Projeto de Graduação

DEE DEPARTAMENTO DE ENGENHARIA ELÉTRICA

13 de Julho de 2018

NAVEGAÇÃO AUTÔNOMA, RECONHECIMENTO E DESVIO DE OBSTÁCULOS COM DRONES MULTIROTORES

Henrique Pinheiro Saraiva

PUC RIO

www.ele.puc-rio.br

Projeto de Graduação

DEE DEPARTAMENTO DE ENGENHARIA ELÉTRICA

NAVEGAÇÃO AUTÔNOMA, RECONHECIMENTO E DESVIO DE OBSTÁCULOS COM DRONES MULTIROTORES

Aluno: Henrique Pinheiro Saraiva

Orientador: Eduardo Costa da Silva

Trabalho apresentado como requisito parcial à conclusão do curso de Engenharia Elétrica na Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, Brasil.

Agradecimentos

Gostaria de agradecer, primeiramente, as minhas avós que tornaram meu sonho realidade e sempre acreditaram na minha capacidade de "construir coisas".

Agradeço, também, aos meus pais, Julia e Luiz, e meu irmão, Filipe, que sempre me incentivaram a continuar fazendo o que eu gosto e me ajudaram bastante durante a minha caminhada na vida.

Faço um agradecimento, também, aos colegas e professores que encontrei na PUC-RIO. Eles, por menor que fosse as nossas interações, sempre me ensinaram algo valioso. Agradeço especialmente aos meus bons amigos Guilherme Gusmão, Lucas Grativol, Carolina Maria e Karine Galvão que me acompanharam até o final da graduação e aos professores Wouter Caarls, Antônio Candea, Eduardo Costa e Mauro Schwanke, cuja os ensinamentos e ajudas foram fundamentais para o desenvolvimento desse trabalho.

Gostaria de agradecer, também, ao pessoal do laboratório de optoeletrônica do CETUC, especialmente ao professor Guilherme Temporão, Gustavo Amaral, Felipe Calliari e Vinícius Lima, cujos ensinamentos práticos e o trabalho em conjunto, foram importantes na minha vida acadêmica.

Por fim gostaria de agradecer toda a equipe do Departamento de Engenharia Elétrica da PUC-RIO por ceder toda a infraestrutura e ajuda que eu precisei para obter o título de engenheiro. Um agradecimento especial aos técnicos do LET, LAMAQ e LCA.

Resumo

Veículos aéreos não tripulados (*VANT*'s) são criações relativamente novas, concebidas em épocas onde as tecnologias disponíveis não permitiam sua adequada construção e operação. Recentemente, com o avanço tecnológico, *VANT*'s deixaram de ser apenas ideias e se tornaram realidade. O emprego de técnicas computacionais que possibilitem que tais dispositivos se tornem capazes de executar tarefas de modo autônomo tem sido um campo de intensos estudos nos dias atuais. Tipicamente, tais técnicas são compostas por algoritmos computacionais executados em unidades de processamento, embarcadas ou não nos *drones*, que tomam decisões em função de informações coletadas por meio sensores embarcados no *drone*. Apesar de, na grande maioria das aplicações, o estado da arte atual ter se mostrado capaz de atender as demandas associadas a capacidade de processamento computacional requerida para execução destes algoritmos, observa-se que o tempo de autonomia de voo desses dispositivos ainda representa um grande desafio, tendo em vista limitações tecnológicas, associadas principalmente na eficiência de motores, armazenamento de energia nas baterias e relação força de empuxo por peso. Dessa forma, técnicas que possibilitem a redução do peso da aeronave, mas ainda permitam um voo autônomo e confiável, são indispensáveis. O IMAV (*International Micro Air Vehicles Conference and Competition*) é uma das mais importantes competições internacionais focada em fomentar o desenvolvimento de tecnologias chave para o desenvolvimento de *drones* inteligentes. Os requisitos da competição demandam o desenvolvimento de sistemas autônomos, leves e energeticamente eficientes. O projeto aqui desenvolvido se insere neste contexto, visando contribuir para o desenvolvimento de um *VANT* autônomo para o IMAV2018, que ocorrerá em novembro de 2018 em Melbourne, Austrália. Em específico, objetiva-se contribuir para a navegação robusta do sistema por um dos elementos do percurso do IMAV2018, focando no desenvolvimento de algoritmos de processamento de imagem capazes de implementar um seguidor de corda, além do reconhecimento e desvio de obstáculos que possam estar na trajetória de navegação do *drone*. Os algoritmos computacionais desenvolvidos são descritos e avaliados. Os resultados dos testes efetuados indicaram um grande potencial da técnica desenvolvida para realização da navegação autônoma do *drone*, baseada em visão computacional.

Palavras-chave: Navegação autônoma, Drones, Reconhecimento de imagem, Visão computacional

AUTONOMOUS NAVIGATION, OBSTACLE RECOGNITION AND AVOIDANCE WITH MULTIROTOR DRONES

Abstract

Unmanned aerial vehicles (UAV's) are relatively new inventions, designed in an era where the available technologies did not allow them to be properly built and operated. Recently, with the technological progress, UAV's are no longer an idea, becoming a reality. Nowadays, the use of computational techniques that allow such devices to execute tasks autonomously has been an intense research field. Typically, such techniques are based on computational algorithms executed in processing units, embedded or not in a drone. Although, in most applications, the current state of the art is capable of meeting the requirements associated to the computational processing capacity required to execute these algorithms, it is observed that the flight autonomy time still represents a big challenge, due to technological limitations, associated primarily to the motors efficiency, the energy storage capacity in batteries and the thrust-to-weight ratio. In this way, weight reduction techniques of the aircraft that allow an autonomous and reliable flight are indispensable. The IMAV (International Micro Air Vehicles Conference and Competition) is one of the most important international competitions aimed at fostering the development of key technologies to the development of intelligent drones. The requirements of the competition demand the development of autonomous, light-weight and energetically efficients systems. The present work belongs to this scope, aiming to contribute to the development of an autonomous UAV to IMAV2018, that will take place in Melbourne, Australia, during November 2018. In particular, this work aims to contribute to the robust navigation of the system through one of the elements that compose the IMAV2018's challenge, focusing on the development of image processing algorithms capable to implement a line-follower, besides recognizing and avoiding possible obstacles that may be on the UAV path. The developed computational algorithms are described and evaluated. The results shows that the techniques used in this project has great potential to perform the autonomous navigation of the drone, based on computational vision.

Keywords: Autonomous navigation, Drones, Image recognition, Computacional Vision

Sumário

Lista de Figuras

1 Introdução

a Linha histórica: da V-1 aos *drones* multirotores

O surgimento de *drones*, ou aeronaves controladas remotamente, ocorreu durante a Segunda Guerra Mundial. Assim como outros grandes avanços na área bélica, os *drones* foram desenvolvidos por engenheiros alemães com o intuito de serem bombas remotamente controladas, com o intuito de permitir que os soldados germânicos pudessem atacar alvos de uma distância segura sem expor suas posições. Essas armas eram conhecidas como *buzz-bombs* devido ao som emitido por seus motores parecer com o zumbido de um abelha (*buzz* em inglês) tendo sido os precursores do míssil V-2, fato pelo qual também são conhecidas como V-1.

Após a Segunda Guerra Mundial, os veículos aéreos não-tripulados só foram ter um grande destaque, em termos de desenvolvimento, nos anos 60, quando um número grande de testes foi realizado pela marinha dos Estados Unidos da América. No entanto, somente no início dos anos 70 é que a Marinha admitiu o uso de *drones* ou *VANT*'s (Veículos Aéreos Não Tripulados). Durante a Guerra Fria os *VANT*'s foram utilizados tanto pelos Estados Unidos quanto pela União Soviética para espionar e fazer reconhecimento de terreno e instalações secretas.

Contudo, para operar esses *drones* era necessário um grande número de pessoas e seu custo era muito alto. Nos anos 80, o engenheiro espacial israelense, Abe Karem, vendo esse problema, resolveu criar um *drone* mais simples de controlar e mais barato. Karem criou o *Albatross* produzido com fibra de vidro, madeira e um motor de *kart* e pilotado por 3 operadores. Abe é considerado por muitos como o pai dos *drones* como conhecemos atualmente.

Figura 1: *Drone Albatross* junto com o seu criador, Abe Karem.

Foi apenas com o avanço da tecnologia de armazenamento de energia em baterias de lítio e com o aumento da eficiência de motores elétricos que foram criados os primeiros multirotores, mais especificamente os quadricópteros, que começaram a ser usados não só por militares e grupos de pesquisa, mas também pelo público em geral.

b Motivação e Justificativa

Nos dias atuais, é possível encontrar *drones* com diferentes características que os tornam mais ou menos adequados para execução de diferentes tipos de tarefas. Consequentemente, *drones* multirotores têm começado a expandir consideravelmente seu nicho de aplicações, sendo possível observar o emprego dos mesmos em diferentes áreas, tais como na indústria, comércio e mesmo para fins recreativos. É cada vez mais comum observar o emprego de multi-rotores em testes de entrega de mercadorias ou correspondência, para escaneamento e levantamento de áreas inóspitas, como observado em [1], bem como áreas desmatadas, além de serem amplamente utilizados em filmagens aéreas.

Considerando os usos já citados para *drones*, o grande desafio de engenharia para os *drones* atuais é o controle autônomo de voo. Grande parte das aplicações ainda é feita de forma não autônoma, onde um operador deve enviar comandos para o *drone*, apesar de cada vez mais a operação manual ser assistida por algoritmos de controle automáticos. Além disso, como exemplificado anteriormente, os *drones*, principalmente os multirotores, têm sido cada vez mais usados para fazer mapeamento de áreas. Tal atividade, é feita tipicamente empregando-se navegação semi- autônoma, na qual um piloto controla o *drone*, por meio de um rádio-controle e auxílio de uma câmera, visualizada remotamente em um visor monitor, ou envia uma rota de varredura pré-programada.

Atualmente, já existem controles autônomos que são usados em larga escala no meio acadêmico. Universidades como a *Penn State University*, nos Estados Unidos da América e a *National University of Singapore*, em Singapura são exemplos de Universidades que estão desenvolvendo projetos nessa área, como pode ser visto em [2], [3], [4]. O desenvolvimento de técnicas de controle autônomo é essencial para o desenvolvimento e incentivo à criação de atividades envolvendo multirotores em âmbito nacional.

c Objetivo

Este trabalho de conclusão de curso focará no desenvolvimento de algoritmos inteligentes para realização de tarefas autônomas com drones multirotores. Em particular, o objetivo deste projeto é criar um módulo de controle responsável por navegar autonomamente a aeronave em um labirinto, sem colidir com obstáculos presentes em seu caminho. Pretende-se que o presente trabalho contribua para o sistema que será desenvolvido para a *10th International Micro Air Vehicle Competition and Conference* (IMAV 2018). Os desenvolvimentos serão baseados no controlador de voo Navio 2, interconectado a um Raspberry Pi 3 B.

d Estrutura Textual

O primeiro capítulo apresenta uma breve contextualização histórica do surgimento e popularização dos *drones*, além de descrever o objetivo deste trabalho. Nas próximas seções serão descritas ferramentas matemáticas que auxiliarão a modelar e controlar um *drone* multirotor, possibilitando extrair pontos chaves de uma imagem que ajudarão a movimentar o multirotor por um labirinto, além de descrever o desenvolvimento de um módulo de reconhecimento e desvio de obstáculos.

O capítulo 2 descreve, de maneira resumida, uma das possíveis modelagens utilizadas para o controle de *drones*, mostrando, além das equações que serão usados no controle, as possíveis limitações do modelo. Além disso, é descrito um tipo de controle, bastante comum na área de VANT's, para o *drone*.

O capítulo 3 mostra, apresentando suas nuances, a técnica utilizada para fazer o reconhecimento de uma linha em um fundo de alto contraste usando visão computacional. Nesse capítulo são apresentados os resultados e possíveis limitações para a técnica escolhida.

O capítulo 4 descreve outra técnica de reconhecimento de objetos em imagens, mas dessa vez focando no reconhecimento de obstáculos. Assim como o capítulo 3 são apresentados resultados e possíveis limitações da técnica.

O capítulo 5 apresenta uma forma de, a partir da localização de um objeto em uma imagem e da localização do *drone*, executar um desvio caso haja uma suspeita de colisão.

O capítulo 6 apresenta as conclusões do trabalho e indica caminhos para possíveis aprimoramentos a serem implementados em trabalhos futuros.

e Visão Geral do projeto

De forma geral, as características do algoritmo de controle autônomo são descritas no algoritmo 1, apresentado na figura 2

Algorithm 1 Controle autônomo

Require: Câmera
  Obter imagem da câmera;
Require: Módulo de reconhecimento de imagem
  Usar imagem para obter a distância do *drone* ao centro da linha
Require: Módulo de reconhecimento de obstáculos **and** Módulo de desvio de obstáculos
  if Detectar objeto na trajetória **then**
    Alterar trajetória
  end if
  return Direção a ser tomada

Figura 2: Algoritmo de controle autônomo.

Os capítulos 3 a 5 apresentam os algoritmos dos módulos requeridos. O algoritmo 2, descrito de modo sucinto na figura 3 e apresentado no Capítulo 3, obtém o quanto o *drone* desviou da linha guia e faz com que o ele retorne para a linha. Os algoritmos 3 e 4 (figuras 4 e 5), descritos respectivamente nos capítulos 4 e 5, atuam em conjunto de modo a alterar a trajetória do *drone* em função da detecção de objetos na trajetória do *drone*, com os quais o mesmo poderia colidir, porém sempre garantindo o retorno do mesmo a trajetória definida pela linha, após o desvio.

Algorithm 2 Módulo de processamento de imagens e reconhecimento de linha

Require: Imagem da câmera
  Tratar a imagem
  Encontrar contornos
  Encontrar equação para contornos
Require: Câmera no centro do *drone*
  Verificar distância do centro da imagem para o centro da linha na abscissa do centro
  return Distância do centro da imagem para o centro da linha na abscissa do centro

Figura 3: Algoritmo do módulo de processamento de imagens e reconhecimento de linha.

Algorithm 3 Módulo de reconhecimento de obstáculos

Require: Imagem da câmera
  Tratar a imagem
  Encontrar contornos de objetos
  Remover a linha guia
  Delinear os objetos
  return Posição dos objetos na imagem

Figura 4: Algoritmo do módulo de processamento de imagens e reconhecimento de linha.

Algorithm 4 Módulo de desvio de obstáculos

Require: Imagem da câmera **and** Distância do centro da imagem para o centro da linha na abscissa do centro **and** Posição dos obstáculos
  if Algum objeto está dentro da área de segurança do *drone* **then**
    calcular para onde o *drone* deve desviar e de quanto deve ser o desvio
  else
    Não faz nada
  end if
  return Direção a ser tomada

Figura 5: Algoritmo do módulo de desvio de obstáculos.

2 Modelagem e controle de multirotores

a Ferramental teórico

Para modelar um *drone* multirotor é necessário, primeiramente, entender algumas ferramentas básicas de cálculo e álgebra linear. A seguir pode-se ver a ferramenta mais importante para a modelagem e o controle do *drone*: a matriz de rotação entre eixos referenciais.

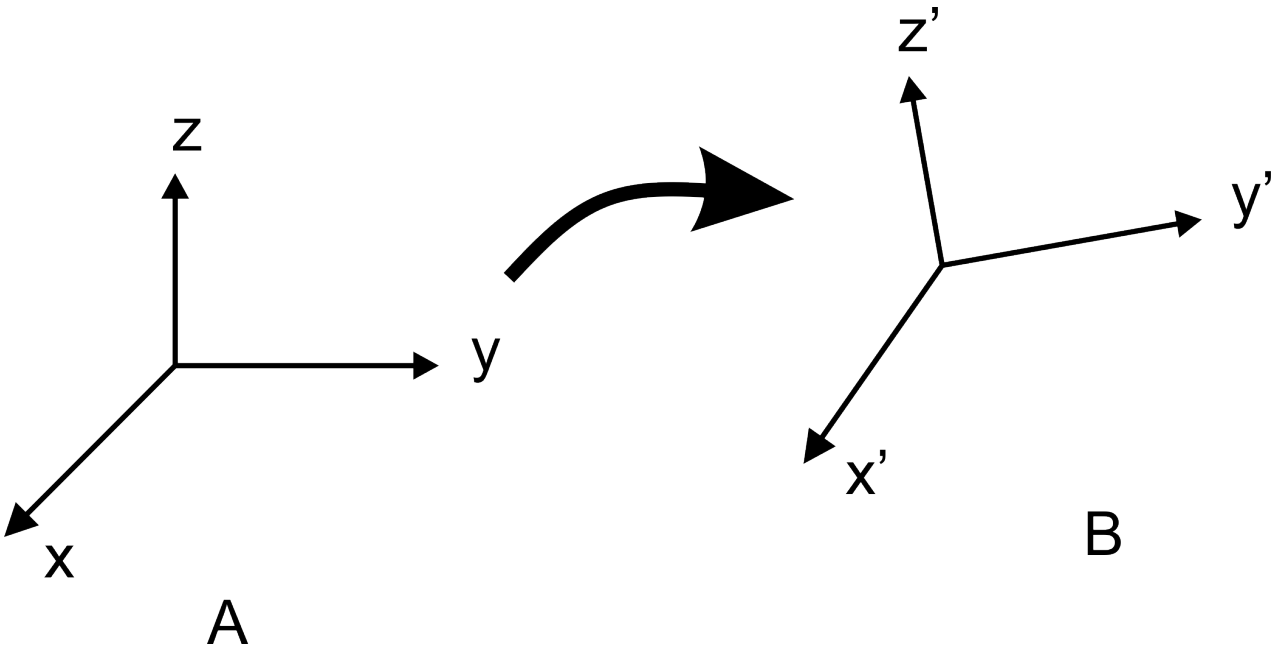


Figura 6: Representação da transformação do *frame* ***A*** para o *frame* ***B***.

Considerando a figura 6 em que pode-se ver a passagem de um *frame*(como referenciado na comunidade de *drones*), ou simplesmente de um grupo eixos cartesianos ortogonais e mutuamente exclusivos, para outro. Nessa transição é possível ver que há a necessidade de alguma operação matemática para transformar os vetores do *frame* A para os vetores do *frame* B. Ou seja,

$$b_1 = r_{11}a_1 + r_{12}a_2 + r_{13}a_3; \tag{1}$$
$$b_2 = r_{21}a_1 + r_{22}a_2 + r_{23}a_3; \tag{2}$$
$$b_3 = r_{31}a_1 + r_{32}a_2 + r_{33}a_3; \tag{3}$$

Observando as equações (1),(2) e (3) pode-se reescrever na forma matricial

$$\begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \times \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}; \tag{4}$$

$$\mathbf{b} = \mathbf{R} \times \mathbf{a}; \tag{5}$$

O motivo de propor essa transição de um *frame* para o outro é que em muitos, se não todos, casos têm-se dois *frames* principais denominados como **eixos do mundo** e **eixos do *drone***. Geralmente as coordenadas de localização são dadas por *G.P.S.* que utilizam os **eixos do mundo** ao invés do *drone*. Assim, para gerar sinais de controle que dependam da posição espacial do *drone* é necessário fazer a conversão de um *frame* para o outro.

Para facilitar o entendimento e a organização desse trabalho, a letra ***W*** é usada em referência ao *frame* do mundo e a letra ***D*** em referência ao *frame* do *drone* multirotor, como demonstra a figura 7.

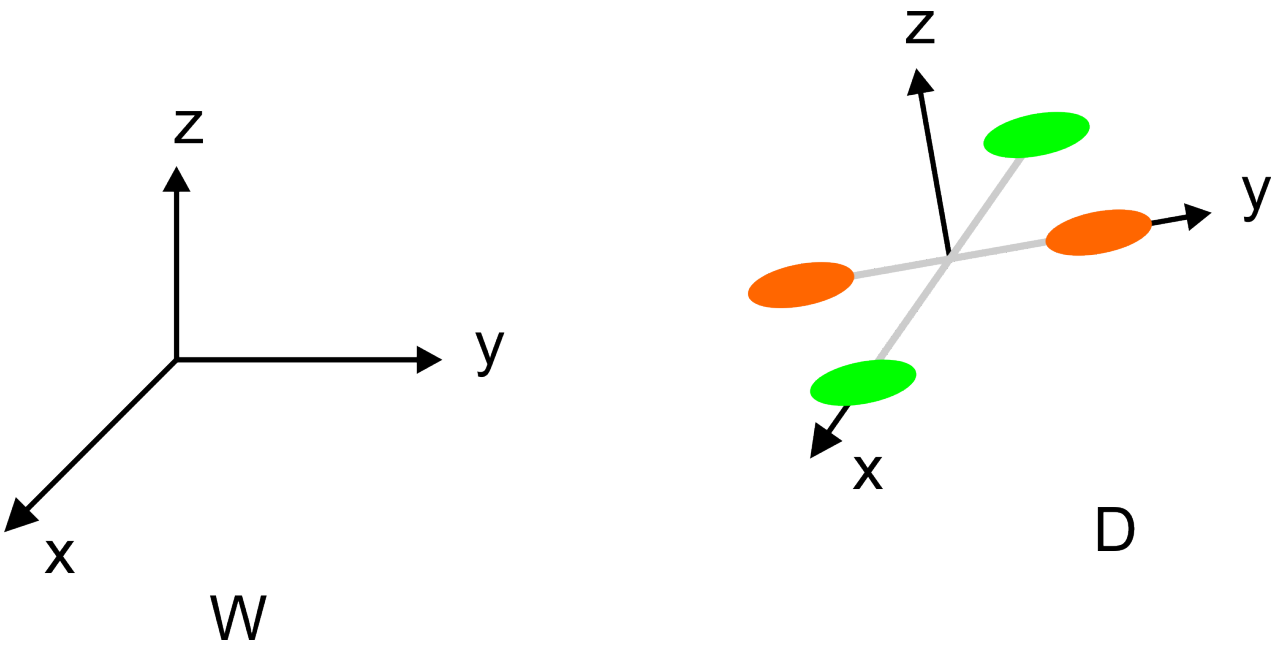


Figura 7: Representação adotada por esse trabalho. O *frame* à esquerda, ***W***, é o *frame* do mundo e à direita, ***D***, o do *drone*.

1 Ângulos de Euler

Tendo apresentado a matriz de rotação, precisa-se, agora, defini-la para que possa ser efetivamente utilizada. Para tal precisa-se de mais um conceito teórico, ângulos de Euler. Euler provou que para representar qualquer vetor de um *frame* em outro, precisa-se de apenas três rotações angulares sucessivas e em eixos diferentes para que isso seja possível. Os ângulos dessas rotações são denominados como ângulos de Euler. Assim, a matriz de rotação **R** (vista na equação (5)) pode ser construída da seguinte forma

$$^{W}R_{D} = {}^{W}R_{A} \times {}^{A}R_{B} \times {}^{B}R_{D}; \tag{6}$$

no qual o supraescrito representa o *frame* de origem e o subscrito o de destino. Os *frames* A e B são estados intermediários pelos quais deve-se criar com o intuito de facilitar a criação da matriz de rotação $^{W}R_{D}$

Dessa maneira, ao fazer uma rotação em torno do eixo z, depois uma rotação em torno do eixo x e por último uma rotação em torno do eixo y, obtêm-se a matriz de rotação que leva de ***W*** para ***D***. Tendo essa ideia em mente, pode-se propor as seguintes rotações

$$^{W}R_{D} = Rot(z,\psi) \times Rot(x,\phi) \times Rot(y,\theta); \tag{7}$$

no qual o operador *Rot(a,ω)* significa uma rotação de ω graus, em radianos, em torno do eixo *z*. Existe ainda uma outra notação bastante usada nas referências [5] e [6] por ser mais compacta

$$^{W}R_{D} = R_{z}(\psi) \times R_{x}(\phi) \times R_{y}(\theta); \tag{8}$$

no qual $R_{a}(\omega)$ representa uma rotação no eixo *a* em ω graus.

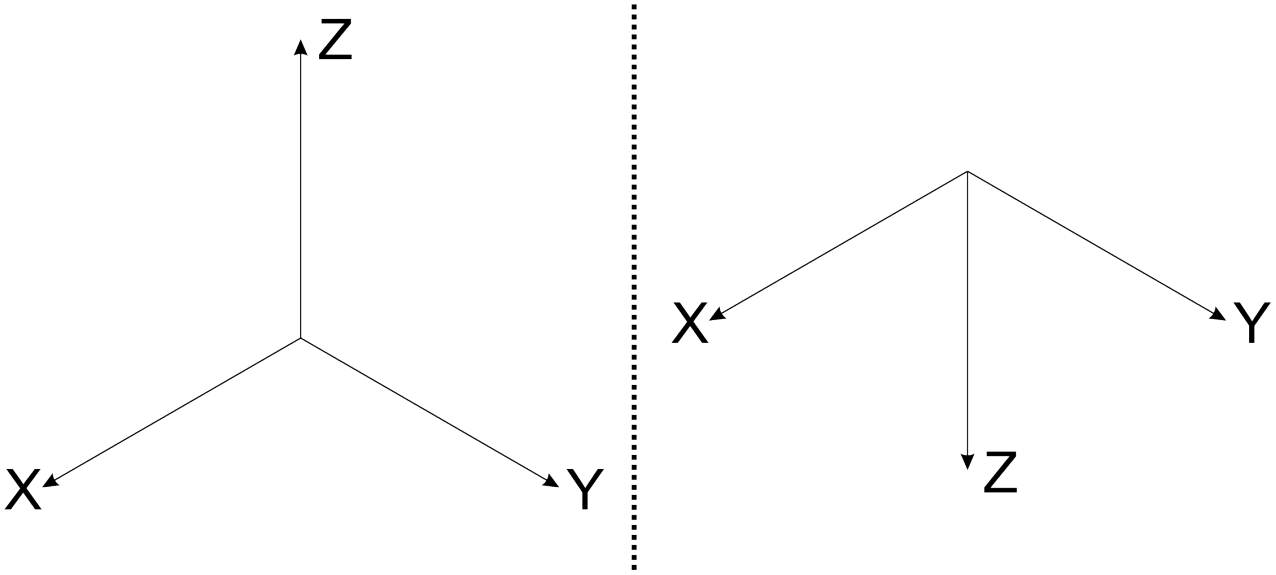


Figura 8: À direita está a convenção de eixos adotada por grande parte dos trabalhos em aeronáutica, à esquerda está a convenção adotada para esse trabalho.

A figura 8 utiliza e apresenta a mesma convenção de eixos adotados nesse trabalho. Usando-a é possível explicar o motivo da escolha da sequência de rotações propostas na equação (7). Considerando que a frente do *drone* esteja alinhada ao eixo *x* da figura 8, Rot(z,ψ) representa o *yaw*, Rot(x,ϕ) representa o *roll* e Rot(y,θ) representa o *pitch*. Assim, essas são as rotações comumente usadas quando há a referência de veículos aéreos, inclusive *drones*. E, além disso, esses ângulos são conhecidos como **ângulos Z-X-Y de Euler**, devido aos seus eixos de rotação.

Pode-se, com ajuda das figuras 9, 10 e 11 e sabendo que os ângulos de rotação são no sentido anti-horário, calcular as rotações dos eixos usando os vetores unitários que representam a base do espaço de cada *frame*. Então, para calcular o *roll* tem-se

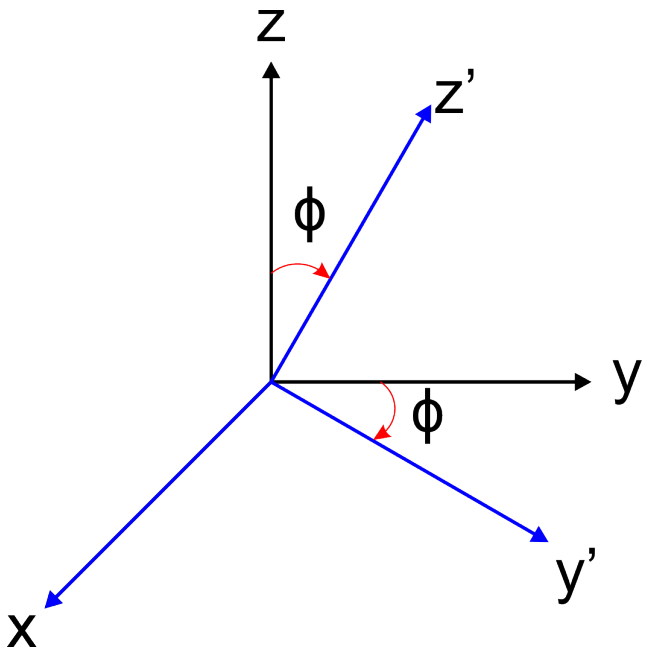


Figura 9: Representação do eixo original com um rotação de ϕ graus.

$$\mathbf{x'} = 1\mathbf{x} + 0\mathbf{y} + 0\mathbf{z} \tag{9}$$
$$\mathbf{y'} = 0\mathbf{x} + \cos(\phi)\mathbf{y} - \sin(\phi)\mathbf{z} \tag{10}$$
$$\mathbf{z'} = 0\mathbf{x} + \sin(\phi)\mathbf{y} + \cos(\phi)\mathbf{z} \tag{11}$$

Reescrevendo as equações (9), (10) e (11) na forma matricial e mais compacta

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{12}$$

nos quais *x'*, *y'* e *z'* são os eixos rotacionados e Rot(x,ϕ) é dado por

$$Rot(x, \psi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix} \tag{13}$$

De maneira similar pode-se encontrar Rot(y,θ) como

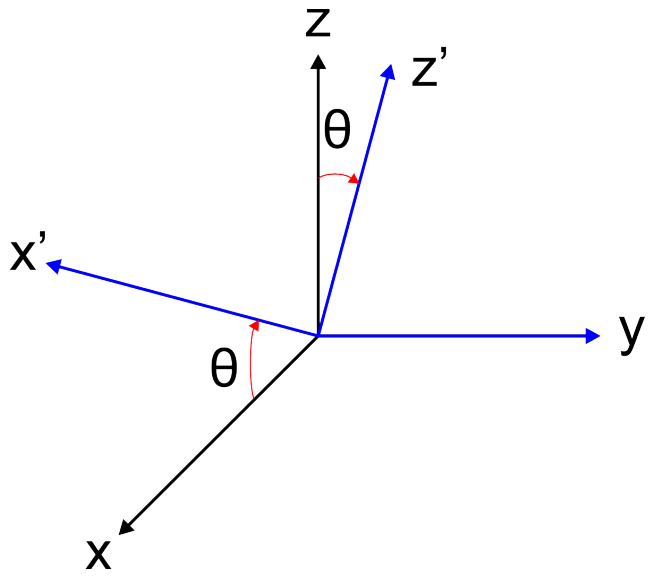


Figura 10: Representação do eixo original com um rotação de θ graus.

$$\mathbf{x'} = \cos(\theta)\mathbf{x} + 0\mathbf{y} - \sin(\theta)\mathbf{z} \tag{14}$$
$$\mathbf{y'} = 0\mathbf{x} + 1\mathbf{y} + 0\mathbf{z} \tag{15}$$
$$\mathbf{z'} = -\sin(\theta)\mathbf{x} + 0\mathbf{y} + \cos(\theta)\mathbf{z} \tag{16}$$

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{17}$$

$$Rot(y, \theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \tag{18}$$

e Rot(z,ψ) como

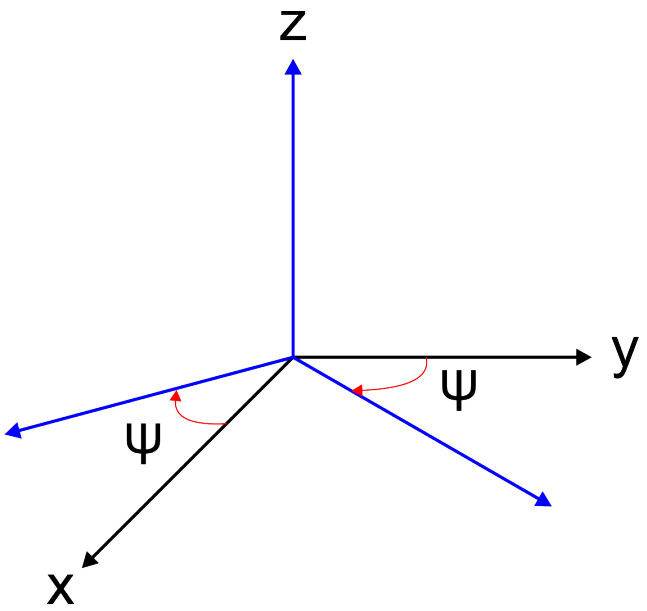


Figura 11: Representação do eixo original com um rotação de ψ graus.

$$\mathbf{x'} = \cos(\psi)\mathbf{x} - \sin(\psi)\mathbf{y} + 0\mathbf{z} \tag{19}$$
$$\mathbf{y'} = \sin(\psi)\mathbf{x} + \cos(\psi)\mathbf{y} + 0\mathbf{z} \tag{20}$$
$$\mathbf{z'} = 0\mathbf{x} + 0\mathbf{y} + 1\mathbf{z} \tag{21}$$

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{22}$$

$$Rot(z,\psi) = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{23}$$

Observe que foi possível criar três matrizes de rotação independentes uma das outras e, assim como previsto na teoria de Euler, pode-se criar a matriz de rotação ${}^{\boldsymbol{W}}R_{\boldsymbol{D}}$ usando as equações (7) e (13), (18) e (23)

$${}^{\boldsymbol{W}}R_{\boldsymbol{D}} = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix} \times \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \tag{24}$$

$${}^{\boldsymbol{W}}R_{\boldsymbol{D}} = \begin{bmatrix} \cos\psi\cos\theta + \sin\phi\sin\psi\sin\theta & \cos\phi\sin\psi & \cos\theta\sin\phi\sin\psi - \cos\psi\sin\theta \\ \cos\psi\sin\phi\sin\theta - \cos\theta\sin\psi & \cos\phi\cos\psi & \sin\psi\sin\theta + \cos\psi\cos\theta\sin\phi \\ \cos\phi\sin\theta & -\sin\phi & \cos\phi\cos\theta \end{bmatrix} \tag{25}$$

A escolha das rotações nessa ordem específica é dada para evitar certos problemas no mapeamento da matriz de rotação. O teorema dos ângulos de Euler permite criar a rotação de ***W*** para ***D***, mas seria interessante que fosse possível passar de ***D*** para ***W*** usando, de alguma forma, essa mesma matriz de rotação. Essa ideia é válida para quase todos os pontos do espaço e para quase todos os ângulos de rotação. Para ilustrar esse problema, pode-se observar a matriz de rotação dada por uma rotação em torno do eixo *z*, depois em torno do eixo *y* e depois em torno do eixo *z*

$$R_{zyz} \begin{bmatrix} \cos\phi\cos\theta\cos\psi - \sin\phi\sin\psi & -\cos\phi\cos\theta\sin\psi - \sin\phi\cos\psi & \cos\phi\sin\theta \\ \sin\phi\cos\theta\cos\psi + \cos\phi\sin\psi & -\sin\phi\cos\theta\sin\psi + \cos\phi\cos\psi & \sin\phi\sin\theta \\ -\sin\theta\cos\psi & \sin\theta\sin\psi & \cos\theta \end{bmatrix} \tag{26}$$

Observando o termo $\cos\theta$ pode-se assumir algumas condições, nas quais há a possibilidade de haver algum tipo de problema. Além disso, observando os termos $\sin\theta\sin\psi$, $-\sin\theta\cos\psi$, $\sin\phi\sin\theta$ e $\cos\phi\sin\theta$ pode-se afirmar que se θ for definido, ou seja, tenha seu valor conhecido, pode-se obter tanto ϕ quanto ψ. A seguir observa-se as condições previamente citadas

$$\cos\theta = 1 \tag{27}$$

$$\cos\theta = -1 \tag{28}$$

$$\|\cos\theta\| < 1 \tag{29}$$

Na primeira equação, para $\theta = 0$ e os termos citados anteriormente serão nulos e a matriz de rotação será função dos ângulos ϕ e ψ, não sendo possível determinar de forma única os valores desses ângulos. A segunda equação é similar à primeira, para $\theta = \Pi$ e os termos citados anteriormente serão nulos. Na terceira equação é possível calcular $\theta = \arccos(\cos\theta)$. No entanto, só se pode garantir o módulo de θ. Dessa forma, pode-se definir ψ e ϕ, contudo teremos duas equações, dois conjuntos de ângulos, que levarão para a mesma resposta. Dessa maneira, fazer a função inversa desses pontos não é possível, pois a função não será bi-univoca.

No caso desse problema, o θ seria o ângulo de *roll* e para solucionar isso, uma das possíveis forma, seria usar outras matrizes de rotação para tratar especificamente o caso explicitado, cujo mapeamento de maneira bi-univoca dos pontos não é possível. O que geraria um aumento na complexidade do problema de modelagem.

b Modelagem de *drones* multirotores

Para começar a modelar um multirotor, precisa-se estabelecer algumas convenções como os eixos que serão utilizados, a quantidade de rotores a serem modelados e os sentidos de rotação para cada hélice. Como dito anteriormente, o eixo da figura 8 será utilizado como padrão (diferentemente do padrão aero-espacial convencional). Além disso, *drones* diferentemente de veículos terrestres, são modelados usando modelos dinâmicos, em termos de forças e torques, ao invés de modelos cinemáticos. Isso ocorre porque usando o modelo dinâmico, obtém-se sistemas lineares e mais simples de controlar. Por fim, o *drone* multirotor usará quatro motores equidistantes entre si e em relação ao centro de massa do veículo, tendo os motores 1 e 3 rotação no sentido anti-horário e os motores 2 e 4, sentido horário, como pode ser visto na figura 12.

Inicialmente usa-se a segunda lei de Newton,

$$\sum \mathbf{F} = m \times \mathbf{a}; \tag{30}$$

no qual **F** é o vetor tri-dimensional de forças resultantes, m é a massa do *drone* e **a** é o vetor tri-dimensional de acelerações, para obter a expressão do *drone* quando parado.

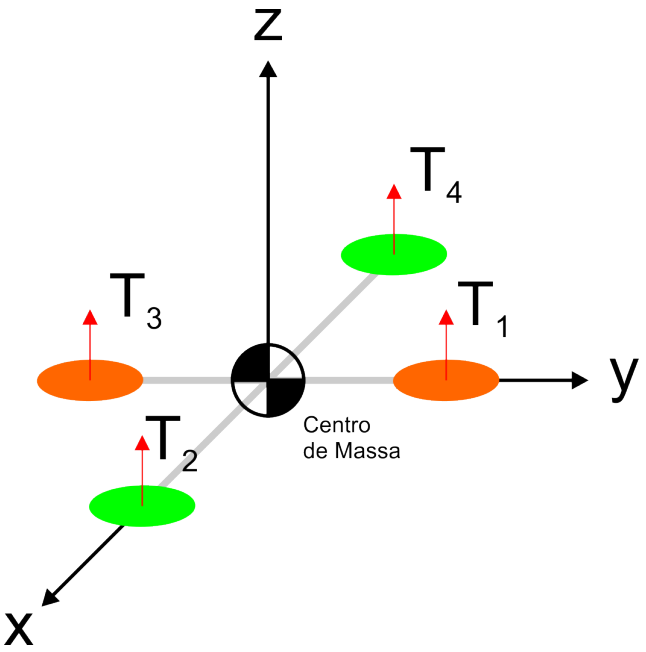


Figura 12: Representação da convenção de eixo adotada e das forças de sustentação geradas pelos rotores.

Usando a figura 12 e a equação (30) pode-se obter, em relação às coordenadas do mundo, a equação dinâmica translacional

$$m \times \dot{v} = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + {}^{\boldsymbol{W}}R_{\boldsymbol{D}} \times \begin{bmatrix} 0 \\ 0 \\ T \end{bmatrix}; \tag{31}$$

no qual $\dot{v}$ é a velocidade do *drone* nas coordenadas do mundo, m é a massa do *drone*, g é a gravidade, ${}^{\boldsymbol{W}}R_{\boldsymbol{D}}$ é a matriz de rotação e T é a soma de todos as forças de sustentação, ou *thurst* (empuxo). Esse último item pode ser definido de maneira simples quando observado a figura 12 como

$$T = T_1 + T_2 + T_3 + T_4; \tag{32}$$
$$T_1 = b\omega_1^2; \tag{33}$$
$$T_2 = b\omega_2^2; \tag{34}$$
$$T_3 = b\omega_3^2; \tag{35}$$
$$T_4 = b\omega_4^2; \tag{36}$$

no qual o subscrito indica o motor que gera aquela força de sustentação e *b* é a constante de sustentação (ou *lift constant*) que depende da densidade do ar, do número de hélices, da largura da pá (mais precisamente do *chord length*) e do cubo do raio das hélices. Sobre essa constante, é importante saber que ela é positiva, $b > 0$, isso será visto mais a frente.

Outro ponto importante é que a diferença de rotação entre os pares de rotores opostos gera rotação, principalmente, nos eixos x e y, ou seja, essas diferenças geram forças resultantes nos eixos citados. Assim, pode-se equacionar dois torques resultantes, τ_x e τ_y, da seguinte forma

$$\tau_x = dT_4 - dT_2; \tag{37}$$
$$\tau_y = dT_1 - dT_3; \tag{38}$$

no qual *d* é a distância entre um rotor e o centro de massa do *drone*. Pode-se, ainda, escrever as equações (37) e (38) em relação as velocidades angulares dos rotores

$$\tau_x = db(\omega_4^2 - \omega_2^2); \tag{39}$$
$$\tau_y = db(\omega_1^2 - \omega_3^2); \tag{40}$$

considerando que *d* e *b* são iguais para todos os rotores. O torque aplicado no eixo x é conhecido como *rolling torque* ou torque de rolagem e o torque aplicado ao eixo y é conhecido como *pitching torque* ou torque de arfagem.

Com o eixo z não é diferente, há um torque aplicado. Esse torque existe opondo o arrasto aerodinâmico. Pode-se equacionar esse torque no eixo z como

$$\tau_z = -Q_1 + Q_2 - Q_3 + Q_4; \tag{41}$$

no qual *Q* é o torque resultante de cada rotor em relação ao arrasto. Note que o sinal de cada torque é dado pela sua rotação. *Q* é definido como

$$Q_1 = k\omega_1^2; \tag{42}$$
$$Q_2 = k\omega_2^2; \tag{43}$$
$$Q_3 = k\omega_3^2; \tag{44}$$
$$Q_4 = k\omega_4^2; \tag{45}$$

no qual *k* é uma constante que depende dos mesmos fatores de *b*. Assim a equação (41) pode ser reescrita como

$$\tau_z = k(-\omega_1^2 + \omega_2^2 - \omega_3^2 + \omega_4^2); \tag{46}$$

Agora adiciona-se mais algumas equações que auxiliarão na modelagem do *drone*. Pretende-se usar as equações de movimento de Euler que é similar à segunda lei de Newton, mas é aplicada usando rotações. Dessa forma

$$I \times \dot{\omega} + \omega \times (I \times \omega) = \begin{bmatrix} \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} = \Gamma; \tag{47}$$

no qual **I** é a matriz de inércia do *drone*, ω é o vetor de velocidades angulares e Γ é o vetor de torque aplicados no *drone*.

Usando as equações (33) à (36), (39),(40) e (46) pode-se criar a expressão matricial

$$\begin{bmatrix} T \\ \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} = \begin{bmatrix} b & b & b & b \\ 0 & -db & 0 & db \\ db & 0 & -db & 0 \\ -k & k & -k & k \end{bmatrix} \begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix} = \mathbf{M} \begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix} \tag{48}$$

Note que se *b*,*d*,*k* são positivo e maiores que zero, a matriz **M** possui *rank* completo e, dentre outras coisa, é invertível. Dessa forma, tem-se

$$\begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix} = \mathbf{M}^{-1} \begin{bmatrix} T \\ \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} \tag{49}$$

$$\begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix} = \begin{bmatrix} \frac{1}{4b} & 0 & \frac{-1}{2db} & \frac{-1}{4k} \\ \frac{1}{4b} & \frac{1}{2db} & 0 & \frac{1}{4k} \\ \frac{1}{4b} & 0 & \frac{1}{2db} & \frac{-1}{4k} \\ \frac{1}{4b} & \frac{-1}{2db} & 0 & \frac{1}{4k} \end{bmatrix} \begin{bmatrix} T \\ \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} \tag{50}$$

Observe que a equação (50) é responsável por calcular os valores das rotações de cada rotor. Além disso, veja que *T* e Γ podem ser calculados independentes das velocidade angulares individuais dos rotores pelas equações (31) e (47).

Com a modelagem feita, o próximo passo é a criação de algum modo de controle de malha fechada para o *drone* multirotor.

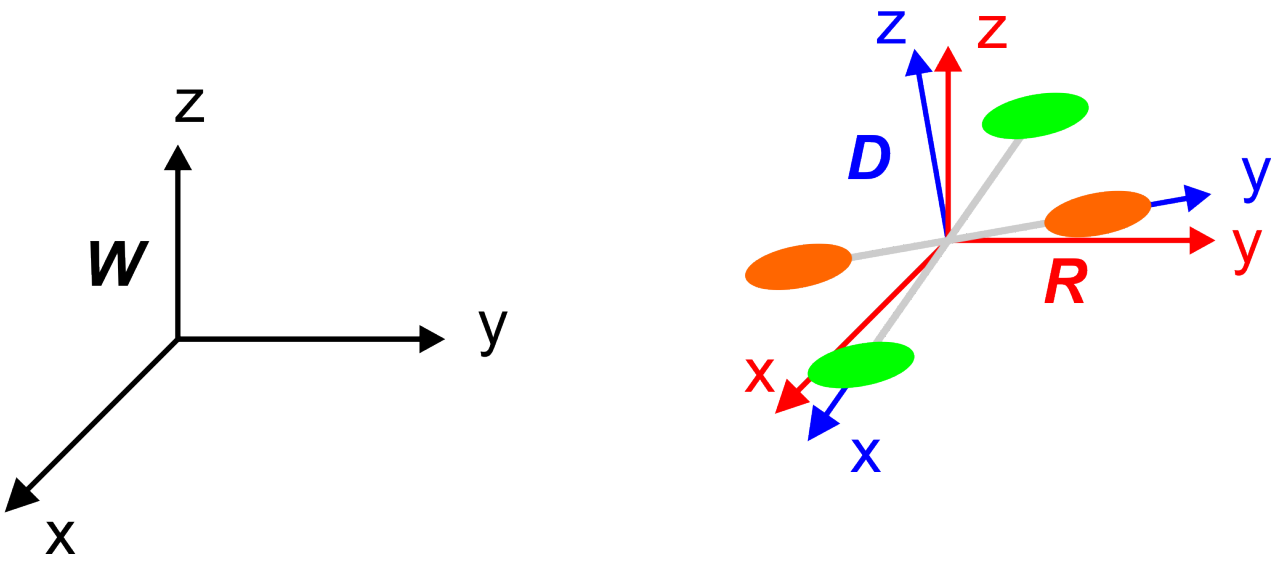


Figura 13: Representação do novo *frame* ***R*** junto com os *frames* vistos na figura 7.

c Controle de *drones* multirotores

Como visto na subseção b, com a equação (50) pode-se, através de torques, gerar um sinal de controle para as velocidades angulares de cada um dos motores do multirotor. Dessa maneira, usando controle clássico, por exemplo, pode-se controlar facilmente o *drone*. Assim, vamos fazer uso de uma estrutura de controle Proporcional, Integral ou Derivativo, para controlar determinadas variáveis.

Diferentemente de alguns controles de *drones*, não será controlada diretamente a posição do multirotor. Ao invés disso, controla-se os ângulos de *pitch*, *roll* e *yaw*. Isso porque se fosse desejado controlar a posição, toda a modelagem considerada não poderia ser aplicada, uma vez que ao usar um modelo cinemático em veículos aéreos, principalmente em quadricópteros, o modelo se torna não linear, aumentando a complexidade do controle e da modelagem. Dessa forma, as considerações de linearidade feitas seriam inutilizadas. Além disso, para o problema proposto, não há a necessidade de uma precisão tão grande (que seria obtida com o modelo cinemático) na movimentação do *drone*.

1 Controle no eixo X e no eixo Y

Agora almeja-se criar e analisar o controle proposto. Para simplificar a analise, será considerado um movimento na direção positiva do eixo x, ou um movimento para frente. Para realizar tal ação seria necessário que o *drone* fizesse um movimento de *pitch*, ou uma rotação no eixo y, de θ graus de maneira que os torques T_1, T_2, T_3 e T_4 (vistos na figura 12) gerassem uma componente na direção positiva de x, ou seja, esses torques, para o *drone* seguir na direção escolhida, devem gerar uma componente para frente.

Ainda para facilitar a analise, será considerado um novo *frame* ***R***, que terá como origem o centro de massa do *drone* (a mesma origem do *frame* ***D***) e terá os eixos paralelos aos do *frame* ***W***, como visto na figura 13. Para fazer o controle desse movimento, propõe-se, primeiramente, usar o controle proporcional-derivativo (ou simplesmente **P.D.**) do *pitching torque*, visto na equação (40). Assim tem-se

$$\tau_y = K_{p_{tau_y}}(\theta_p^* - \theta_p) + K_{d_{tau_y}}(\dot{\theta}_p^* - \dot{\theta}_p); \tag{51}$$

na qual $K_{p_{tau_y}}$ é a constante proporcional, $K_{d_{tau_y}}$ é a constante derivativa, θ_p^* representa o ângulo de rotação que deseja-se atingir, θ_p é o ângulo atual do *drone* em relação ao *frame* ***R*** ou ***W***, $\dot{\theta}_p^*$ é a velocidade angular desejada, $\dot{\theta}_p$ é a velocidade angular atual. O subscrito *p* de θ indica apenas que é um ângulo de *pitch*. Resumindo, o controle de τ_y é dado baseado no erro entre o ângulo que deseja-se e o ângulo atual, seja o próprio ângulo ou sua velocidade. O ângulo atual e a velocidade angular serão estimadas por uma unidade de medição de inércia, ou do acrônimo *I.M.U*(*Inertial Measurement Unit*). Para esse trabalho, a velocidade angular, que o *drone* precisa fazer para ir para frente, desejada é nula, assim,

$$\dot{\theta}_p^* = 0; \tag{52}$$

Agora com a lei de controle dada pela equação (51) e pela restrição dada por (52) falta apenas o ângulo θ_p^*. Observando mais uma vez a figura 13 e levando em consideração o torque total gerado pelos motores, tem-se

$$F = R_y(\theta_p^*) \begin{bmatrix} 0 \\ 0 \\ T \end{bmatrix} = \begin{bmatrix} T\sin\theta_p^* \\ 0 \\ T\cos\theta_p^* \end{bmatrix}; \tag{53}$$

Querendo verificar apenas a componente no eixo x,

$$F_x = T\sin\theta_p^*; \tag{54}$$

e como deseja-se que o movimento seja suave, pode-se considerar que o θ_p será um ângulo pequeno e assim pode-se fazer

$$F_x \approx T\theta_p^*; \tag{55}$$

É possível controlar a velocidade do *drone*, no *frame* ***R*** na direção x com apenas um controle proporcional, ou **P**. Não é necessário controlar a aceleração do *drone* nesse *frame* e nessa direção porque, como já está sendo controlado o angulo de *pitch*, indiretamente já controla-se a aceleração. O controle de F_x pode ser dado através do momento linear (lembrando que a derivada de um momento linear é uma força; discretamente, a variação de um momento é uma força), assim, tem-se uma equação para controlar a velocidade do *drone*. Então, a lei de controle para essa velocidade pode ser dada por

$$F_x^* = K_{p_{F_x}}(p_x^* - p_x); \tag{56}$$
$$F_x^* = K_{p_{F_x}}(m^{\boldsymbol{R}}v_x^* - m^{\boldsymbol{R}}v_x); \tag{57}$$
$$F_x^* = mK_{p_{F_x}}({}^{\boldsymbol{R}}v_x^* - {}^{\boldsymbol{R}}v_x); \tag{58}$$

no qual p_x^* é o momento linear desejado na direção x, p_x é o momento linear atual na direção x, *m* é a massa do *drone*, $K_{p_{F_x}}$ é a constante proporcional, ${}^{\boldsymbol{R}}v_x$ e ${}^{\boldsymbol{R}}v_x^*$ são as velocidades atual e desejada, respectivamente . Combinando as equações (55) e (58) tem-se

$$\theta_p^* = \frac{m}{T}K_{p_{F_x}}({}^{\boldsymbol{R}}v_x^* - {}^{\boldsymbol{R}}v_x); \tag{59}$$

Assim como o ângulo θ_p, pode-se estimar a velocidade atual usando um sistema de navegação por *G.P.S.* ou uma *I.M.U.*. Além disso, $\frac{m}{T}$ pode ser descoberto sabendo que o torque total quando o *drone* está parado, ou *hovering*, é igual à massa do veículo, assim $\frac{m}{T} \approx \frac{1}{g}$, na qual *g* é a gravidade.

Agora é necessário achar ${}^{\boldsymbol{R}}v_x^*$ para completar o controle da lei de equação (51). Para tal, considera-se algo similar ao que foi feito para o momento linear. Considera-se o ponto $\mathbf{a} \in \Re^2$. O controle da velocidade é **P** e dado pela lei

$${}^{\boldsymbol{W}}v_x^* = K_{p_{v_x}}({}^{\boldsymbol{W}}a^* - {}^{\boldsymbol{W}}a) \tag{60}$$

na qual $K_{p_{v_x}}$ é a constante proporcional, a^* e a são os pontos desejado e atual do *drone*. Observe que a velocidade é a variação dos pontos em um determinado período.

Finalmente, resta apenas converter o resultado da velocidade para o *frame R*. Para isso deve-se usar a matriz de rotação ${}^{\boldsymbol{W}}R_{\boldsymbol{R}}(\theta_p)$. Essa matriz é definida como

$${}^{\boldsymbol{R}}R_{\boldsymbol{W}}(\theta_p) = {}^{\boldsymbol{W}}R_{\boldsymbol{R}}^T(\theta_p) \tag{61}$$

e dessa forma é obtido a velocidade desejada no *frame* ***R*** fazendo a transformação

$${}^{\boldsymbol{R}}\mathbf{v}^* = {}^{\boldsymbol{R}}R_{\boldsymbol{W}}(\theta_p){}^{\boldsymbol{W}}\mathbf{v}^* = {}^{\boldsymbol{W}}R_{\boldsymbol{R}}^T(\theta_p){}^{\boldsymbol{W}}\mathbf{v}^* \tag{62}$$

Matricialmente

$$\begin{bmatrix} {}^{\boldsymbol{R}}v_x \\ {}^{\boldsymbol{R}}v_y \end{bmatrix} = {}^{\boldsymbol{W}}R_{\boldsymbol{R}}^{T}(\theta_p) \begin{bmatrix} {}^{\boldsymbol{W}}v_x \\ {}^{\boldsymbol{W}}v_y \end{bmatrix} \tag{63}$$

Agora tem-se um controlador para movimentos no eixo x. De maneira resumida, primeiramente deve-se calcular a equação (60), em seguida transforma-se esse resultado com a equação (63) e com a estimativa da velocidade calcula-se a equação (59), finalmente pode-se obter o *pitching torque* com as equações (52) e (51).

Para um movimento no eixo y o cálculo é similar apenas trocando o *pitching torque* pelo *rolling torque* e o ângulo θ por ϕ

Observe novamente que os ângulos de *pitch* e *roll* são responsáveis por controlar indiretamente a posição e a velocidade do multirotor. Isso ocorre por que o sistema é sub-atuado, ou seja, existem mais saídas do que entradas para os controladores. Em um *drone* multirotor controlam-se quatro velocidades angulares para movimentar o veiculo com seis graus de liberdade. Um ponto muito importante é que por optar-se por um controle dessa forma existem algumas limitações para os ângulos que o *drone* deve se inclinar. No entanto, para esse projeto, essas limitações não influenciarão no resultado final.

2 Controle de *yaw* e de altitude

Propõe-se agora criar o controle para um movimento de *yaw*. Assim como calculado na subseção 1, o controle será feito com um controlador PD apenas, sendo sua lei dada por

$$\tau_z = K_p(\psi^* - \psi) + K_d(\dot{\psi}^* - \dot{\psi}) \tag{64}$$

nas quais K_p e K_d são as constantes proporcionais e derivativa, ψ^* e ψ são os ângulos de *yaw* desejado e atual, $\dot{\psi}^*$ e $\dot{\psi}$ são as velocidades angulares de *yaw* desejada e atual. O termo $\dot{\psi}^*$, geralmente, é considerado nulo, pois não é desejado que o *drone* vire muito rápido.

Para realizar, agora, o controle da altitude usa-se também um controlador PD. Sua lei é dada por

$$T = K_p(z^* - z) + K_d(\dot{z}^* - \dot{z}) + \omega_0 \tag{65}$$

nas quais z^* e z são as alturas desejadas, $\dot{z}^*$ e $\dot{z}$ são as velocidades no eixo z, ω_0 é o termo que considera o empuxo (*thrust*) necessário para manter o multirotor parado, em relação ao eixo z.

$$\omega_0 = \sqrt{\frac{mg}{4b}} \tag{66}$$

na qual *m* é a massa do *drone*, *g* é a gravidade, *b* é a constante de sustentação.

d Controle proposto

Existem diversas placas controladoras de voo para *drones*. Dentre elas estão *NAVIO* e *PixHawk*. Ambas possibilitam o uso de dois *firmwares* consagrados no campo, *PX4* e *Ardupilot*. Esses *softwares* controladores são de código aberto, *open source*, bem documentados e testadas. Ainda não foi decidido o uso de um conjunto plataforma-*firmware*, mas ambas possuem características similares e atraentes para a execução desse trabalho. Dentre essas caraterísticas, estão os modos de voo, já implementados nesses *firmwares*. O modo *follow me*, ou siga-me, está presente tanto no *Ardupilot* quanto no *PX4* e é extremamente atraente, pois já implementa uma forma de converter pontos no espaço em sinais de controle (velocidades angulares dos motores).

A conversão citada anteriormente é algo um pouco mais complexo, pois envolve trabalhar com controle não linear. Assim, foge um pouco do foco desse trabalho. Usando um *software* no qual tem-se implementado esse tipo de conversão é algo interessante, pois evita uma fase de criação e, principalmente, teste desse controle.

Tendo em mente o que foi dito anteriormente, além disso, a ideia para o controle não é usar os pontos cartesianos, mas distâncias para controlar o *drone* pelo percurso. Assim, basta alterar alguns parâmetros dos modos *follow-me* bem como codificar as distâncias de modo que o controlador as entendam como comandos convencionais. Basicamente, será deito um *hacking* da função *follow-me*.Isso deve-se ao fato

da função citada, usar sinais de *GPS* como entrada, dessa forma, se, manipulando o sinas de distância, for possível criar um sinal de *GPS* simulado, o controle será executado da maneira esperada, seguindo a linha através do percurso (presente na figura 14).

3 Módulo de processamento de imagens e reconhecimento de linha

Nessa seção é mostrado o funcionamento do módulo de reconhecimento de imagens, que reconhecerá a linha de referência a ser seguida pelo multirotor (como pode ser visto na figura 14). Essa linha, na competição, será uma corda colocada sobre um fundo de maneira que ela tenha um alto contraste e seja facilmente identificada.

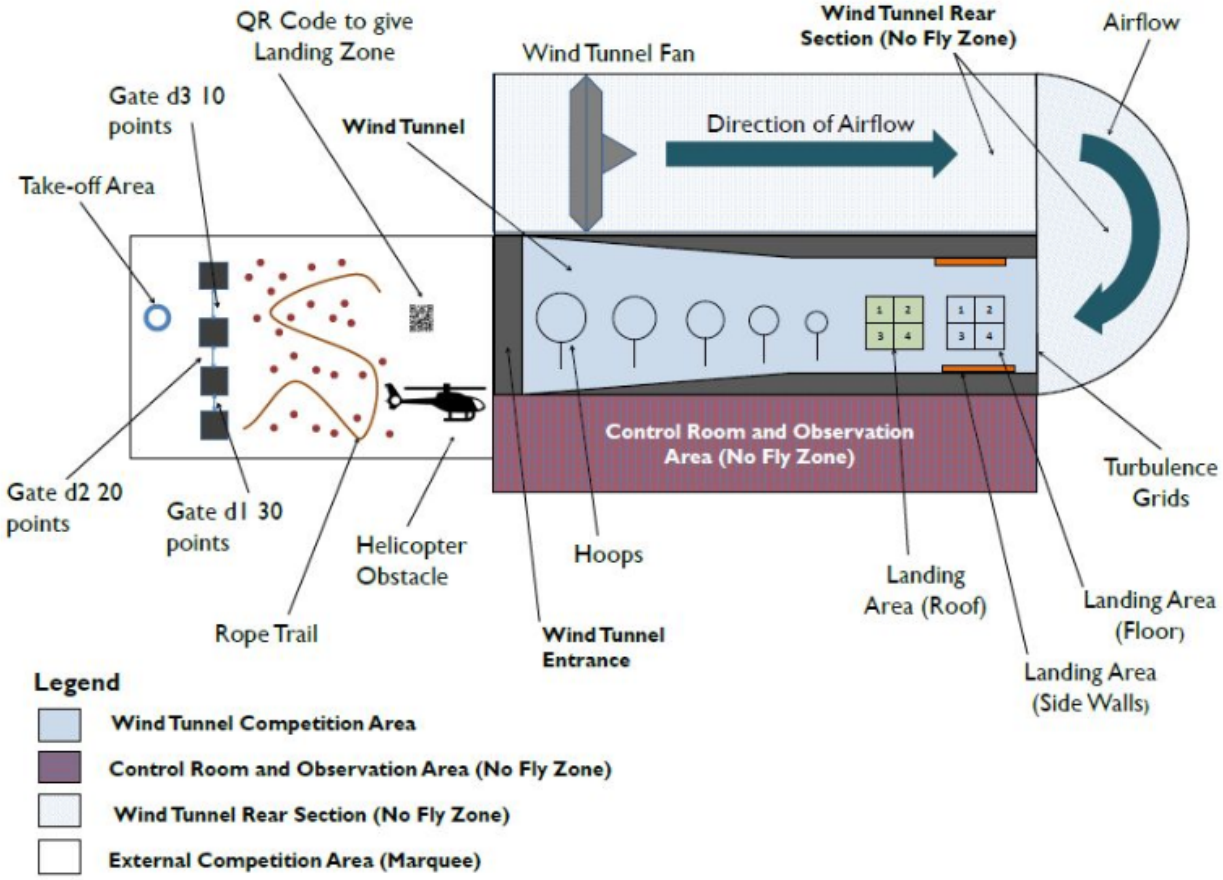


Figura 14: Esquema do percurso proposto pelos organizadores da competição IMAV2018.

Esse módulo será responsável, dentre outras funções, por indicar ao *drone* se ele deve manter o percurso, caso esteja sobre a linha, ou para onde ele dever seguir para retornar à linha, caso esteja fora da linha guia. Nas subseções a e b serão apresentadas a ideia e a implementação desse módulo.

a Metodologia

Existem diversas formas de resolver esse problema, ou seja, de reconhecer uma linha ou corda sobre um fundo de alto contraste. Dentre elas, pode-se usar um algoritmo de detecção de contornos ou bordas em imagens e depois traçar essas bordas.

Para aplicar esses algoritmos de maneira mais eficiente, é preciso um pre-tratamento da imagem, primeiramente. Assim, pode-se dividir esse módulo nas etapas

1. Conversão das cores da imagem;
2. Aplicação de um filtro de desfoque;
3. Aplicação de um algoritmo de detecção de bordas;
4. Aplicação de um algoritmo de construção debordas a partir da etapa 3;

que serão explicadas mais detalhadamente nas seções3.a.1 - 4.

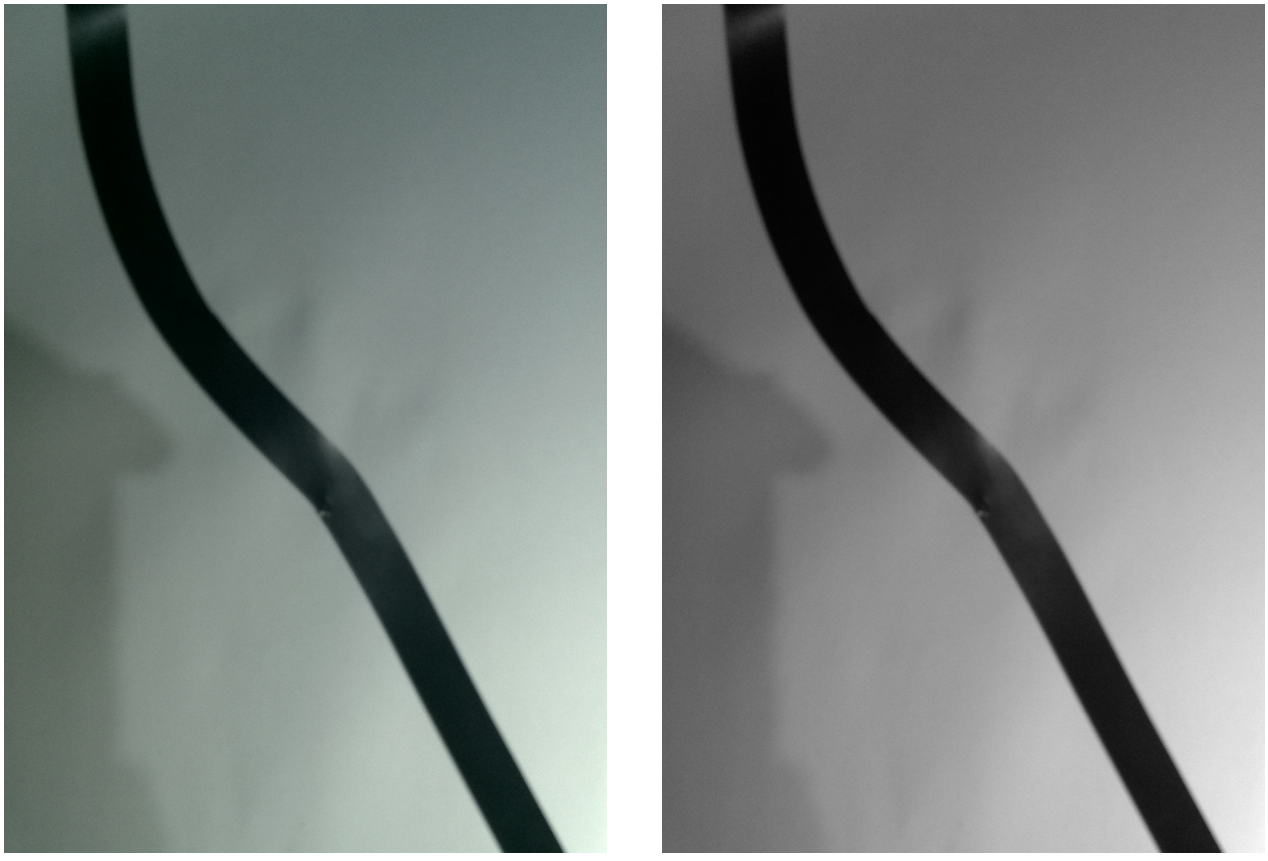
Um ponto importante é que no multirotor será acoplada um câmera policromática que registrará em tempo real os movimentos do *drone*. Essa câmera capta imagens à 30 quadros por segundo, ou seja, 1 quadro a cada 33.33 ms. Assim, para a execução das etapas anteriormente citadas usa-se apenas 1 desses quadros a cada segundo, dando tempo para o processamento de imagem e a resposta do sistema.

Outro ponto importante é saber que esse módulo foi criado e pensado para a utilização na linguagem Python^{Tm} com o auxílio da biblioteca *OpenCV* [7]

1 Conversão das cores da imagem

A primeira etapa a ser cumprida é a conversão dos canais de cores da imagem recebida. Essa imagem chegará com os três canais de cores, *RGB*(*Red, Green and Blue*, ou Vermelho, Verde e Azul) e sairá apenas com tons de cinza.

Isso é realizado, porque as imagens em *RGB* são codificadas como matrizes tridimensionais nas quais duas dimensões representam os pontos cartesianos discretizados da imagem e a terceira dimensão, uma das cores. Ao aplicar os algoritmos nessas matrizes, tem-se que o mesmo demorará duas vezes mais do que o necessário para realizar sua função e assim, consumirá mais recursos do processador e tempo. Além disso, como o algoritmo analisa cada canal independentemente, a detecção pode não ser precisa e acabar por detectar contornos que não estão nos locais certos, bem como contornos inexistentes.



(a) Imagem original; (b) Imagem após a conversão;

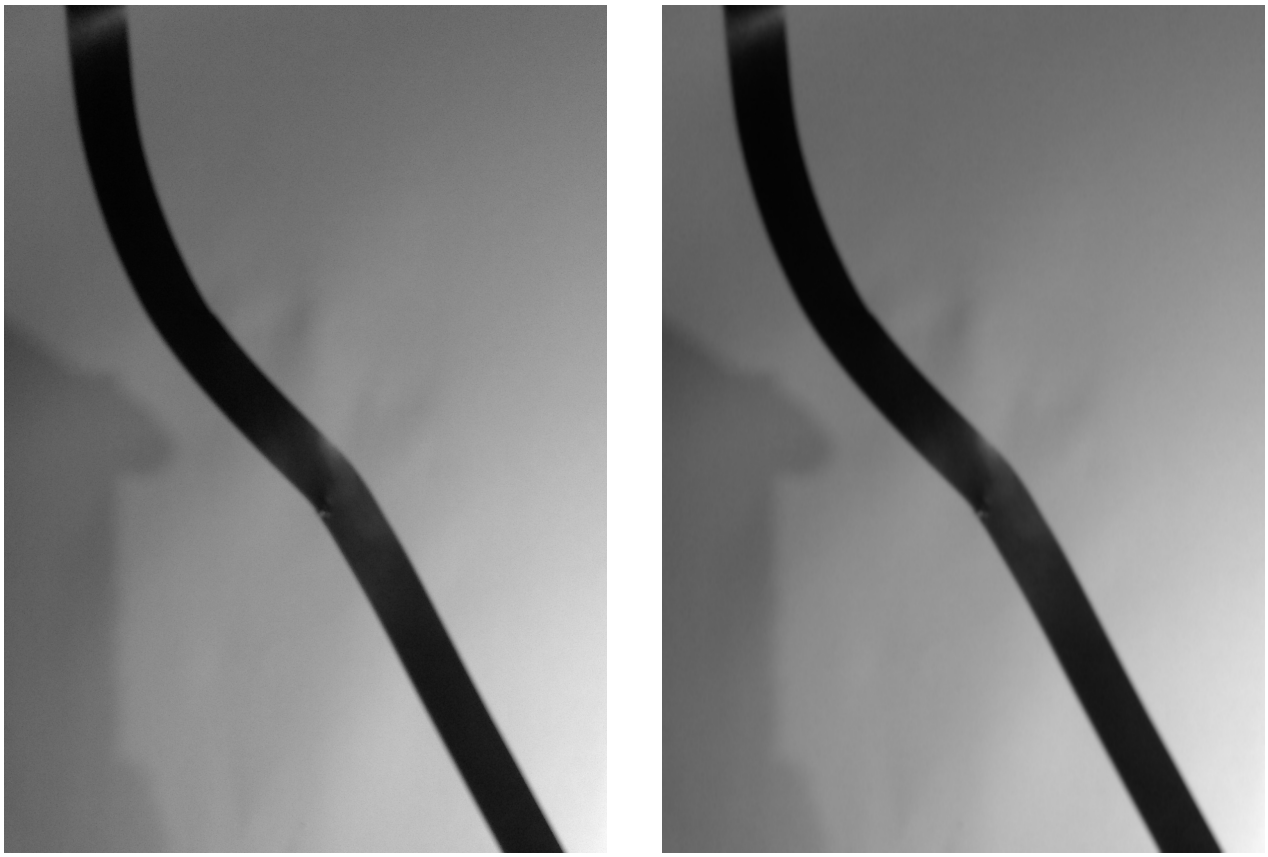
Figura 15: Resultado da etapa de conversão à direita e imagem de entrada da etapa à esquerda.

2 Aplicação de um filtro de desfoque

Após a conversão da imagem para tons de cinza, deve-se, para ajudar a etapa 3, aplicar algum efeito de desfoque. Isso servirá para que as derivadas dos pontos dos contornos sejam acentuadas, facilitando,

assim, a sua detecção

O filtro escolhido para essa etapa foi o *Gaussian blur*. O que esse filtro faz é usar uma função gaussiana em cada pixel, distribuindo, em todas as dimensões, a cor do pixel, com a intensidade dada por uma função gaussiana. Existem diversos filtro de desfoque ou *blur*, mas a grande vantagem do *Gaussian blur* é que os pontos de contorno, nos quais existem basicamente a cor da linha(preta, na maioria das vezes) e o fundo (branco, na maioria das vezes), são acentuados pela distribuição gaussiana. Assim, a derivada nos pontos de contorno sofrem uma variação abrupta (comparando com a derivada dos pontos do fundo) facilitando sua detecção. Além disso, esse filtro ajuda a eliminar ruídos na imagem.



(a) Imagem em tons de cinza; (b) Imagem após a aplicação do desfoque Gaussiano;

Figura 16: Resultado da etapa de desfoque à direita e imagem de entrada da etapa à esquerda.

3 Aplicação de um algoritmo de detecção de bordas

Com a imagem já tratada, pode-se aplicar o algoritmo de detecção de contornos. Existem diversos algoritmos de detecção de bordas como *Canny*, *Sobel*, *Prewitt*, *Deriche*. Nesse trabalho foi escolhido o *Canny*, por uma questão de custo-benefício em relação ao esforço computacional, facilidade de uso e qualidade do resultado.

A imagem resultante do algoritmo *Canny* é apenas uma imagem preta com traços brancos no possível lugar das bordas da imagem. Observando a figura 3(b) é possível ver que o contorno foi obtido com qualidade suficientemente boa. Obviamente existem alguns pontos de *outliers*, mas em sua maioria o contorno foi detectado.

(a) Imagem original; (b) Imagem após execução do algoritmo *Canny*;

Figura 17: Resultado da etapa de desfoque à direita e imagem de entrada da etapa à esquerda.

4 Aplicação de um algoritmo de construção de bordas a partir da etapa anterior

A partir da detecção deseja-se obter os pontos cartesianos que formam esse contorno. Assim, pode-se usar algum algoritmo de criação desses pontos. O algoritmo escolhido foi o *Probabilistic Hough Transform*.

Para entender o funcionamento do *Hough Transform* precisa-se, primeiro, entender que a maiorias das formas geométricas que tenham representação matemática podem ser representadas de forma paramétrica. Nesse trabalho, foca-se na detecção de linhas, então usa-se uma linha como exemplo.

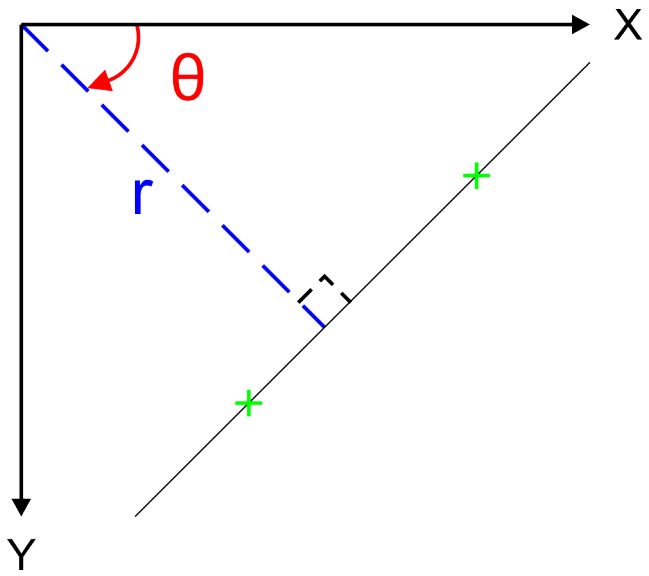


Figura 18: Representação dos eixos cartesianos e o espaço de Hough.

Linhas ou retas podem ser representadas da forma

$$y = ax + b; \tag{67}$$

nos quais x e y são abscissa e coordenada. Podem ser representadas também na forma polar, como visto na figura 18, da seguinte maneira

$$r = x\cos\theta + y\sin\theta; \tag{68}$$

na qual *r* é a distância de uma linha perpendicular a reta de parametrização entre a origem do plano, θ é o ângulo entre linha perpendicular à reta e a origem. No caso desse trabalho, como será utilizado a biblioteca *OpenCV*, o plano cartesiano tem a representação adotada pela figura 18 e, dessa maneira, o ângulo θ é dado pela rotação no sentido horário. Note que todos os pontos que pertencem a mesma reta apresentarão o mesmo ângulo θ e a mesma distância *r*. Na figura 18 as cruzes verdes são dois pontos distintos da reta apresentada e que possuem mesmo *r* e θ. Outro ponto interessante de ressaltar é que o espaço no qual temos os *r*'s e θ's é conhecido como espaço de *Hough*.

Agora foca-se em entender como o algoritmo *Hough Transform* funciona. Primeiramente é criada uma matriz cujas linhas representam um dos valores de *r*, discretizados e distintos, e cada coluna um valor de θ, também discretizado e com a precisão desejada. Essa matriz será a matriz de acumulação de "votos".

Agora, para todos os pontos é feito o seguinte: como tem-se cada valor de x e y é desejado, deve-se substituí-los na equação (68) e verificar para cada θ o valor de *r*. Então, para cada par (r, θ) incrementa-se 1 voto na sua posição correspondente na matriz de acumulação.

Ao final desse processo, observa-se na matriz de acumulação e nota-se qual par possui maior número de votos. Essa será uma das retas detectadas.

Apresentado o funcionamento do *Hough Transform*, compreender como funciona o *Probabilistic Hough Transform* é mais simples. Ao invés de serem utilizados todos o pontos, usa-se apenas uma parte deles para que seja criada a reta. Essa última transformada é mais eficiente, pois utiliza um número menor de pontos para criar a reta, isso gera, também, um consumo menor de processamento.

O resultado dessa etapa são dois vetores de pontos e não uma imagem, diferentemente das outras etapas. Basicamente, tem-se os pontos presentes na imagem resultante da etapa anterior, um vetor para os x's e um para os respectivos y's. Para facilitar a explicação, a partir de agora caso haja uma referência a esses vetores, será considerado apenas o vetor de x's, exceto quando explicitamente citado os dois vetores.

Para ratificar o entendimento, pode-se observar o site [8], cuja explicação está apresentada, também, em forma visual.

b Resultados e implementação

Dada a ideia apresentada na subseção anterior (subseção a), a implementação foi desenvolvida na linguagem Python™, usando, principalmente, a biblioteca *OpenCV* versão 3.4.1. A escolha da linguagem, bem como da biblioteca foi feita baseada na facilidade de uso, disponibilidade de material e, especialmente, pelo fato de existir a intenção de utilizar os resultados desse trabalho em um *drone* real usando um *Raspberry Pi* como parte de um sistema de controle.

Um dos problemas pré-concebidos é a limitação de processamento do *Raspberry Pi*. Assim, a otimização de recursos é algo essencial para esse trabalho. Com isso em mente, para obter uma maior eficiência e velocidade de execução dos algoritmos, pode-se limitar o processamento de imagem em uma região menor e mais importante da imagem completa. Então, antes de seguir para a próxima etapa, todo o processamento de imagem é feito apenas em pedaço da imagem, uma região de interesse. No caso desse trabalho, como ainda não há o uso real em um *drone*, essa região foi escolhida sem muito critério. No caso de uma aplicação real, pode-se dimensionar essa região para que seja uma área onde o *drone* está contido e há poucos ou (preferencialmente) nenhum obstáculo.

Além disso, para esse trabalho foram considerados que as linhas visualizadas pela câmera do *drone* podem ser aproximadas por linhas retas. Apesar de não ser uma aproximação perfeita, é boa o suficiente para a execução das tarefas. Assumir isso é plausível porque dado que no labirinto, parte onde o *drone* deve seguir uma linha guia, as ordens de grandeza das distâncias entre obstáculos, do percurso e da altura de voo é grande o suficiente para um *drone* de 1 m de largura passar (esse valor foi obtido considerando que a aeronave passará por todas as etapas, por pelo menos um dos obstáculos propostos pelos organizadores da competição citada na seção 1 e visto no manual de regras [9]) então é normal assumir que dado uma altura razoável, entre 1 m e 1.5 m, a câmera enxergará apenas partes suficientemente lineares.

Com o intuito de dar uma noção espacial do que está ocorrendo nos resultados, considere que a linha tem 1.8 cm de largura e que o comprimento é de uma folha de papel A4.

Pode-se destacar um resultado extremamente importante desse módulo que é o vetor de pontos de onde foram encontradas as bordas da linha. Com esse resultado, pode-se obter pontos de referência para o *drone* e assim controlar sua posição.

1 Cálculo da distância do *drone* multirotor em relação à linha

Essa etapa é feita usando o vetor previamente citado, nela é feito o cálculo da distância do *drone* em relação a linha guia, recém descoberta pelas etapas anteriores. Apesar de estar no módulo de reconhecimento de imagens, essa etapa não é parte de nenhum tipo de reconhecimento de imagem, apenas usa o resultado do reconhecimento feito anteriormente.

Para que possa ser calculada a distância desejada, é preciso primeiramente, de alguma maneira, criar uma equação que modele a linha. Assim, o que pretende-se é, a partir dos dados obtidos na etapa 4 criar uma função que melhor se ajuste aos dados (também conhecido como *fitting* de dados). Note, no entanto, que os dados estão bem separados visualmente, como pode ser visto na figura 19

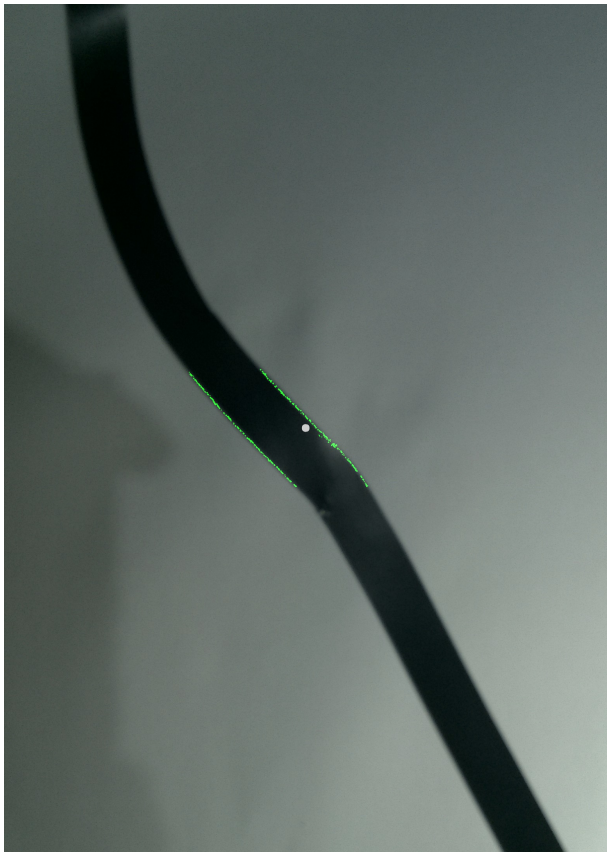
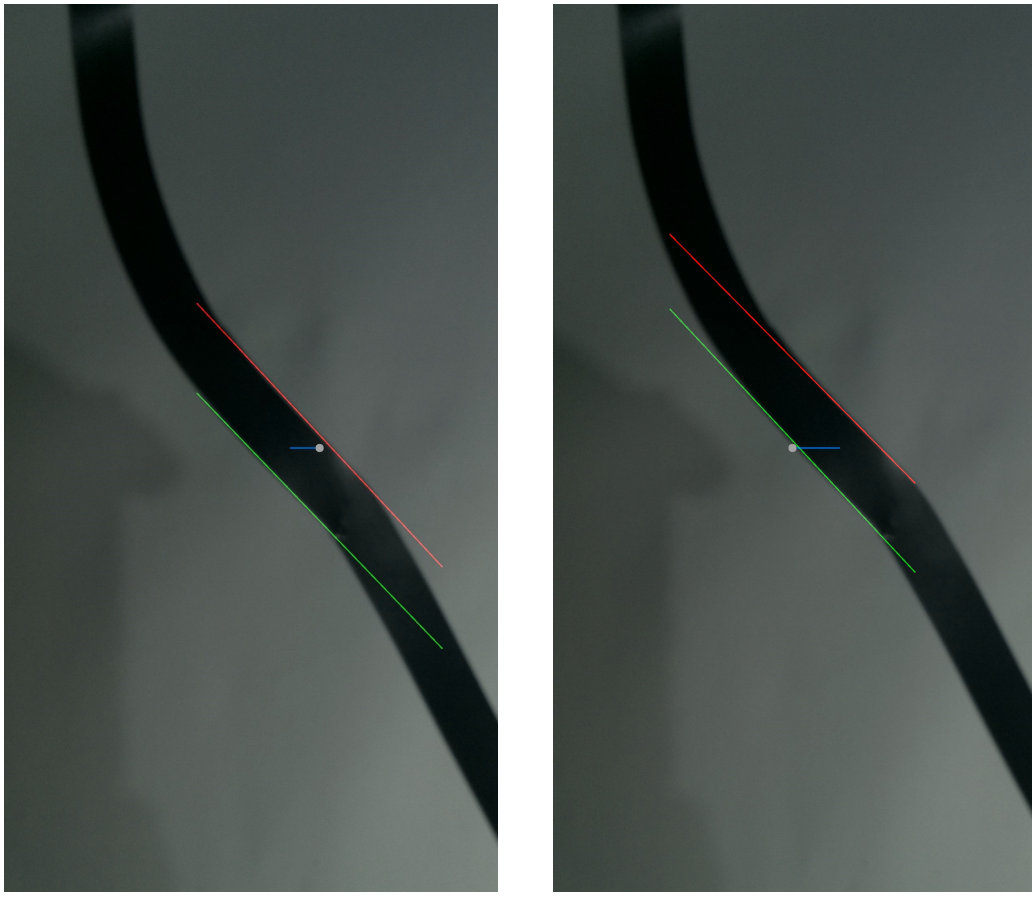


Figura 19: Resultado da etapa 4 mesclado com a imagem original. Em verde estão os pontos das bordas e em branco, o centro da imagem.

O vetor de pontos não é organizado por nenhuma lógica, ou seja, não há como distinguir a qual reta um determinado ponto pertence só olhando ou escolhendo, de maneira lógica ou automática, algum índice. A partir desse instante, foi feita a separação dos pontos levando em consideração a distância cartesiana entre eles. Basicamente, foi comparado se a distância entre dois pontos consecutivos era menor que um limiar. Caso fosse, esses pontos pertenciam a mesma reta, caso contrário, não pertenciam. O limiar de seleção citado é baseado na largura da linha e a distancia, observada visualmente, entre os pontos que estão do mesmo lado e são consecutivos. Com os dados já separados, foi utilizada uma função que consegue gerar um polinômio que melhor represente os dados. Para o caso desse trabalho, para facilitar e simplificar o programa, foi assumido que dentro da região delimitada anteriormente a linha pode ser aproximada por um polinômio de primeiro grau.

A partir desse ponto, com o polinômio já criado, bastou fazer a média das funções na altura do ponto central e depois, a partir desse ponto, fazer a distância até o ponto central da imagem. Com isso obtém-se o quão distante o *drone* está da linha central e, consequentemente, o quanto ele deve corrigir em sua rota para voltar a seguir a linha.

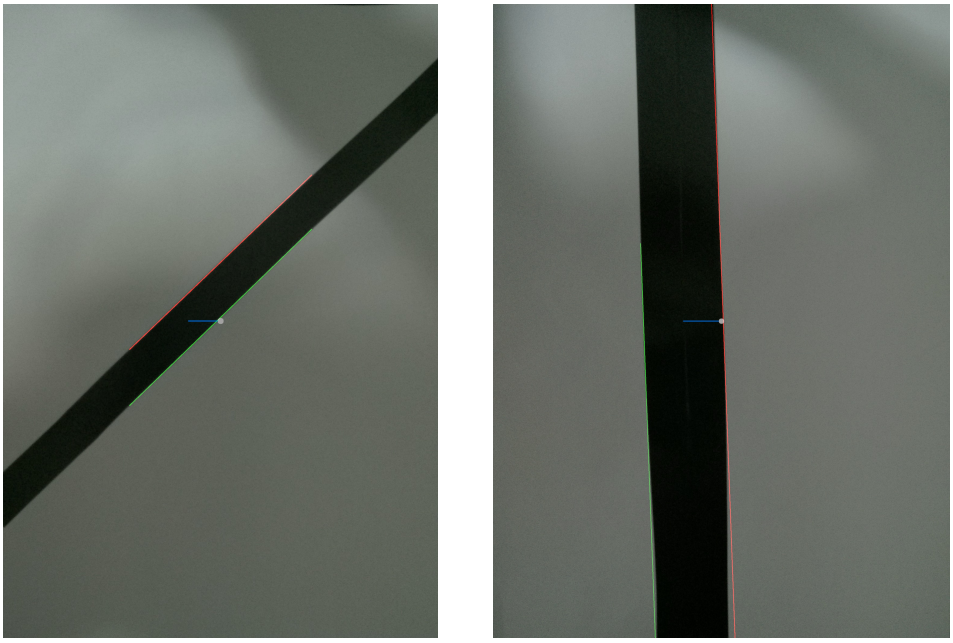


(a) Linha à esquerda do centro (b) Linha à direita do centro

Figura 20: Resultado da etapa do cálculo da distância. Em verde e vermelho estão as funções encontradas pelo *fitting*, em branco está o centro da imagem e em azul, a distância do centro da linha ao da imagem.

2 Resultado para várias formas diferentes

Apesar de ter sido mostrado anteriormente resultados com apenas uma forma geométrica para a linha, o módulo consegue lidar com outras formas, distintas da apresentada. A seguir estão alguns exemplos



(a) Linha diagonal com inclinação diferente da apresentada;

(b) Linha paroximadamente vertical c;

Figura 21: Resultado para formas diferentes.

3 Possíveis limitações

Pela forma como os pontos para o *fitting* são adquiridos e preprocessados, existem algumas limitações e resultados inesperados como o mostrado na figura 22. Observa-se que em alguns casos, ainda não especificados, é possível que as linhas de contorno, principalmente a linha mais afastada do centro da imagem, não sofra um *fitting* da maneira correta. Notando a figura 22 é possível ver que apesar do erro, a direção de onde se encontra a linha está correta fazendo com que o *drone*, mesmo que não esteja indo para o centro a linha, se aproxime o suficiente da linha e o módulo volta a funcionar de maneira esperada.

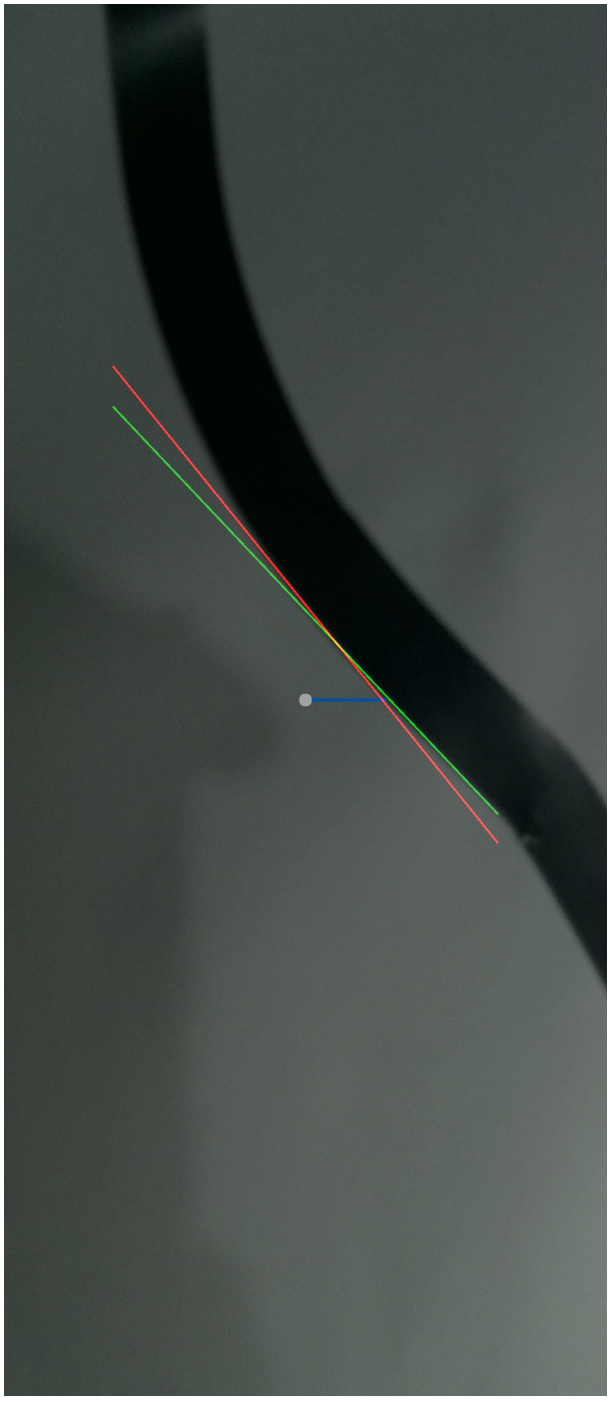


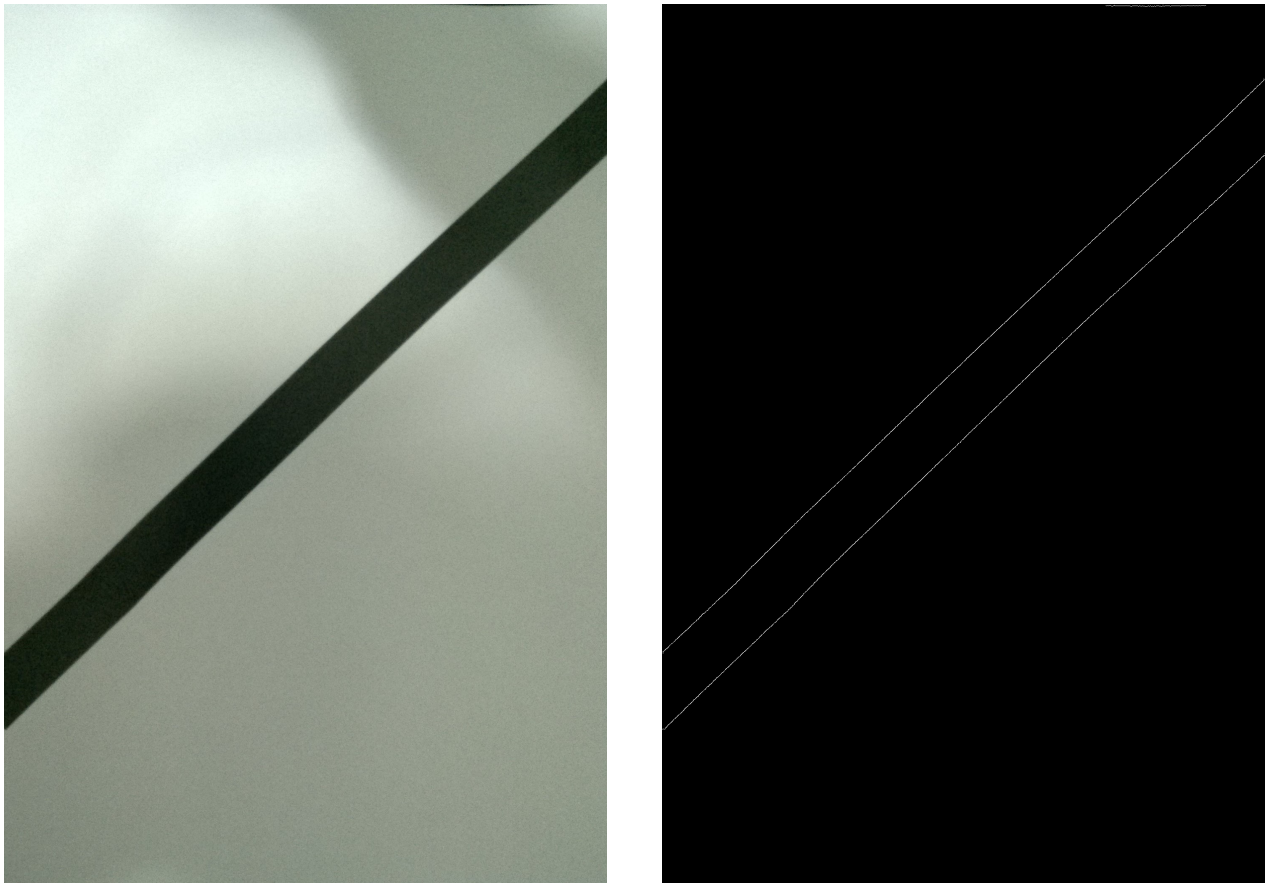
Figura 22: Resultado inesperado. A direção de onde se encontra a linha ainda continua correta.

Outra limitação encontrada nesse modulo é que a linha não pode ser reflexiva. Isso porque quando o algoritmo de *Canny* for aplicado os pontos que refletirem luzes claras, branca principalmente, serão considerados como bordas. Essa limitação pode ser vista na figura 23

(a) Linha reta com reflexo acentuado; (b) Resultado do algoritmo *Canny*;

Figura 23: Resultado com erro proveniente de reflexos.

Como pode ser visto na figura 23(b) o reflexo é considerado como borda e isso gera um erro no *fitting* da função, como apresentado na figura 23(a). Para comprovar que o reflexo influencia, a seguir é apresentado resultado do *Canny* para a figura 21 que é a mesma forma geométrica, mas sem o reflexo.

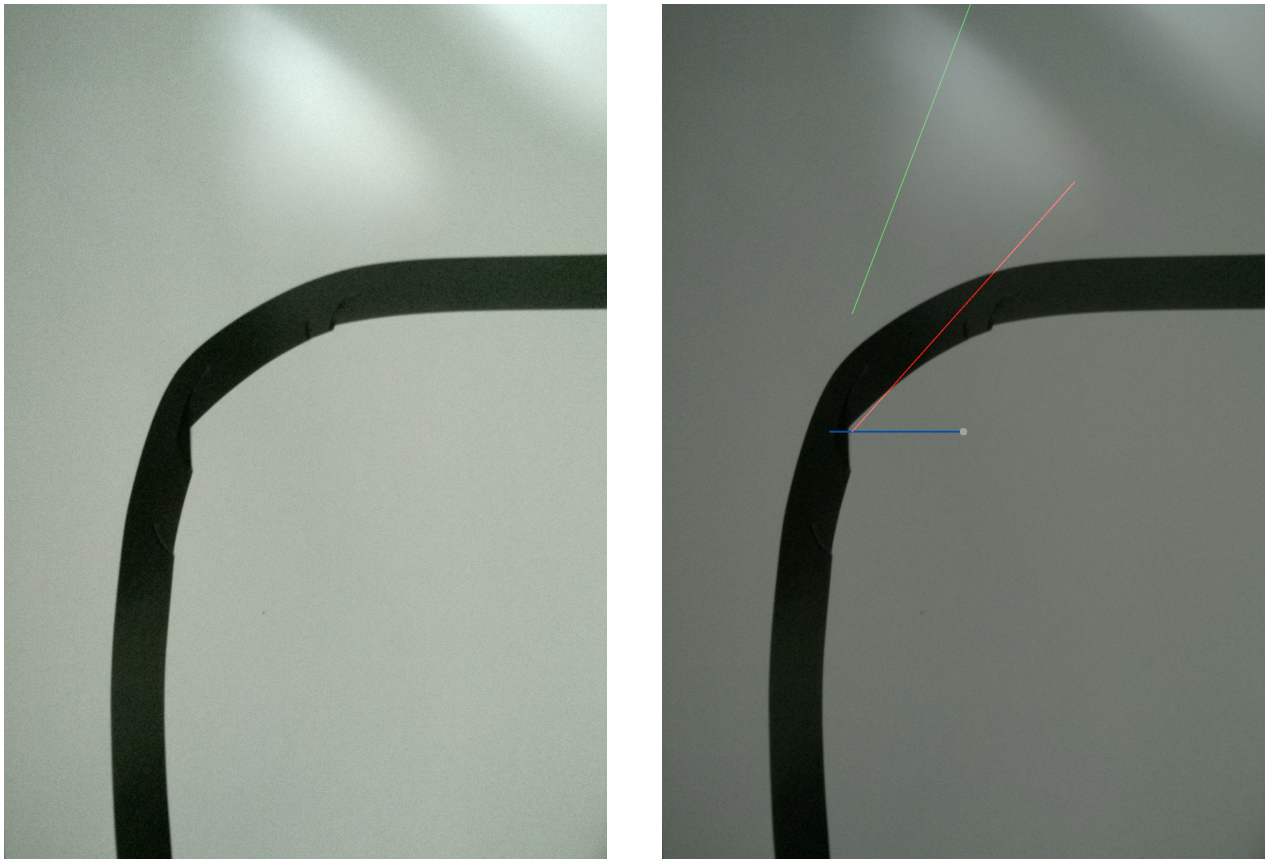


(a) Imagem original; (b) Resultado do algoritmo *Canny*;

Figura 24: Resultados sem erros de detecção.

Na figura 24 é possível ver que o reflexo não está presente e consequentemente não é detectado.

Por fim, o módulo não foi feito para em lidar com curvas (pelos motivos mostrados no início dessa subseção) e, dessa maneira, não há garantias de que o reconhecimento de linha funcione bem com essa forma. Observando a figura 25 pode-se observar uma das respostas do módulo para uma curva. É importante salientar, mais uma vez, que esse tipo de geometria não é esperado ser encontrado na competição cujo *drone* foi projetado e foco desse trabalho.



(a) Imagem original; (b) Resultado do módulo;

Figura 25: Resultado para uma curva acentuada.

4 Módulo de reconhecimento de obstáculos

Seguindo as etapas do processo apresentado na seção visão geral do projeto, na seção 1, tem-se, após o reconhecimento da linha, o reconhecimento de objetos ou obstáculos na mesma imagem obtida e já processada no módulo anterior. Esse módulo é responsável por encontrar obstáculos no caminho do *drone*. Nas subseções a e b serão mostradas as técnicas usadas para o reconhecimento, bem como resultados interessantes. É importante lembrar que o processamento feito nessa seção leva em consideração o resultado da etapa anterior, ou seja, é conhecido o lugar da linha, assim, pode-se facilitar e melhorar o desempenho do módulo. Isso será, também, visto a seguir.

a Metodologia

Para realizar o reconhecimento citado existem, assim como para o módulo apresentado na seção 3, diversas formas. A escolhida para a resolução desse problema nesse trabalho é utilizar um pré-tratamento de imagem, seguindo parte da a metodologia implementada na seção 3, a binarização da imagem e a partir da análise de áreas brancas, detectar um objeto. Nesse módulo, diferentemente do módulo de processamento de imagem e reconhecimento de linha, dentro do pré-processamento da imagem será eliminado da análise parte da imagem que contem a linha, pois não é possível que haja um objeto em cima da linha. Essa decisão foi feita baseada na economia de recursos do processador.

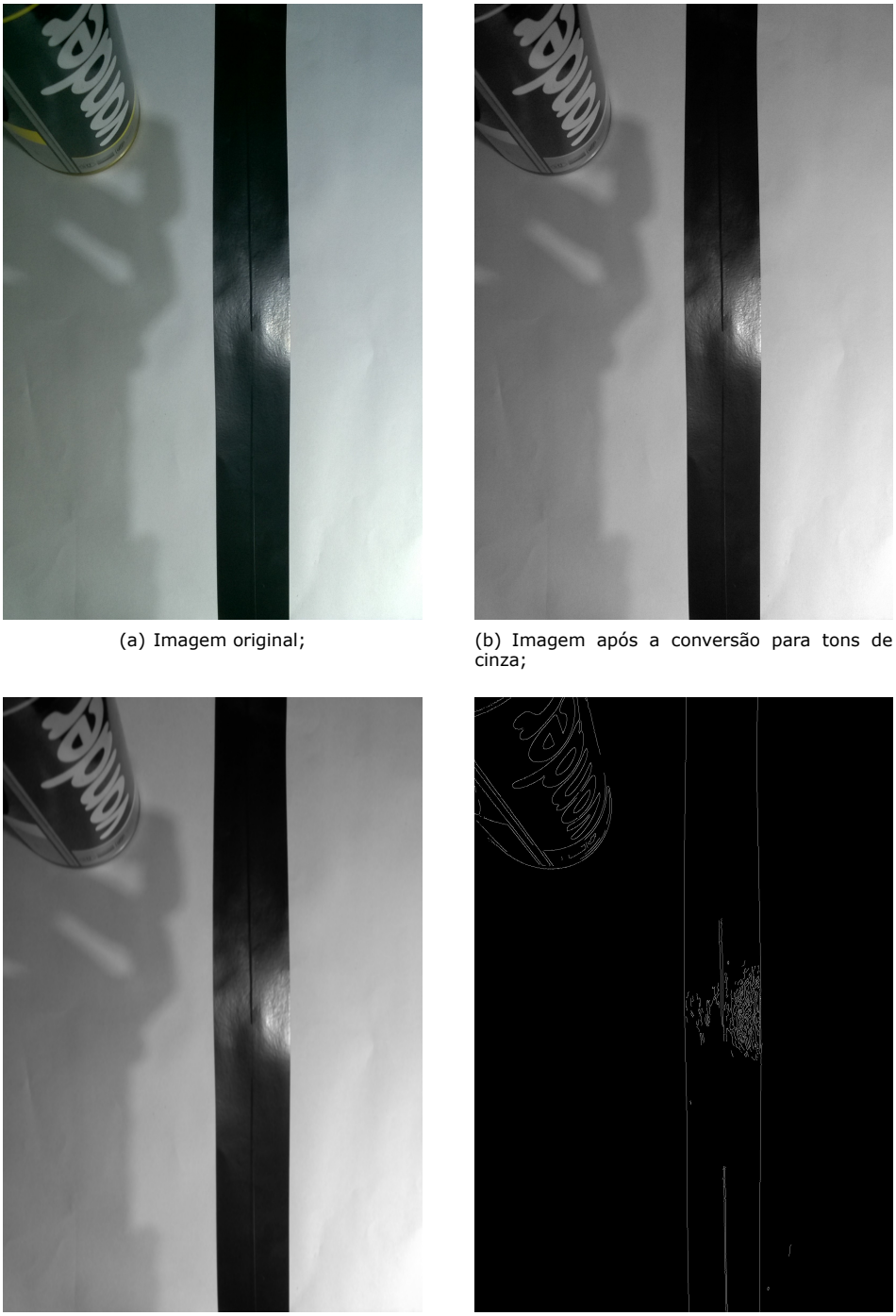
Pode-se, finalmente, enumerar as etapas presentes nesse módulo como

1. Conversão das cores da imagem;
2. Aplicação de um filtro de desfoque;
3. Aplicação de um algoritmo de detecção de bordas;
4. Aplicação de um algoritmo de dilatação da imagem;
5. Aplicação de um algoritmo de contração da imagem;
6. Aplicação de uma máscara,eliminando a área que contem a linha mais uma margem de segurança;
7. Detecção e destaque de objeto;

A seguir cada item listado será explicado mais detalhadamente.

1 Conversão das cores da imagem, aplicação de um filtro de desfoque e aplicação de um algoritmo de detecção de bordas

Essas 3 etapas são exatamente iguais e executados pelos mesmos motivos apresentados na seção 3, subseções 1, 2 e 3. Nesse caso, o resultado obtido é apresentado na figura 26.



(a) Imagem original;

(b) Imagem após a conversão para tons de cinza;

(c) Imagem após a aplicação de um filtro gaussiano de desfoque;

(d) Imagem após execução do algoritmo *Canny*;

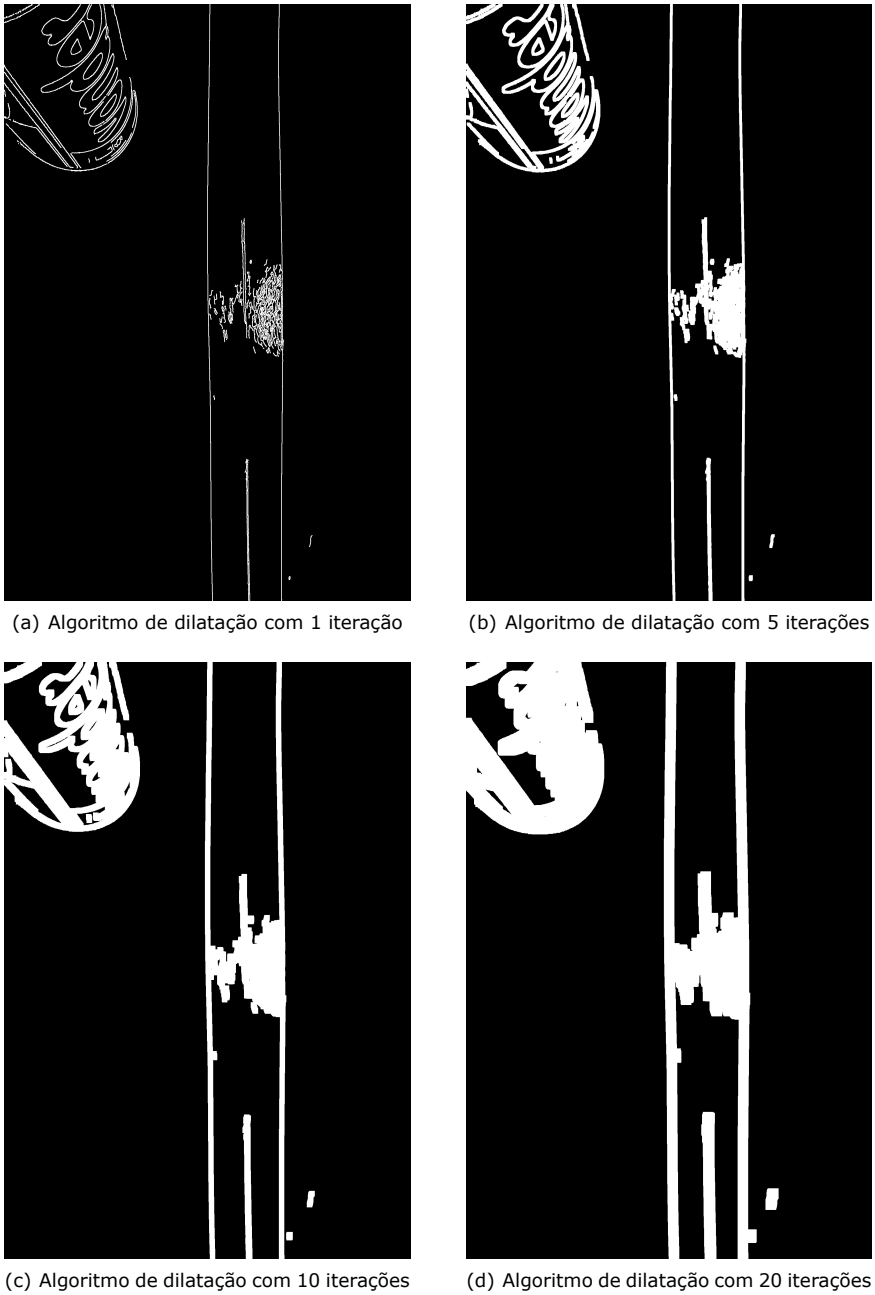
Figura 26: Resultado da etapa de desfoque à direita e imagem de entrada da etapa à esquerda.

Nessa etapa foi introduzida, como obstáculo, uma lata cilíndrica de aproximadamente 20 cm de altura e 5.5 cm de diâmetro. Além disso, a fim de dar noção espacial, a linha apresentada na figura 26 tem aproximadamente 3.5 cm de largura e o comprimento de uma folha A4. Nota-se que ao final dessa etapa, observando a figura 26(d), foram detectados, tanto a linha quanto os contornos da lata, incluindo desenho e letras presentes na embalagem.

A próxima etapa, pretende tratar a figura 26(d) visando obter melhores resultados na detecção de objetos.

2 Aplicação de um algoritmo de dilatação da imagem

Como dito anteriormente, propõe-se nessa etapa aplicar mais um tratamento na imagem resultando das etapas anteriores. Para tal foi selecionado um algoritmo que expande a área de todos os *pixels* brancos. O intuito de fazer isso é que pequenos furos ou áreas pretas dentro de áreas brancas sumam, limpando, assim, a imagem. No final quer-se retirar ruído da imagem e facilitar as etapas posteriores. Considera-se ruído, pequenas descontinuidades na detecção de bordas do objeto e contornos internos de desenhos, palavras, reflexos.



(a) Algoritmo de dilatação com 1 iteração
(b) Algoritmo de dilatação com 5 iterações
(c) Algoritmo de dilatação com 10 iterações
(d) Algoritmo de dilatação com 20 iterações

Figura 27: Resultados do algoritmo de dilatação para diferentes números de iterações.

Nota-se que quanto maior o número de iterações, mais os ruídos somem. Observando a lata na figura 27, a palavra *vonder* vai sumindo conforme o número de iterações aumenta, chegando ao ponto de não conseguir mais ser distinguida. No caso desse trabalho, isso é um ponto positivo, pois a palavra é um

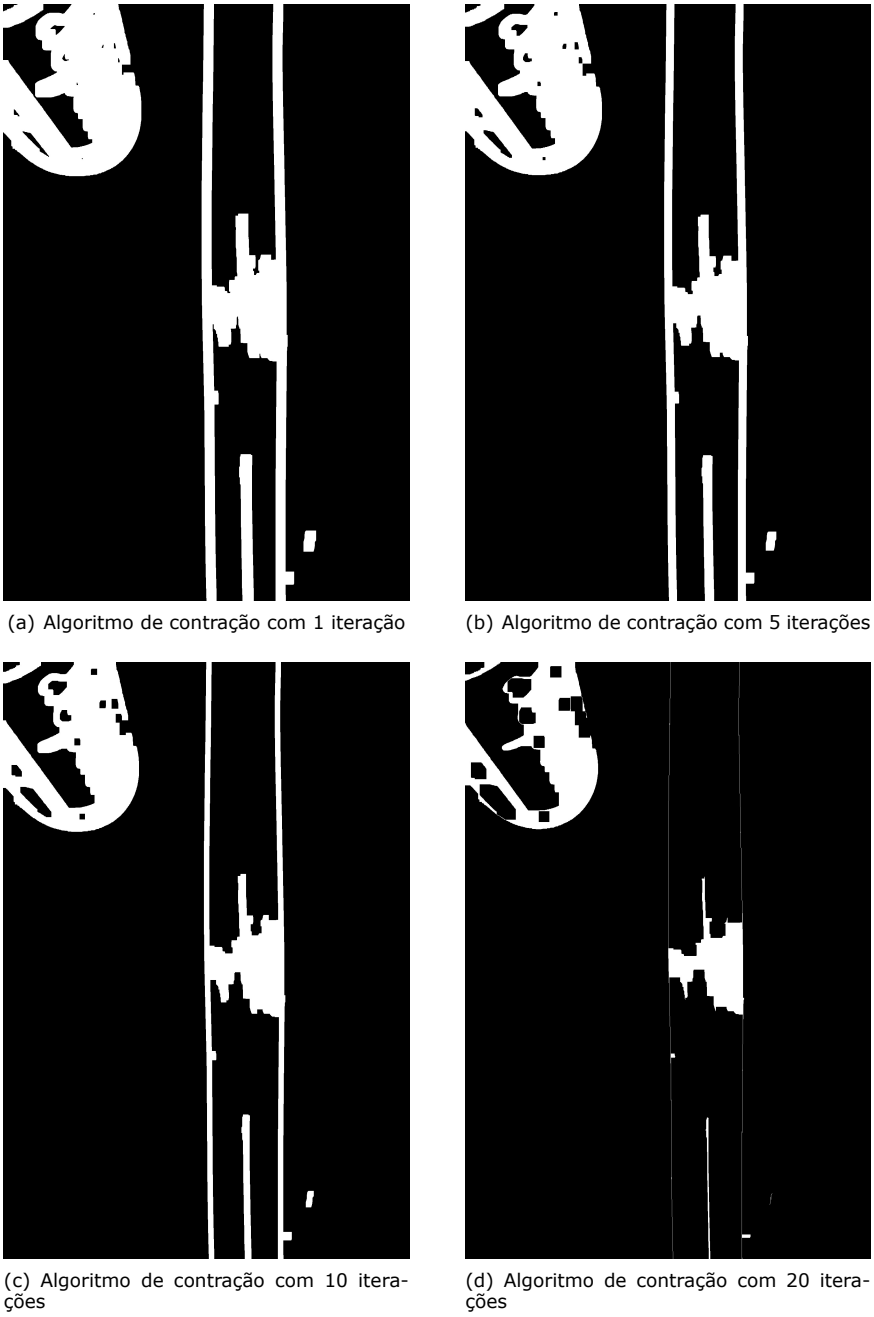
ruído, tendo em vista que deseja-se obter apenas o contorno da lata. Para esse trabalho foram utilizadas 20 iterações desse algoritmo.

Observa-se também que o contorno do objeto desejado perde parte de sua definição, sofrendo uma distorção. Essa última, pode ser um problema tendo em vista que a perda de definição da forma pode gerar uma detecção de um objeto que não é o desejado e, assim, pode-se gerar um comando de desvio de maneira errônea culminando numa colisão do *drone*, por exemplo.

Essa etapa por si só não é capaz, de forma eficiente, de reduzir os ruídos, ela apenas prepara a imagem para a etapa que elimina de maneira efetiva os ruídos, evitando os possíveis problemas citados anteriormente.

3 Aplicação de um algoritmo de contração da imagem

Essa etapa é complementar à etapa anterior, contraindo e tentando encontra uma forma original de contorno. Teoricamente, os contornos internos encontrados na etapa anterior nos objetos, quando a imagem for contraída, devem ser eliminados.



(a) Algoritmo de contração com 1 iteração

(b) Algoritmo de contração com 5 iterações

(c) Algoritmo de contração com 10 iterações

(d) Algoritmo de contração com 20 iterações

Figura 28: Resultados do algoritmo de erode para diferentes números de iterações; A entrada para essa etapa é a figura 27(d).

Observando o resultado final, figura 28(d), nota-se que o contorno da lata está mais delineado e com uma menor distorção. Nota-se que, apesar de haver furos (áreas brancas dentro de pretas) na lata, grande parte do objeto é branco, facilitando a detecção nas próximas etapas.

4 Aplicação de uma máscara, eliminando a área que contem a linha mais uma margem de segurança

Essa é a etapa responsável apenas por reduzir parte do custo computacional do projeto. Nela é criada uma máscara que a região que contem a linha, deixando menos *pixels* para serem tratados. O resultado dessa etapa pode ser visto a seguir na figura

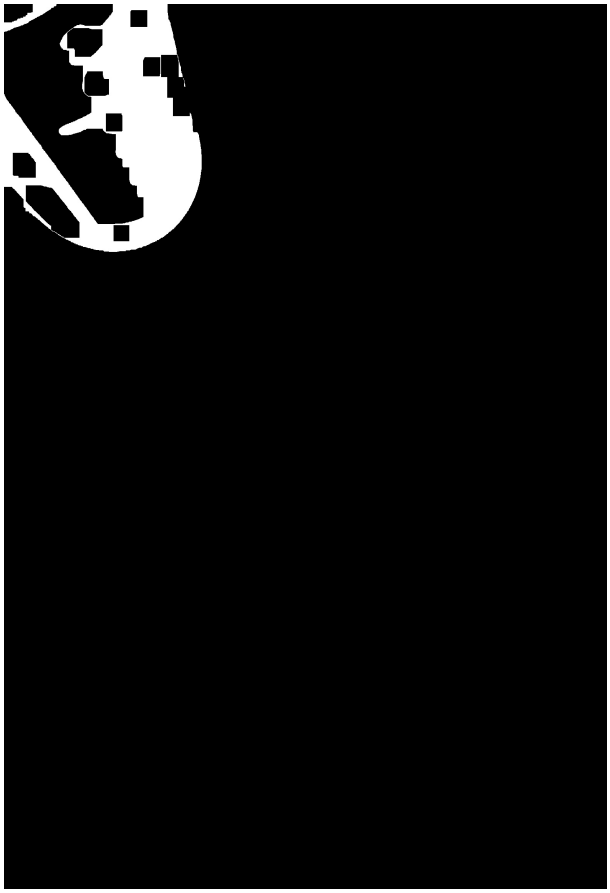


Figura 29: Resultado da aplicação da máscara na imagem resultante da etapa anterior.

5 Detecção e destaque de objeto

Com o contorno do objeto pré-destacado, pode-se partir para essa etapa. Para detectar o que considera-se um objeto, será utilizado a seguinte técnica: mede-se as áreas brancas no objeto e considera-se somente a maior delas, em seguida cria-se um quadrilátero com a menor área que contem a área branca considerada e esse quadrilátero será a área efetiva do objeto. Assume-se que as áreas brancas desconsideradas vão estar dentro desse novo quadrilátero, isso porque a área maior é vista como o contorno do objeto e as descontinuidades (áreas brancas) dentro do contorno são pequenos ruídos que não conseguiram ser filtrados. Assumir isso é plausível.

Aplicando a ideia exposta e usando a biblioteca *OpenCV* [7], pode-se obter o seguinte resultado

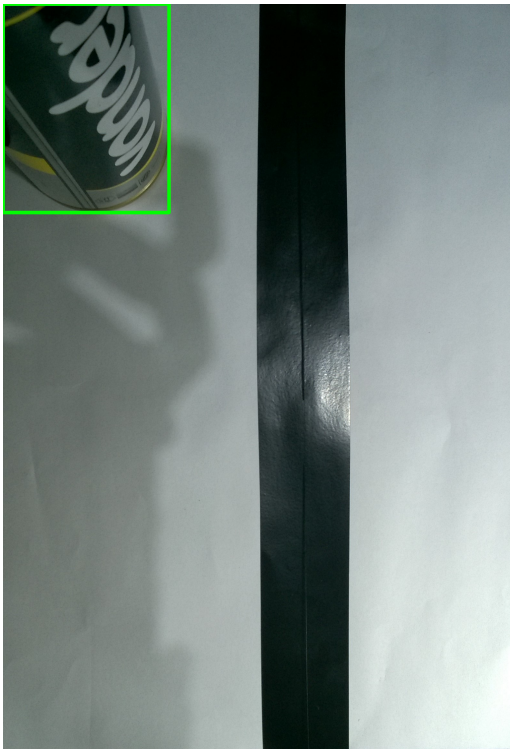


Figura 30: Resultado da detecção do objeto visto na imagem original.

nota-se que a área detectada é maior que o objeto em si, no pior caso é igual. Isso, no caso desse trabalho, é bom pois a detecção já inclui uma pequena área de segurança. Como o objetivo é identificar objetos e desviar deles, ter uma área de segurança é fundamental para que o multirotor tenha tempo o suficiente para desviar de possíveis obstáculos.

b Resultados e implementação

Esse módulo é de extrema importância para o módulo de desvio de objetos, isso porque as localizações dos objetos detectados serão usadas para determinar o deslocamento necessário para evitar uma colisão entre o *drone* e um obstáculo. Dentre os resultados obtidos tem-se

1 Localização de obstáculos

Como apresentando anteriormente, a última etapa desse módulo detecta e destaca os objetos. Para isso, foram utilizadas algumas funções da *OpenCV* [7] que, dentre seus resultados, apresenta as coordenadas cartesianas dos vértices do retângulo que contém o objeto.

2 Múltiplos objetos

Apesar desse trabalho ter mostrado na subseção a apenas um objeto, o código funciona com mais objetos presentes na imagem. Para isso, bastou reaplicar a etapa de detecção somada a uma etapa de eliminação de objetos, já detectados, na máscara até não haver mais objetos a serem detectados. A etapa de eliminação consiste em usar o quadrilátero de detecção para criar uma máscara desse objeto e apagar na máscara principal (que contém todos os objetos) o obstáculo. Pela forma como foi implementado, há a garantia de que todos os objetos serão detectados, mas não há garantia de nenhuma ordem lógica para as detecções.

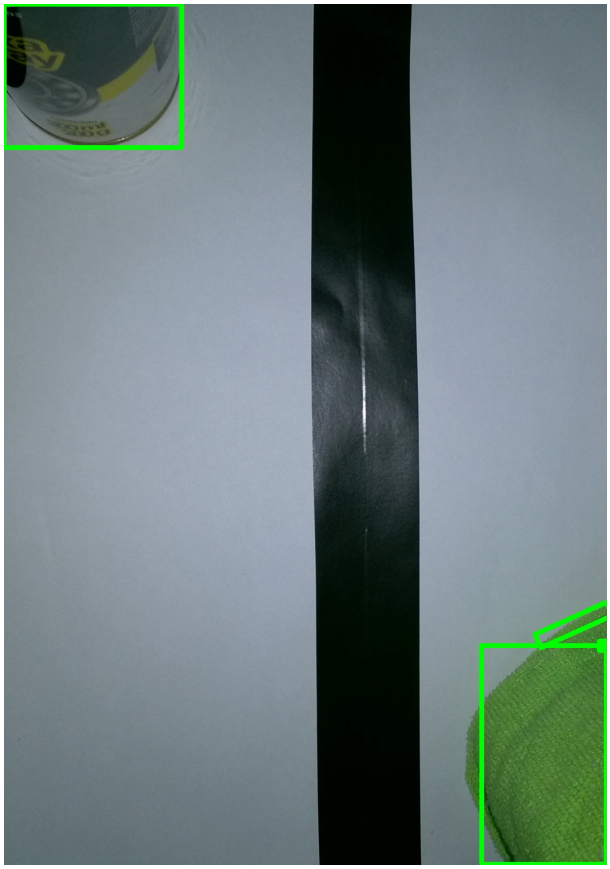


Figura 31: Resultado para a detecção de múltiplos objetos.

Nesse caso vemos a detecção da lata apresentada nos exemplos iniciais dessa seção e a detecção de um pano verde. Um ponto a ser levado em consideração é que apesar de o pano não ter sido considerado com um único objeto, a soma dos objetos detectados faz o contorno do pano e não impedem a execução da próxima etapa. É esperado que haja pequenos erros desse tipo, mas esses não atrapalharão o módulo de desvio, como será visto na seção 5.

3 Possíveis limitações

Como citado em seções anteriores, o reconhecimento de linhas pode ser feito de maneiras diferentes, dentre elas (além da apresentada na seção 3), usando o método de dilatação e contração, mas não pode ser feita como apresentado nessa seção, subseção a. É necessário um tipo de tratamento diferente do mostrado. Essa limitação pode ser vista no contorno da linha encontrada na figura 28(d). Observa-se, nesse caso, um grande nível de ruído que dificultará o reconhecimento da linha.

Uma outra limitação desse módulo é que em raras ocasiões, as sombras dos objetos, como visto na figura 32, pode ser detectada como um objeto. Isso é decorrente da forma como o algoritmo de dilatação e contração funcionam. Em lugares que a luz é baixa o suficiente para que a sombra e o objeto tenham quase os mesmos tons esse tipo de erro acontece. Mas isso pode ser corrigido com o auxílio de uma fonte de luz presente na hora de captura de imagem. Além disso a cor do objeto pode influenciar no algoritmo. Cores mais chamativas são mais simples de detectar.

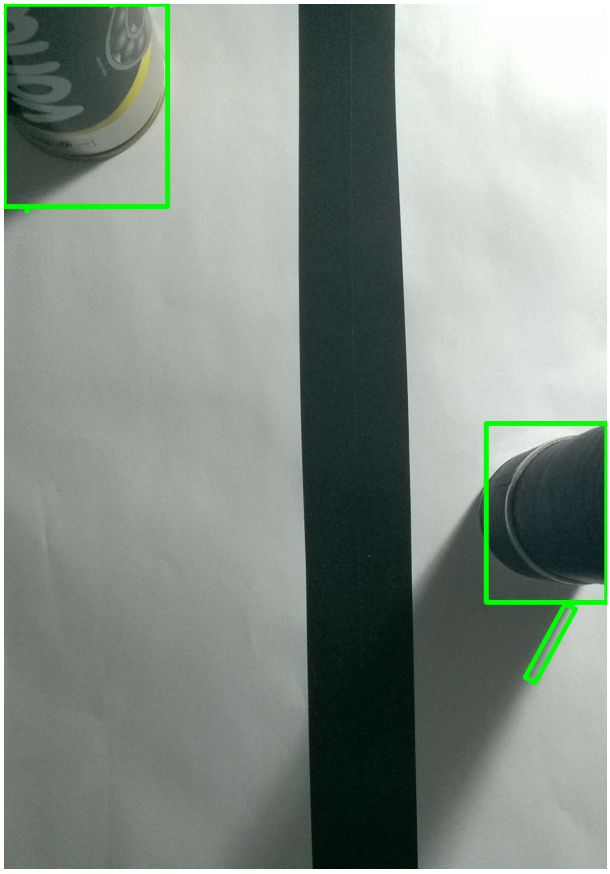


Figura 32: Resultado para a detecção de múltiplos objetos em condições não ideais.

5 Módulo de desvio de obstáculos

Fazendo uma breve revisão do que foi visto até essa seção tem-se o reconhecimento de um linha, o reconhecimento de objetos e agora, como sucessão lógica do processo, tem-se o desvio de obstáculos. Esse módulo é complementar ao apresentado na seção 4 e como dito anteriormente é fundamental para o projeto, uma vez que ao reconhecer um objeto, pode haver a necessidade do multirotor fazer um desvio. Esse módulo será incumbido da função de decisão de quando executar uma manobra de desvio. Nas subseções a e b será destacado o funcionamento do módulo e os resultados encontrados

a Metodologia

A ideia geral desse módulo é bem simples, levando em consideração uma zona proibida (que contém a área do *drone* acrescida de uma área de segurança), os locais dos obstáculos e a distância desses para o centro da imagem (onde, como descrito na seção 3, é o ponto que considera-se o centro do *VANT*).

De forma concisa, tendo a localização dos objetos pode-se encontrar as distâncias desses para o centro e verificar se essas distâncias estão dentro do raio de segurança. Se estiverem, um desvio (baseado na diferença do raio de segurança e da distância do objeto) é executado. Caso contrário, nenhum desvio é aplicado. Listando as etapas tem-se

1. Obtenção da menor distância do objeto até o centro da imagem;
2. Verificação de possível colisão;

A seguir tem-se explicações mais detalhadas sobre cada uma das etapas destacadas

1 Obtenção da menor distância do objeto até o centro da imagem

A ideia para essa etapa é bem simples, considera-se que o objeto foi detectado no módulo anterior (seção 4) e que a posição dos vértices é conhecida. Dessa maneira, é possível encontrar a menor distância entre as bordas e o centro da imagem fazendo uma triangulação de todos os pares de vértices e o centro do *drone*. Basicamente o que tem-se é, para cada par de vértices consecutivos, um triângulo cuja menor distância está contida no espaço entre as arestas. Visualmente, pode-se observar a figura 33 para alguma clarificação.

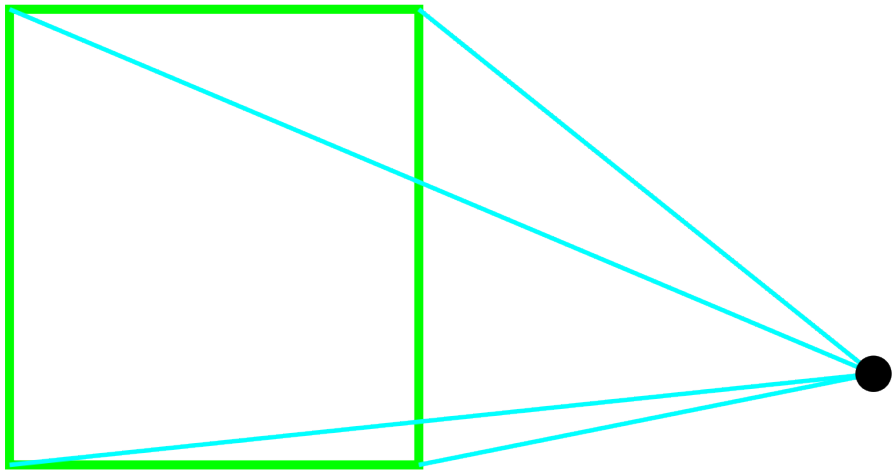


Figura 33: Distâncias das arestas de um quadrilátero um ponto; As distâncias estão representadas por linhas azuis.

Nota-se que a distância pode ser calculada facilmente usando apenas geometria. Contudo, essa seria uma abordagem mais perfeccionista e precisa, que no caso desse trabalho não se faz necessária. Pode-se, então, facilitar e simplificar bastante se for considerada apenas a menor distâncias entre os vértices. Esse ideia se faz viável e plausível porque considerando o pior caso, onde a distância mínima está dentro do triângulo, como visto na figura 34, considerar a distância do vértice mais próximo do centro do multirotor apenas adiciona uma margem de segurança. Além disso, é importante salientar que, como os objetos

que serão detectados pelo *drone* na competição são cilindros e a área de detecção gera um quadrilátero, o projeto desse e do módulo anterior adiciona margens de segurança que, além de facilitar e reduzir a complexidade o código, servem para compensar imperfeições nas detecções.

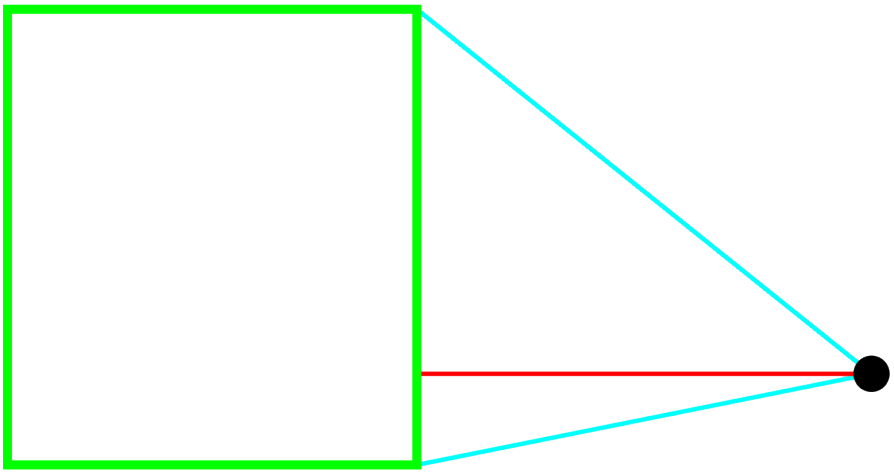


Figura 34: Distâncias das duas arestas de um quadrilátero mais próximas de um ponto.

Executando essa ideia tem-se o seguinte resultado

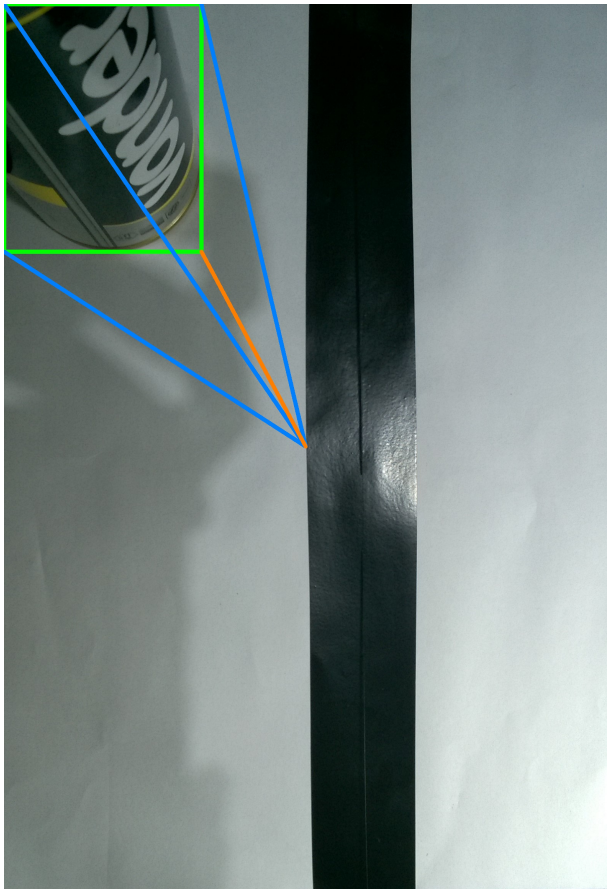


Figura 35: Resultado, gráfico, da verificação da menor distância; Em azul estão todas as distâncias e em laranja a menor distância.

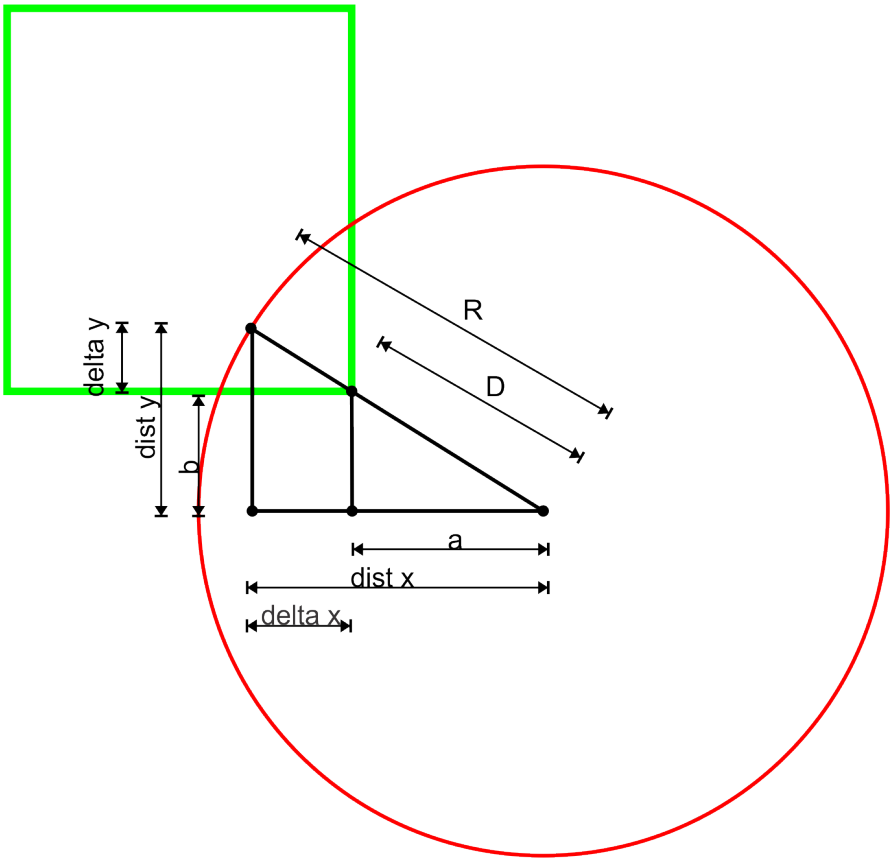


Figura 36: Problema de semelhança de triângulos. Em vermelho está a zona proibida, em verde o quadrilátero de detecção.

2 Verificação de possível colisão

Para essa etapa é necessário fazer algumas considerações, com o intuito de explicar algumas decisões de projeto tomadas. A primeira consideração é que a zona proibida será uma área circular cujo raio é a maior distância do centro geométrico do *drone* visto de cima até o ponto mais distante ainda pertencente à aeronave acrescida de uma margem de segurança de 5 cm (essa margem foi escolhida sem nenhum motivo específico), mas como ainda não foi decidido ao tamanho do *drone*, um raio arbitrário foi escolhido para representar o modelo real. A segunda consideração é que o desvio ocorrerá considerando apenas o objeto mais próximo detectado, isso é plausível porque o perigo de colisão eminente provem do objeto mais próximo. A terceira consideração é que o desvio, quando necessário será multiplicado por um fator de 1.2 visando dar mais segurança para a navegação autônoma do *drone*

Tendo as considerações citadas em mente, propõe-se para essa etapa, com o vértice mais próximo do centro já detectado, resolver o problema descrito geometricamente na figura 36, que descreve um problema de semelhança de triângulos

$$dist_x = x_{centro} - x_{mais\ perto} \tag{69}$$

$$dist_y = y_{centro} - y_{mais\ perto} \tag{70}$$

$$a = \frac{D \times dist_x}{R} \tag{71}$$

$$b = \frac{D \times dist_y}{R} \tag{72}$$

$$\Delta_x = (dist_x - a) \times fs \tag{73}$$

$$\Delta_y = (dist_y - b) \times fs \tag{74}$$

na qual D é a distância do centro ao vértice mais próximo, R é o raio da zona proibida e fs é o fator de segurança (nesse caso, 1.2).

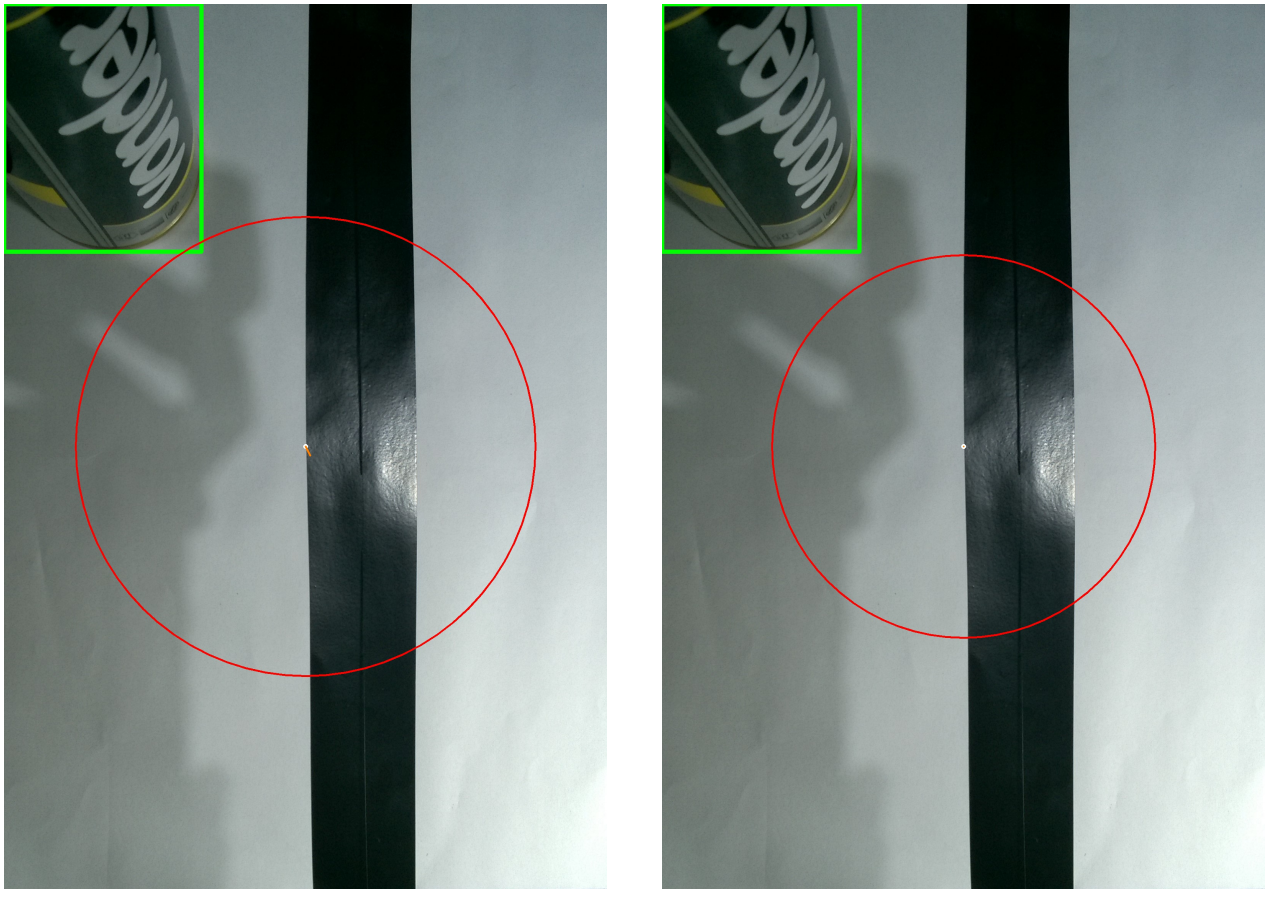
Usando geometria básica ve-se que a solução para esse problema é trivial. Para encontrar os deslocamentos nos eixos $x(\Delta_x)$ e $y(\Delta_y)$ basta seguir as equações de (69) à (74). Um ponto importante que deve ser observado é que pela forma como foram determinadas as equações (69) e (70), o sinal do deslocamento em x (Δ_x) e em y (Δ_y) já informa para onde o multirotor deve desviar. Isso, pois foi considerado que pontos à esquerda do centro possuem abscissas com módulo menor que a dos centro e pontos acima do centro apresenta coordenadas com módulo menor que a do centro.

b Resultados e implementação

Um ponto importante a ser levado em conta é que os resultados aqui apresentados não levam em consideração a trajetória encontrada no módulo da seção 3, mas considera a localização da linha. A intenção é apresentar resultados que mostram que o módulo funciona independente do resultado da trajetória e que esse resultado, caso seja necessário desviar, será acrescido na trajetória. Levando isso em consideração, pode-se destacar alguns resultados importantes.

1 Desvio

O módulo de desvio é acionado apenas quando um objeto entra na zona proibida. Se isso não ocorrer, não será necessário desviar a trajetória. A figura 37 mostra os possíveis resultados.



(a) Resultado com desvio;

(b) Resultado sem desvio;

Figura 37: Resultado para o desvio do *drone*; Em vermelho está a zona proibida, em banco, o centro da imagem e em laranja o desvio necessário.

Veja que para fins de ilustração e para mostrar que o módulo funciona, o raio de segurança foi alterado de maneira que o objeto não entrasse na zona proibida e assim não seria necessário um desvio.

2 Desvios com múltiplos objetos

Como exposto na subseção anterior, o módulo leva em consideração apenas o desvio de um objeto, aquele que estiver mais próximo. Isso é plausível porque é esperado que na competição não exista a possibilidade de existirem dois objetos de lados diferentes da linha e com distância pequena o suficiente para que os dois entrem na zona proibida. Contudo o módulo é capaz de lidar com isso. A seguir tem o resultado usando a detecção de múltiplos objetos apresentadas na seção 4.b.3

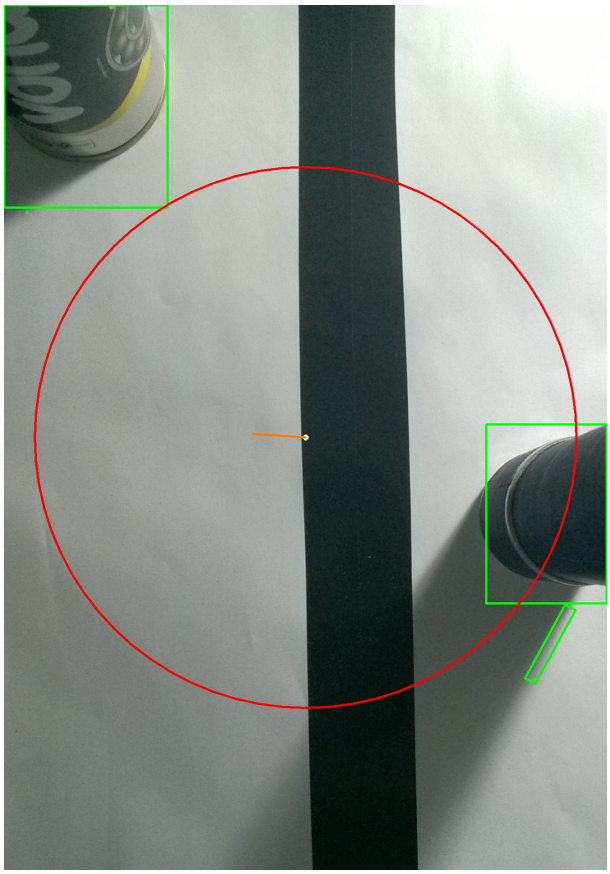


Figura 38: Resultado para detecção de múltiplos objetos.

3 Possíveis limitações

Assim como exposto anteriormente, o módulo do jeito que está não funciona levando em consideração todas as distâncias dos objetos, isso para uma navegação mais precisa não é bom. Mas para o intuito que foi desenvolvido esse trabalho, não tem problema.

Além disso, o módulo não possui nenhum tipo de inteligência computacional em termos de ajuste de parâmetros, algo que pode ser interessante em implementações futuras.

6 Conclusão e Trabalhos Futuros

Os resultados obtidos ao longo desse trabalho demonstram a viabilidade de implementação de um módulo de controle autônomo, baseado apenas em visão computacional, capaz de permitir que um *drone* realize de forma autônoma a tarefa de seguir uma linha e desviar de obstáculos inseridos ao longo de sua trajetória. O módulo desenvolvido utiliza-se apenas de uma câmera (Pi Cam V2) de baixo custo, leve (3 g) e com baixo consumo de energia, que captura imagens no espectro visível com 8 megapixels de resolução a taxas de até 60 *fps*. Dessa forma vai-se de encontro aos objetivos inicialmente traçados, que buscavam a implementação de um sistema leve e energeticamente eficiente, capaz de realizar as tarefas propostas. Na área de *VANT*'s esse tipo de trabalho mostra um avanço, visto que foca na utilização do menor número possível de recursos de hardware (como câmeras estéreo, sensores ultrassônicos, infravermelhos, *LIDAR*'s, etc), a fim de reduzir o peso da aeronave e, consequentemente, diminuir a carga que os motores devem sustentar.

Futuramente, como o trabalho aqui apresentado não chegou a testar os algoritmos desenvolvidos em testes com a câmera embarcada em um *drone*, pretende-se usar uma aeronave para testar o desempenho dos módulos em um ambiente real. Além disso, pode-se pensar em possíveis melhorias para o sistema. Dentre elas, pode-se destacar uma possível mudança no algoritmo de *fitting* de pontos presente no módulo de processamento de imagens e reconhecimento de linha (seção 3). Foi utilizada uma regressão linear, o que limita as formas geométricas delineadas. Uma sugestão é o uso de um *fitting* usando polinômios com graus maiores que 1 ou, até mesmo, um algoritmo adaptativo que tenta identificar o grau do polinômio que reduz o erro entre os pontos provenientes dos dados medidos e os pontos provenientes da equação. Entretanto, deve-se avaliar a capacidade de execução de tal algoritmo em tempo real, tendo em vista a capacidade computacional do sistema embarcado no *drone*. Sugere-se, também, o desenvolvimento de outra forma de reconhecimento de linha, com o intuito de comparar desempenhos e verificar os pontos fortes e fracos de cada uma. Além disso, ainda sobre esse módulo, serão realizados estudos mais abrangentes sobre o raro problema explicitado na seção 3.b.3, no primeiro parágrafo, de modo a identificar sua causa e solucioná-lo.

Adicionalmente, também se sugere o uso de uma fonte luminosa, um LED por exemplo, acoplado à câmera com o intuito de evitar que sombras de objetos sejam confundidas com possíveis obstáculos. O LED não terá um impacto significante no consumo da bateria, pois além de necessitar de pouca corrente, pode ser acionado apenas quando necessário. Além disso, seu peso é insignificante em relação ao peso do *drone* e de seus componentes, impactando muito pouco no consumo dos motores – lembrando que os motores devem gerar empuxo suficiente para levantar o multirotor e seus componentes.

Referências

[1] A. Rango, A. Laliberte, J. E. Herrick, C. Winters, K. Havstad, C. Steele, and D. Browning, "Unmanned aerial vehicle-based remote sensing for rangeland assessment, monitoring, and management," *Journal of Applied Remote Sensing*, vol. 3, 2009.

[2] S. Lakhmani, J. W. Langelaan, and A. Wagner, "Human-intuitable collision avoidance for autonomous and semi-autonomous aerial vehicles."

[3] B. R. Geiger, J. F. Horn, A. m DeLullo, and L. N. Long, "Optimal path planning of uavs using direct collocation with nonlinear programming."

[4] K. Peng, G. Cai, B. M. Chen, M. Dong, K. Y. Lum, and T. H. Leeb, "Design and implementation of an autonomous flight control law for a uav helicopter," *Automatica*, vol. 5, no. 10, 2009.

[5] B. Siciliano and O. Khatib, Eds., *Springer Handbook of Robotics*. Springer-Verlag Berlin Heidelberg, 2008.

[6] P. Corke, *Robotics, Vision and Control*, ser. Springer Tracts in Advanced Robotics. Springer International Publishing, 2017.

[7] Open source computer vision library website. [Online]. Available: https://opencv.org/

[8] Hough transform. [Online]. Available: http://homepages.inf.ed.ac.uk/amos/hough.html

[9] Imav 2018 competition rules. [Online]. Available: https://imav2018.org/wp-content/uploads/2018/04/IMAV2018-Rules-1.pdf

[10] H. Anton and C. Rorres, *Álgebra Linear com Aplicações*. bookman, 2012.

[11] D. Halliday, R. Resnick, and J. Walker, *Fundamentos de Física: Mecânica*, ser. Fundamentos de Física. LTC, 20132.

[12] Open source computer vision library documentation website. [Online]. Available: https://docs.opencv.org/3.4.1/index.html

[13] Ardupilot documentation website. [Online]. Available: http://ardupilot.org/copter/index.html

[14] Aerial torpedoes, buzz bombs, and predators: The long cultural history of drones. [Online]. Available: http://origins.osu.edu/article/aerial-torpedoes-buzz-bombs-and-predators-long-cultural-history-drones