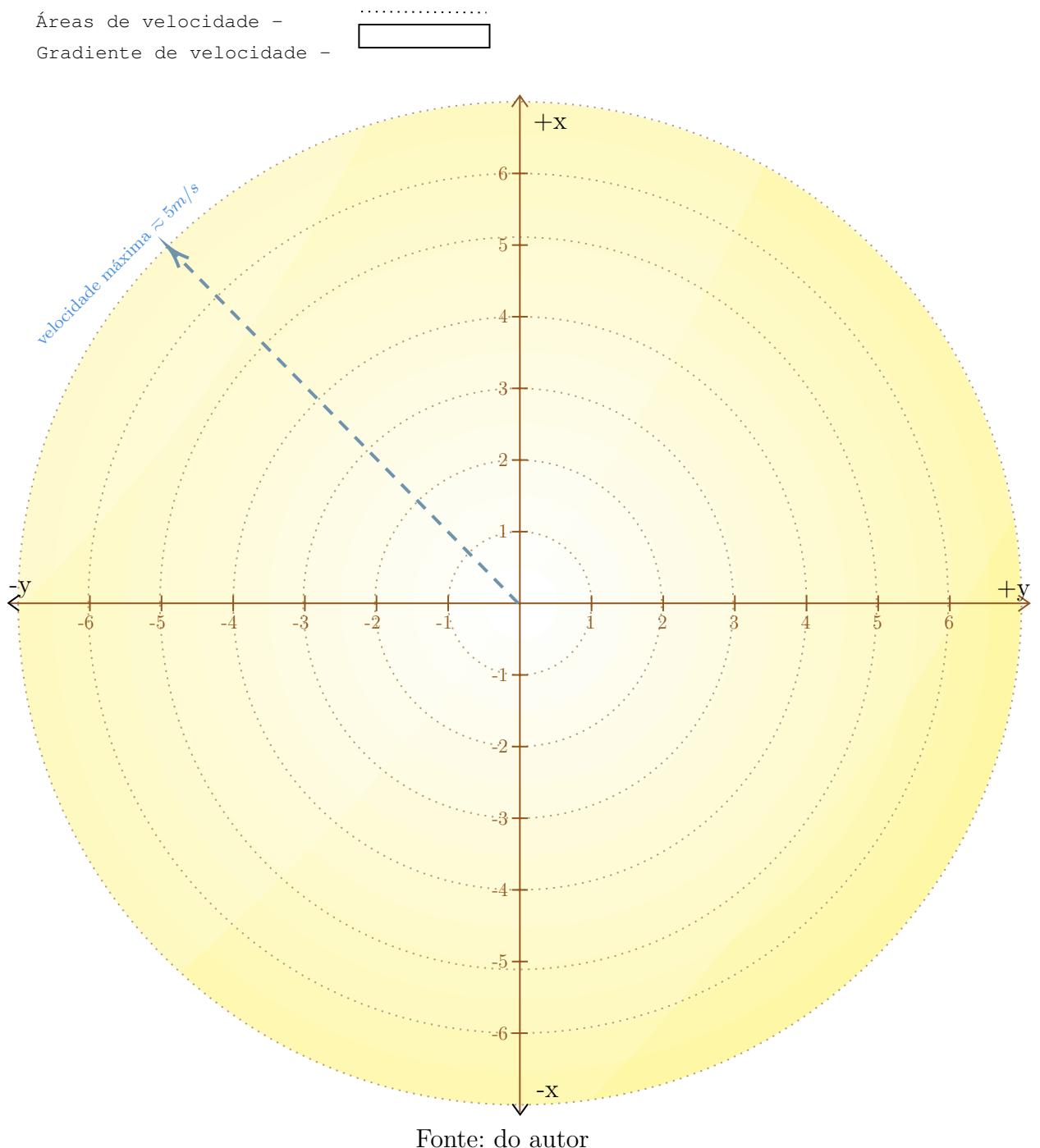
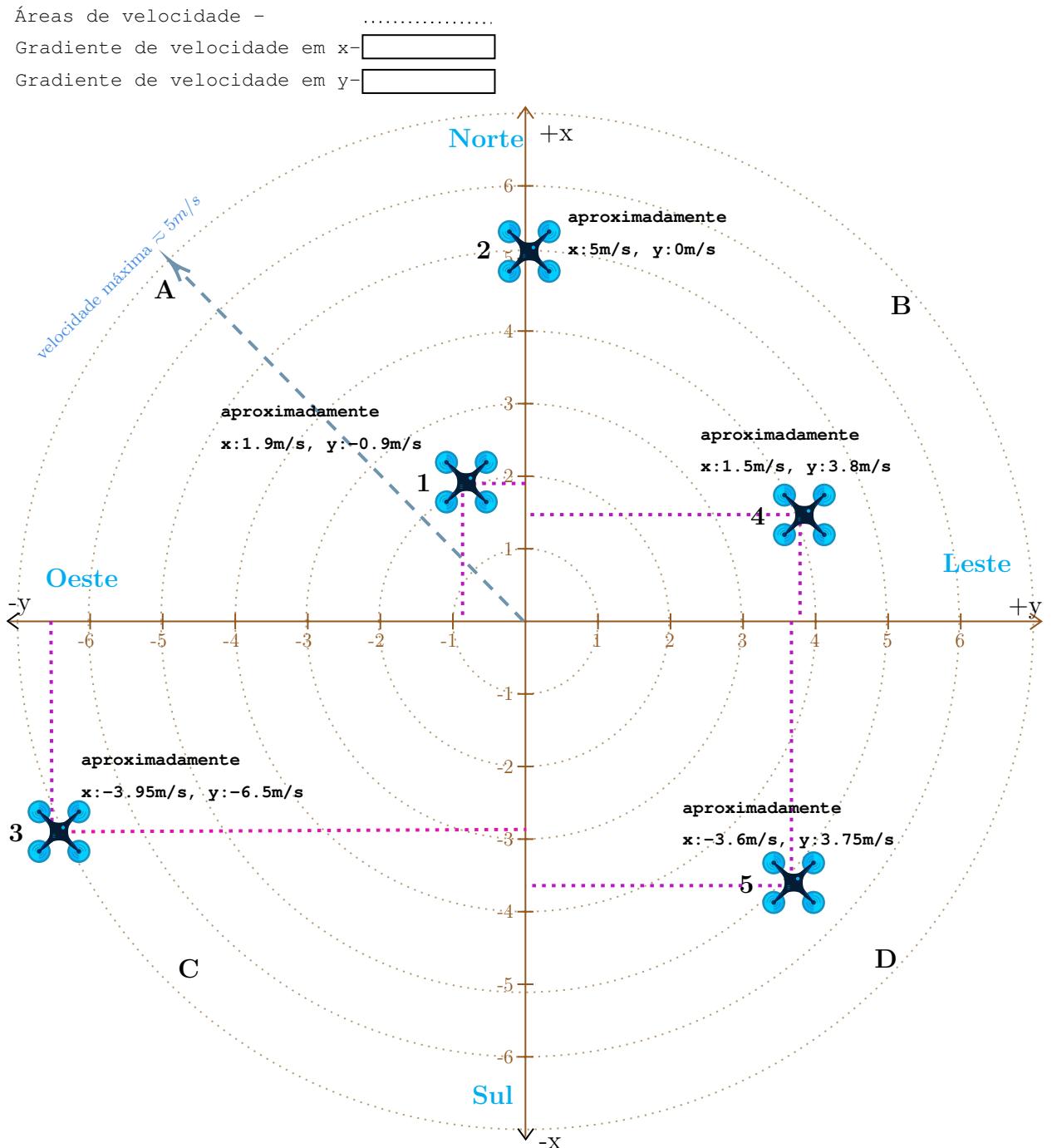


Figura 46 – Gráfico que Representa o Gradiente de Velocidade para X e Y



Fonte: do autor

3.3 Treinamento da rede Neural

Para o sistema de visão computacional foi treinada uma rede neural do tipo CNN utilizando tecnologias da NVIDIA (CUDA, CUDNN) e as bibliotecas DLIB com *face-recognition* para as chamadas de detecção facial já implementadas nelas.

O treinamento gera um arquivo que cria um vetor contendo os dados 128-D para cada face inserida no banco de dados do método de busca, assim como foi abordado no capítulo 2.2.5 tendo como exemplo a ilustração 9.

3.4 Controles do VANT

Para controlar o VANT foi utilizado o kit de ferramentas para controle e desenvolvimento de VANTS Dronekit juntamente do *firmware* Ardupilot, nele esta contido o software SITL que é utilizado para simular o funcionamento de VANTS. Foi utilizada uma API² desenvolvida para integrar o SITL do Ardupilot com o software de simulação Gazebo. A partir dai foi possível usar as chamadas de funções do Dronekit no código fonte para enviar comandos de movimento para o VANT, também é possível interceptar mensagens de telemetria do VANT como velocidade, altura e direção. E ao integrarmos essa ferramenta com a parte desenvolvida em visão computacional obtivemos o sistema de controle de VANT.

3.5 Componentes Físicos

Os componentes físicos são o computador utilizado nas simulações, computador complementar (Raspberry Pi)e o dispositivo de captura de imagem utilizado nos testes.

3.5.1 Computador Utilizado nas Simulações

Devido ser necessário executar simultaneamente varias ferramentas de simulação que na sua maioria se utilizam de interfaces gráficas que necessitam de renderização em tempo real, e com um sistema de visão computacional que necessita de uma GPU e boa quantidade de memoria RAM para conseguir executar sua tarefa de reconhecimento facial, foi preciso um computador com hardware de tecnologia de vanguarda.

- Especificações do Computador:
 - Processador:
 - * Intel Core i7 8700
 - * 6 núcleos de processamento
 - * 12 threads
 - * 3.20GHz de frequência base
 - * 4.60GHz de frequência máxima
 - * 12MB de memória cache

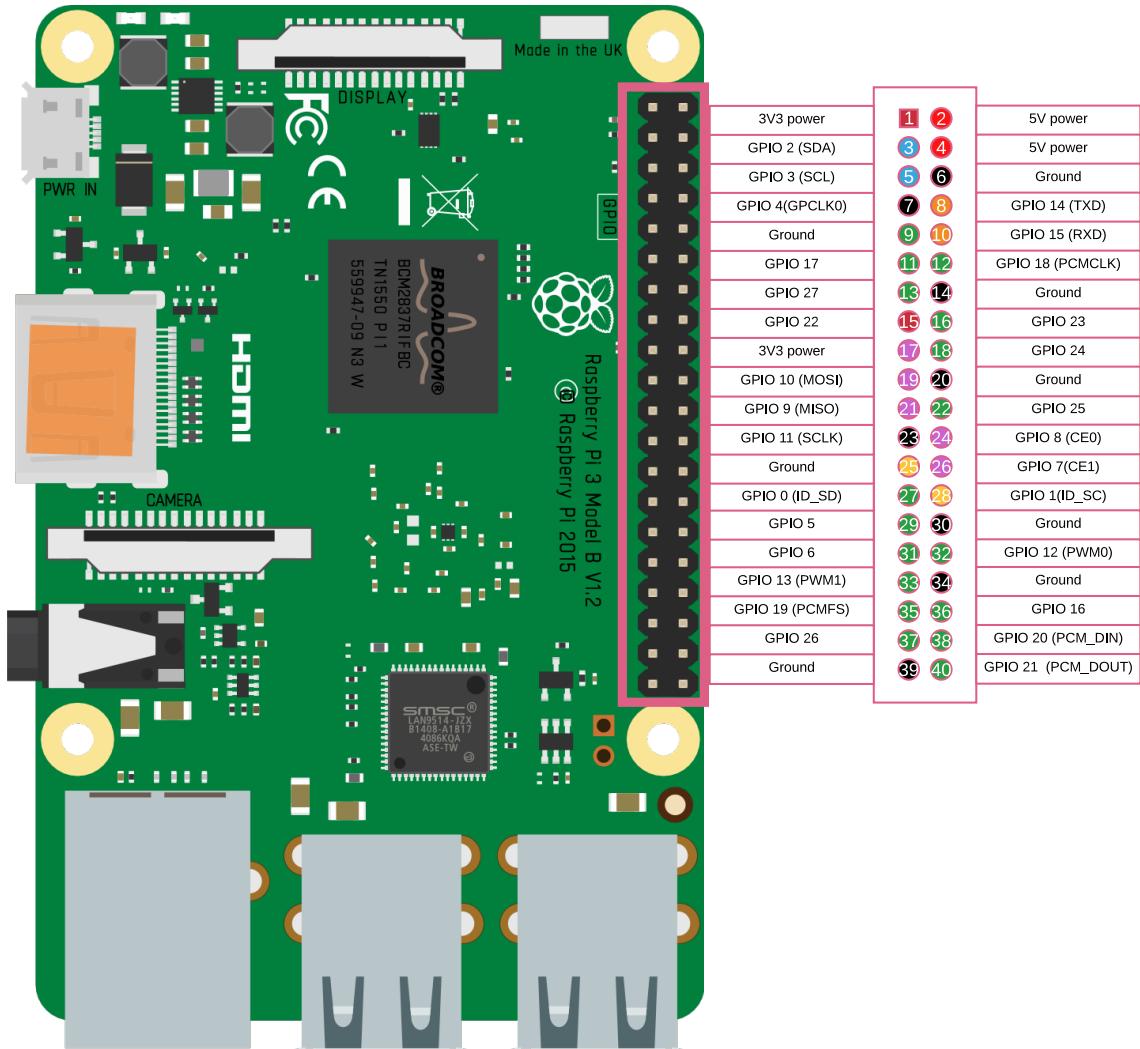
²<<https://ardupilot.org/dev/docs/using-gazebo-simulator-with-sitl.html>>

- L1 (Data) cache 6x32 KBytes
 - L1 (Instruction) cache 6x32 KBytes
 - L2 cache 6x256 KBytes
 - L3 cache 12 MBytes
- Placa de Video:
- * NVIDIA GFORCE RTX2070
 - * 2304 NVIDIA CUDA Cores
 - * 1620 MHz de clock máximo
 - * 1410 MHz de clock base
 - * 8GB de memoria GDDR6
 - * Interface de memoria de 256bit
 - * Taxa de transferencial 448GB/s
- Memória RAM:
- * 16GB de capacidade
 - * Taxa de transferência 3200Mb/s
 - * Tecnologia DDR4
- Armazenamento:
- * 1TB SSD
 - * Taxa de leitura gravação 500MB/s e 450MB/s
 - * Interface SATA 3

3.5.2 Computador Complementar

Devido ao fato da pesquisa propor um vanto que acople um computador que comportaria o software de visão computacional, foi estipulado que este seria um Raspberry Pi, pelo seu reduzido tamanho e peso, Foi feita uma implementação do sistema de reconhecimento facial que foi utilizado na simulação, com o Raspberry Pi para provar seu funcionamento em uma possível simulação real.

Figura 47 – Raspberry Pi 3 B



Fonte: do autor com base em ([RASPBERRYPI.ORG](https://www.raspberrypi.org), 2020b).

- CPU Broadcom BCM2837 de 64 bits Quad Core de 1.2GHz
- 1GB RAM
- LAN sem fio BCM43438 e Bluetooth Low Energy (BLE)
- 100 Base Ethernet
- Extensão GPIO de 40 pinos
- 4 portas USB 2
- Saída estéreo de 4 polos e porta de vídeo composto
- HDMI em tamanho real
- Porta de conexão para câmera CSI Raspberry Pi

- Porta de conexão para display DSI Raspberry Pi sensível ao toque
- Porta Micro SD para carregar seu sistema operacional e armazenar dados
- Fonte de alimentação Micro USB comutada atualizada até 2.5^a

3.5.3 Dispositivo de Captura

Para os testes foi utilizado um dispositivo de captura da figura 48 do tipo webcam da *Microsoft* modelo life-cam HD-5000.

Figura 48 – Dispositivo de Captura



Fonte: do autor

3.6 Sistemas Operacionais

Para implementar as simulações no computador foi utilizado o sistema operacional linux Mint 19, baseado no linux Ubuntu Bionic. A maioria das ferramentas são desenvolvidas para sistemas UNIX e que se baseiam no Linux Debian e Ubuntu por isso se chegou nessa escolha.

3.7 Computador Complementar vs Computador Utilizado nas Simulações (Benchmarking)

Uma simulação *benchmarking* para aferir o desempenho do algoritmo de visão computacional do protótipo foi implementada, comparando os resultados entre o computador utilizado nas simulações [3.5.1](#) e o computador complementar (Raspberry Pi) [3.5.2](#).

Para o computador das simulações foi utilizada a técnica de treinamento CNN³ e para o computador complementar foi utilizada a técnica HOG⁴, isso porque a Raspberry Pi não tem capacidade computacional para treinar uma rede do tipo CNN.

Se sabe que uma HOG tem desempenho inferior ao de uma CNN então realizou-se essa comparação extraindo FPS⁵ e numero de detecções positivas utilizando um video de aproximadamente um minuto.

Enquanto a instalação do OpenCV e treinamento da rede neural no computador das simulações durou algo em torno de meia hora, na Raspberry Pi foram necessárias 3 horas para instalação do OpenCV e 20 minutos para treinar a HOG.

³Rede Neural Convolucional Profunda

⁴Histograma de Gradiente Orientado<<https://www.learnopencv.com/histogram-of-oriented-gradients/>>

⁵Quadros por segundo

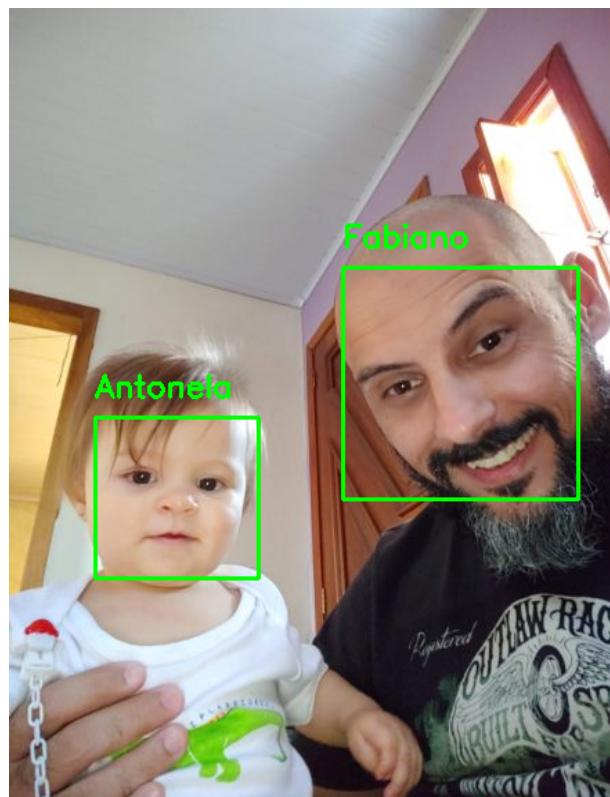
4 Analise dos Resultados e Considerações Finais

4.1 Resultados

Para demonstrar a capacidade de reconhecimento facial do protótipo, foi utilizada uma foto com dois personagens (Fabiano, Antonela). A rede neural foi treinada duas vezes; a primeira para reconhecer apenas o personagem Fabiano e a segunda para reconhecer os dois personagens. A ilustração 50 mostra o reconhecimento de apenas o personagem Fabiano, e a ilustração 50 o reconhecimento dos dois personagens

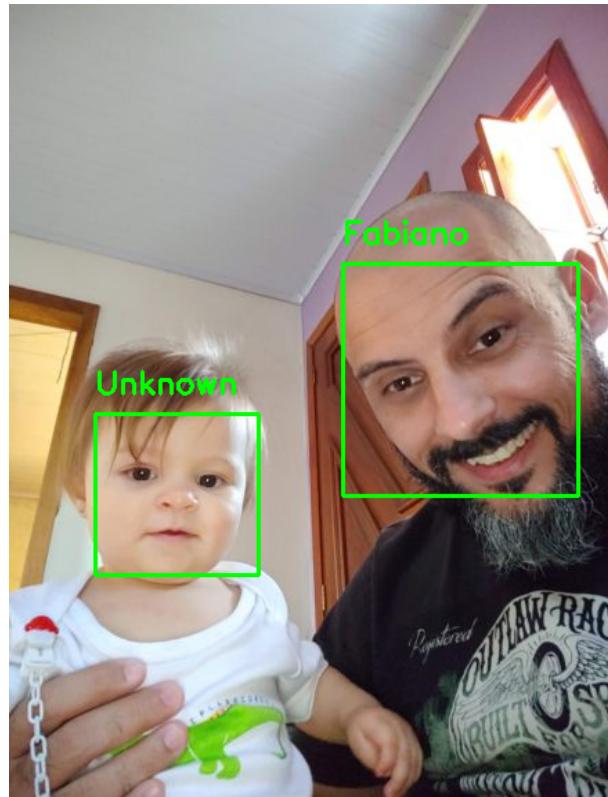
Isso mostra a capacidade do sensoriamento populacional, pelo fato de ser capaz de distinguir entre uma pessoa que foi inserido no conjunto de dados e uma pessoa que não foi.

Figura 49 – Teste de Reconhecimento Facial (2 Pessoas)



Fonte: do autor

Figura 50 – Teste de Reconhecimento Facial (1 Pessoa)



Fonte: do autor

Para mostrar o funcionamento do sistema de controle que diz para o sistema a direção em que o VANT deve se deslocar, algumas ilustrações foram inseridas para interpretar os dados adquiridos no *software* de simulação Gazebo.

Analisando as imagens conseguimos obter as seguintes informações para interpretar o funcionamento:

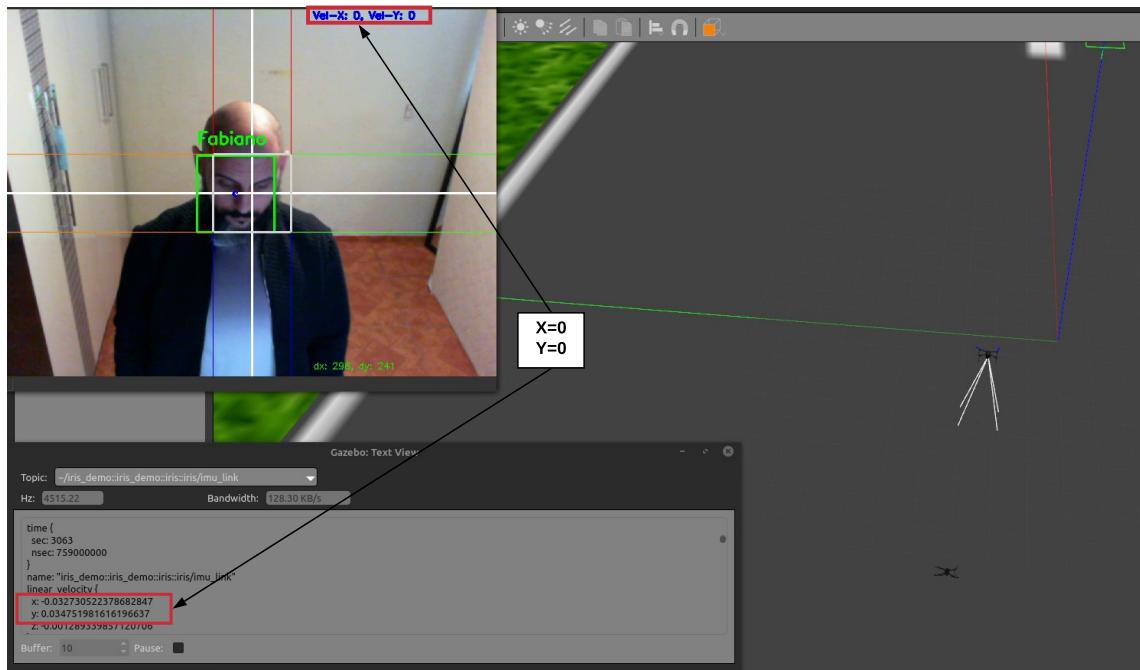
- **Simulação de Nenhum Deslocamento:** Na ilustração 51 o VANT esta parado seguindo a lógica do sistema de sempre que o VANT estiver no centro da tela ele não terá reações de deslocamento.
- **Simulação de Deslocamento para o Norte:** Na ilustração 52 o VANT esta na região de interesse Norte se deslocando nesse sentido à uma velocidade de $\approx 3,75m/s$ em X e não a componente de velocidade em Y nessa região.
- **Simulação de Deslocamento para o Sul:** Na ilustração 53 o VANT esta na região de interesse Sul se deslocando nesse sentido à uma velocidade de $\approx -2,1m/s$ em X e não à componente de velocidade em Y nessa região.

- Simulação de Deslocamento para o Leste:** Na ilustração 54 o VANT esta na região de interesse Leste se deslocando nesse sentido à uma velocidade de $\approx 2,8m/s$ em Y e não à componente de velocidade em X nessa região.
- Simulação de Deslocamento para o Nordeste:** Na ilustração 55 o VANT esta na região de interesse Nordeste se deslocando nesse sentido à uma velocidade de $\approx 2,7m/s$ em X e $\approx 1,5m/s$ em Y.
- Simulação de Deslocamento para o Noroeste:** Na ilustração 56 o VANT esta na região de interesse Noroeste se deslocando nesse sentido à uma velocidade de $\approx 2,8m/s$ em X e $\approx 2,7m/s$ em Y.

Com isso podemos analisar que a teoria de deslocamento mostrada na ilustração 46 é no mínimo proficiente para ser utilizada em um sistema de controle e automação para VANT, utilizando visão computacional.

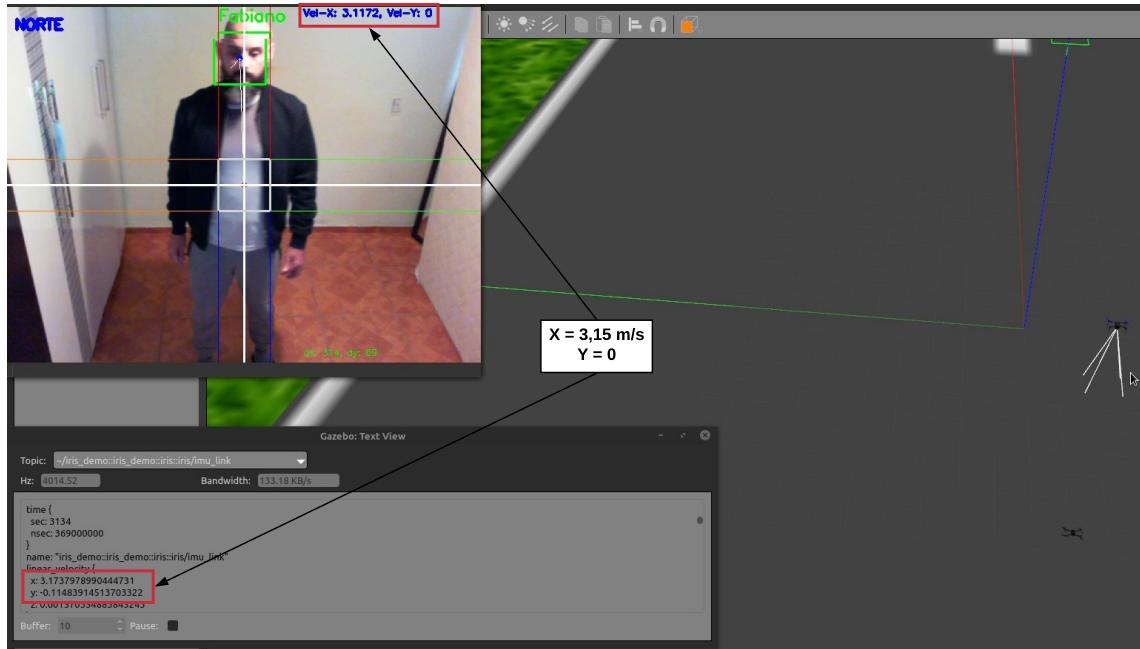
No Gazebo a velocidade no eixo Y fica negativo em relação a componente de velocidade calculada no sistema, mas isso é porque o sistema de coordenadas para VANT do Gazebo esta incorreto, porem isso não altera o funcionamento em geral.

Figura 51 – Simulação de Nenhum Deslocamento



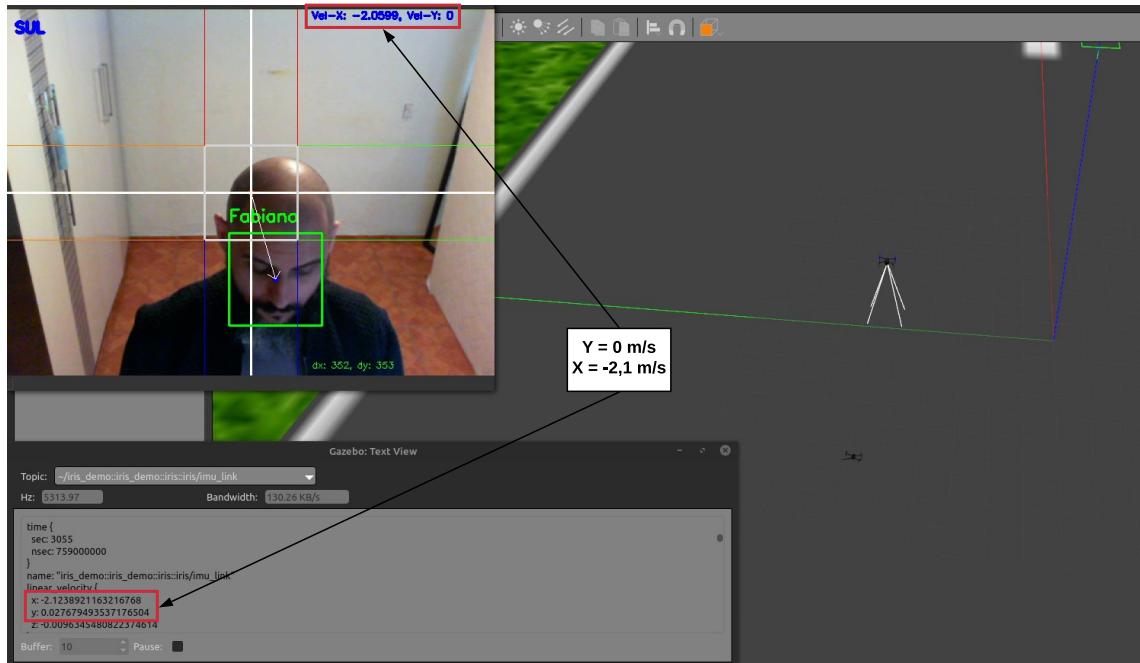
Fonte: do autor

Figura 52 – Simulação de Deslocamento para o Norte



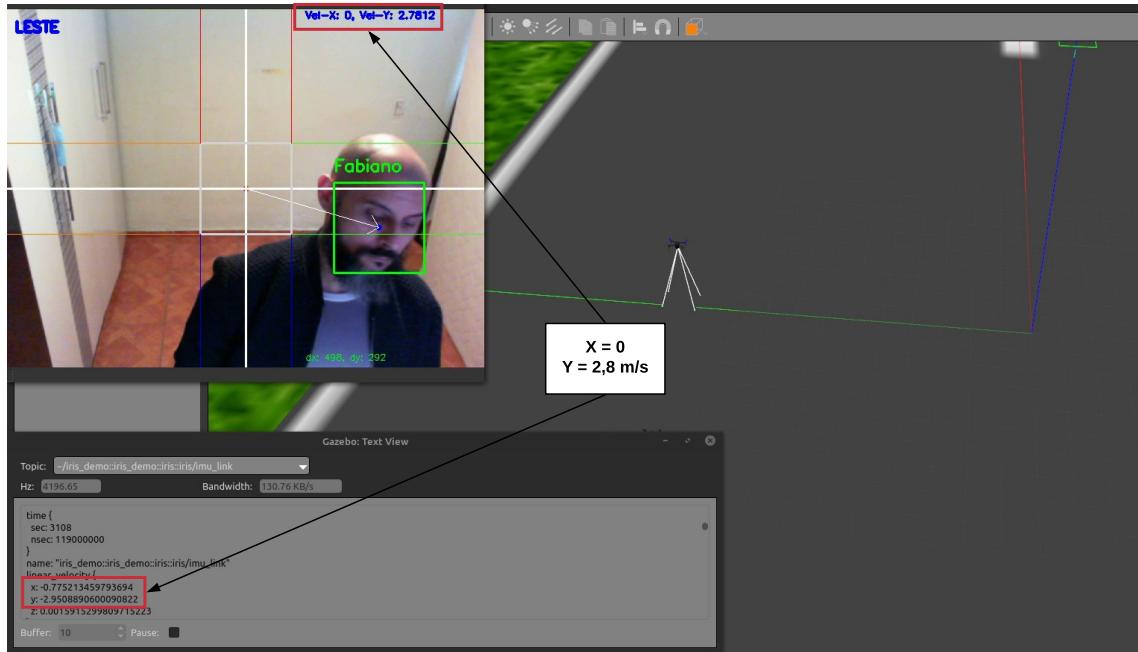
Fonte: do autor

Figura 53 – Simulação de Deslocamento para o Sul



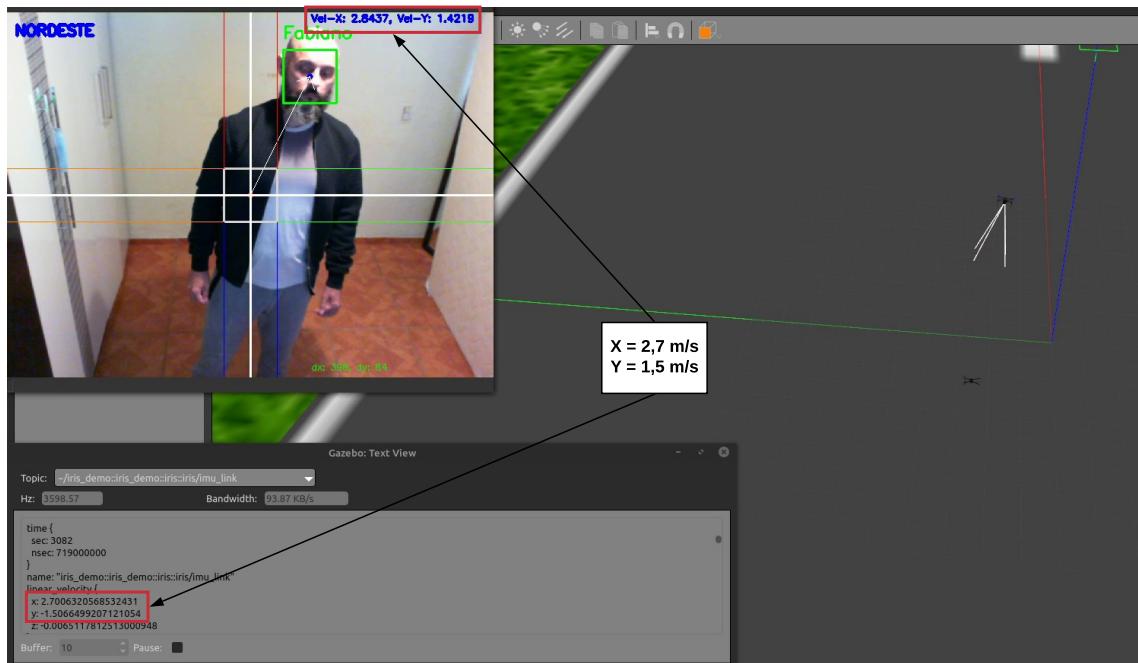
Fonte: do autor

Figura 54 – Simulação de Deslocamento para o Leste



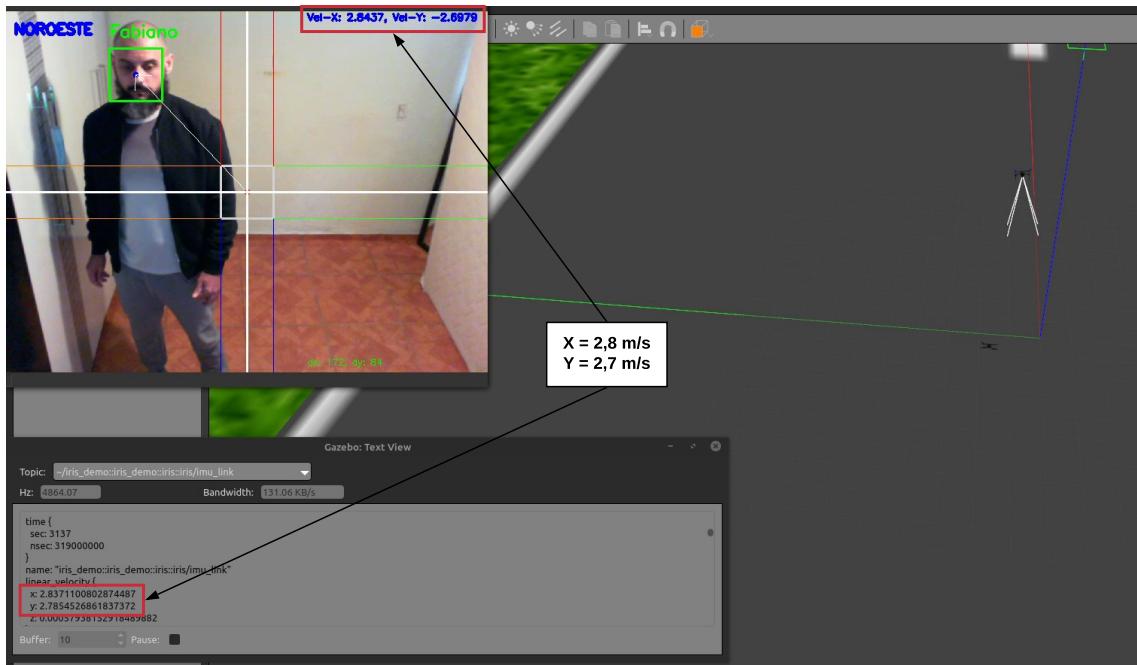
Fonte: do autor

Figura 55 – Simulação de Deslocamento para o Nordeste



Fonte: do autor

Figura 56 – Simulação de Deslocamento para o Noroeste



Fonte: do autor

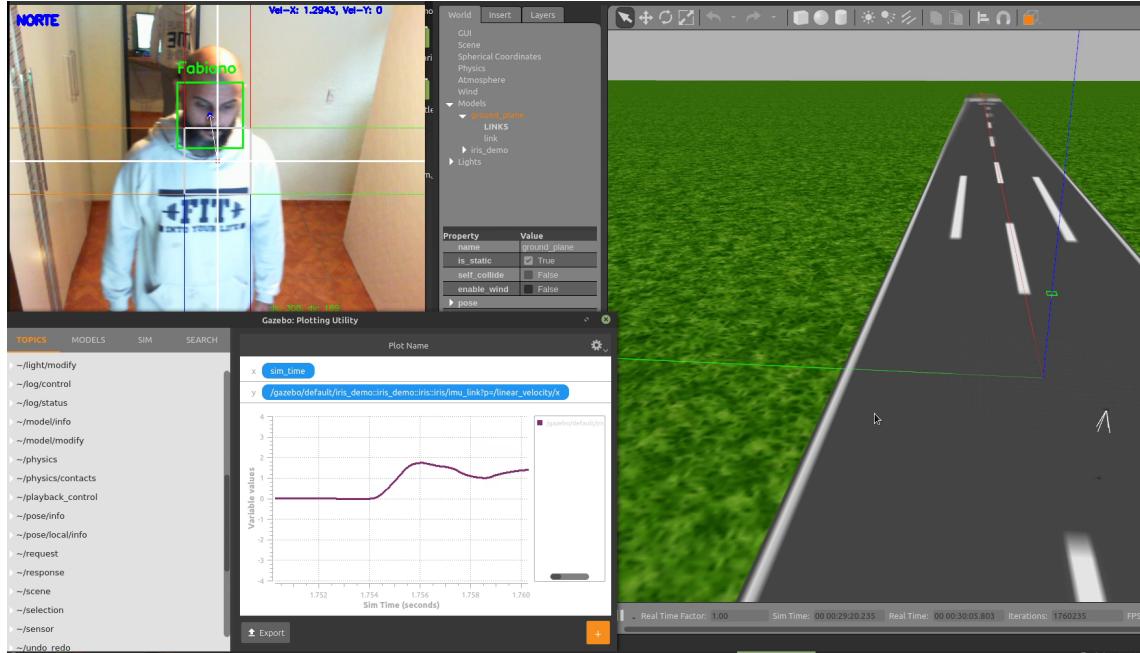
Agora analisamos a aceleração do VANT em relação a teoria que fui utilizada na ilustração 45, dizendo que o VANT possui maior velocidade nas bordas e menor velocidade no centro do sistema de referência.

Através das ilustrações abaixo podemos analisar as seguintes informações extraídas do *software Gazebo*:

- **Primeira Simulação de Velocidade para o Norte:** Na ilustração 57 observando o gráfico podemos verificar que o VANT saiu de um estado aonde não possuía velocidade e acelerou até atingir $\approx 1,3m/s$ em X e o personagem alvo do sistema de visão computacional esta bem próximo do centro da tela aonde as componentes de velocidade são baixas.
- **Segunda Simulação de Velocidade para o Norte:** Na ilustração 58 observando o gráfico o VANT estava em aproximadamente $1,3m/s$ e acelerou até atingir velocidade de $\approx 2,0/s$.
- **Terceira Simulação de Velocidade para o Norte:** Na ilustração 59 observando o gráfico o VANT ja esta a uma velocidade de $\approx 3,7/s$ e o personagem alvo do sistema de visão computacional esta bem próximo a borda da tela aonde as componentes de velocidade são maiores.

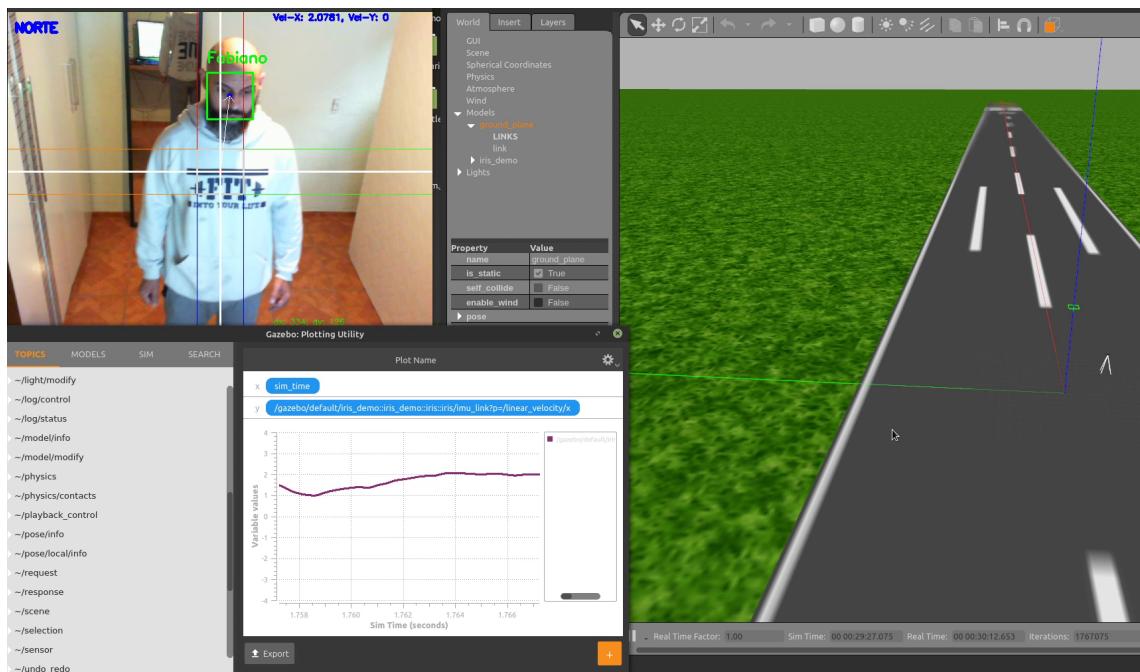
- Simulação de Velocidade para o Sul:** Na ilustração 60 o gráfico tem uma inversão de sinal obtendo velocidade de $\approx -2,5/s$ em X, isso porque o personagem alvo esta na região de interesse Sul invertendo a componente de velocidade do eixo X.

Figura 57 – Primeira Simulação de Velocidade para o Norte



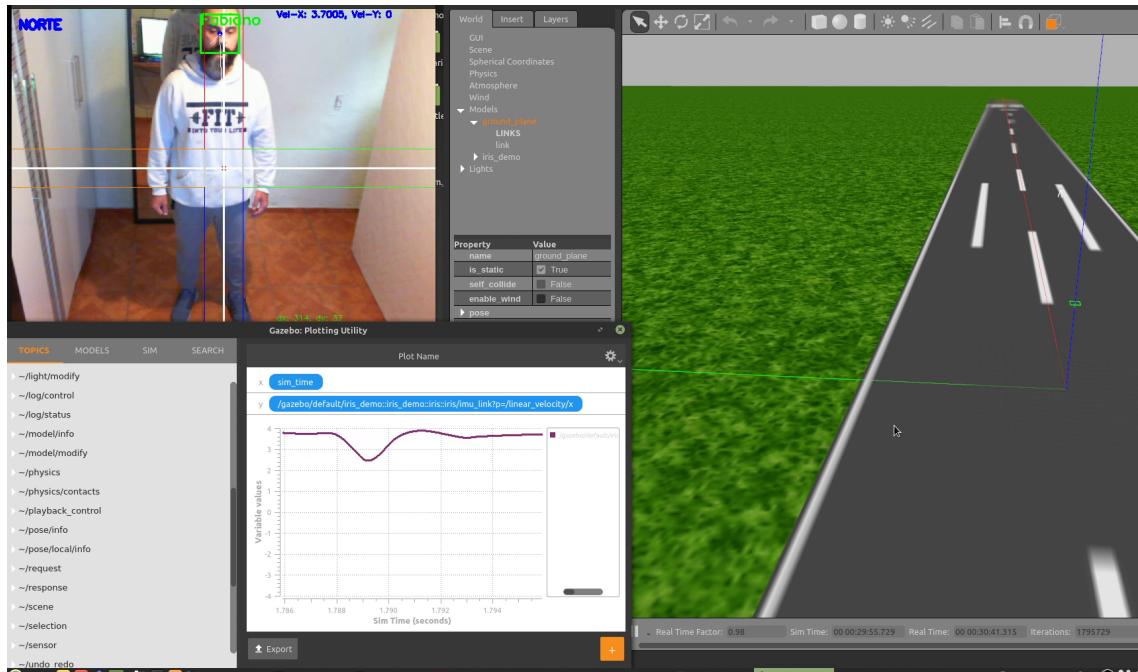
Fonte: do autor

Figura 58 – Segunda Simulação de Velocidade para o Norte



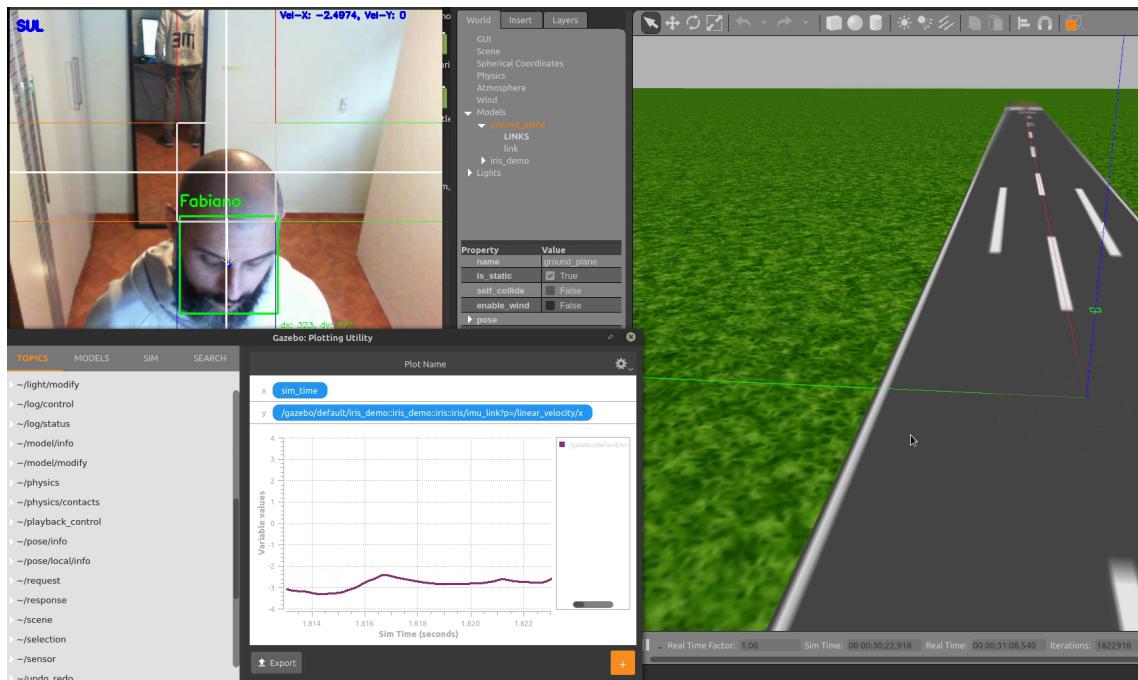
Fonte: do autor

Figura 59 – Terceira Simulação de Velocidade para o Norte



Fonte: do autor

Figura 60 – Simulação de Velocidade para o Sul



Fonte: do autor

4.2 Considerações Finais

Com tudo isso se chegou a conclusão de que é possível controlar um VANT usando visão computacional, porem como a maioria dos trabalhos nessa área são de caráter experimental, se decidiu criar uma bateria de simulações utilizando softwares e implementar um algoritmo que une visão computacional (OpenCV para reconhecimento facial) com tecnologia de controle para VANT (Dronekit) e checar se as reações da simulação são suficientemente satisfatórias. Logo se determinou que o VANT ir na direção pré-determinada no algoritmo é o ponto de satisfação para uma conclusão positiva do protótipo.

Apesar do sistema de simulação não ter sido totalmente satisfatório, é possível evoluir o produto aplicando mais testes, evoluindo o sistema de simulação, agregando mais recursos e utilizando equipamentos mais específicos para este fim.

Em relação aos componentes usados a maioria eram do próprio autor e não foram adquiridos outros devido ao custo devidamente auto para uma pessoa física.

Em relação aos métodos usados na analise, a simulação por software comprovou ser ideal para os primeiros testes, porem se fez necessário possuir um computador com uma alta capacidade de processamento, e foi utilizado o computador que o autor ja possuía.

Alguns paradigma em relação ao desenvolvimento desse projeto:

- Em relação ao VANT o principal paradigma a ser vencido é a autonomia que na atualidade fica em torno de no máximo 45 minutos de operação.
- Em relação a tecnologia de visão computacional o paradigma a ser vencido é o de capacidade de processamento.

Alguns aspectos que podem ser melhorados em versões futuras para tornar o projeto mais eficiente e utilizável são:

- Utilizar um equipamento com maior poder de processamento e específico para o fim.
- Maior cronograma de tempo para o desenvolvimento de um sistema de simulação mais completo e eficiente.
- Estender as técnicas de visão computacional para melhoria do sistema de sensoriamento de pessoas.
- Utilizar um sistema de controle do tipo PID ou lógica Fuzzi para controlar o VANT.
- Adquirir os componentes físicos para testes reais.

Referências

- ARDUPILOT. *ArduPilot Documentation*. [S.l.], 2019. Disponível em: <<https://ardupilot.org/ardupilot/>>. Acesso em: 2019. Citado 7 vezes nas páginas 21, 24, 27, 29, 30, 31 e 32.
- ARDUPILOT.ORG. *Communicating with Raspberry Pi via MAVLink*. 2020. Disponível em: <<https://ardupilot.org/dev/docs/raspberry-pi-via-mavlink.html>>. Citado 2 vezes nas páginas 2 e 3.
- BRUM, H. K. M. *Brasil não soluciona nem 10% dos seus homicídios*. Paraná, 2018. Disponível em: <<https://www.gazetadopovo.com.br/ideias/brasil-nao-soluciona-nem-10-dos-seus-homicidios-d726kw8ykpwh6xm41zakgzoue/>>. Acesso em: 17 set. 2019. Citado na página 1.
- BURGGRÄF ALEJANDRO RICARDO PÉREZ MARTÍNEZ, H. R. J. W. P. uadrotors in factory applications: design and implementation of the quadrotor's p-pid cascade control system. *SN Applied Sciences*, Springer, v. 1, n. 722, 2019. Disponível em: <<https://doi.org/10.1007/s42452-019-0698-7>>. Citado 2 vezes nas páginas 36 e 39.
- CYPRIANO, R. M. M. I. M. A. *ESTUDO DAS FORÇAS GERADAS POR UMA HÉLICE*. [S.l.], 2014. Disponível em: <http://www.engenhariamecanica.ufes.br/sites/engenhariamecanica.ufes.br/files/field/anexo/2015.1_-_marcos_cypriano_roberto_makio.pdf>. Citado na página 35.
- DRONECODE. *mRo Pixhawk Flight Controller (Pixhawk 1)*. 2020. Disponível em: <https://docs.px4.io/v1.9.0/en/flight_controller/mro_pixhawk.html>. Citado 2 vezes nas páginas 3 e 26.
- GEITGEY, A. Machine learning is fun! part 4: Modern face recognition with deep learning. Medium.com, 2016. Disponível em: <<https://medium.com/@ageitgey/machine-learning-is-fun-part-4-modern-face-recognition-with-deep-learning-c3cffc121d78>>. Citado 3 vezes nas páginas 14, 15 e 16.
- HE, K. et al. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. Disponível em: <<http://arxiv.org/abs/1512.03385>>. Citado na página 15.
- JSBSIM. *JSBSim Reference Manual*. 2020. Disponível em: <<https://jsbsim-team.github.io/jsbsim-reference-manual/mypages/user-manual-frames-of-reference/>>. Citado na página 43.
- KING, D. High quality face recognition with deep metric learning. 2017. Disponível em: <<http://blog.dlib.net/2017/02/high-quality-face-recognition-with-deep.html>>. Citado na página 16.
- KISSAI, A.; SMITH, M. Electronic navigation system based on the use of alternate coordinate system and polar stereographic projection for uavs operating in polar regions. *International Journal of Aeronautics and Aerospace Engineering*, v. 1, p. 46–53, 05 2019. Citado na página 43.

- LONGARINI, A. *Redes Neurais Artificiais / As máquinas pensam?* [S.l.], 2019. Disponível em: <<https://www.lambda3.com.br/2019/09/redes-neurais-artificiais-as-maquinas-pensam/>>. Acesso em: 2019. Citado na página 11.
- MENG, X. et al. Contact force control of an aerial manipulator in pressing an emergency switch process. In: . [S.l.: s.n.], 2018. p. 2107–2113. Citado na página 37.
- MSARDONINI. *The Basic Physics of the Quad-Copter*. [S.l.], 2015. Disponível em: <<https://beaglequad.wordpress.com/2015/10/03/the-basic-physics-of-the-quad-copter/>>. Acesso em: 2020. Citado na página 40.
- MUNGUÍA, R. A gps-aided inertial navigation system in direct configuration. *Journal of Applied Research and Technology*, v. 12, p. 803–814, 08 2014. Citado na página 43.
- NVIDIA.COM. *Nvidia Jetson Nano*. 2020. Disponível em: <<https://www.nvidia.com/pt-br/autonomous-machines/embedded-systems/jetson-nano/>>. Nenhuma citação no texto.
- PX4.IO. *Px4 Documentation*. [S.l.], 2020. Disponível em: <<https://px4.io/>>. Acesso em: 2019. Citado na página 27.
- RAMPAZZO, L. *Metodologia científica*. [S.l.]: Edições Loyola, 2005. Citado na página 44.
- RASPBERRYPI.ORG. *Raspberry Pi 3 Model B*. 2020. Disponível em: <<https://www.raspberrypi.org/products/raspberry-pi-3-model-b/>>. Citado na página 3.
- RASPBERRYPI.ORG. *Raspberrypi documentation*. 2020. Disponível em: <<https://www.raspberrypi.org/documentation/>>. Citado na página 64.
- ROSEBROCK, A. Face recognition with opencv, python, and deep learning. pyimagesearch.com, 2018. Disponível em: <<https://www.pyimagesearch.com/2018/06/18/face-recognition-with-opencv-python-and-deep-learning/>>. Citado na página 15.
- SANTOS, A. M. S. P. *Políticas sociais em xeque: impactos da crise sobre as finanças municipais*. [S.l.], 2018. 35-46 p. Disponível em: <<https://royaltiesdopetroleo.ucam-campos.br/wp-content/uploads/2018/12/boletim-royalties-N61-dezem-2018-artigo-4.pdf>>. Citado na página 5.
- SANTOS, L. A. *Artificial intelligence and machine learning*. GitBook, 2018. 65 p. Disponível em: <https://leonardoaraujosantos.gitbooks.io/artificial-intelligence/content/linear_algebra.html>. Citado na página 8.
- SILVA, V. H. *Rio de Janeiro identificou 8 mil pessoas com reconhecimento facial no Carnaval*. 2019. Disponível em: <<https://tecnoblog.net/289696/rio-de-janeiro-identificou-8-mil-reconhecimento-facial/>>. Citado na página 5.
- SOUZA, D. A. de et al. Câmeras de segurança e seus sistemas tecnológicos: Percepções sobre os motivos da utilização. p. 1–13, 2016. Disponível em: <<https://www.aedb.br/seget/arquivos/artigos17/12425136.pdf>>. Citado na página 5.

TEFAY, B. et al. Design of an integrated electronic speed controller for compact robotic vehicles. In: *Proc. of Australasian Conf. Robotics and Automation*. [S.l.: s.n.], 2011. p. 1–8. Citado na página 30.

TIERNEY ORACLE GROUNDBREAKER AMBASSADOR, O. A. D. B. *Understanding, Building and Using Neural Network Machine Learning Models using Oracle 18c*. [S.l.], 2018. Disponível em: <<https://developer.oracle.com/databases/neural-network-machine-learning.html>>. Acesso em: 2019. Citado na página 10.

TURING, A. M. Computer machinery and intelligence. *Parsing the Turing Test*, Springer, DORDRECHT, p. 23–65, 2009. Citado na página 6.

ZMOGINSKI, F. *A sociedade mais vigiada do mundo: como a China usa o reconhecimento facial*. [S.l.], 2019. Disponível em: <<https://www.uol.com.br/tilt/noticias/redacao/2019/01/19/a-sociedade-mais-vigiada-do-mundo-como-a-china-usa-o-reconhecimento-facial.htm>>. Acesso em: 20 set. 2019. Citado na página 1.

Apêndices

APÊNDICE A – Código Fonte da Visão Computacional

```

# USAGE
# python recognize_faces_video.py --encodings encodings.pickle

# importação das bibliotecas
# import the necessary packages
from collections import deque
from imutils.video import VideoStream
from imutils.face_utils import rect_to_bb
from decimal import Decimal
from dronekit import connect, VehicleMode, LocationGlobalRelative, APIException
from functionsControlDrone import connectMyCopter,
arm_and_takeoff, send_local_ned_velocity, send_global_ned_velocity
import face_recognition
import argparse
import imutils
import pickle
import time
import numpy as np
import cv2

# contrução dos argumentos passados
# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-e", "--encodings", default="encodings.pickle",
help="path to serialized db of facial encodings")
ap.add_argument("-o", "--output", type=str,
help="path to output video")
ap.add_argument("-y", "--display", type=int, default=1,
help="whether or not to display output frame to screen")
ap.add_argument("-d", "--detection-method", type=str, default="cnn",
help="face detection model to use: either `hog` or `cnn`")
ap.add_argument("-b", "--buffer", type=int, default=64,
help="max buffer size")
ap.add_argument('--connect')
args = vars(ap.parse_args())

# conectar o VANT (MAVLink)
# connect the UAV (MAVLink)
vehicle = connectMyCopter()

```

```
#def vision():
# load the known faces and embeddings
# carrega o arquivo de faces conhecidas pré-treinada
print("[INFO] loading encodings...")
data = pickle.loads(open(args["encodings"], "rb").read())

pts = deque(maxlen=args["buffer"])
counter = 0
(dX, dY) = (0, 0)
direction = ""
auxX=""
auxY=""
auxVelX=""
auxVely=""
VELMAX = 5

# allow the camera sensor to warm up
# aguarda a camera inicializar
print("[INFO] starting video stream...")
vs = VideoStream(src=0).start()
writer = None
time.sleep(2.0)

# loop sobre quadros do fluxo de arquivos de vídeo
# loop over frames from the video file stream
while True:

    # pegue o quadro do fluxo de vídeo encadeado
    # grab the frame from the threaded video stream
    frame = vs.read()

    # converta o quadro de entrada de BGR para RGB e redimensione-o para ter
    # uma largura de 750px (para acelerar o processamento)
    # convert the input frame from BGR to RGB then resize it to have
    # a width of 750px (to speedup processing)
    rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    rgb = imutils.resize(frame, width=750)
    r = frame.shape[1] / float(rgb.shape[1])

    # calcula varias coordenadas de posicionamento x,y em pixels na tela do opencv
    # calculates various x, y positioning coordinates in pixels on the opencv screen
    (height, width) = frame.shape[:2]
    startX = int(height-height)
    startY = int(width-width)
    centerScreenX = int(width/2)
```

```
centerScreenY = int(height/2)

# converte em string
# max_w_str = str(width)
# max_h_str = str(height)

#cv2.circle(frame,(width,height),10, (0,21,255),-2)
#cv2.putText(frame, "dx: {} , dy: {}".format(max_w_str, max_h_str),
#(frame.shape[0]+40, frame.shape[0] - 10), cv2.FONT_HERSHEY_SIMPLEX,0.40,
#(0,21,255), 1)

cv2.circle(frame,(centerScreenX,centerScreenY),4, (0,21,255),-2)

# detecta as coordenadas (x, y) das caixas delimitadoras
# correspondente a cada face no quadro de entrada e depois calcule
# os tratamentos faciais para cada rosto
# detect the (x, y)-coordinates of the bounding boxes
# corresponding to each face in the input frame, then compute
# the facial embeddings for each face
boxes = face_recognition.face_locations(rgb, model=args["detection_method"])
encodings = face_recognition.face_encodings(rgb, boxes)
center = None
names = []

cv2.line(frame,(centerScreenX,0),(centerScreenX,480), (255,255,255),2)
cv2.line(frame,(0,centerScreenY),(640,centerScreenY), (255,255,255),2)

# loop sobre os revestimentos faciais
# loop over the facial embeddings
for encoding in encodings:

    # tente combinar cada rosto na imagem de entrada com os nossos conhecidos
    # attempt to match each face in the input image to our known
    # encodings
    matches = face_recognition.compare_faces(data["encodings"],
        encoding)
    name = "Unknown"

    # verifique se encontramos uma correspondência
    # check to see if we have found a match
    if True in matches:

        # encontre os índices de todas as faces correspondentes e inicialize um
        # dictionary para contar o número total de vezes que cada face
        # foi correspondido
        # find the indexes of all matched faces then initialize a
        # dictionary to count the total number of times each face
```

```

# was matched
matchedIdxs = [i for (i, b) in enumerate(matches) if b]
counts = {}

# faz um loop sobre os índices correspondentes e mantém uma contagem para
# cada rosto reconhecido
# loop over the matched indexes and maintain a count for
# each recognized face
for i in matchedIdxs:
    name = data["names"][i]
    counts[name] = counts.get(name, 0) + 1

# determinar a face reconhecida com o maior número
# numero de votos (nota: no caso de um empate improvável em Python
# seleciona a primeira entrada no dicionário)
# determine the recognized face with the largest number
# of votes (note: in the event of an unlikely tie Python
# will select first entry in the dictionary)
name = max(counts, key=counts.get)

# atualiza a lista de nomes
# update the list of names
names.append(name)

# passe pelas faces reconhecidas
# loop over the recognized faces
for ((top, right, bottom, left), name) in zip(boxes, names):

    # redimensionar as coordenadas do rosto
    # rescale the face coordinates
    top = int(top * r)
    right = int(right * r)
    bottom = int(bottom * r)
    left = int(left * r)

    topS = str(top)
    rightS = str(right)
    bottomS = str(bottom)
    leftS = str(left)

    # Pontos X e Y do centro do retângulo da face
    # X and Y points in the center of the face rectangle
    cx = int((left + right)/2.0)
    cy = int((top + bottom)/2.0)
    center = (cx, cy)
    cx_str = str(cx)
    cy_str = str(cy)

```

```

# desenha o retangulo, nome do rosto, centro do quadro
# draw the predicted face name on the image
cv2.rectangle(frame, (left, top), (right, bottom),
(0, 255, 0), 2)
y = top - 15 if top - 15 > 15 else top + 15
cv2.putText(frame, name, (left, y), cv2.FONT_HERSHEY_SIMPLEX,
0.75, (0, 255, 0), 2)
cv2.circle(frame,(cx,cy),4, (255),-2)
cv2.putText(frame, "dx: {} , dy: {}".format(cx_str, cy_str),
(400, frame.shape[0] - 10), cv2.FONT_HERSHEY_SIMPLEX,0.40, (0, 255, 0), 1)
cv2.arrowedLine(frame,(centerScreenX,centerScreenY),(cx,cy),
(255, 255, 255),1, 8, 0, 0.1)
pts.appendleft(center)

x1=1
y1=1

# determina a região (retângulo) de não interesse
# determines the region (rectangle) of no interest
l=int((cx-left)*1.025) #delta width
a=int((cy-top)*1.025) #delta height
PaX = int(centerScreenX-a)
PaY = int(centerScreenY-l)
PbX = int(centerScreenX+a)
PbY = int(centerScreenY+l)

# desenha as regioes de interesse
# draw the regions of interest
cv2.rectangle(frame, (PaX,PaY), (PbX,PbY), (211,211,211), 2)
cv2.line(frame,(startX,PaY),(PaX,PaY), (0,128,255),1) #laranja orange
cv2.line(frame,(startX,PbY),(PaX,PbY), (0,128,255),1) #laranja orange
cv2.line(frame,(PaX,height),(PaX,PbY), (255),1) #azul blue
cv2.line(frame,(PbX,height),(PbX,PbY), (255),1) #azul blue
cv2.line(frame,(PbX,PbY),(width,PbY), (20,255,57),1) #verde green
cv2.line(frame,(PbX,PaY),(width,PaY), (20,255,57),1) #verde green
cv2.line(frame,(PaX,PaY),(PaX,startY), (0,0,255),1) #vermelho red
cv2.line(frame,(PbX,PaY),(PbX,startY), (0,0,255),1) #vermelho red

# valor da componente de velocidade
# speed component value
COMPVX=Decimal(VELMAX/centerScreenY)
COMPVY=Decimal(VELMAX/centerScreenX)
COMPV=Decimal((COMPVX+COMPVY)/2)

# verifica em qual regiao de interesse a face esta,
# calcula as velocidades

```

```

# envia a informação para o VANT
# check in which region of interest the face is
# calculates speeds
# sends the information to the UAV
if cx in range(PaX,PbX) and cy in range(startY,PaY):
    print("NORTE")
    direction = "NORTE"
    auxX = int(centerScreenY-cy)
    auxVelY = 0
    auxVelX = round(Decimal(auxX*COMPV),4)
    print(auxVelX)
    #send_local_ned_velocity(vehicle, auxX , 0, 0)
elif cx in range(PaX,PbX) and cy in range(PbY,height):
    print("SUL")
    direction = "SUL"
    auxX = int(centerScreenY-cy)
    auxVelY = 0
    auxVelX = round(Decimal(auxX*COMPV),4)
    print(auxVelX)
elif cx in range(PbX,width) and cy in range(PaY,PbY):
    print("LESTE")
    direction = "LESTE"
    auxVelX = 0
    auxY = int(cx-centerScreenX)
    auxVelY = round(Decimal(auxY*COMPVY),4)
    print(auxVelY)
elif cx in range(startX,PaX) and cy in range(PaY,PbY):
    print("OESTE")
    direction = "OESTE"
    auxVelX = 0
    auxY = int(cx-centerScreenX)
    auxVelY = round(Decimal(auxY*COMPVY),4)
    print(auxVelY)
elif cx in range(PbX,width) and cy in range(startY,PaY):
    print("NORDESTE")
    direction = "NORDESTE"
    if cx >= centerScreenX and cy <= centerScreenY:
        auxY = int(cx-centerScreenX)
        auxX = int(centerScreenY-cy)
        auxVelX = round(Decimal(auxX*COMPV),4)
        auxVelY = round(Decimal(auxY*COMPV),4)
elif cx in range(startX,PaX) and cy in range(startY,PaY):
    print("NOROESTE")
    direction = "NOROESTE"
    if cx <= centerScreenX and cy <= centerScreenY:
        auxY = int(cx-centerScreenX)
        auxX = int(centerScreenY-cy)

```

```

        auxVelX = round(Decimal(auxX*COMPV),4)
        auxVelY = round(Decimal(auxY*COMPV),4)

    elif cx in range(startX,PaX) and cy in range(PbY,height):
        print("SUDOESTE")
        direction = "SUDOESTE"
        if cx <= centerScreenX and cy >= centerScreenY:
            auxY = int(cx-centerScreenX)
            auxX = int(centerScreenY-cy)
            auxVelX = round(Decimal(auxX*COMPV),4)
            auxVelY = round(Decimal(auxY*COMPV),4)

    elif cx in range(PbX,width) and cy in range(PbY,height):
        print("SUDESTE")
        direction = "SUDESTE"
        if cx >= centerScreenX and cy >= centerScreenY:
            auxY = int(cx-centerScreenX)
            auxX = int(centerScreenY-cy)
            auxVelX = round(Decimal(auxX*COMPV),4)
            auxVelY = round(Decimal(auxY*COMPV),4)

    else:
        direction = ""
        auxX = 0
        auxY = 0
        auxVelX = 0
        auxVelY = 0

# desenha os valores das componentes de velocidade e a direção
# draws the values of the speed components and the direction
cv2.putText(frame, direction, (10, 30), cv2.FONT_HERSHEY_SIMPLEX,
0.65, (255), 3)
cv2.putText(frame, "Vel-X: {}, Vel-Y: {}".format(auxVelX, auxVelY),
(10, frame.shape[0] - 10), cv2.FONT_HERSHEY_SIMPLEX,
0.35, (255), 1)
counter += 1

# se o gravador de vídeo for None * AND *, devemos escrever
# o vídeo de saída em disco inicializa o gravador
# if the video writer is None *AND* we are supposed to write
# the output video to disk initialize the writer
if writer is None and args["output"] is not None:
    fourcc = cv2.VideoWriter_fourcc(*"MJPG")
    writer = cv2.VideoWriter(args["output"], fourcc, 20,
                           (frame.shape[1], frame.shape[0]), True)

# se o gravador não for None, escreva o quadro com as
# faces no disco
# if the writer is not None, write the frame with recognized
# faces to disk

```

```
if writer is not None:
    writer.write(frame)

# verifique se devemos exibir o quadro de saída
# na tela
# check to see if we are supposed to display the output frame to
# the screen
if args["display"] > 0:
    cv2.imshow("Frame", frame)
    key = cv2.waitKey(1) & 0xFF

# se a tecla `q` foi pressionada, interrompa o loop
# if the `q` key was pressed, break from the loop
if key == ord("q"):
    break

# faça uma limpeza
# do a bit of cleanup
cv2.destroyAllWindows()
vs.stop()

# verifique se o ponto do gravador de vídeo precisa ser liberado
# check to see if the video writer point needs to be released
if writer is not None:
    writer.release()
```

APÊNDICE B – Código Fonte dos Controles do VANT

```
#####
#DEPENDENCIES#####
#####

import dronekit
from dronekit import connect, VehicleMode, LocationGlobalRelative, APIException
import time
import socket
import math
import argparse
from pymavlink import mavutil
from click import exceptions
#####

#FUNCTIONS#####
#####

vehicle = None

def connectMyCopter():

    global vehicle
    try:

        parser = argparse.ArgumentParser(description='commands')
        parser.add_argument('--connect')
        args = parser.parse_args()

        connection_string = args.connect

        if not connection_string:
            import dronekit_sitl
            sitl = dronekit_sitl.start_default()
            connection_string = sitl.connection_string()

        vehicle = connect(connection_string, wait_ready=True, timeout=5)

    except socket.error:
        print ('SEM SERVIDOR SITL EXISTENTE!')
    except exceptions.OSError as e:
        print ('SEM COMINICAÇÃO SERIAL EXISTENTE!')
    except dronekit.APIException:
        print ('TEMPO DE CONEXAO EXCEDIDO!')
    except:
        print ('ACONTEREAM OUTROS ERROS DE CONEXAO!')

    return vehicle
```

```

def arm_and_takeoff(vehicle, targetHeight):
    while vehicle.is_armable!=True:
        print("Esperando o veiculo se armar")
        time.sleep(1)
    print("Veiculo armado")

    vehicle.mode = VehicleMode("GUIDED")

    while vehicle.mode!='GUIDED':
        print("Aguardando entrar em modo GUIDED")
        time.sleep(1)
    print("Veiculo em modo GUIDED")

    vehicle.armed = True
    while vehicle.armed==False:
        print("Esperando o veiculo se armar")
        time.sleep(1)
    print("Cuidado as helices virtuais estao em funcionamento")

    vehicle.simple_takeoff(targetHeight) ##meters

while True:
    print("Current Altitude: %d"%vehicle.location.global_relative_frame.alt,
          targetHeight)

    if vehicle.location.global_relative_frame.alt>=.92*targetHeight:
        break
    time.sleep(1)
print("Target altitude reached!!")
return None

def send_local_ned_velocity(vehicle, vx, vy, vz):
    """
    Move vehicle in direction based on specified velocity vectors.

    """
    msg = vehicle.message_factory.set_position_target_local_ned_encode(
        0,          # time_boot_ms (not used)
        0, 0,      # target system, target component
        mavutil.mavlink.MAV_FRAME_BODY_OFFSET_NED, # frame
        0b0000111111000111, # type_mask (only speeds enabled)
        0, 0, 0, # x, y, z positions (not used)
        vx, vy, vz, # x, y, z velocity in m/s
        0, 0, 0, # x, y, z acceleration (not supported yet, ignored in GCS_Mavlink)
        0, 0)     # yaw, yaw_rate (not supported yet, ignored in GCS_Mavlink)

```

```
# send command to vehicle on 1 Hz cycle
print(" Airspeed: %s" % vehicle.airspeed)
vehicle.send_mavlink(msg)
vehicle.flush()

def send_global_ned_velocity(vehicle, vx, vy, vz):
    """
    Move vehicle in direction based on specified velocity vectors.

    msg = vehicle.message_factory.set_position_target_local_ned_encode(
        0,          # time_boot_ms (not used)
        0, 0,       # target system, target component
        mavutil.mavlink.MAV_FRAME_LOCAL_NED, # frame
        0b000011111000111, # type_mask (only speeds enabled)
        0, 0, 0, # x, y, z positions (not used)
        vx, vy, vz, # x, y, z velocity in m/s
        0, 0, 0, # x, y, z acceleration (not supported yet, ignored in GCS_Mavlink)
        0, 0)      # yaw, yaw_rate (not supported yet, ignored in GCS_Mavlink)

    # send command to vehicle on 1 Hz cycle
    vehicle.send_mavlink(msg)
    vehicle.flush()
```