UNIVERSIDADE FEDERAL DO PARANÁ

ADRIANO PENICHE DOS SANTOS

AUXÍLIO DE NAVEGAÇÃO DE UM DRONE DE COMPETIÇÃO POR PROCESSAMENTO DE IMAGEM

CURITIBA

2019

ADRIANO PENICHE DOS SANTOS

AUXÍLIO DE NAVEGAÇÃO DE UM DRONE DE COMPETIÇÃO POR PROCESSAMENTO DE IMAGEM

Projeto apresentado como requisito à obtenção de nota na disciplina de Trabalho de Conclusão de Curso, do Curso de Engenharia Elétrica com ênfase em Eletrônica e Telecomunicações.
Orientador: Prof. Dr. Marlio Bonfim

CURITIBA
2019

TERMO DE APROVAÇÃO

ADRIANO PENICHE DOS SANTOS

AUXÍLIO DE NAVEGAÇÃO DE UM DRONE DE COMPETIÇÃO POR PROCESSAMENTO DE IMAGEM

Trabalho de Conclusão de Curso aprovado como requisito parcial à obtenção do título de Bacharel, Curso de Bacharelado em Engenharia Elétrica, Setor de Tecnologia, Universidade Federal do Paraná, pela seguinte banca examinadora:

_____

Prof. Dr. Marlio J. do C. Bonfim (Orientador)
Departamento de Engenharia Elétrica
UNIVERSIDADE FEDERAL DO PARANÁ

_____

Prof. Dr. Eduardo Parente Ribeiro
Departamento de Engenharia Elétrica
UNIVERSIDADE FEDERAL DO PARANÁ

_____

MSc. Luiza Beana Chipanski Freire
Departamento de Engenharia Elétrica
UNIVERSIDADE FEDERAL DO PARANÁ

Curitiba, 25 de Junho de 2019

AGRADECIMENTOS

Agradeço a minha família, minha mãe Denise Peniche dos Santos, meu pai Anibal dos Santos, minha irmã Amanda Peniche dos Santos e a minha companheira Alessandra Harumi Anami por toda dedicação, paciência e carinho durante essa jornada de graduação, cujo fim se materializa por este trabalho.

Agradeço ao meu orientador Prof. Dr. Marlio J. do C. Bonfim pelo auxílio durante a elaboração deste e de outros trabalhos nos últimos anos.

Agradeço também aos meus amigos que encontrei durante o curso e em outros continentes nos intercâmbios, em especial para Guilherme Cordeiro Vogt e Luís Guilherme Dias de Souza pelos bons momentos e pelo apoio durante os desafios.

“Quem não pensa sobre o futuro,
resolve o presente com ferramentas do passado.
- Tiago Mattos

RESUMO

A estimação da posição de objetos através de imagens obtidas por câmeras tem diversas aplicações no mundo atual, desde indústrias à navegação de veículos autônomos. A partir disso, este trabalho propõe um algoritmo capaz de calcular parâmetros de distância e ângulos que informem a posição de objetos referente a pontos conhecidos no espaço. A principal aplicação deste algoritmo é a competição de drones que acontece anualmente na UFPR, onde pilotos devem capturar e carregar no menor tempo possível um cubo de dimensões e características conhecidas. A partir dessas características, o sistema de visão calcula a posição relativa do drone ao cubo e informa o piloto sobre o posicionamento do equipamento para que o processo de pega seja o mais rápido possível. O projeto foi desenvolvido em python e embarcado em uma plataforma Raspberry Pi 3. Devido a instabilidade na detecção da cor, causada por variações de iluminação e erros aplicados por ajustes morfológicos e ambiguidade no cálculo dos parâmetros, sujere-se o auxílio de outros sensores externos para melhorar o cálculo dos parâmetros.

Palavras-chaves: Processamento de imagens. Posicionamento por imagem. Drone

ABSTRACT

Object pose estimation trough camera images has multiple applications in the actual world, from industry to autonomous guided vehicles. This project aims the development of an algorithm to calculate the parameters of distance and angles that inform the object pose regarding to known points in space. Its main application is the annual drone competition that occurs at UFPR, where pilots have to catch an transport a cube in the least amount of time possible. The dimensions and characteristics of the cube are previously known. From these characteristics, the vision system calculates the relative position from the drone to the cube, and informs the pilot about where it is, so that the catch process occurs as fast as possible. The project was developed in python, and embedded in a Raspberry Pi 3. Due to instability in the color segmentation with illumination changes, errors added by morphological adjustments and ambiguity in the parameters estimation, the use of auxiliary sensors are suggested to improve the stipulation of the parameters.

Key-words: Image processing. Pose estimation. Drone

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

CPU	*Central Processing Unit*
GPIO	*General Porpouse Input/Output*
CSI	*Camera Serial Interface*
DSI	*Display Serial Interface*
GUI	*Graphical User Interface*
RGB	*Red, Green, Blue*
PnP	*Perspective n-Point*

SUMÁRIO

1 INTRODUÇÃO

A partir da década de 70 as pesquisas sobre processamento de imagem começam a ganhar relevância. Na indústria as câmeras já são usadas em diversos pontos da cadeia produtiva, desde a análise da qualidade, com medição de cores, de tamanho e forma, até no auxílio de posicionamento em processos automatizados. Além das aplicações já bem consolidadas na indústria, estão em desenvolvimento algoritmos baseados em inteligência artificial para aplicações em veículos autônomos, como carros e veículos aéreos não tripulados, onde além de identificar objetos, pessoas e situações, as câmeras podem fornecer medições de posicionamento e velocidade para os algorítimos de controle desses veículos, a análise de fluxo óptico. (KURKA, 2019)

Este trabalho se encontra entre as duas aplicações de processamento de imagem citadas anteriormente. O processamento de imagem é usado para estimar a localização da câmera em relação a um objeto de interesse e gerar dados que podem ser usados no controle e operação de um drone autônomo.

Acontece anualmente na Universidade Federal do Paraná competição de drones em parceria com três universidades francesas, cujo objetivo é pegar e carregar com o drone por 100 m de distância um cubo de 10 cm de lado. A equipe vencedora é aquela que completa o desafio no menor tempo. O desafio de engenharia se traduz na confecção da garra que apanhará o cubo, e na própria pilotagem do drone. Nas edições de 2018 e 2019, o drone foi controlado manualmente por pilotos, mas no futuro essa competição contará com a modalidade de drones autônomos, quando o desafio de engenharia se tornará muito mais ambicioso. Visando as próximas edições da competição e para o auxílio da modalidade pilotada, o processamento de imagem se torna importante. Como o piloto deve ficar a uma certa distância da área onde o cubo se encontra, ter informações sobre a distância e ângulo entre o drone e o cubo é um diferencial competitivo importante. Partindo do pressuposto de que as características do alvo são conhecidas, como a cor e a dimensão, assim como o conhecimento prévio de como o objeto aparecerá na imagem. Os métodos aqui estudados, explicados e implementados são a base para o processamento de imagens onde se deseja extrair informações que indiquem o posicionamento relativo entre a câmera e o objeto a ser

detectado.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

Desenvolver um algoritmo de processamento de imagem para auxílio da navegação de drone de competição, que forneça dados de ângulos e distância do alvo.

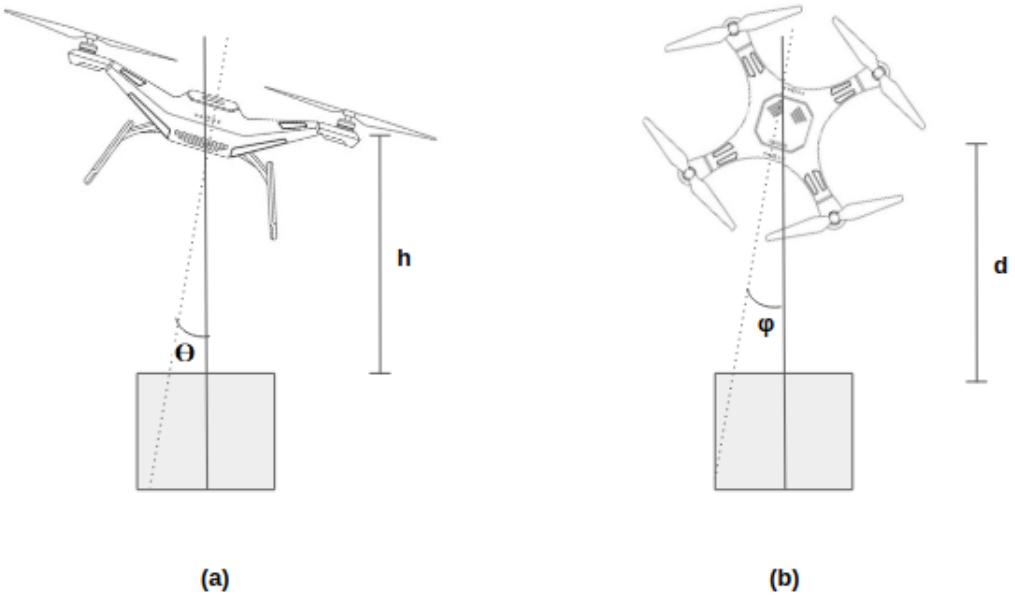
1.1.2 Objetivos Específicos

A partir do objetivo geral foram delimitados os seguintes objetivos específicos a serem atingidos durante o desenvolvimento do trabalho:

- Detecção do alvo em tempo real;
- Cálculo e apresentação na tela dos parâmetros em tempo real;
- Teste de uma solução inicial baseada em por segmentação por cor, identificação de bordas e cálculo dos parâmetros;
- Realizar uma análise comparativa entre os diferentes métodos de detecção de bordas e de ajuste de imagem (Morfologia, treshold, blur);
- Implementar algoritmo na plataforma Raspberry pi;

Os parâmetros a serem calculados são mostrados na figura 1.1. A figura (a) mostra a visão frontal do drone e do cubo, sendo h a altura do drone em relação ao cubo e θ o ângulo entre as normais do cubo e do drone. A figura (b) mostra a vista superior do conjunto, sendo ϕ o ângulo entre os eixos perpendiculares do cubo e do drone em relação ao chão. O parâmetro d é a distância entre os centros do cubo e do drone no mesmo eixo.

FIGURA 1.1 – *Parâmetros*



Fonte: O autor

1.2 JUSTIFICATIVA

Um algorítimo de processamento de imagens que detecte o alvo e forneça informações sobre o posicionamento da câmera em relação a ele tem diversas aplicações, tanto na indústria como na pesquisa de novos produtos e processos. A partir disso, o levantamento e aprendizagem dos métodos de processamento de imagens que possibilite tais aplicações contribui muito para o futuro. Cada vez mais a automatização de processos, tanto industriais como no próprio dia-a-dia das pessoas, é dependente de sistemas de visão capazes de inferir diferentes parâmetros do mundo a sua volta, possibilitando melhoria na precisão, custos e conforto.

2 MATERIAIS E MÉTODOS

2.1 HARDWARE

Nessa seção são abordados os itens de hardware usados no desenvolvimento do trabalho. Foram usados itens de baixo custo, peso e tamanho. Estes últimos para que não impactassem no desempenho do drone durante a competição.

2.1.1 *Raspberry Pi 3 B*

O *Raspberry Pi* é um produto da *Raspberry Pi Foundation* cujo principal propósito é colocar a computação e a cultura *make* nas mãos das pessoas por todo o mundo. O primeiro microcomputador começou a ser vendido em 2012 por U$35 . Desde então a comunidade ganhou força e a plataforma já está na sua terceira versão. Além do microcomputador, a fundação Raspberry oferece diversos módulos, materiais e conferências para que as pessoas possam aprender e aplicar vários conceitos relacionados à computação e a eletrônica de forma rápida e acessível.

FIGURA 2.1 – *Raspberry Pi 3 B*

Fonte: (RASPBERRYPIFOUNDATION, 2019b)

Esse microcomputador conta com um processador *Broadcom* BCM2837 atualizado, memória RAM de 1GB, 900MHz de velocidade e faz uso de uma CPU *ARM*

Cortex-A53 *quad-core* de 64 *bits*, que funciona com frequência nominal de 1,2 GHz. Além disso, consta também na placa: entradas USB, conectividade por Bluetooth 4.1 de baixo consumo e WLAN, conector para áudio, saída HDMI, e 40 pinos GPIO de baixo nível que podem ser utilizadas de diversas maneiras definidas por software, interface CSI para câmera e DSI para display.

2.1.2 *Pi NoIR Camera V2*

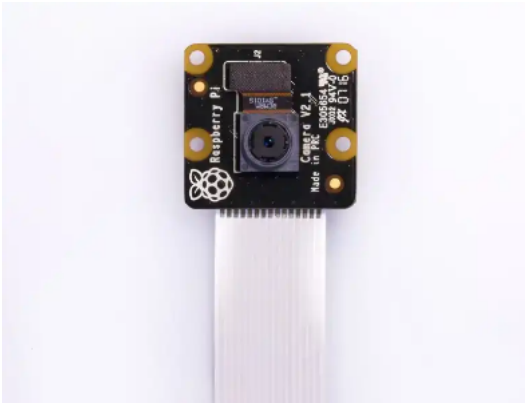
Câmeras digitais são dispositivos que transformam sinais luminosos em sinais digitais. Os sensores são matrizes de sensores de luz (tipicamente CCDs), cuja tensão de saída é proporcional à intensidade luminosa aplicada sobre eles. Combinando diferentes filtros anteriormente ao elemento sensor da câmera são obtidos os três canais de uma foto RGB.

O módulo *Pi NoIR Camera V2* é da própria marca Raspberry e utiliza um sensor Sony IMX219 que fornece uma imagem de resolução de 8 Mega pixeis através da tecnologia CMOS. Esse módulo tem tamanho reduzido (25 mm x 23 mm x 9 mm) e pesa pouco mais de 3 g. É usado amplamente em diversos equipamentos onde o tamanho é restrito, mas ainda assim oferece imagens com alta sensibilidade e alta taxa de *frames* por segundo. O módulo conta com foco da lente fixo. (SONY, 2019)

Uma diferença entre o módulo usado nesse projeto e câmeras comuns é a falta do sensor infra-vermelho. Isso não influencia em nada a qualidade da imagem e não apresenta nenhum impacto nos métodos descritos a seguir, salvo a segmentação por cores, onde os valores usados para as cores precisam ser corrigidos para essa diferença.

A câmera é conectada ao Raspberry Pi pela porta CSI através de um cabo *flat*.

FIGURA 2.2 – *Pi NoIR Camera V2*



Fonte: (RASPBERRYPIFOUNDATION, 2019a)

2.2 SOFTWARE

Nessa seção serão apresentados e descritos os componentes de software usados no desenvolvimento do projeto.

2.2.1 *Raspbian*

O sistema operacional usado no Raspberry Pi 3 é o *Raspbian*. Esse sistema é um Linux Debian customizado para ser usado na plataforma raspberry, onde as instruções que gerenciam a memória, o processador e os periféricos da plataforma foram modificadas para se alinharem aos requisitos da arquitetura ARM. Esse sistema já traz diversas facilidades para a operação do Raspberry, como bibliotecas, apoio da comunidade desenvolvedora, a própria instalação na plataforma e é inclusive *open source*.

2.2.2 Linguagem Python

O python é uma linguagem de programação de alto nível interpretada e orientada a objetos. Foi lançada por Guido Von Rossum em 1991 e prioriza a legibilidade sobre a velocidade. Isso a torna de fácil e rápida aprendizagem, além de ser amplamente utilizada na plataforma Raspberry. O que possibilita o fácil acesso a diversas bibliotecas, sejam de processamento de imagem, de cálculo numérico ou manipulações e criação de interfaces gráficas. Essa linguagem possui diferentes versões ainda em uso pelos desenvolvedores. A versão utilizada no trabalho foi a 3.7.

2.2.3 OpenCV

A biblioteca OpenCV foi originalmente desenvolvida pela Intel em 2000 para avançar em aplicações que usam os CPUs de forma intensa. Ela é multiplataforma e completamente aberta para usos acadêmicos e comerciais. Possui diversas funções para processamento de imagens e video I/O em tempo real, estrutura de dados, álgebra Linear e GUI básica. A segunda versão, usada nesse projeto, conta com diversas melhorias referente a primeira, como novas funções e melhoria na usabilidade.

2.3 PROCESSAMENTO DIGITAL DE IMAGENS

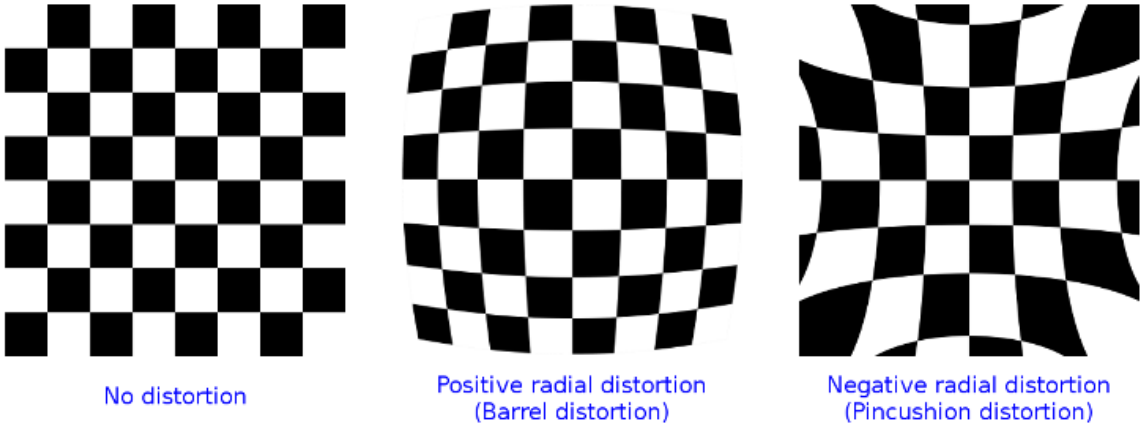
Processamento de imagens são um conjunto de métodos para alterar, melhorar, analisar, comprimir e reconstruir imagens (BRITANNICAACADEMIC, 2019). Esses métodos aplicados às imagens muitas vezes estão interconectados. Tudo se inicia na captura da imagem. A luz reflete sobre o objeto ou cena em questão e é traduzida para estruturas de dados que permitem o processamento computacional. Essas estruturas são matrizes bidimensionais, onde os elementos representam a intensidade da luz que atinge os elementos sensores das câmeras digitais através de lentes.

Esse processo não é perfeito, por isso, o primeiro passo no processamento de imagens é o pré-processamento, onde métodos matemáticos são aplicados às matrizes que representam o mundo real, de forma a minimizar as imperfeições introduzidas na imagem tanto pela lente das câmeras quanto pelos elementos sensores. Após o pré-processamento, características da imagem podem ser inferidas através de algoritmos específicos, como detecção de objetos ou formas. Após isso o objeto ou forma em questão podem ser separados da imagem, a chamada segmentação. Essa tarefa é extremamente dependente das características do objeto em análise. A forma e as descontinuidades entre o objeto e o fundo da imagem são elementos importantes na facilidade e na acuidade dessa operação. A tarefa de maior nível no processamento de imagens é a classificação. Através desta é inferida a identidade do objeto em análise na imagem, cuja tarefa é muitas vezes realizada por algoritmos de aprendizagem de máquina e *deep learning*.

2.3.1 Calibração

A calibração da câmera é um passo importante no processamento de imagens e através dela são corrigidas distorções causadas pela lente. Nenhum dispositivo é perfeito, erros de fabricação nas lentes, desalinhamento entre a lente e o sensor são defeitos facilmente encontrados e perceptíveis nas câmeras provenientes de fabricação em massa, onde o preço é muitas vezes balanceado de forma tendenciosa em relação à qualidade. (RICOLFE-VIALA, 2011) A figura 2.3 mostra a distorção causada pela lente em uma foto.

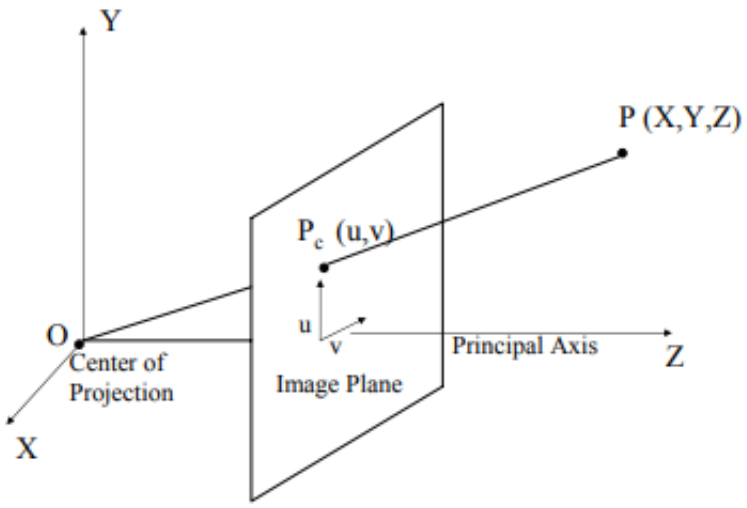
FIGURA 2.3 – Distorção em câmeras



Fonte: (MAJUMDER, 2019a)

Para poder explicar como acontece a calibração de câmeras digitais matematicamente primeiro faz-se necessário um modelo da própria câmera. Um modelo muito utilizado na literatura é o *pin hole*. Ele descreve a relação matemática entre um ponto descrito por coordenadas em 3D no mundo real e sua projeção no plano de imagem de uma câmera ideal. A figura 2.4 ilustra a projeção da imagem pela câmera.

FIGURA 2.4 – Modelo *Pin Hole*



Fonte: (MAJUMDER, 2019b)

A figura 2.4 mostra a câmera com o centro de projeção em *O*. A imagem está em foco, ou seja, o plano da imagem está a uma distância maior que uma distância focal *f* do centro de projeção. Primeiro encontra-se a matriz de calibração da câmera, a qual mapeia o ponto 3D P para 2D P_c. É possível achar o ponto P_c por simetria de triângulos mostrado na equação 2.1.

$$\frac{f}{Z} = \frac{u}{X} = \frac{v}{Y} \tag{2.1}$$

Reescrevendo de forma matricial:

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \tag{2.2}$$

Considerando que a origem do sistema de coordenadas do plano 2D pode não coincidir com o eixo Z do plano da imagem adicionam-se fatores de translação à matriz:

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} f & 0 & t_u \\ 0 & f & t_v \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \tag{2.3}$$

O ponto P_c é expressado em milímetros. Por esse motivo, faz-se necessário a adição de mais um fator que correlaciona as coordenadas medidas em pixeis com o mundo real em milímetros. Sabendo a resolução da câmera e considerando por generalização que a relação milímetros/pixeis não é a mesma nas direções x e y, são adicionados os fatores m_u e m_v. Além disso os eixos u e v podem não ser ortogonais. Por esse motivo é adicionado o fator de inclinação s. Com isso a equação da câmera nesse modelo pode ser escrita na seguinte forma:

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} m_u f & s & m_u t_u \\ 0 & m_v f & m_v t_v \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} \alpha_x & s & u_o \\ 0 & \alpha_y & v_o \\ 0 & 0 & 1 \end{pmatrix} P = KP \tag{2.4}$$

Na matriz K estão os parâmetros intrínsecos da câmera como a distância focal. Visto que a câmera pode estar rotacionada e transladada em relação ao plano da imagem, faz-se necessário a aplicação de uma matriz de rotação e translação para realinhar à configuração mostrada na figura 2.4.

A multiplicação da matriz K com a matriz de rotação e translação da a matriz completa de calibração, com os parâmetros extrínsecos. A calibração é o método de encontrar esses parâmetros, tanto intrínsecos como extrínsecos.

Isso é feito por um algoritmo da própria biblioteca do OpenCV. Para isso, é necessário um gabarito em padrão de tabuleiro de xadrez. O algoritmo detecta as intersecções das casas do tabuleiro, e sabendo que cada quadrado tem um certo tamanho em centímetros, o algoritmo determina todos os parâmetros intrínsecos e extrínsecos.

2.3.2 Segmentação por cor

Um método comum para o primeiro passo de detecção de objetos em imagens é a segmentação por cor. A segmentação em processamento de imagens é a separação de partes da imagem baseada por características comuns. Em imagens complexas, a segmentação se demostra um desafio muito grande. Porém, sabendo que o objeto em questão tem uma determinada cor, a segmentação pode ser simples. Ela é feita a partir da criação de uma máscara com os pixeis que contém as cores em um intervalo que englobam a cor do objeto a ser detectado. A partir disso, criam-se regiões na imagem que contém, ou não pixeis com os determinados intervalos de cor. Considerando então que o objeto tem uma cor específica, e homogênea, o primeiro passo para a identificação do objeto é isolar a área da imagem que contenha os pixeis com a determinada cor.

A figura 2.5 ilustra a segmentação por cor. A figura na direita mostra a imagem original, a figura a esquerda mostra apenas os pixeis de cor azul.

FIGURA 2.5 – Segmentação por cor

Fonte: (STANFORD, 2019)

2.3.3 Limiarização

A limiarização é um processo bastante usado no processamento de imagens. Ele pode ser descrito como a obtenção de uma imagem binária a partir de uma imagem em tons de cinza. A partir da imagem em tons de cinza, estipula-se um limiar, um valor máximo do tom de cada pixel, que caso seja superior ou inferior ao valor limite, é convertido em um valor saturado (branco ou preto) ou o próprio valor limite. Como mostra a equação 2.5.

$$G[n,m] = \begin{cases} 255, f[n,m] > k \\ 0, c.c. \end{cases} \tag{2.5}$$

Sendo k o valor limite, $f[n,m]$ o valor do pixel nas coordenadas $[n,m]$ e G o resultado.

A figura 2.6 mostra o resultado da operação, onde a figura a esquerda é a original, e a figura da direita é o resultado da aplicação do método de limiarização.

FIGURA 2.6 – Limiarização

Fonte: (STANFORD, 2019)

Esse método de processamento de imagens reduz consideravelmente a complexidade da imagem e possibilita de forma mais direta e rápida a detecção de outras características como bordas e cantos, cujos métodos de obtenção de baseiam na derivada dos valores dos pixeis.

2.3.4 Morfologia

Na biologia, a palavra morfologia é normalmente usada para nomear a área que descreve e classifica a forma e a estrutura de animais e plantas. Em processamento de imagem, morfologia se refere a técnicas matemáticas que extraem importantes características das imagens como os contornos, por exemplo. (GONZALEZ, 1993)

Existem várias técnicas compreendidas na morfologia (dilatação, erosão, fechamento e abertura) sendo que cada uma delas modifica a imagem a seu modo, evidenciando características, filtrando ruídos e ajustando contornos. Cada um desses

métodos citados será explicado com mas detalhes. Mas antes de entrar em detalhes das operações, alguns conceitos são necessários. Elementos estruturantes são conjuntos de pixeis usados nas operações. Translação é a movimentação de algum conjunto de pixeis. E reflexão é a inversão de coordenadas dado um eixo.

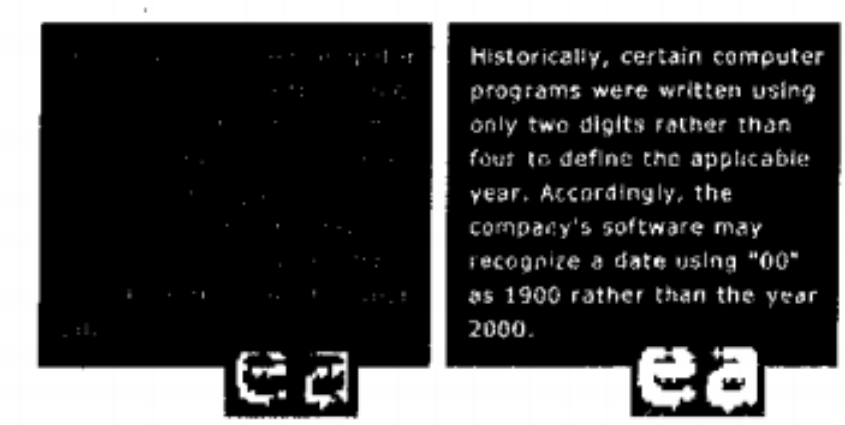
2.3.4.1 Erosão

Erosão é uma operação morfológica fundamental. A erosão retira pixeis de certas partes da imagem. A fórmula 2.6 mostra matematicamente a operação. Em palavras, a erosão de um conjunto A pelo elemento estruturante B é o conjunto de todos os pontos c, tais que, B transladado por c, esteja contido em A. (GONZALEZ, 1993)

$$A \ominus B = \{z | \, (B)_z \subseteq A\} \tag{2.6}$$

A figura 2.7 mostra o resultado da operação morfológica erosão. A esquerda, o texto não é legível. Mas após a aplicação da erosão, pixeis são retirados de dentro das letras, tornando o texto legível.

FIGURA 2.7 – Erosão



Fonte: (GONZALEZ, 1993)

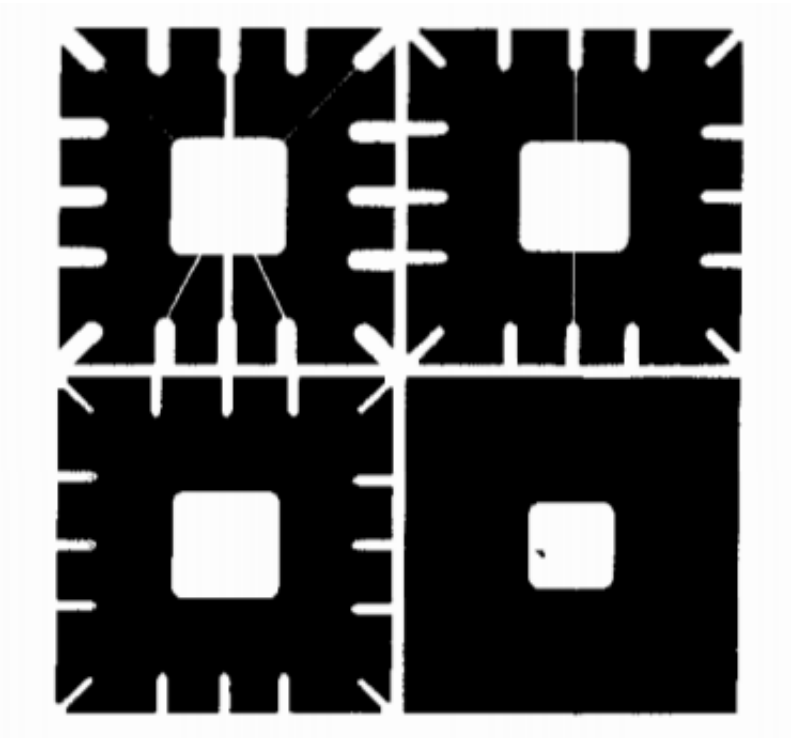
2.3.4.2 Dilatação

A dilatação, assim como a erosão, também é uma operação morfológica fundamental e é contrária à erosão. Ela adiciona pixeis à conjuntos. A fórmula 2.7 mostra matematicamente esta operação. Em outras palavras, a dilatação é o conjunto de todos os elementos z , de forma que as translações da reflexão de B por z, sobreponham-se ao conjunto A por pelo menos um elemento não-nulo. (GONZALEZ, 1993)

$$A \oplus B = \left\{ z | \left(\hat{B} \right)_z \cap A \neq \varnothing \right\} \tag{2.7}$$

A figura 2.8 ilustra a operação. A imagem original é a do canto superior esquerdo. As outras três subsequentes são resultados da operação de dilatação, com elementos estruturantes de diferentes tamanhos.

FIGURA 2.8 – Dilatação



Fonte: (GONZALEZ, 1993)

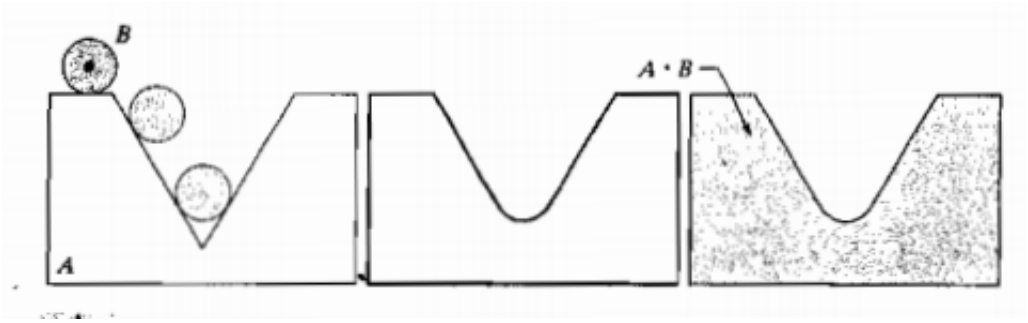
2.3.4.3 Fechamento

A operação de fechamento é uma combinação de uma dilatação e uma erosão. O objetivo dessa operação é reestabelecer conexões sem modificar significativamente o tamanho do conjunto. Eliminando pequenos buracos e suavizando o contorno. A fórmula 2.8 mostra matematicamente a operação. (GONZALEZ, 1993)

$$A \cdot B = (A \oplus B) \ominus B \tag{2.8}$$

A figura 2.9 ilustra a operação de fechamento. Nesse caso o elemento estruturante é um circulo que ao ser transladado pelo contorno, preenche os espaços vazios, suavizando a forma em questão.

FIGURA 2.9 – Fechamento

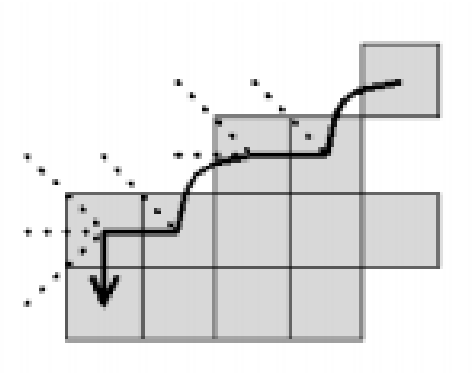


Fonte: (GONZALEZ, 1993)

2.3.5 Detecção de contornos

Detecção de contornos é uma técnica fundamental no processamento digital de imagens. O algoritmo descrito aqui é a base da função *findContours* da biblioteca OpenCV. O algoritmo se baseia no seguimento das bordas. A partir de um primeiro pixel, o algoritmo calcula a derivada com os pixeis vizinhos. Os pixeis que contém uma derivada nula, ou muito baixa são considerados como parte do contorno. São considerados parte do contorno aqueles pixeis que possuem um vizinho com a derivada de alta. A figura 2.10 ilustra a busca pelo contorno, onde as linhas pontilhadas representam as comparações para a definição de quais pixeis fazem parte da borda e quais não fazem. O algoritmo termina o processo ao encontrar o pixel inicial. (SUZUKI, 1985)

FIGURA 2.10 – Seguimento de bordas



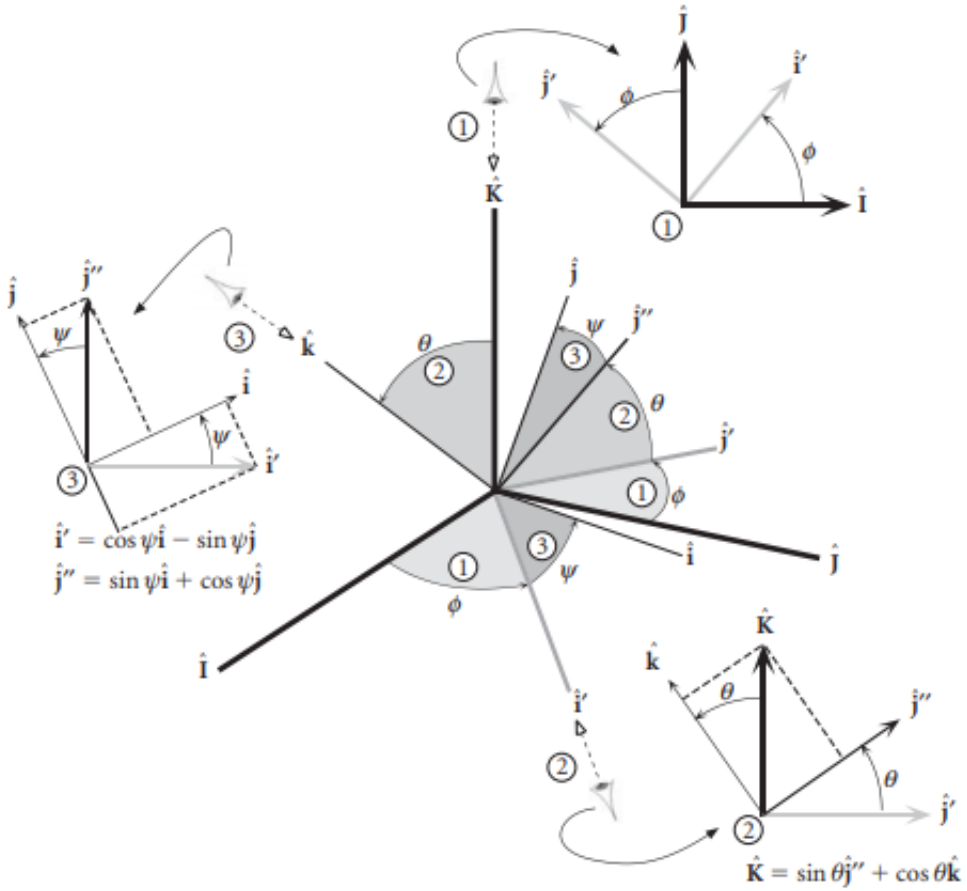
Fonte: (IOWAUNIVERSITY, 2019)

2.4 MÉTODOS

2.4.1 Ângulos de Euler

Três ângulos são necessários para determinar a orientação de um corpo rígido no espaço relativa a um sistema de coordenadas inercial. A figura 2.11 mostra os ângulos de Euler. Existem diferentes formas de se representar a orientação de um corpo rígido no espaço. Os ângulos de Euler são três ângulos, onde o primeiro se refere a rotação em Z, o segundo em Y e o terceiro em X (ZYX). Outra forma de representação são os ângulos de inclinação, guinada e rolamento (*pitch, yaw and roll* em inglês). Esses ângulos são também usados para descrição de orientação, mas a ordem é diferente (XYZ).(CURTIS, 2005)

FIGURA 2.11 – Ângulos de Euler



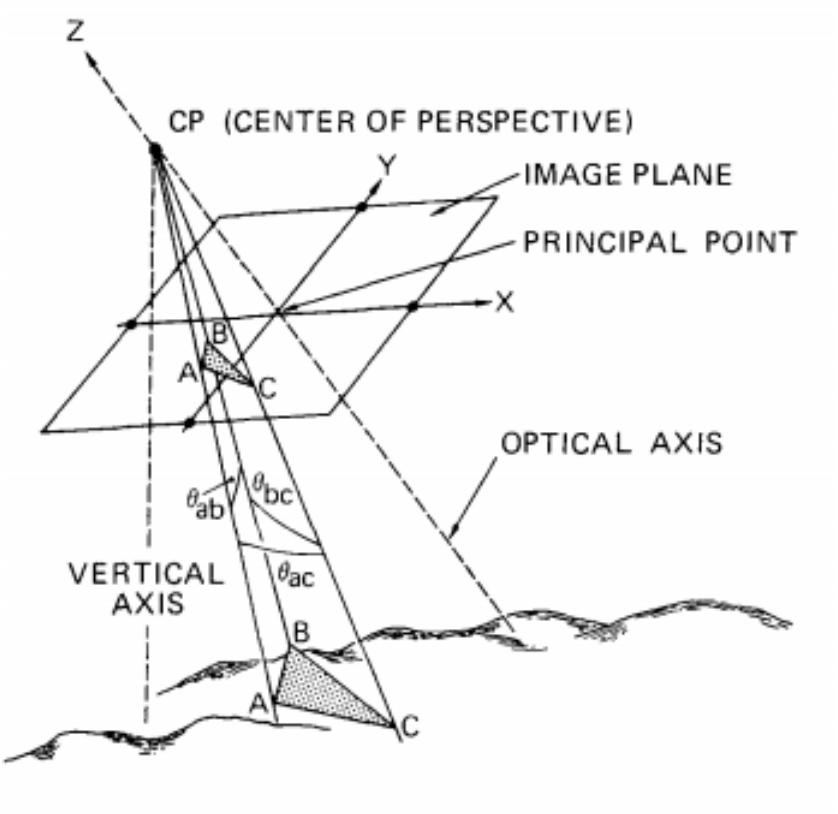
Fonte: (CURTIS, 2005)

2.4.2 Perspectiva n-pontos

Em 1981, Martin A. Fischler and Robert C. Bolles publicaram um artigo que descrevia um método para determinação de localização de uma câmera baseado em pontos referenciados na terra para aplicações cartográficas. Até esse ponto, isso era feito completamente de forma manual, iterativa, propensa a erros e demorada.

O problema da determinação de localização pode ser formalmente definido da seguinte maneira: Dado um conjunto de m pontos de controle, cujas coordenadas tridimensionais são conhecidas em um sistema de coordenadas, e dado uma imagem na qual os pontos de controle são visíveis, determine a localização (relativa ao sistema de coordenadas dos pontos de controle) de onde a imagem foi obtida. Situação essa ilustrada na figura 2.12. Na figura, nota-se que os ângulos formados entre as retas que cortam o centro de perspectiva e os pontos de controle são facilmente calculados caso as distâncias sejam conhecidas.

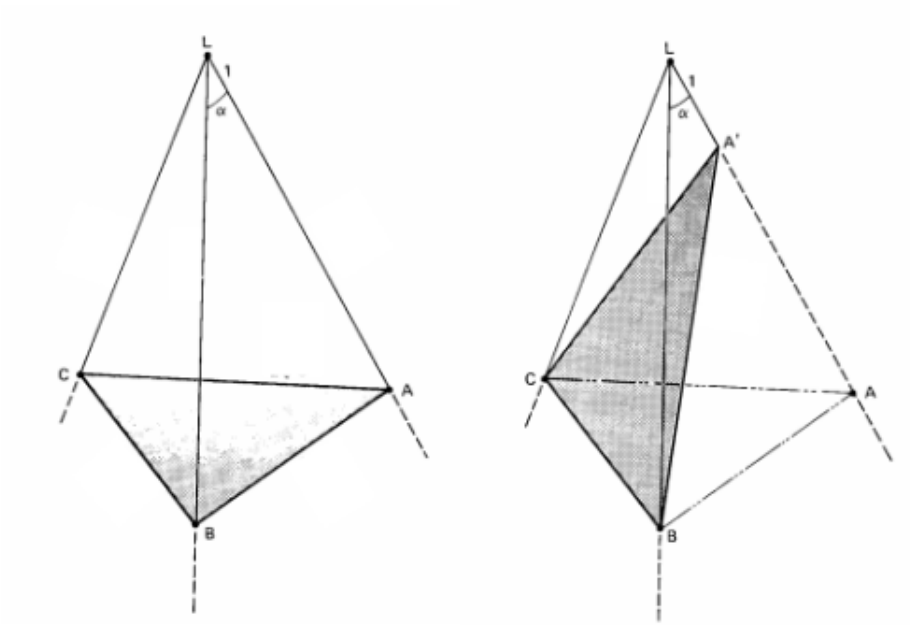
FIGURA 2.12 – Problema de determinação de localização



Fonte: (FISCHLER A., 1981)

O problema PnP então consiste em achar o menor número de correspondências que resulte na localização irresoluta da câmera em relação aos pontos de controle. Com $n = 1$, ou seja, P1P, existe apenas um ponto de controle, e as possibilidades de resultado são infinitas. Com $n = 2$, a posição da câmera está contida em um círculo onde o eixo formado pelos pontos de controle se encontram transfixando seu centro. Com $n = 3$, a possibilidade de resultados se reduz a 4 localizações, como mostra a figura 2.13. A figura mostra a solução real a esquerda e uma possível solução a direita. As outras três possíveis soluções são quando as outras pontas do triângulo (C e B) são deslocadas em direção ao ponto de perspectiva L seguindo o exemplo da figura a direita.

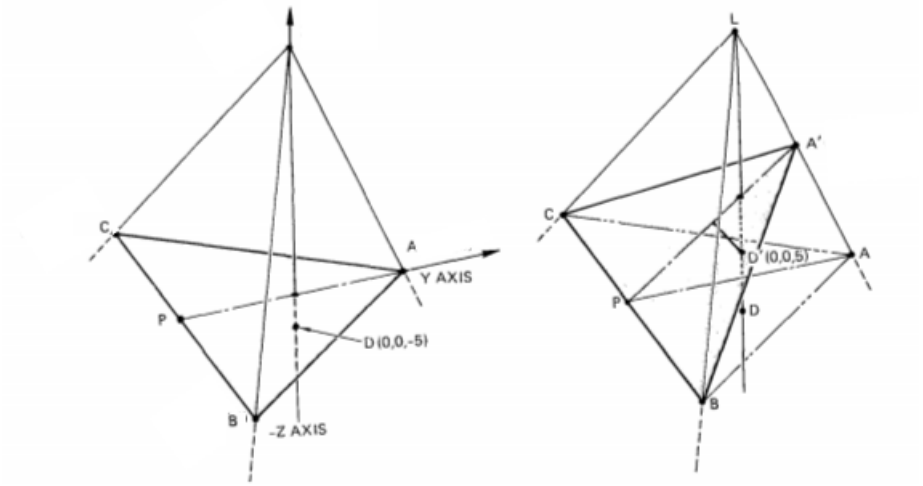
FIGURA 2.13 – Geometria do P3P



Fonte: (FISCHLER A., 1981)

Para $n = 4$ a solução ainda não é única, como mostra a figura 2.14. Nela o ponto A é deslocado sobre a reta, assim como o ponto D, que também é solução do sistema.

FIGURA 2.14 – Geometria do P4P



Fonte: (FISCHLER A., 1981)

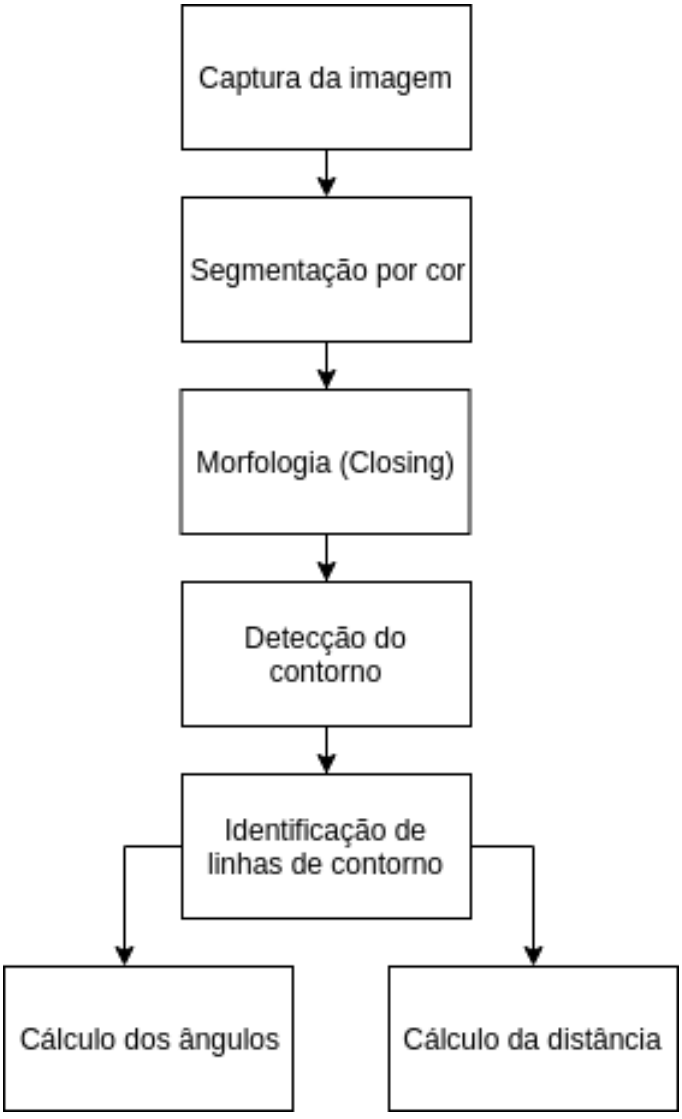
3 DESENVOLVIMENTO

Este capítulo descreve o desenvolvimento do projeto como um todo, desde a metodologia, até cada método usado na obtenção dos resultados.

3.1 METODOLOGIA

A figura 3.1 mostra o diagrama no qual o desenvolvimento foi baseado.

FIGURA 3.1 – Diagrama da metodologia



Fonte: O autor

Cada bloco do diagrama representa um passo do algoritmo de detecção dos parâmetros do cubo em relação ao drone, já descritos na figura 1.1. Para cada bloco,

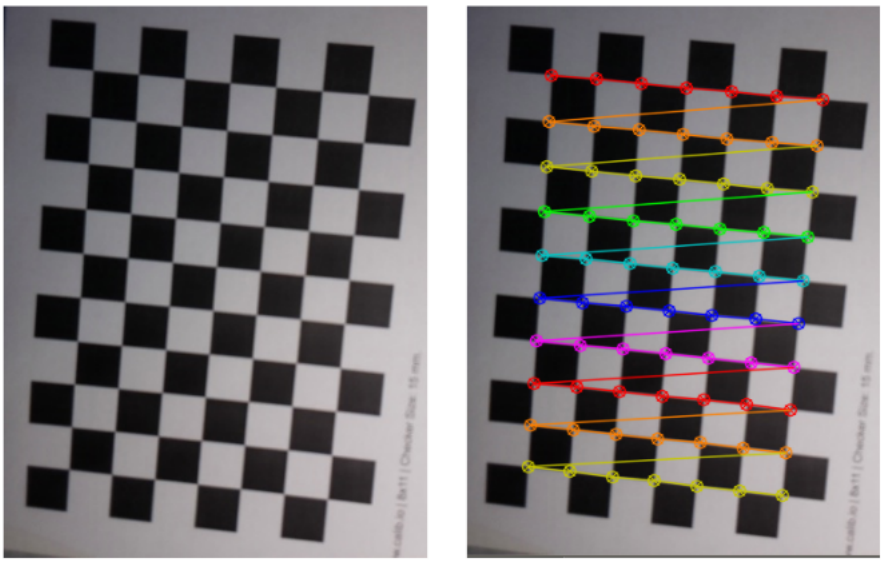
diferentes parâmetros, algoritmos e métodos foram testados, usando proncipalmente os métodos já implementados na biblioteca OpenCV em python, diretamente na plataforma Raspbery Pi, já completando um dos objetivos específicos.

As próximas secções explanam os métodos usados em cada um dos blocos mostrados na figura 3.1.

3.2 CAPTURA DA IMAGEM

O projeto começa com a captura da imagem. Para isso, na plataforma Raspberry Pi, o pacote *picamera* contém todas as funções para controle da câmera. Mas antes de poder usar as imagens diretamente da câmera, ela precisa ser calibrada. O método de calibração usado foi o mesmo descrito anteriormente. A figura 3.2 mostra um exemplo de imagem do gabarito usada na calibração. A esquerda a imagem original, a direita a imagem dos pontos detectados pelo algoritmo de calibração fornecido pela própria biblioteca OpenCV.

FIGURA 3.2 – Calibração



Fonte: O autor

Foram usadas ao todo 36 imagens do gabarito para a calibração, e o resultado da matriz de calibração foi:

$$K = \begin{pmatrix} 502,251 & 0 & 319,607 \\ 0 & 503,008 & 237,054 \\ 0 & 0 & 1 \end{pmatrix}$$

3.3 SEGMENTAÇÃO POR COR

O passo de segmentação por cor é responsável por detectar o objeto. Como no caso da competição, a cor do cubo é conhecida e diferente do fundo, essa abordagem se mostra simples e eficiente. A figura 3.3 mostra a imagem original na esquerda, e a imagem já segmentada na direita.

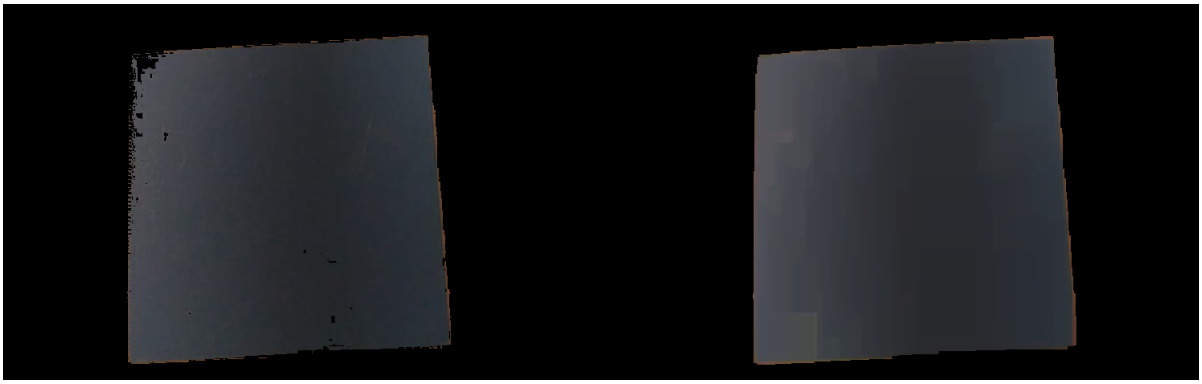
FIGURA 3.3 – Segmentação por cor

Fonte: O autor

3.4 MORFOLOGIA

Devido a iluminação, muitas vezes alguns pixeis podem ser considerados como fora dos limites estipulados para a segmentação por cor. Por isso a morfologia se faz necessária, além de suavizar as bordas. A figura 3.4 mostra a imagem segmentada na esquerda, e a imagem após a aplicação da morfologia. Nesse caso, o método aplicado foi o de fechamento, usando um elemento estruturante de 50 pixeis por 50 pixeis. As dimensões do elemento estruturante foram encontradas empiricamente para essa solução. Percebe-se que o método foi eficiente em corrigir os pixeis que estavam faltantes na imagem segmentada.

FIGURA 3.4 – Fechamento



Fonte: O autor

3.5 DETECÇÃO DE CONTORNO

O contorno do cubo é detectado pela função *findContours* do OpenCV que usa o método de seguimento de contornos. Considerando que o cubo é o único objeto azul na imagem, o contorno usado para o cálculo dos parâmetros é aquele que apresenta a maior área, visto podem existir pequenas áreas identificadas como azuis na imagem. Essa função retorna todos os contornos achados na imagem em ordem decrescente de área. Por isso, apenas o primeiro contorno da lista e usado. A imagem 3.5 mostra o contorno detectado pela função. A esquerda vê-se a imagem binária da do cubo, e a direita o contorno encontrado em vermelho.

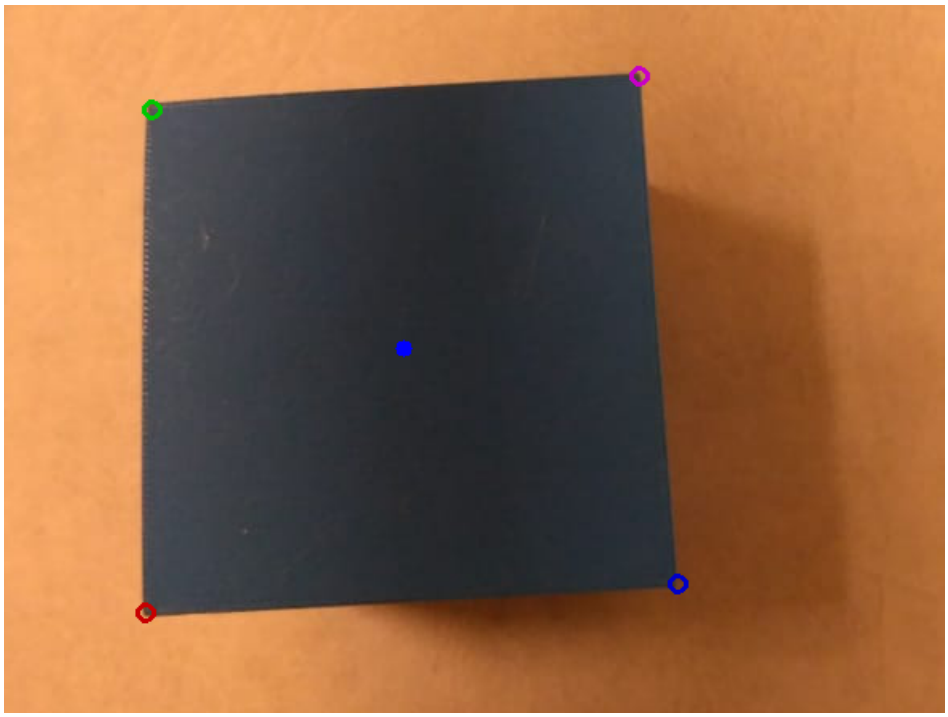
FIGURA 3.5 – Detecção do contorno

Fonte: O autor

3.6 IDENTIFICAÇÃO DE CANTOS

O contorno detectado anteriormente contém vários pontos. Para os cálculos dos parâmetros, são necessários pontos conhecidos do cubo, ou seja, são necessários pontos do cubo em pixeis que correspondam a pontos do cubo no mundo real. Os únicos pontos que contém essa característica são os pontos dos cantos, visto que as dimensões do cubo são conhecidas. Além disso, esses pontos precisam ser sempre encontrados em uma determinada ordem. A ordem escolhida é a horária, começando pelo ponto mais a esquerda da parte superior da imagem. As bordas detectadas são mostradas na imagem 3.6.

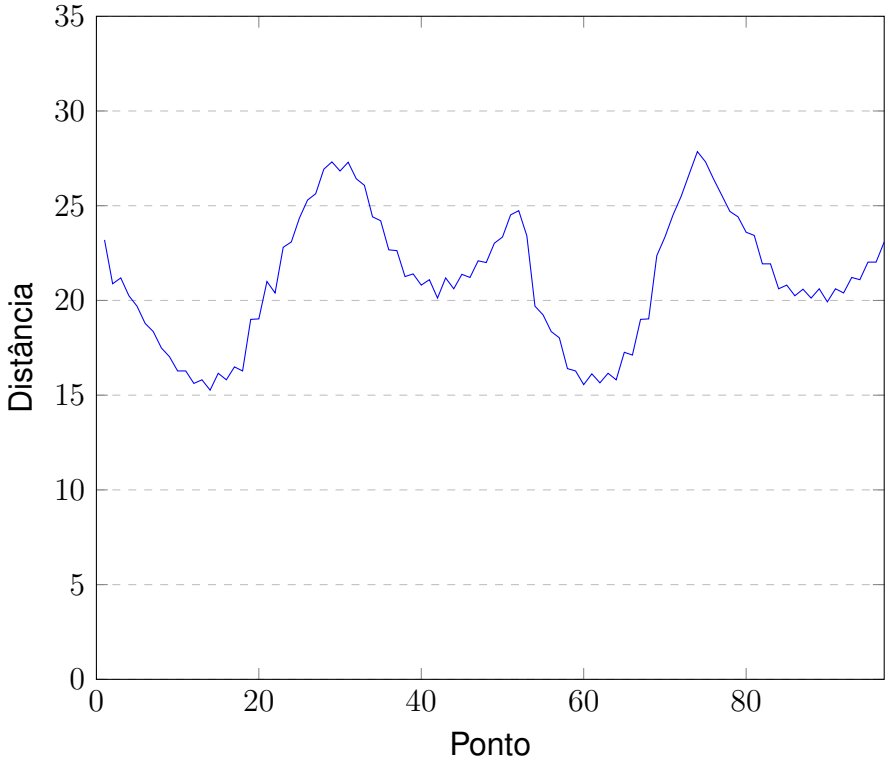
FIGURA 3.6 – Identificação dos cantos



Fonte: O autor

Esses cantos são encontrados em três passos. O primeiro consiste em calcular todas as distâncias entre todos os pontos do contorno e o centro do contorno. Esses valores são armazenados temporariamente em um vetor, que se plotado, gera um padrão mostrado na figura 3.7. Nesse padrão, os pontos mais altos, quatro no total, serão sempre os cantos do cubo, pois estes são os pontos com a maior distância até o centro. O segundo passo é encontrar os picos. Para isso, faz-se o uso da função da biblioteca *Scipy* do python, a *findPeaks*. Essa função retorna os máximos locais do

FIGURA 3.7 – Distâncias entre os pontos do contorno e o centro



Fonte: O autor

vetor de distâncias. Os primeiro quatro valores são os cantos do cubo. O terceiro passo é ordenar os pontos de acordo com a ordem requerida pelo algoritmo de cálculo dos parâmetros. Para isso, as coordenadas do centro são subtraídas das coordenadas dos pontos. Nesse caso, o ponto que tiver a coordenada ajustada no eixo x maior que zero, e a coordenada em y menor que zero é o canto superior esquerdo.

3.7 CÁLCULO DOS ÂNGULOS E DISTÂNCIAS

O método usado no cálculo dos parâmetros é a função *solvePnP* da biblioteca OpenCV. Ela toma como argumentos os pares de correspondências dos cantos do cubo, detectados anteriormente e as dimensões do cubo no mundo real. Visto que as dimensões do cubo são conhecidas, e considerando o canto mais a esquerda sempre como a origem, a saída da função são dois vetores, um vetor contém informações sobre a translação da imagem, e o outro as informações sobre a rotação. A partir desses dois vetores, a matriz que mapeia os pontos em 2D para 3D, ou seja, uma

matriz de rotação junto a um vetor de translação. A partir dessa matriz, os ângulos de Euler são determinados a partir da função *decomposeProjectionMatrix*, que toma como argumento a matriz de projeção, que é a combinação entre a matriz da câmera, o vetor de rotação e o vetor de translação resultantes da função *solvePnP*.

4 RESULTADOS

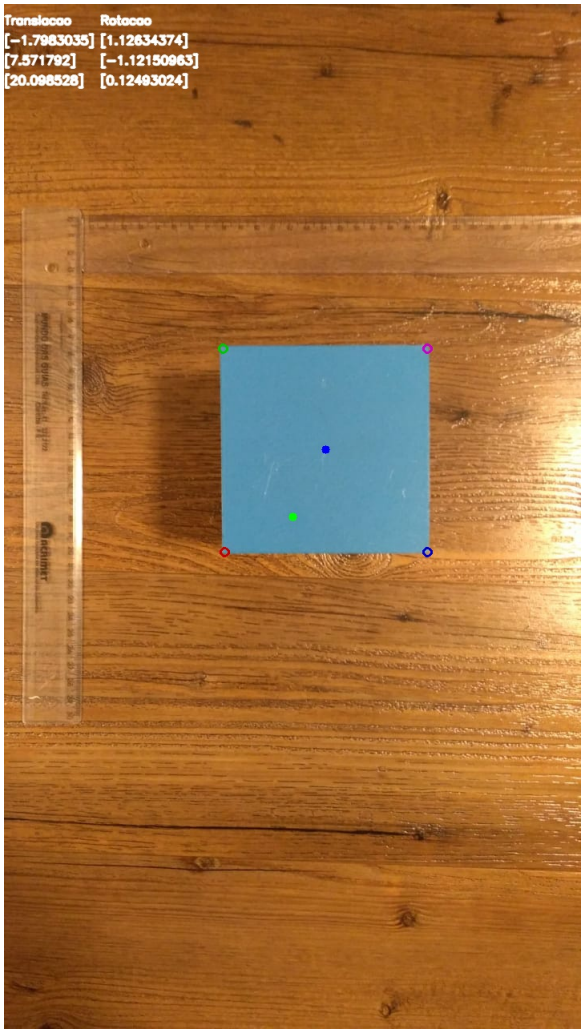
As imagens apresentadas nesse capítulo mostram os resultados dos cálculos dos parâmetros feitos pelo método *solvePnP*. Os resultados são mostrados na própria imagem, no canto superior esquerdo. A primeira coluna se refere à translação (XZY), no referencial da câmera, e a segunda coluna mostra os ângulos, de guinada, inclinação e rolagem.

Em relação a performance, o algoritmo apresentou uma média de 2,5 fps. Para aplicações em tempo real, essa taxa é baixa. Existem vários pontos no aplicativo que oferecem potencial de otimização, principalmente na identificação dos cantos e do contorno.

4.0.1 Ãngulos

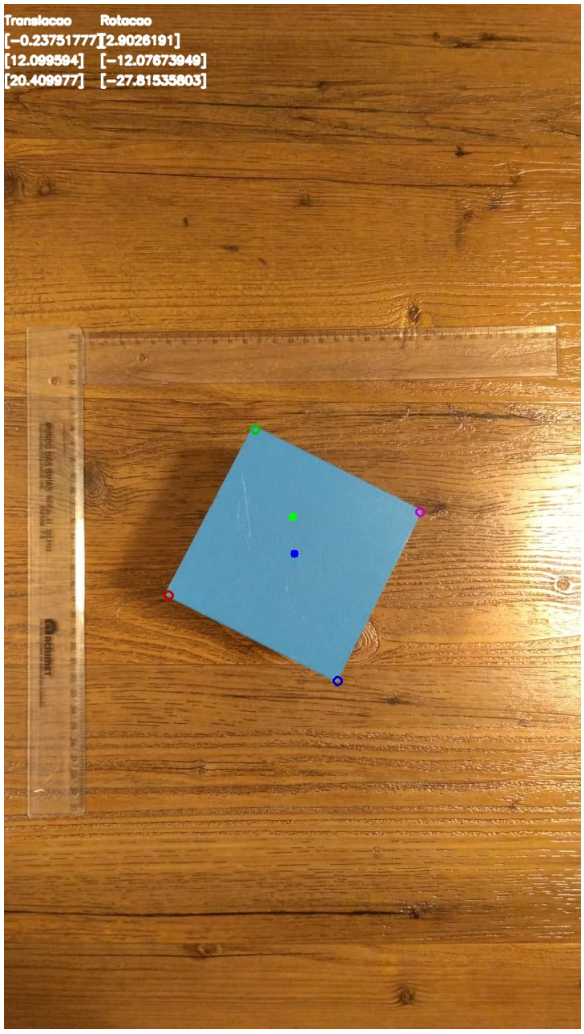
Os cálculos dos ângulos de rolagem se mostraram eficientes, como mostram as imagens 4.1 e 4.2. Porém as outras medições dos outros ângulos não bateram com a realidade. Um exemplo disso é a figura 4.2. O ângulo de inclinação calculado foi de -12°

FIGURA 4.1 – Ângulo 0°



Fonte: O autor

FIGURA 4.2 – Ângulo 30°

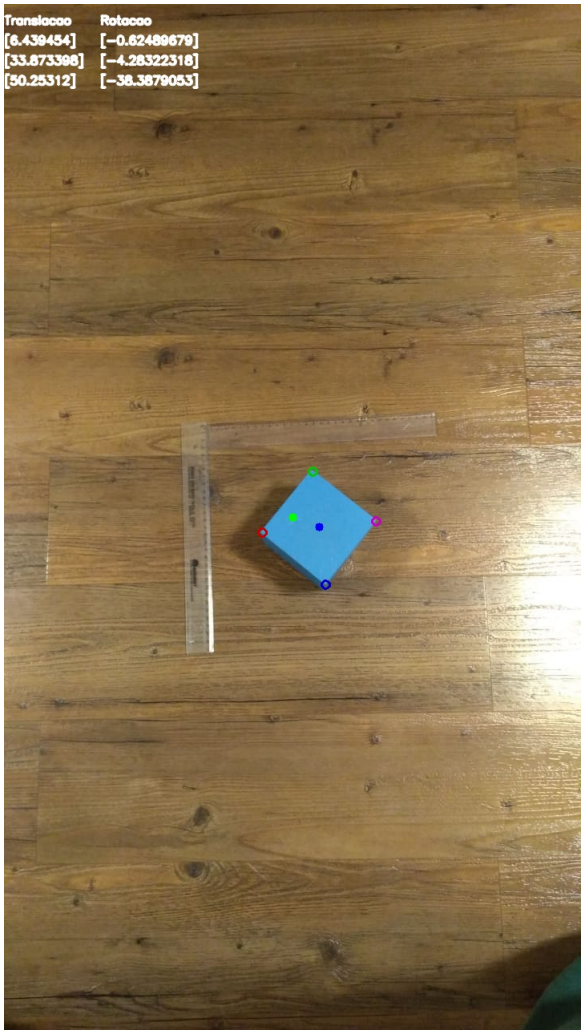


Fonte: O autor

4.0.2 Translação

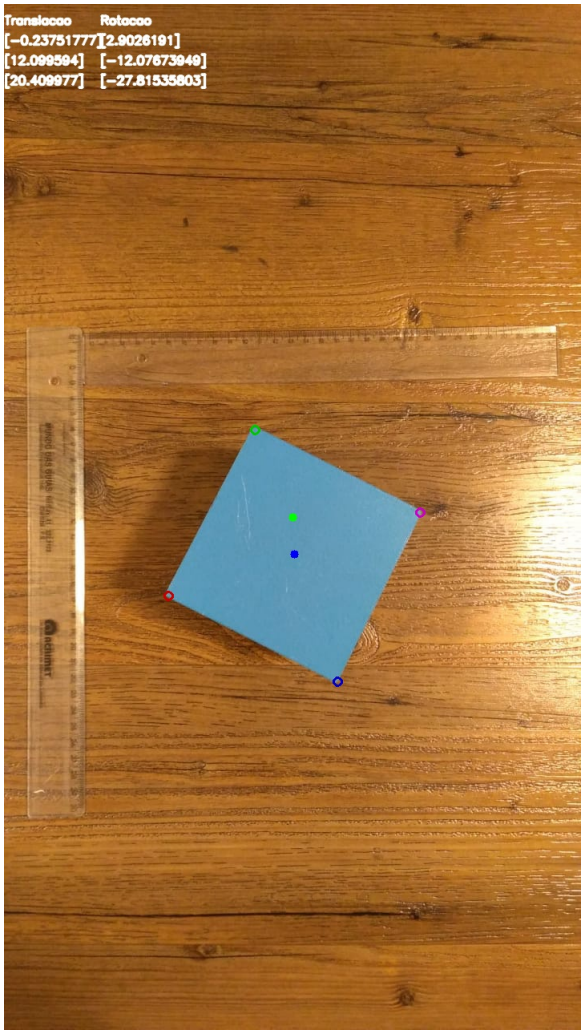
Em relação a translação, a medida em Z se mostrou eficiente. Isso é visível nas figuras 4.3 e 4.4

FIGURA 4.3 – Longe



Fonte: O autor

FIGURA 4.4 – Perto



Fonte: Do autor

O processamento das imagens até a detecção dos cantos se mostrou bastante efetiva. Porém a segmentação por cor se mostrou instável em situações de baixa luminosidade ou em situações onde sombras passavam sobre o cubo. Quando a proximidade da câmera e o cubo são grandes, a curva mostrada no gráfico 3.7 mostra picos menos proeminentes. Isso dificulta a detecção dos cantos, como mostrado na figura 4.4.

5 CONCLUSÃO

A estimação da posição de câmeras referentes a pontos conhecidos no espaço é um problema com diversas aplicações. Dentre elas a navegação de drones. No caso da competição que ocorre anualmente na UFPR, o sistema de visão proposto por esse trabalho representa um diferencial competitivo. Mas devido a incertezas referentes à iluminação e a instabilidade dos cálculos dos parâmetros, ainda precisa de refinamentos para poder ser colocado em prática.

Os grandes pontos que influenciaram o resultado negativo foram em primeiro lugar a própria iluminação, que implicou no aumento do tamanho do elemento estruturante do ajuste de fechamento, que adicionou incerteza na detecção do contorno. Em segundo, a co-planaridade e a baixa quantidade de pontos conhecidos que são detectados no cubo. A solução para o problema PnP requer pelo menos seis pontos para uma solução sem ambiguidades. As únicas medidas que foram acretivas, foram aquelas em que o centro da câmera e o centroide do cubo estavam colineares, e perpendiculares à face superior do cubo. Essa configuração pode ser de grande ajuda quando o drone está bem próximo do cubo, mas nesses casos a identificação indeterminada dos cantos influencia o resultado negativamente. Isso acontece por que quanto mais próximo a câmera está do cubo, as distâncias do contorno até o centro tendem a ficar mais uniformes, influenciando a identificação de máximos locais.

Apesar disso, dadas as devidas condições, os objetivos foram cumpridos. Para futuros trabalhos o foco deve ser concentrado na estabilização da detecção do cubo. Para isso redes neurais podem ser uma alternativa robusta. Além disso, o cálculo dos parâmetros pode ser combinado com outros sensores presentes também no drone, como acelerômetros, giroscópios, magnetômetros e barômetros. Dessa forma, mesmo com poucos pontos, as ambiguidades do algoritmo que soluciona o problema PnP podem ser mais facilmente resolvidas.

REFERÊNCIAS

BRITANNICAACADEMIC. *Definição processamento de imagem*. 2019. <https://academic-eb-britannica.ez22.periodicos.capes.gov.br/levels/collegiate/article/image-processing/472572>. Acesso em 3 jun. 2019. Citado na página 16.

CURTIS, H. D. *Orbital mechanics for engineering students*. [S.l.]: Elsevier, 2005. Citado 2 vezes nas páginas 25 e 26.

FISCHLER A., B. C. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, v. 24, p. 381–395, 1981. Citado 2 vezes nas páginas 27 e 28.

GONZALEZ, R. *Digital Image Processing*. [S.l.]: Pearson, 1993. Citado 4 vezes nas páginas 21, 22, 23 e 24.

IOWAUNIVERSITY. *Segmentation: Edge-based segmentation*. 2019. <http://user.engineering.uiowa.edu/ dip/LECTURE/Segmentation2.html>. Acesso em 14 jun. 2019. Citado na página 25.

KURKA, P. R. G. Applications of image processing in robotics and instrumentation. *Mechanical Systems and Signal Processing*, Elsevier, v. 124, p. 142–169, 2019. Citado na página 10.

MAJUMDER, A. *Distorção*. 2019. <https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d//_reconstruction.html>. Acesso em 3 jun. 2019. Citado na página 17.

____. *Modelagem câmera Pin Hole*. 2019. <https://www.ics.uci.edu/ majumder/vispercep/cameracalib.pdf>. Acesso em 3 jun. 2019. Citado na página 18.

RASPBERRYPIFOUNDATION. *Pi NoIR Camera V2*. 2019. <https://www.raspberrypi.org/products/pi-noir-camera-v2/>. Acesso em 2 jun. 2019. Citado na página 15.

____. *Raspberry Pi 3 Model B*. 2019. <https://www.raspberrypi.org/products/raspberry-pi-3-model-b/>. Acesso em 2 jun. 2019. Citado na página 13.

RICOLFE-VIALA, C. Using the camera pin-hole model restrictions to calibrate the lens distortion model. *Optics and Laser Technology*, Elsevier, v. 43, p. 996–1005, 2011. Citado na página 17.

SONY. *Módulo usado na câmera Pi NoIR V2*. 2019. <https://www.sony-semicon.co.jp/products_en/new_pro/april_2014/imx219_e.html >. Acesso em 3 jun. 2019. Citado na página 14.

STANFORD. *Distorção*. 2019. <https://ai.stanford.edu/ syyeung/cvweb/tutorial1.html>. Acesso em 14 jun. 2019. Citado 2 vezes nas páginas 20 e 21.

SUZUKI, S. Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing*, Elsevier, v. 30, p. 32–46, 1985. Citado na página 24.

APÊNDICE A – CÓDIGO

```
# coding: utf-8

import cv2
import numpy as np
import sys
from math import hypot
from math import atan2
from math import asin


from picamera.array import PiRGBArray
from picamera import PiCamera
import time
import cv2


import matplotlib.pyplot as plt
from scipy.signal import find_peaks


# initialize the camera and grab a reference to the raw camera capture
camera = PiCamera()
camera.resolution = (640, 480)
camera.framerate = 32
rawCapture = PiRGBArray(camera, size=(640, 480))
time.sleep(0.1)



def mostraImg(titulo1, titulo2, img1, img2):
```

```
        cv2.imshow(titulo1, img1)
        cv2.imshow(titulo2, img2)
        key = cv2.waitKey(0)

def main():


    #Limites pra segmentação por cor
    boundaries = [([127, 122, 62], [177, 172, 192])]

    for frame in camera.capture_continuous(rawCapture, format="bgr",
                                           use_video_port=True):

        image=frame.array
        #print(image)
        t1 = time.time()
        try:
            for (lower, upper) in boundaries:
                # create NumPy arrays from the boundaries
                lower = np.array(lower)
                upper = np.array(upper)

                # find the colors within the specified boundaries
                # and apply the mask
                mask = cv2.inRange(image, lower, upper)

                # Cor detectada
                output = cv2.bitwise_and(image, image, mask = mask)
                mask = output

                kernel = np.ones((50,50),np.uint8)
                closing = cv2.morphologyEx(output, cv2.MORPH_CLOSE, kernel)
                output = closing
```

```
# Preto e branco
output = cv2.cvtColor(output,cv2.COLOR_BGR2GRAY)

output = cv2.threshold(output, 1, 255, cv2.THRESH_BINARY)[1]

# Pegar coordenadas das bordas do contorno por contours
contours, hierarchy = cv2.findContours(output, cv2.RETR_TREE,
                                       cv2.CHAIN_APPROX_NONE)

# COnsidera que o maior contorno é o quadrado
c = contours[0]
#print(c)
cv2.drawContours(closing,contours,0,(0,0,255), 2)

# calculate moments of binary image
M = cv2.moments(contours[0])

# calculate x,y coordinate of center
cX = int(M["m10"] / M["m00"])
cY = int(M["m01"] / M["m00"])

# put text and highlight the center
cv2.circle(image, (cX, cY), 5, (255, 0, 0), -1)

# Calcula as distâncias entre o centroide e
# todos os pontos do contorno
dist = []
coord = []
for p in contours[0]:
    temp = hypot(cX - p[0][0], cY- p[0][1])
    coord.append(p[0])
    dist.append(temp)
```

```
# Faz com que a lista de distâncias comece com um valor mínimo,
# Caso contrário um pico pode não ser identificado pelo find peaks
offset = dist.index(min(dist))
dist = dist + dist[:offset]
dist = dist[offset:]



# Acha os picos com a função find_peaks
peaks, propr = find_peaks(dist, prominence=2, width=2)

# Arruma os indices da lista pra tirar o offset
# colocado por ter mudado por começar com um minimo
cantos = []
cantosPre = []
for p in peaks:
    indx = p + offset
    if indx >= len(dist):
        indx = indx - len(dist)

    # Printa os pontos na tela
    pt = (coord[indx][0],coord[indx][1])
    cantosPre.append(pt)

# Desenha centro da imagem
cv2.circle(image, (360, 640), 5, (0, 255, ), -1)

# Coloca os cantos em ordem
it = 0
canto0 = 1
while canto0:
    #print(it)
```

```
    if cantosPre[it][0] - cX < 0 and cantosPre[it][1] - cY < 0:
        for kit in range(0,4):
            cantos.append(cantosPre[it])
            it = it + 1
            if it == 4:
                it = 0
        canto0 = 0
    it = it + 1


cv2.circle(image,cantos[0],5,(0,200,0),2)
cv2.circle(image,cantos[1],5,(0,0,200),2)
cv2.circle(image,cantos[2],5,(200,0,0),2)
cv2.circle(image,cantos[3],5,(200,0,200),2)

cantos.append((cX, cY))
# Teste com o pnp
objectPoints = np.array([(0,0,0), (0,10,0), (10,10,0),
                         (10,0,0)],dtype=np.float32)
cantos = np.array(cantos,dtype=np.float32)



cameraMatrix = np.array([(5.022514281632450661e+02,
                          0,
                          3.196071369571616287e+02),
                         (0,
                          5.030083998572816881e+02,
                          2.370542264434168658e+02),
                         (0,
                          0,
                          1)],dtype=np.float32)
```

```
distCoeffs = np.array([2.059048620706134536e-01,
                       -4.892609345962896095e-01,
                        5.239882661353135193e-04,
                       -8.352698096665962758e-04,
                        3.416575120330669901e-01],
                        dtype=np.float32)



_, rVec, tVec = cv2.solvePnP(objectPoints, cantos, cameraMatrix,
                             distCoeffs)

Rt = cv2.Rodrigues(rVec)[0]
R = Rt.transpose()


proj_matrix = np.hstack((R, tVec))
euler_angles = cv2.decomposeProjectionMatrix(proj_matrix)[6]
yaw   = euler_angles[1]
pitch = euler_angles[0]
roll  = euler_angles[2]

# convert to centimeters
translation = tVec

# convert to degree
rotation = rVec * 180./np.pi


axis = np.float32([[5,0,0], [0,5,0], [0,0,-5]]).reshape(-1,3)
```

```
            imgpts, jac = cv2.projectPoints(axis, rVec, tVec,
                                            cameraMatrix, distCoeffs)


            font = cv2.FONT_HERSHEY_SIMPLEX
            cv2.putText(image,'Translacao',(0,25), font,
                0.5,(255,255,255),2,cv2.LINE_AA)
            cv2.putText(image,str(tVec[0]),(0,50), font,
                0.5,(255,255,255),2,cv2.LINE_AA)
            cv2.putText(image,str(tVec[1]),(0,75), font,
                0.5,(255,255,255),2,cv2.LINE_AA)
            cv2.putText(image,str(tVec[2]),(0,100), font,
                0.5,(255,255,255),2,cv2.LINE_AA)

            cv2.putText(image,'Rotacao',(120,25), font,
                0.5,(255,255,255),2,cv2.LINE_AA)
            cv2.putText(image,str(yaw),(120,50), font,
                0.5,(255,255,255),2,cv2.LINE_AA)
            cv2.putText(image,str(pitch),(120,75), font,
                0.5,(255,255,255),2,cv2.LINE_AA)
            cv2.putText(image,str(roll),(120,100), font,
                0.5,(255,255,255),2,cv2.LINE_AA)

    except:
        print("cubo nao encontrado")

    t2 = time.time()
    print(1/(t2-t1))
    cv2.imshow("Frame", image)
    key = cv2.waitKey(1) & 0xFF
    rawCapture.truncate(0)
    if key == ord("q"):
        break
```

```
if __name__=='__main__':
    main()
```