TRABALHO DE GRADUAÇÃO

DESENVOLVIMENTO DE MÓDULO DE TRIANGULAÇÃO PARA RECONSTRUÇÃO 3D DE SUPERFÍCIES COM VISÃO COMPUTACIONAL

Lucas Fernandes das Neves

Brasília, Dezembro de 2020

ENGENHARIA
MECATRÔNICA
UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASILIA
Faculdade de Tecnologia
Curso de Graduação em Engenharia de Controle e Automação

TRABALHO DE GRADUAÇÃO

DESENVOLVIMENTO DE MÓDULO DE TRIANGULAÇÃO PARA RECONSTRUÇÃO 3D DE SUPERFÍCIES COM VISÃO COMPUTACIONAL

Lucas Fernandes das Neves

Relatório submetido como requisito parcial de obtenção de grau de Engenheiro de Controle e Automação

Banca Examinadora

Prof. Jones Yudi, ENM/UnB _____
Orientador

Prof. Gerardo Antonio Idrobo Pizo, FGA/UnB _____
Examinador interno

Prof. Carlos Humberto Llanos Quintero, ENM/UnB _____
Examinador interno

Brasília, Dezembro de 2020

FICHA CATALOGRÁFICA

LUCAS F., DAS NEVES
Desenvolvimento de Módulo de Triangulação para Reconstrução 3D de Superfícies com Visão Computacional,

[Distrito Federal] 2020.

ix, 80p., 297 mm (FT/UnB, Engenheiro, Controle e Automação, 2020). Trabalho de Graduação – Universidade de Brasília. Faculdade de Tecnologia.

I. Mecatrônica/FT/UnB

REFERÊNCIA BIBLIOGRÁFICA

DAS NEVES, LUCAS F., (2020). Desenvolvimento de Módulo de Triangulação para Reconstrução 3D de Superfícies com Visão Computacional. Trabalho de Graduação em Engenharia de Controle e Automação, Publicação FT.TG-$n°$022, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF, 80p.

CESSÃO DE DIREITOS

AUTOR: Lucas Fernandes das Neves

TÍTULO DO TRABALHO DE GRADUAÇÃO: Desenvolvimento de Módulo de Triangulação para Reconstrução 3D de Superfícies com Visão Computacional.

GRAU: Engenheiro ANO: 2020

Lucas Fernandes das Neves

Cond. São José, Conj. A, Casa 9, Setor Contagem, Sobradinho.

73090-925 - Brasília, DF – Brasil.

Agradecimentos

Agradeço primeiramente aos meus pais, Carlos e Graziela, que sempre me apoiam e se dedicam incondicionalmente ao meu desenvolvimento acadêmico e pessoal. Agradeço também aos meus irmãos, em especial a minha irmã Mariana, que sempre apoiou todas as decisões que tomei na vida.

Quero agradecer também ao meu grande amigo Pedro Leoncio, que sempre esteve presente em todas as minhas dificuldades e sempre está disposto a me ajudar. Além disso, ao Thiago Hirano, meu sócio nos principais projetos em que me envolvo e um grande amigo, além de ser um coautor neste trabalho.

Por fim, quero agradecer ao meu orientador Jones Yudi, que sempre esteve pronto para responder minhas dúvidas rapidamente e sempre com grande paciência em relação ao trabalho.

Lucas Fernandes das Neves

RESUMO

Inspeção e medição de superfícies de geometria complexa é uma grande área de estudo devido à dificuldade na obtenção de dados de forma manual. Uma aplicação na visão computacional que está constantemente em pesquisa é a reconstrução 3D de objetos e superfícies através da obtenção de sua nuvem de pontos para reconhecimento e análise.

A reconstrução tridimensional é um processo complexo devido à falta de conectividade entre os pontos na nuvem gerada pela inspeção do objeto e à presença de ruídos na obtenção da amostra. Estes problemas podem ser resolvidos com a aplicação de técnicas de filtragem e de algoritmos que definem relações de conectividade entre os pontos.

Este trabalho tem como objetivo dissertar e definir técnicas para reconstrução de superfícies por meio da inspeção à laser por um módulo de triangulação, a obtenção de uma nuvem de pontos e sua concatenação, e formação de vértices e faces para a construção de um objeto tridimensional digital. Uma estrutura de *hardware* de baixo custo e robusto para obtenção da nuvem de pontos da superfície escaneada é desenvolvida, assim como o uso de técnicas de filtragem com visão computacional para a redução de ruídos e distorções nas imagens capturadas.

O projeto utiliza técnicas de visão computacional para a segmentação das imagens e para definição da nuvem de pontos, utiliza a técnica de *Hough Transform* para detecção da reta de referência (feixe do laser sobre a base do scanner), e *Marching Cubes Algorithm* para obter os vértices e as faces que definem um modelo digital tridimensional.

Este trabalho é capaz de reconstruir um modelo 3D escaneado, gerando um arquivo que pode ser renderizado em diversos *softwares* de visualização 3D. O projeto poderá ser usado para diferentes finalidades, como por exemplo: inspeção de linhas de transmissão de energia, de edificações e de fundo de barragens.

Palavras-chave: Reconstrução 3D, Escaneamento 3D, Visão Computacional, Módulo de Triangulação.

ABSTRACT

Inspection and measurement of complex geometry surfaces is a large area of study due to the difficulty in obtaining data manually. An application in computer vision that is constantly being researched is the 3D reconstruction of objects and surfaces by obtaining their point cloud for recognition and analysis.

Three-dimensional reconstruction is a complex process due to the lack of connectivity between points in the cloud generated by object inspection and the presence of noise in obtaining the sample. These problems can be solved by applying filtering techniques and algorithms that define connectivity relationships between points.

This paper aims to discuss and define techniques for surface reconstruction through laser inspection by a triangulation module, the acquisition of a point cloud and its concatenation, and formation of vertices and faces for the construction of a digital three-dimensional object. A low cost and robust Hardware structure for obtaining the point cloud of the scanned surface is developed, as well as the use of computer vision filtering techniques to reduce noise and distortions in the captured images.

The project uses computer vision techniques for image segmentation and point cloud definition, uses the Hough Transform technique for detection of the reference line (laser beam over the base of the scanner), and Marching Cubes Algorithm to obtain the vertices and faces that define a three-dimensional digital model.

This work is able to reconstruct a scanned 3D model, generating a file that can be rendered in several 3D visualization softwares. The work can be used for different purposes, such as: inspection of power transmission lines, buildings and dam bottoms.

Keywords: 3D Reconstruction, 3D Scanning, Computer Vision, Triangulation Module.

SUMÁRIO

LISTA DE FIGURAS

Lista de Tabelas

LISTA DE SÍMBOLOS

Símbolos Latinos

x,y,z	eixos de coordenadas em pixels da imagem
X,Y,Z	eixos de coordenadas em milímetros do espaço real
n	constante que inclui a ampliação ótica de lentes da câmera e o conversor de milímetro para pixels na captação de uma câmera
k_1,k_2,k_3	coeficiente de distorção radial
p_1,p_2	coeficiente de distorção tangencial
f_x,f_y	distância focal
c_x,c_y	centro óptico
P	matriz 3x4 de projeção
K	matriz de extrínsecos
R	matriz de rotação 3x3
t	vetor de translação
h	matriz de coeficientes de filtros lineares
f	função
g	matriz de pixels gerada por filtros lineares
m	coeficiente angular de uma reta
b	coeficiente linear de uma reta
fps	taxa de frames por segundo
v	velocidade do deslocamento da estrutura câmera/laser

Símbolos Gregos

Δ	variação e/ou dimensão
α	ângulo em graus do laser em relação à câmera
∇	cálculo vetorial
θ, ρ	coeficientes no sistema polar de coordenadas

Abreviações

ASME	*American Society of Mechanical Engineers*
RGB	Sistemas de canais de imagens digitais. R (*Red*), G (*Green*), B (*Blue*)

Capítulo 1

Introdução

Este capítulo descreve a contextualização e motivação, e apresenta os objetivos do trabalho realizado. A estrutura do manuscrito é apresentada ao final do capítulo.

1.1 Contextualização e Motivação

A necessidade da digitalização de objetos e estruturas físicas só cresce com o passar dos anos. Hoje, modelos digitalizados tridimensionais são praticamente fundamentais para uma inspeção detalhada de alguma entidade física. A análise de superfícies, como por exemplo em edifícios, pontes, viadutos e barragens, pode ser crucial para evitar custos futuros, replanejamento e até mesmo desastres.

Por isso, justifica-se a busca constante do aperfeiçoamento de algoritmos e de hardware para se obter um baixo custo operacional, melhor desempenho e um custo computacional baixo em inspeções. Uma das técnicas que podem ser empregadas para auxiliar essas inspeções é a reconstrução 3D.

A reconstrução 3D pode ser explicada como a modelagem de características geométricas e da forma de objetos a partir de imagens com o intuito de analisá-los e descrevê-los por meio de modelos matemáticos. É uma tecnologia complexa que está associada a diversos fatores. Dentre esses fatores estão:

> Os requisitos da aplicação alvo, como exatidão geométrica, realismo visual, ou rapidez na aquisição do modelo; a escolha de uma entre muitas opções de digitalização; adversidades como a complexidade topológica inerente a superfícies de forma livre; a degradação da qualidade dos dados sensoriados por medições errôneas; a solução do difícil problema da correspondência e a indisponibilidade de informações importantes para a reconstrução (Albuquerque, 2006).

Com o advento da visão computacional, a construção e o desenvolvimento de um sistema de digitalização tridimensional de superfícies torna-se mais eficiente, com custos relativamente reduzi-

dos devido à utilização de menos sensores, e com um menor custo computacional. Hoje, o processo de reconstrução pode ser feito por meio da utilização de imagens que representam a geometria tridimensional do objeto (nuvem de pontos), que são obtidas normalmente por equipamentos sem a necessidade de contato.

1.2 Objetivos

1.2.1 Objetivo Geral

O objetivo geral do projeto é o estudo e desenvolvimento de um sistema de módulo de triangulação. O projeto deve ser capaz de preparar e obter uma nuvem de pontos que representam características geométricas de uma superfície e, a partir dela, gerar vértices e faces para a construção de um objeto tridimensional digital.

1.2.2 Objetivos Específicos

O objetivo geral abrange os seguintes objetivos específicos:

- construção do *scanner* baseado em um módulo de triangulação;
- calibração da câmera para remoção de distorções radiais e tangenciais;
- aplicação de filtros e segmentação das imagens captadas pela câmera para uma melhor detecção do feixe gerado pelo laser;
- obtenção da reta de referência que define o feixe do laser sem interferência do objeto a ser escaneado;
- construção da nuvem de pontos no espaço tridimensional;
- construção da malha, formada por triângulos e vértices, que representa a superfície do objeto; e
- reconstrução 3D a partir da malha criada.

1.3 Metodologia

Os meios utilizados para alcançar os objetivos definidos deste projeto basearam-se no estudo e revisão bibliográfica de técnicas mais populares utilizadas no ramo de visão computacional.

O primeiro objetivo específico é a construção de um *scanner* capaz de captar imagens da influência de um laser sobre a superfície de um objeto. O scanner foi desenvolvido em parceria com um projeto de graduação de engenharia mecânica da Universidade de Brasília (Hirano, 2020).

Como o projeto tinha como objetivo a construção de um scanner de baixo custo, técnicas de impressão 3D, corte a laser e outras técnicas de fabricação digital de baixo custo foram utilizadas.

O tratamento das imagens obtidas pelo *scanner* desenvolvido, com o intuito da obtenção de um modelo digital com características geométricas que se aproximam ao objeto real escaneado, é fundamental para atingir o objetivo geral deste trabalho.

A Figura 1.1 apresenta um diagrama de blocos com os passos que foram aplicados no tratamento das imagens e reconstrução 3D.

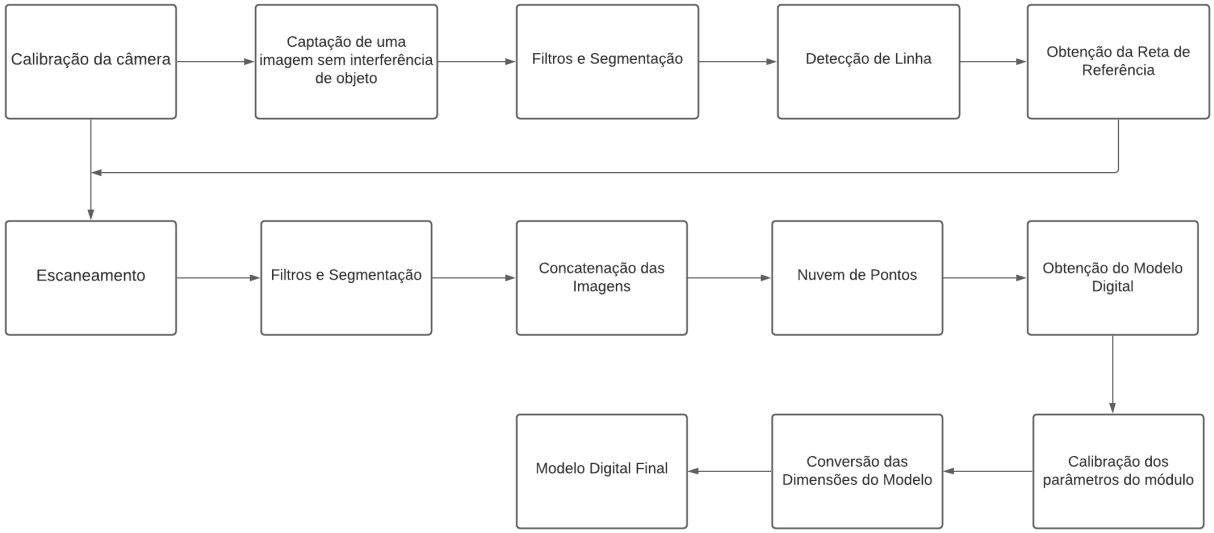


Figura 1.1: Diagrama de blocos do projeto.

O tratamento começa pela calibração da câmera do *scanner* para remover possíveis distorções, geradas pela lente da câmera, nas imagens.

Depois de feita a calibração, uma imagem foi captada da câmera sem a influência de nenhum objeto dentro do scanner. Essa imagem foi captada para que fosse possível obter a reta que representa a linha do laser sobre a base da estrutura, ou seja, a posição do laser quando não há nenhum objeto presente.

Então, filtros e segmentações foram aplicadas sobre a imagem para que apenas o laser seja representado na imagem de resultado. O filtro *Gaussian Filter* foi aplicado, por ser um filtro linear que remove ruídos e preserva bem os contornos na imagem. Depois disso, é aplicado o *Canny Filter*, que segmenta a imagem de forma que apenas os contornos ficam na imagem de resultado.

Sobre a imagem de resultado, foi aplicado então o *Hough Transform* para detecção de linhas. *Hough Transform* foi utilizado neste trabalho por ser uma técnica computacionalmente eficiente e por ser a técnica mais popular no ramo de visão computacional para detecção de linhas em imagens. Por meio desta técnica foi obtida então a reta de referência, que representa a altura 0 do *scanner*.

Com a reta de referência definida, o processo de escaneamento pode ser iniciado. A máquina então percorre objeto captando várias imagens que representam a superfície do objeto. Em cada imagem obtida, são aplicados filtros e segmentações para que apenas o laser esteja presente nas imagens de resultado.

Com o laser bem definido em todas as imagens, fez-se então a concatenação das mesmas, para que um representação tridimensional da superfície do objeto seja possível. Feita a concatenação, tem-se então a nuvem de pontos do objeto.

Aplicou-se então sobre a nuvem de pontos, o *Marching Cubes Algorithm* que é um algoritmo para definir as vértices e faces que representam o modelo digital do objeto escaneado. Esse algoritmo foi utilizado, pois o autor deste projeto já possui uma grande familiaridade na sua utilização em outros projetos.

Com isso tem-se o modelo digital definido, porém as dimensões do modelo ainda não estão corretas. É necessário que os parâmetros de conversão de pixels para milímetros em cada eixo no sistema de coordenadas do modelo sejam obtidos. Para isso, escaneou-se dois objetos de dimensões já conhecidas para o processo de calibração. Com isso, foi possível obter os parâmetros.

Ao final do processo de reconstrução, os parâmetros obtidos no processo de calibração são então utilizados para a conversão das dimensões do modelo digital. Com isso, é formado o resultado final do modelo digital.

1.4 Estrutura do Documento

Este documento é constuído por 7 capítulos, as referências bibliográficas e os apêndices de códigos utilizados no decorrer do desenvolvimento do projeto.

O primeiro capítulo apresenta a contextualização, motivação, objetivos e metodologia do trabalho realizado.

O segundo capítulo consiste na fundamentação teórica do trabalho. É feita a revisão bibliográfica da literatura já existente acerca do assunto, a descrição do *hardware* utilizado, a descrição das ferramentas e algoritmos, além das bibliotecas e linguagens de programação utilizadas.

O terceiro capítulo apresenta os materiais utilizados na construção do *scanner* de triangulação a laser, tanto sua parte mecânica, quanto os eletrônicos utilizados.

O quarto capítulo apresenta técnicas que foram utilizadas para o otimização das imagens captadas pela câmera do *scanner*. Explica como filtros e segmentações foram aplicadas para que a construção da nuvem de pontos fosse possível.

O quinto capítulo contêm a explicação do processo de reconstrução 3D de superfícies que foi aplicado. Nesse capítulo também é explicada a formação da nuvem de pontos no espaço tridimensional, além da aplicação do algoritmo *Marching Cubes* sobre esta nuvem para construção da malha de triângulos e vértices que define um modelo digital tridimensional. Ao final do capítulo a calibração dos parâmetros do módulo de triangulação é realizada.

A sexto capítulo apresenta os resultados da reconstrução 3D de superfícies de alguns corpos de prova. Apresenta também uma análise desses resutados, como também a precisão obtida no final do projeto.

O sétimo capítulo apresenta a conclusão do projeto, assim como os possíveis trabalhos que podem complementar o desenvolvimento do que aqui foi proposto.

Capítulo 2

Fundamentos

Este capítulo tem como objetivo mostrar toda a fundamentação teórica na qual o trabalho foi baseado. É feita a revisão bibliográfica da literatura já existente acerca do assunto.

Primeiramente, uma definição geral sobre digitalizadores tridimensionais é contextualizada, mostrando os principais tipos que são utilizados no mercado. Depois disso, uma explicação acerca do funcionamento padrão de um módulo de triângulação a laser. Além disso, definem-se aqui diferentes técnicas de visão computacional para segmentação de imagens e remoção de ruídos obtidos pelo *scanner* de triângulação a laser.

Ao final do capítulo, é feita uma explicação breve sobre a biblioteca OpenCV e a descrição do *Marching Cubes Algorithm*, que é utilizado para reconstrução 3D.

2.1 Digitalizadores Tridimensionais

A digitalização tridimensional é realizada normalmente pela obtenção de dados do objeto analisado e posteriormente pela construção de um modelo tridimensional. Os dados fornecidos pelo digitalizador são sempre uma série de dados que são transformados em coordenadas de pontos que podem estar no formato (FREITAS, 2006):

- $[x, y, z]$;
- $[x, y, z, c]$: quando podem obter outros parâmetros como cor ou temperatura;
- $[x, y, z, i, j, k]$: quando incluem a inclinação ou orientação do digitalizador.

As técnicas de digitalização 3D podem ser classificadas, de maneira geral, em duas categorias principais: sistemas com contato ou sistemas sem contato com o objeto a ser digitalizado (ZÚÑIGA, 2013).

Em sistemas de digitalização com contato, as coordenadas de pontos são obtidas pelo deslocamento de um sensor sobre a superfície a ser digitalizada. Este processo é mais antigo, usado

principalmente para o controle de qualidade de peças (ZÚÑIGA, 2013). Em sistemas como esse, o objeto analisado deve possuir uma rigidez considerável para que a sua deformação não ocorra no contato. Com o tempo, esses digitalizadores, por serem de contato, podem apresentar desgaste.

Um exemplo de sistemas com contato são as Máquinas de Medir Coordenadas, conhecidas como MMC (*Coordinate Measuring Machine*). Essas máquinas funcionam com o deslocamento de três guias ortogonais sobre a peça, obtendo as coordenadas X,Y,Z dos pontos sobre a superfície do objeto (ZÚÑIGA, 2013).

Segundo a Norma Americana ASME B89. 4.1(1995) são onze os tipos de MMC:

- braço horizontal como mesa fixa;
- cantilever (braço em balanço) como mesa móvel;
- braço horizontal com mesa móvel;
- ponte L;
- cantilever (braço em balanço) com mesa fixa;
- ponte fixa;
- coluna;
- ponte móvel;
- braço horizontal;
- gantry (ou pórtico); e
- modo Duplex.

Os sistemas sem contato podem ser classificados em sistemas não-ópticos e ópticos. Sistemas não-ópticos de detecção podem ser formados por sensores acústicos, eletromagnéticos e outros. Em sistemas de sensores eletromagnéticos, por exemplo, a medição de distâncias ocorre através da medição do tempo em que a radiação emitida retorna do objeto (ZÚÑIGA, 2013).

Em sistemas sem contato que utilizam sensores ópticos 3D, são várias as formas de obter os dados de coordenadas de uma superfície de objeto. Geralmente, o sistema projeta sobre o objeto uma energia eletromagnética (forma ativa) ou adquire uma forma de energia eletromagnética do objeto (forma passiva), e então grava a energia transmitida ou refletida (Sansoni et al., 2009). Nestes sistemas, a luz é a responsável por carregar a informação de medição.

Segundo Sansoni, algumas das técnicas que mais se mostram eficientes nesse setor são:

- Triangulação a laser - é baseada no princípio de triangulação ativa. A técnica de triangulação a laser está baseada na iluminação de uma peça com um padrão laser que pode ser uma linha laser, ou um ponto laser, ou um plano laser. A luz projetada da superfície do corpo produz um único padrão que depende da forma e das dimensões do corpo (ZÚÑIGA, 2013);

- Luz estruturada: sensores baseados em luz estruturada compartilham a abordagem de triangulação ativa mencionada acima. Contudo, ao invés de escanear a superfície, eles projetam padrões bidimensionais de luz, e os elabora para obter as informações de alcance para cada ponto visualizado simultaneamente. As projeções sobre o objeto constumam ser formados por planos geométricos. As projeções de padrão de grade, de pontos, de múltiplas linhas verticais e múltiplas cores foram amplamente estudadas (Sansoni et al., 2009);

- Visão Estéreo - duas ou mais câmeras são posicionadas para que capturem simultaneamente o mesmo objeto, porém de diferentes ângulos. A reconstrução 3D por visão estéreo é divida nas seguintes etapas: aquisição das imagens; modelagem da câmera; extrações de características; análise de correspondências de pontos entre as imagens; e triangulação. As vantagens são a simplicidade e o baixo custo. Porém, existe o problema complexo de identificação de pontos comuns entre as imagens (Sansoni et al., 2009);

- Fotogrametria - é uma técnica que está em constante evolução no mercado, e se baseia na obtenção da nuvem de pontos por meio de um conjunto de fotografias digitalizadas do objeto. Como as posições das câmeras não são conhecidas, é necessário um processo interativo de indicação da correspondência entre os pontos das diferentes imagens utilizadas (FREITAS, 2006). O processo segue os seguintes passos: calibração e orientação da câmera; medidas de ponto na imagem; geração da nuvem de pontos; geração de superfície; e mapeamento de textura (Sansoni et al., 2009). As vantagens dessa técnica são o baixo custo na operação, a possibilidade de digitalização de modelos de grande escala e a facilidade na obtenção de modelos poliédricos texturizados. Porém, existe uma dificuldade de digitalizar aberturas profundas e concavidades, além de uma grande incerteza dimensional no resultado (FREITAS, 2006).;

- Time of Flight - o emissor gera um pulso de laser que colide com a superfície do objeto escaneado. O receptor detecta o pulso refletido, e o sensor computa então o tempo de viagem (*Time of Flight*) do sinal e sua intensidade. Sensores pontuais realizam a medição de distâncias ponto a ponto. Para medições de grandes objetos sensores são combinados (Sansoni et al., 2009).

Em virtude da necessidade de certeza dimensional no processo e de inspeção de superfícies amplas com alto nível de detalhes, o presente trabalho utilizou-se da técnica de triangulação a laser, que é melhor detalhada no próximo tópico deste documento.

2.2 Módulo de Triangulação

Método de triangulação é uma técnica de visão computacional usada para capturar medidas tridimensionais usando a luz refletida da superfície do alvo desejado.

Uma configuração comumente utilizada é representada na Figura 2.1.

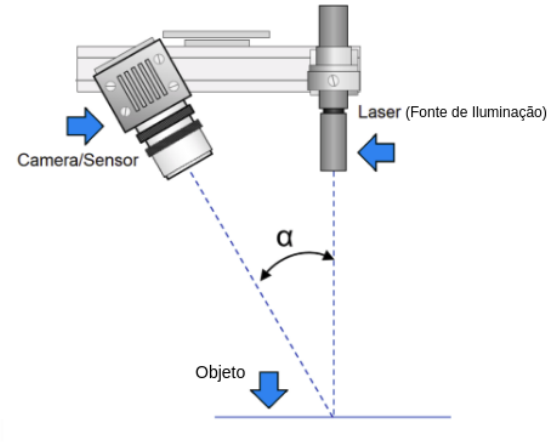


Figura 2.1: Configuração do módulo de triangulaçao (Movimed, 2018 - modificada).

Como pode ser observado, o sistema é formado por uma câmera e uma fonte de iluminação a laser. O feixe de laser e a câmera são ambos direcionados para o alvo de inspeção. Adotando um *offset* angular conhecido (α) entre a fonte laser e a câmera, é possível medir diferenças de profundidade usando trigonometria (Movimed, 2018).

Sensores que utilizam o método de triangulação a laser são frequentemente empregados para detectar um componente e para contagem de objetos (Azosensors, 2014). Eles são sensores mais acessíveis se comparados com outras tecnologias de alta performance.

Uma outra configuração possível para o método de triangulação é com a presença de múltiplas fontes de laser. Como apresentada na Figura 2.2.

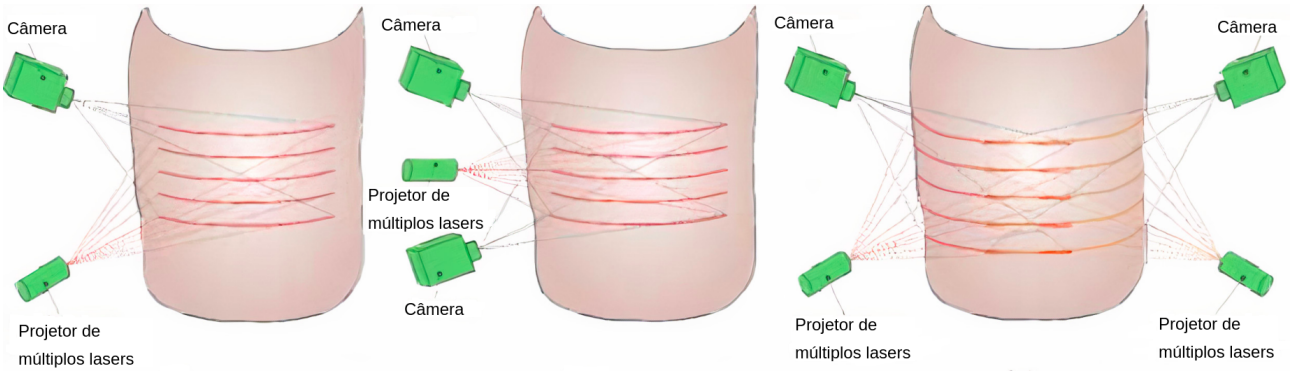


Figura 2.2: Configurações do módulo de triangulação com múltiplos feixes (Poredoš et al, 2015 - modificada).

A Figura 2.2 mostra a presença de diferentes métodos de escaneamento por meio de múltiplas iluminações da superfície. Nessa configuração, o laser ilumina a superfície com múltiplos feixes, enquanto a câmera capta a iluminação refletida de diferentes ângulos. Com isso, a luz captada pela câmera possui o formato da superfície analisada com uma certa precisão, dependendo da

quantidade de lasers e sua orientação. Esse processo torna mais rápido o escaneamento do objeto, pois a superfície inteira é iluminada simultaneamente sem a necessidade de deslocamento da peça.

Porém, existe uma outra configuração, a qual foi utilizada por este trabalho. Ela consiste na presença de apenas um laser e uma câmera que se deslocam com o intuito de percorrer toda a superfície do objeto. A estrutura câmera/laser é comumente acoplada a um motor que, por meio de uma correia dentada, possibilita tal deslocamento. Com isso, o projeto se torna de menor custo se comparado à configuração de múltiplos feixes. A Figura 2.3 representa a configuração utilizada no presente trabalho, em que a estrutura câmera/laser se desloca paralelamente ao eixo Zr do objeto.

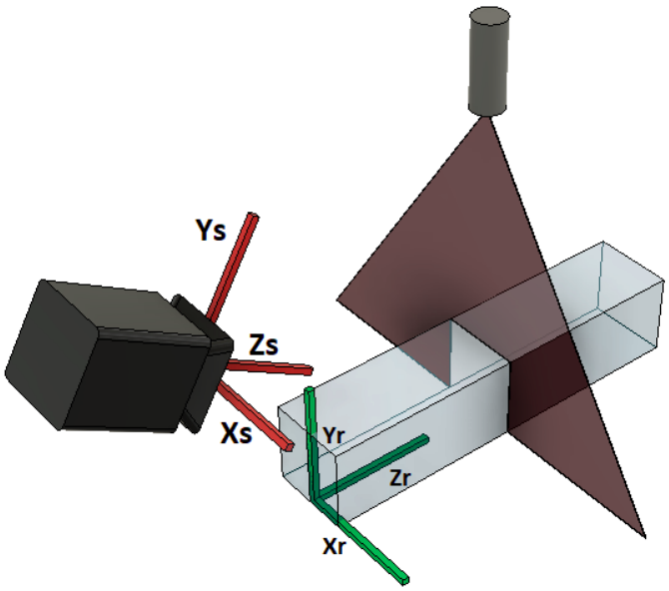


Figura 2.3: Representação tridimensional de módulo de triangulação utilizado neste trabalho.

Como pode ser visto na Figura 2.4, a expressão básica, usando o método de triangulação de configuração simples, para o cálculo da translação da luz refletida captada $\Delta(y)$ pela câmera é (Pears et al, 2012):

$$\Delta(y) = \Delta(Y) \times n \times sin(\alpha), \tag{2.1}$$

em que $\Delta(y)$ representa a variação real no plano vertical do objeto, n representa a ampliação ótica das lentes da câmera e α, como explicado no início desta seção, é o ângulo entre a fonte laser e a câmera.

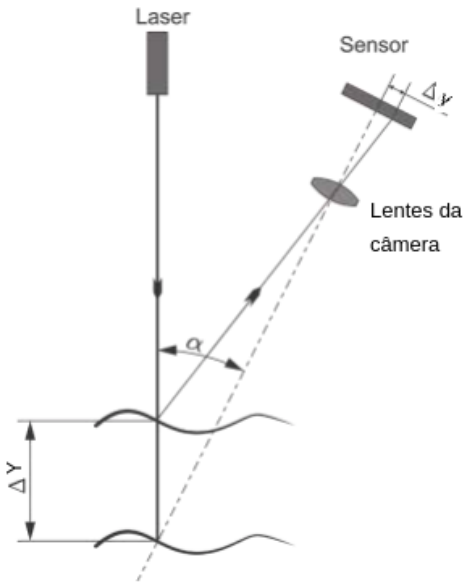


Figura 2.4: Representação básica do módulo de triangulação (Poredoš et al, 2015 - modificada).

O valor de $\Delta(y)$ pode ser obtido pela aplicação de técnicas de visão computacional sobre as imagens obtidas pela câmera no sistema de triangulação a laser. Com o intuito da formação de uma nuvem de pontos que representa a superfície do objeto, cada ponto do laser sobre o objeto na imagem é convertido para o seu valor no sistema de coordenadas do objeto com o uso da equação 2.1.

2.3 Visão Computacional no Processamento de Imagens

Para tentar gerar uma melhor compreensão do processamento de imagens, GONZALES e WOODS (2002) definiram três níveis de processamento:

- Processamento de nível baixo - neste nível o processo envolve, além da obtenção das imagens, o seu aprimoramento com o uso de filtros, realce e outras formas de pré-processamento da imagem (GONZALEZ e WOODS, 2002). Dentre as particularidades que devem ser levadas em conta nessa tarefa estão a escolha do meio de capturar a imagem, as cores, níveis de cinza e condições do ambiente (PEDRINI e SCHWARTZ, 2008; GONZALEZ e WOODS, 2002; GHELLERE, 2015). O pré-processamento das imagens é responsável por operações primitivas para redução ou inclusão de ruídos, realce e restauração da imagem para que a mesma possa se enquadrar melhor no domínio do problema, tendo ao final apenas resultados com relação a qualidade da imagem (CONCI et al., 2008; GHELLERE, 2015).
- Processamento de nível médio - neste nível, o processo envolve diferentes tarefas. Entre elas, a segmentação, descrição e o reconhecimento (GHELLERE, 2015). O processo de segmentação consiste em identificar regiões ou objetos, que sejam pertinentes ao domínio do problema e separá-los da imagem, com base em características como bordas e texturas, por

exemplo (PEDRINI e SCHWARTZ, 2008). O próximo passo neste nível, é a caracterização de detalhes da imagem, com a obtenção de objetos e regiões de interesse, normalmente representados por vetores com atributos numéricos (PEDRINI e SCHWARTZ, 2008).

- Processamento de nível alto - neste nível, o processo envolve, enfim, a determinação de objetos presente na imagem, com sua comparação, detecção e posição na imagem. Porém, é possível pensar, além dessas tarefas, como classificar imagem baseando-se em conceitos abstratos, como pacificidade (CIPOLLA et al., 2012;GHELLERE, 2015). Tarefas como esta estão ligadas com a área de cognição e de Inteligência Artificial. Elas buscam dar sentido ao que a máquina vê, ao mesmo tempo que tentam emular a inteligência humana, de modo a possibilitar que a mesma aprenda, faça inferências e tire conclusões com base no que está enxergando no ambiente (GONZALEZ e WOODS, 2002; GHELLERE, 2015).

As características de uma imagem podem ser extraídas a partir de informações locais ou globais. Características globais tem a propriedade de generalizar um objeto inteiro com apenas um vetor (LISIN et al., 2005) e geralmente algoritmos que extraem informações globais tendem a ser mais rápidos e simples (PENATTI, 2009) e com boa tolerância a ruídos (DE ARAÚJO, 2009). Alguns exemplos de características globais extraídas de uma imagem pode ser: cor (RAOUI et al., 2011) geralmente extraídas por meio de um histograma (PENATTI, 2009), textura (LIBERMAN, 1997) e forma (DESELAERS, 2003; LISIN et al., 2005; RAOUI et al., 2011; GHELLERE, 2015).

Mas para realizar a correspondência para classificação a partir do reconhecimento de um objeto (ou vários) é preciso descrever as características detectadas (extraídas) do conjunto de pixels (ou pontos de interesse), geralmente representadas por um conjunto de números (vetor) que representam as características do objeto, de modo que seja possível classificar uma imagem (NIXON e AGUADO, 2002).

Por isso, é fundamental um pré-processamento em nível baixo para ter um resultado satisfatório. Tanto o controle do ambiente (luz, posição, hardware), como a determinação das características globais a serem capturadas e processadas.

As características globais a serem obtidas são determinadas pelo uso do módulo de triangulação. É, então, necessário o controle do ambiente e dos objetos analisados para a captura da cor vermelha (laser) pela câmera. Por isso é fundamental o uso de filtros no desenvolvimento do projeto para que apenas a luz do laser refletida seja processada.

A segmentação é necessária para separar a imagem em setores onde a cor vermelha está presente. A segmentação da imagem pode ser feita de duas formas:

- Descontinuidade - as alterações bruscas de intensidade são usadas como base para a segmentação da imagem (ex., detecção de contornos); e
- Similaridade - A similaridade entre pixels, seguindo um determinado critério, é a base para a partição ser efetuada.

2.3.1 Calibração da Câmera

Na prática, não existem lentes perfeitas. Lentes, principalmente com o advento de câmeras de baixo custo, apresentam distorções que devem ser consideradas em projetos de visão computacional. São duas as principais distorções presentes: distorção radial e distorção tangencial.

Na distorção radial, as lentes de câmeras costumam distorcer consideravelmente a localização de pixels perto das bordas da imagem. Esse fenômeno é muito conhecido como "*fish-eye*" ou "olho de peixe" em português. A Figura 2.5 demonstra a razão porque a distorção ocorre. A luz mais afastada do centro da lente se curva mais do que a que incide ao centro.

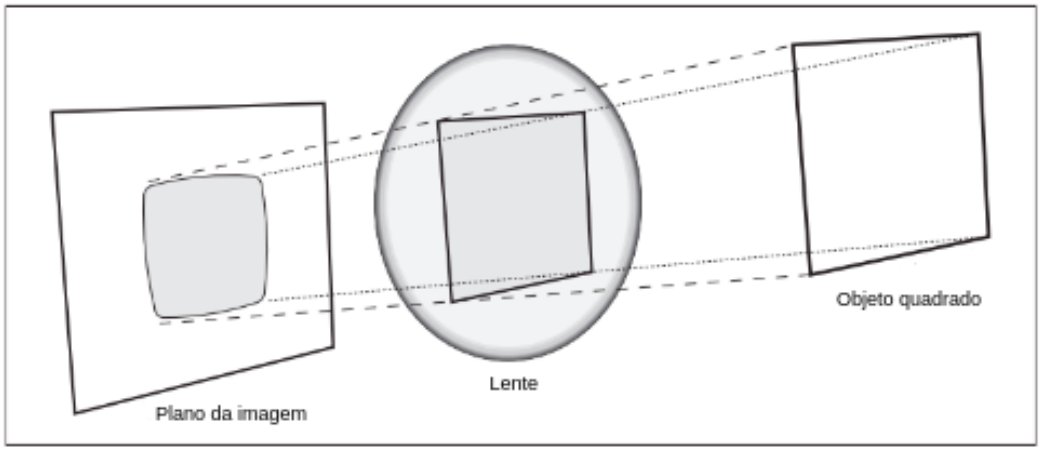


Figura 2.5: Representação da distorção radial em uma lente (BRADSKI E KAEHLER, 2008 - modificada).

A distorção radial pode ser resolvida pelas equações abaixo:

$$x_{\text{corrigido}} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6); \tag{2.2}$$

$$y_{\text{corrigido}} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6), \tag{2.3}$$

em que k_1, k_2 e k_3 são os coeficientes de distorção radial da lente, r é o módulo da distância do pixel (x, y) em relação ao centro da imagem, e $\text{x}_{\text{corrigido}}$ e $\text{y}_{\text{corrigido}}$ são as coordenadas dos pixels ajustados.

A distorção tangencial ocorre quando a lente não está paralela ao plano da imagem. A Figura 2.6 demonstra essa distorção.

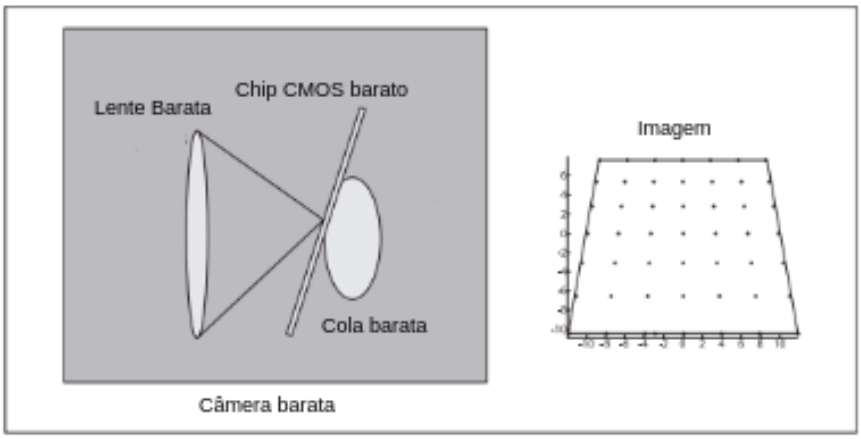


Figura 2.6: Representação da distorção tangencial em uma lente (BRADSKI E KAEHLER, 2008 - modificada).

A distorção tangencial pode ser resolvida pelas equações:

$$x_{\text{corrected}} = x + [2p_1xy + p_2(r^2 + 2x^2)]; \tag{2.4}$$

$$y_{\text{corrected}} = y + [p_1(r^2 + 2y^2) + 2p_2xy], \tag{2.5}$$

em que p_1 e p_2 são os coeficientes de distorção tangencial, r é o módulo da distância do pixel (x,y) em relação ao centro da imagem, e $\text{x}_{\text{corrigido}}$ e $\text{y}_{\text{corrigido}}$ são as coordenadas dos pixels ajustados.

Com isso, tem-se que os parâmetros a serem obtidos, conhecidos como coeficientes de distorção, são:

$$(k_1 \quad k_2 \quad p_1 \quad p_2 \quad k_3). \tag{2.6}$$

Além desses parâmetros é necessário também obter os parâmetros intrínsecos e extrínsecos da câmera.

Parâmetros intrínsecos são os parâmetros internos específicos da câmera. Incluem informações como: distância focal, centro óptico, distorção das lentes, etc. Podem ser modelados pela seguinte matriz:

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}, \tag{2.7}$$

onde f_x e f_y definem a distância focal, enquanto c_x e c_y definem o centro óptico.

Parâmetros extrínsecos são aqueles que representam a posição relativa da câmera em relação ao sistema de coordenadas do objeto. Correspondem às matrizes de rotação e aos vetores de

translação que relacionam as coordenadas de um ponto 3D ao sistema de coordenadas da imagem. A Figura 2.7 demonstra a conversão do sistema de coordenadas do objeto para a câmera. O ponto P no objeto é visto como o ponto p no plano da imagem; o ponto p está relacionado com o ponto P pela aplicação da matriz de rotação R e o vetor de translação t sobre P (BRADSKI E KAEHLER, 2008).

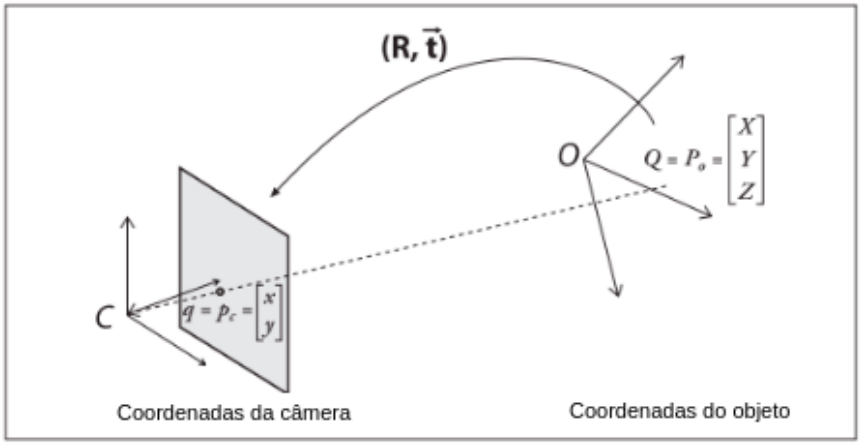


Figura 2.7: Conversão do sistema de coordenadas do objeto para a câmera (BRADSKI E KAEHLER, 2008 - modificada).

As equações que relacionam os pontos 3D nas coordenadas reais (X,Y,Z) para sua projeção em coordenadas de imagem (x,y) são mostradas abaixo:

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = P \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}; \tag{2.8}$$

$$x = x'n; \tag{2.9}$$

$$y = y'n; \tag{2.10}$$

$$P = K \times \Big[R|t\Big], \tag{2.11}$$

em que, P é a matriz 3x4 de projeção que consiste em duas partes: matriz de intrínsecos K e a matriz de extrínsecos ($[R|t]$) que é a combinação da matriz de rotação R (3x3) e o vetor de translação t (3x1).

Para encontrar todos os parâmetros, é necessário o uso de imagens de um padrão com características geométricas bem definidas e de fácil reconhecimento para algoritmos de visão computacional. É muito comum o uso de um *grid* que alterna entre quadrados pretos e brancos,

muito conhecido como “tabuleiro de xadrez" (Figura 2.8). Os dados de entrada necessários para a calibração da câmera são um conjunto de pontos 3D do mundo real (X,Y,Z) e seus pontos de imagem 2D (x,y) correspondentes. Os pontos 3D são chamados de pontos do objeto e os pontos 2D são chamados de pontos da imagem. As coordenadas dos vértices dos quadrados que formam o tabuleiro de xadrez são os pontos utilizados.

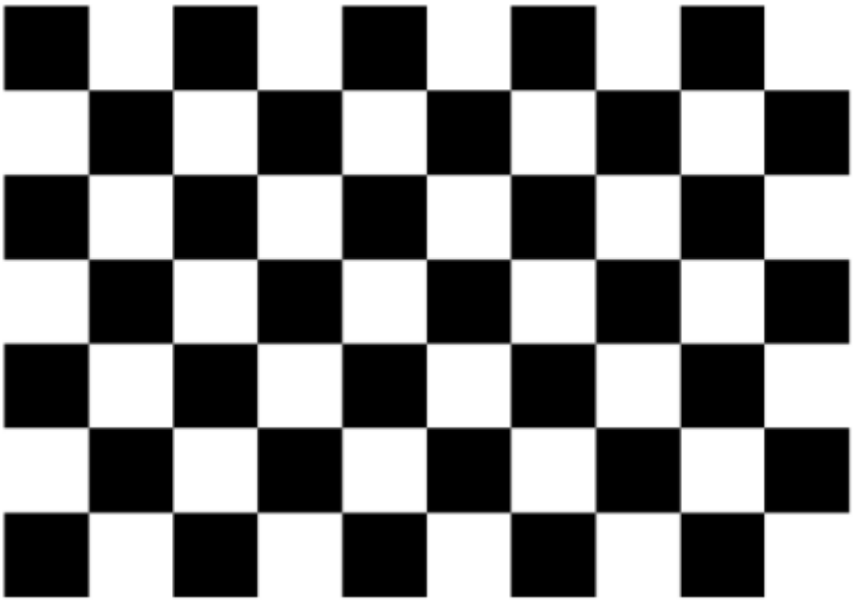


Figura 2.8: Tabuleiro de xadrez.

Os pontos da imagem podem ser obtidos facilmente analisando a imagem do tabuleiro. Porém é necessário também obter os pontos do objeto. Para isso, por simplicidade, pode-se considerar que o tabuleiro se manteve estático no plano XY (valor em Z é sempre 0) e a câmera se moveu adequadamente. Essa consideração facilita a obtenção dos valores X,Y. Então, os valores de X,Y podem ser definidos como (0, 0), (1, 0), (2, 0), etc, que representam a localização dos pontos[1]. O resultado final da calibração estará, então, em relação a escala do quadrado do tabuleiro de xadrez.

O tabuleiro deve ser captado pela câmera de diferentes ângulos para que possa fornecer informações suficientes para uma completa solução da localização dessas imagens no sistema de coordenadas em relação à câmera (Figura 2.9).

[1]Os valores reais em milímetros também podem ser passados para localização dos pontos, porém não é necessário.

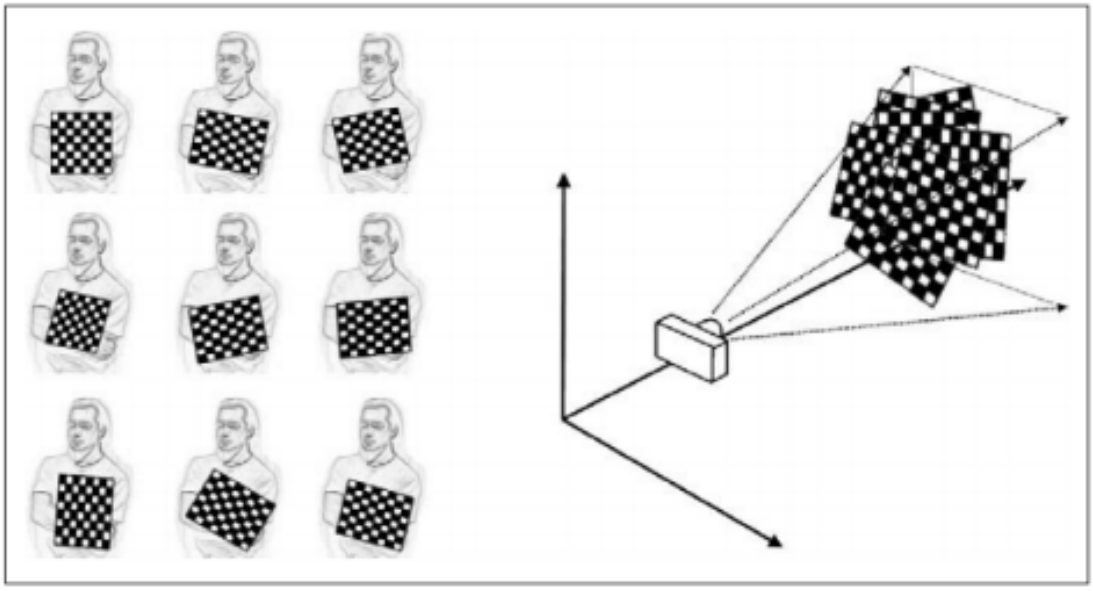


Figura 2.9: Processo de captação de diferentes ângulos do tabuleiro de xadrez pela câmera (BRADSKI E KAEHLER, 2008).

Com esses dados obtidos, são aplicados processos matemáticos com base nas equações 2.8, para se obter os coeficientes de distorção.

2.3.2 Histogramas para Segmentação

Segmentação é o processo de dividir imagens em consistentes regiões ou objetos. Essas regiões podem ser definidas por uma similaridade de seus pixels em relação a características, como cor, intensidade, textura, etc.

A construção de histogramas é um dos meios para se analisar uma imagem com o intuito de segmentá-la. Histograma de uma imagem fornece uma representação gráfica da distribuição da intensidade de pixel de uma imagem. O eixo horizontal representa os valores diferentes de cor, que podem estar entre 0 a 255, e o eixo vertical representa o número de vezes que um determinado valor de intensidade ocorre na imagem.

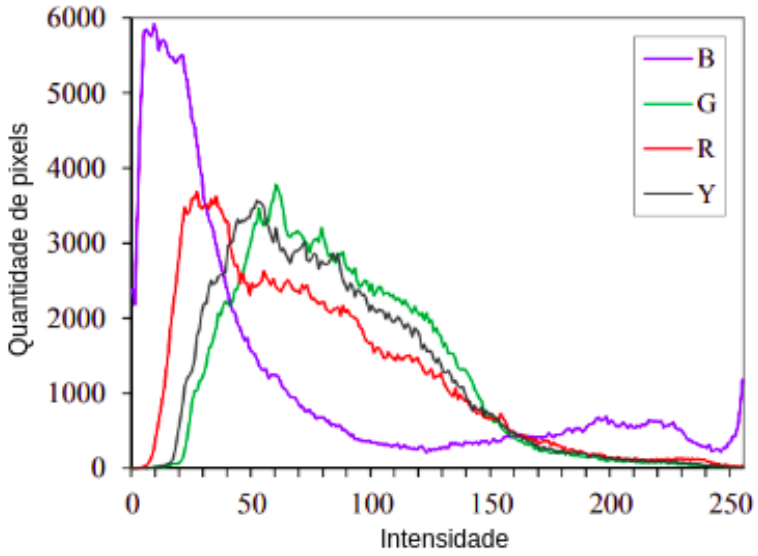


Figura 2.10: Exemplo de histograma com separação por canais RGB e Y (representação da imagem em apenas um canal) (DAVIES, 2018).

A Figura 2.10 apresenta um exemplo de histograma com a representação separada dos canais RGB[2] da imagem e a intensidade (brilho) dos pixels. Pelo histograma apresentado, analisando a curva de intensidade, pode-se dizer que a maioria dos pixels possui uma baixa intensidade e estão presentes na faixa de 0 a 50. A mesma análise pode ser feita para cada canal RGB, como por exemplo, o canal R, em que se percebe que boa parte dos pixels estão próximos ao valor 30.

Após uma análise do histograma, pode-se determinar a faixa de valores em que a segmentação deve ser feita. Na segmentação, cada pixel é comparado com um valor de *threshold*, ou seja, uma faixa de valores definida previamente. Se a intensidade do pixel é menor que o valor de *threshold*, então o pixel correspondente na imagem de saída é definido como 0. Se a intensidade do pixel é maior que o valor *threshold*, então o pixel correspondente na imagem de saída é definido como 1 (CHITYALA e PUDIPEDDI, 2014). A segmentação pode ser feita tanto para apenas um canal RGB, quanto para a imagem em escala de cinza ou até mesmo para imagem original.

2.3.3 Filtragem

Filtragem é uma ferramenta muito utilizada no processamento de imagem. Filtros são usados normalmente para remover ruídos ou impurezas indesejadas. Cada filtro possui uma utilidade específica e é projetado tanto para remover um tipo de ruído quanto para otimizar certos aspectos da imagem.

Os filtros normalmente são formados por uma matriz quadrada de duas dimensões que percorre uma imagem afetando apenas um pixel de cada vez. Esse processo de varredura de pixels e

[2]Sistema de cores em que o Vermelho (*Red*), o Verde (*Green*) e o Azul (*Blue*) são combinados de várias formas e intesidades para reproduzir um largo espectro cromático. Amplamente utilizado em imagens digitais.

sua substituição na imagem original, de acordo com o filtro utilizado, se chama convolução no domínio espacial (todos os pixels de uma imagem). Por isso, os filtros também são conhecidos como *convolution kernels*. Cada número do filtro é conhecido como coeficiente, que determina os efeitos sobre a imagem (CHITYALA e PUDIPEDDI, 2014).

Existem dois tipos de filtros: lineares e não-lineares. Entre os lineares, podem ser incluídos os filtros: *mean*, *Gaussian*, *Laplacian* e *Laplacian of Gaussian*. Entre os não-lineares estão: *median*, *maximum*, *minimum*, *Sobel*, *Prewitt* e *Canny*.

Nas formas mais comuns de filtros lineares, o valor de saída do pixel é determinada pela soma ponderada dos valores de entrada do pixel (SZELISKI, 2010),

$$g(i,j) = \sum_{k,l} f(i+k, j+l)h(k,l), \tag{2.12}$$

onde $h(k,l)$ representa os coeficientes do filtro e f representa os pixels da imagem. A Figura 2.11 mostra um exemplo desse processo.

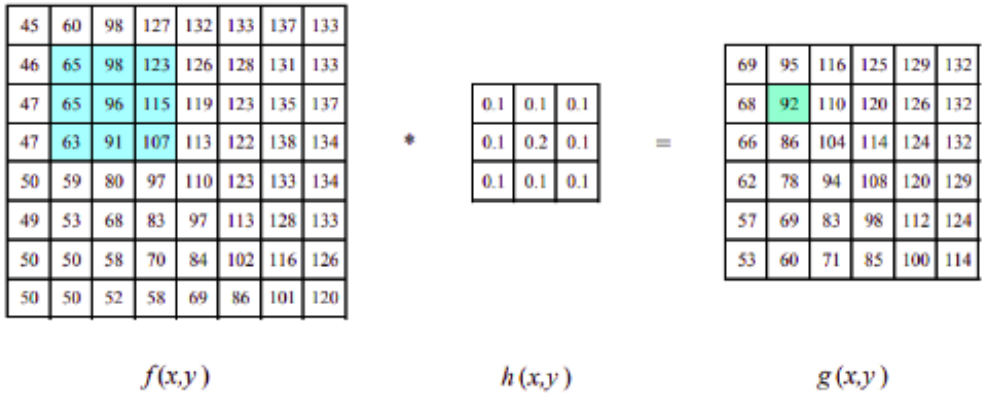
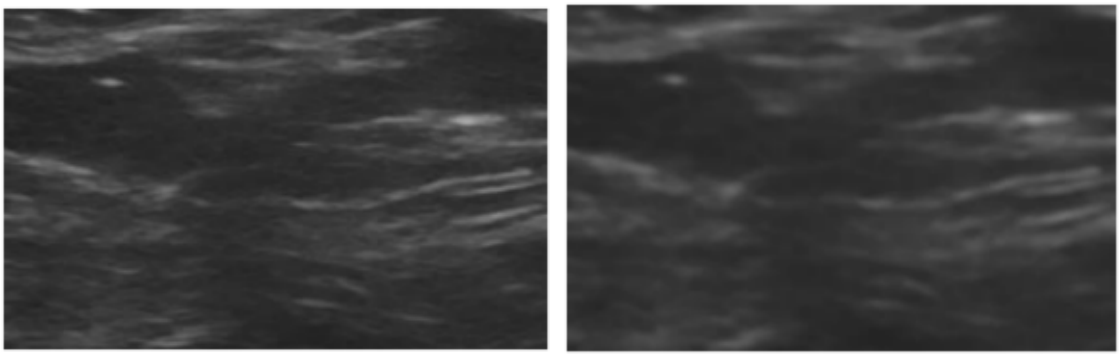


Figura 2.11: Processo de Convolução - A imagem na esquerda passa pelo processo de convolução com o filtro ao centro para gerar a imagem à direita. O pixel verde representa a saída referente aos pixels azuis (SZELISKI, 2010).

Um ótimo exemplo de filtro linear é o *Mean Filter*. Todos os coeficientes do *Mean Filter* possuem o valor de 1. Para evitar um aumento na intensidade após a filtragem, a imagem inteira é dividida pelo número de pixels do filtro. Por exemplo, no caso de um filtro 3x3, a imagem seria dividida por nove (CHITYALA e PUDIPEDDI, 2014).

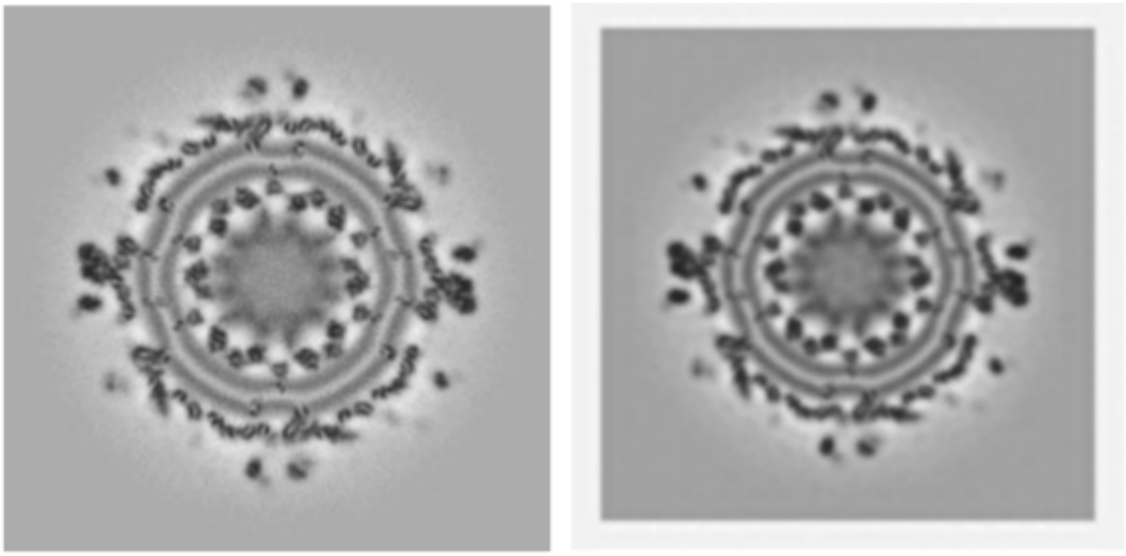


(a) Imagem de entrada do *Mean Filter*; (b) Saída obtida com um filtro 5x5;

Figura 2.12: Exemplo de *Mean Filter* (CHITYALA e PUDIPEDDI, 2014).

A Figura 2.12 representa um exemplo da entrada e saída de um *Mean Filter*. Percebe-se que o filtro conseguiu remover parte do ruído, porém a imagem, principalmente os contornos, ficou mais suave e embaçada.

Um filtro semelhante ao *Mean Filter* é o *Gaussian Filter*. O objetivo do Gaussian Filter é desfocar e suavizar uma imagem, assim como o Mean Filter. O grau de suavização da imagem é determinado pelo desvio padrão de Gauss. Ou seja, o filtro tem como saída a média ponderada da vizinhança de cada pixel com a média ponderada em relação aos pixels centrais. Isso se diferencia do filtro uniformemente distribuído do *Mean Filter*. Por causa disso, o *Gaussian Filter* é mais suave e preserva melhor as bordas do que um *Mean Filter* de mesma dimensão.



(a) Imagem de entrada do *Gaussian Filter*; (b) Saída obtida com um filtro 5x5;

Figura 2.13: Exemplo de *Gaussian Filter* (CHITYALA e PUDIPEDDI, 2014).

A Figura 2.13 representa um exemplo de entrada e saída de um *Gaussian Filter*.

Em muitos casos, entretanto, uma melhor performance pode ser obtida com o uso de combinações não-lineares entre pixels. Um dos filtros não-lineares mais populares em visão computacional é o *Median Filter*. Este filtro utiliza o cálculo da mediana da vizinhança do pixel para a obtenção de sua saída.

(a) Imagem com ruídos distribuídos aleatoriamente; (b) Saída com Median Filter;

Figura 2.14: Exemplo de Median Filter (SZELISKI, 2010).

A Figura 2.17 mostra um exemplo de resultado do *Median Filter*. É um filtro usado normalmente para remoção de ruídos pontuais distribuídos aleatoriamente sobre a imagem. Uma vez que o valor do ruído geralmente fica bem fora dos valores reais na vizinhança, o *Median Filter* é capaz de filtrar esses pixels ruins (SZELISKI, 2010).

Outros filtros não-lineares importantes e muito usados em visão computacional são: *Maximum Filter* e *Minimum Filter*.

O *Maximum Filter* aumenta os pontos brilhantes em uma imagem. Neste filtro o valor máximo da vizinhança de um pixel é a sua saída. No *Minimum Filter* ocorre o oposto, os pontos mais escuros são aumentados, pois o valor mínimo da vizinhança é utilizado (CHITYALA e PUDIPEDDI, 2014).

2.3.4 Detecção de Borda

Pontos de borda são abundantes e frequentemente carregam associações semânticas importantes. Por exemplo, as bordas de um objeto são geralmente delineadas por linhas bem visíveis. Pontos isolados de borda podem também ser agrupados em curvas ou contornos, assim como segmentos de linhas retas. O que é muito interessante, pois até mesmo pequenas crianças não têm dificuldade em reconhecer objetos familiares ou animais em desenhos de linhas simples (SZELISKI, 2010). A Figura 2.15 demonstra alguns exemplos de obtenção de contornos em imagens.

Figura 2.15: Exemplo de detecção de borda (SZELISKI, 2010).

As bordas ocorrem no limite de regiões com uma diferença significativa de intensidade, cor ou textura. Na Matemática, sabe-se que mudanças de intensidade podem ser medidas usando a derivada primeira ou segunda.

Uma imagem não é uma função contínua e por isso as derivadas são calculadas usando aproximações discretas. É importante então definir o gradiente de uma função contínua para, assim, expandir para casos discretos (CHITYALA e PUDIPEDDI, 2014). Se $f(x, y)$ é uma função contínua, o gradiente de f, como um vetor, é dado por

$$\nabla f = \begin{bmatrix} f_x \\ f_y \end{bmatrix}, \tag{2.13}$$

onde $f_x = \dfrac{\partial f}{\partial x}$ é a derivada parcial de f em relação a x, ou seja, representa a mudança de f na direção horizontal. $f_y = \dfrac{\partial f}{\partial y}$ é a derivada parcial de f em relação a y, que representa a variação de f na direção vertical.

O módulo do gradiente é uma grandeza escalar e é dada por

$$|\nabla f| = \sqrt{\left[(f_x)^2 + (f_y)^2\right]}. \tag{2.14}$$

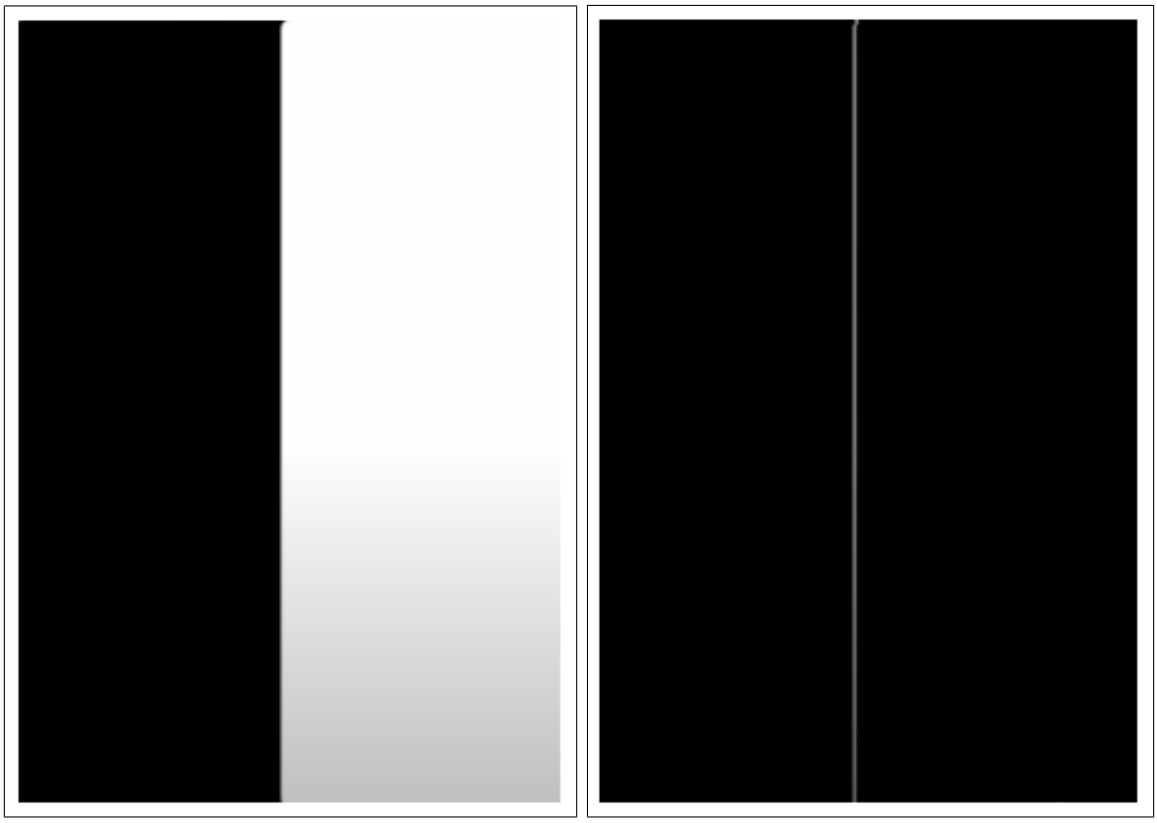
Para propósitos computacionais, pode-se considerar a versão simplificada do gradiente e o seu ângulo, respectivamente, abaixo:

$$|\nabla f| = |f_x| + |f_y|, \tag{2.15}$$

$$\theta = tan^{-1}(\frac{f_y}{f_x}). \tag{2.16}$$

O gradiente $|\nabla f|$ é responsável por definir a variação de intensidade dos pixels em relação aos

seus vizinhos. O objetivo da aplicação desse gradiente é reconhecer regiões de grande variação e representá-las na imagem, como pode ser visto na Figura 2.16.



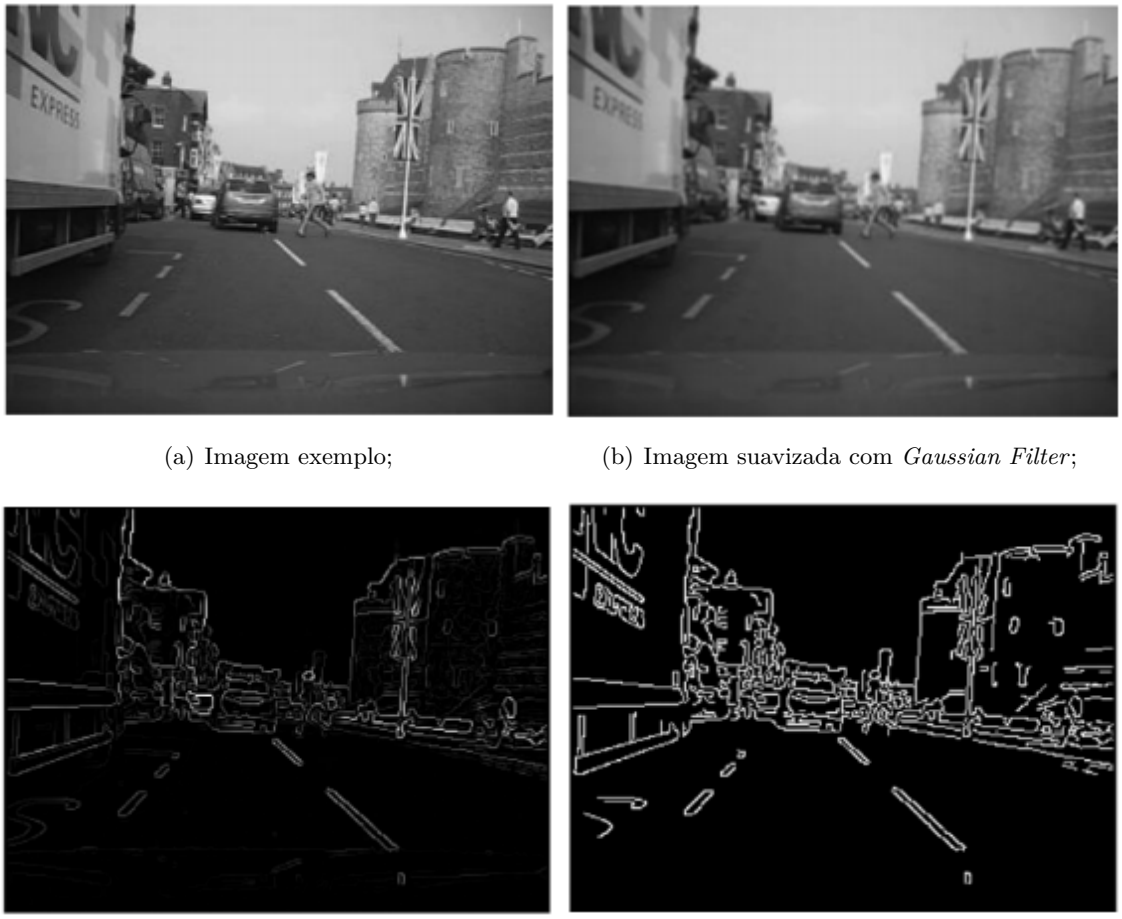
(a) Imagem exemplo; (b) Detecção de borda;

Figura 2.16: Objetivo no processo de detecção de borda.

Um dos mais populares filtros de derivada primeira é o *Canny Filter*. De acordo com Chityala e Pudipeddi, esse filtro possui três parâmetros de entrada para detectar contornos. O primeiro parâmetro é o desvio padrão para um *Gaussian Filter*. O segundo e terceiro parâmetros são valores de *threshold*, ou seja, são os valores de máximo e mínimo que definem a classificação dos contornos como fortes ou fracos. O *Canny Filter* pode ser descrito pelos seguintes passos:

- a imagem de entrada para o *Canny Filter* deve estar em escala de cinza;
- o *Gaussian Filter* é usado para suavizar e remover ruídos na imagem;
- uma importante propriedade do pixel de borda é que ele terá uma magnitude máxima de gradiente na direção do gradiente. Então, para cada pixel, a magnitude do gradiente dada pela Equação 2.15 e a direção correspondente dada pela Equação 2.16 são calculados;
- o valor absoluto (magnitude) do gradiente da imagem nos pontos de borda são o máximo, ou seja, os pixels de pico. Para identificar pontos de borda e suprimir outros pontos, apenas pixels de pico são mantidos, enquanto outros pixels são definidos com o valor zero. Esse processo pode ser chamado de supressão não-máxima (DAVIES, 2018);

- os picos são definidos pelos *thresholds* informados na entrada do filtro. Pixels de pico são classificados como fortes pixels de borda quando o valor supera o maior *threshold*, enquanto pixels de pico que estão entre o maior e o menor *threshold* são definidos como fracos pixels de borda; e
- os pixels fracos de borda são, então, conectados com fortes pixels de borda por 8-connected[3].



(a) Imagem exemplo; (b) Imagem suavizada com *Gaussian Filter*;

(c) Resultado da supressão não-máxima; (d) Resultado final com os *thresholds*.

Figura 2.17: Exemplo de aplicação de *Canny Filter* (DAVIES, 2018)

2.3.5 *Hough Transform* para Detecção de Linhas

Um problema recorrente em visão computacional é a detecção de linhas retas em imagens. Essa detecção é fundamental, por exemplo, para o presente trabalho, pois a detecção do feixe de reflexão do laser é muito importante para a construção da nuvem de pontos. O problema aqui é detectar a presença de grupos de pontos colineares ou perto de serem colineares. Este problema pode ser resolvido, por exemplo, por qualquer grau de acurácia testando as linhas formadas por cada par

[3] 8-connected é uma forma de conexão entre pixels, onde os pixels são conectados horizontalmente, verticalmente e diagonalmente.

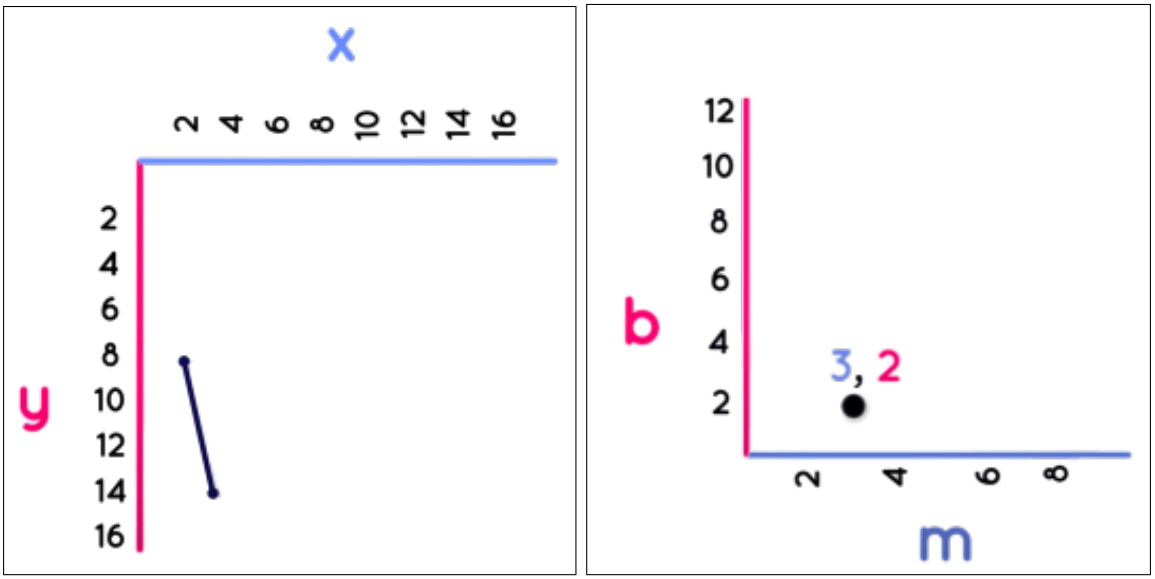
de pontos. Porém, o processamento computacional exigido para n pontos é aproximadamente n^2, o que pode se tornar inviável para um valor grande de n (Duda e Hart, 1972).

Hough (Hough, 1962) propôs um procedimento interessante e eficiente computacionalmente para a detecção de linhas em imagens. O procedimento pode ser descrito como a substituição do problema original de encontrar pontos colineares para o problema de se encontrar retas concorrentes. Esse método consiste em transformar cada ponto da imagem em uma reta no espaço de parâmetro (*Hough Space*) (Duda e Hart, 1972).

Uma reta pode ser descrita pela seguinte equação:

$$y = mx + b, \tag{2.17}$$

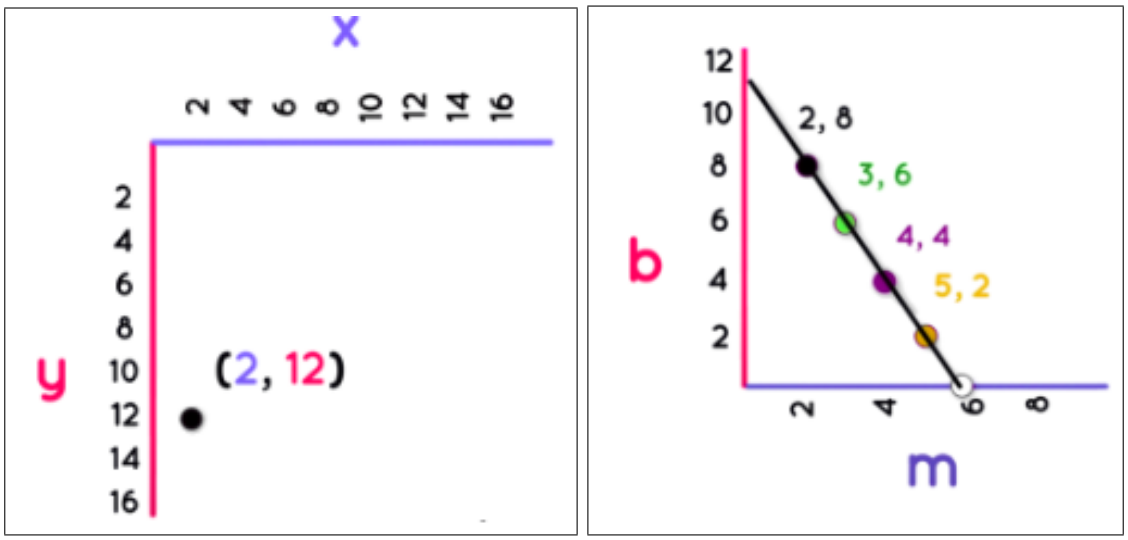
onde m é o coeficiente angular da reta e b é o coeficiente linear da reta. Normalmente a reta é traçada em função de x e y. Porém, essa reta também pode ser traçada no espaço de parâmetros (*Hough Space*), ou seja, traçada em função de seus parâmetros m e b. A Figura 2.18 demonstra um exemplo desse processo.



(a) $y = 3x + 2$; (b) Representação da reta no *Hough Space*;

Figura 2.18: Exemplo de representação de reta no *Hough Space*.

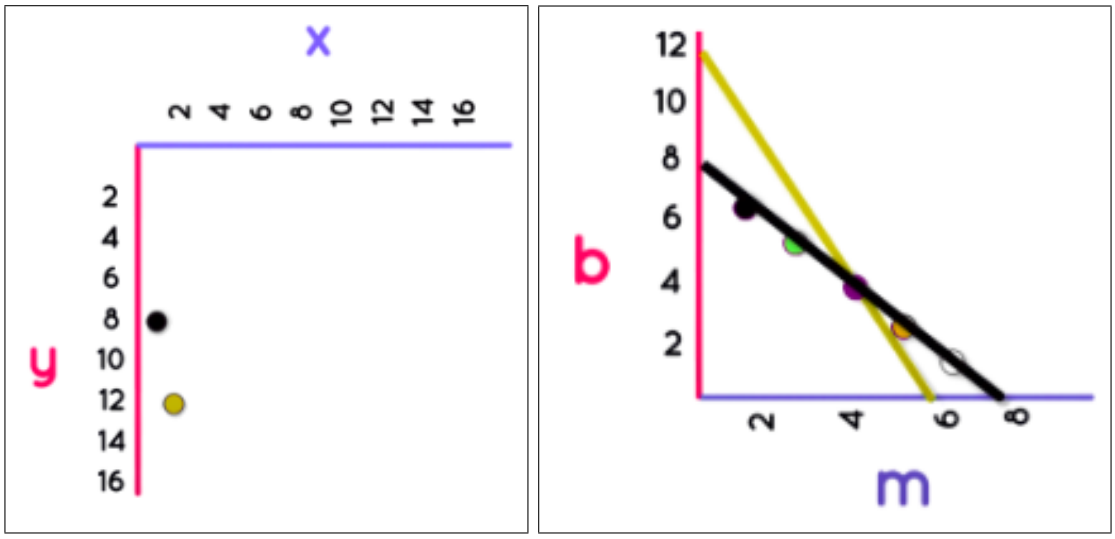
O que torna interessante essa transformação é a representação de um ponto no *Hough Space*. Definido um ponto em função de x e y, sabe-se que infinitas retas podem cruzar esse ponto, ou seja, retas com diferentes coeficientes m e b. Na representação no *Hough Space* então, o ponto se torna uma reta que define os diferentes valores de m e b das retas que cruzam esse ponto. A Figura 2.19 demonstra um exemplo desse processo.



(a) Ponto (2, 12); (b) Representação do ponto no *Hough Space*;

Figura 2.19: Exemplo de representação de um ponto no *Hough Space*.

Pode se perceber pela Figura 2.19, que o *Hough Space* representa por meio de uma reta a família de retas que cruza o ponto no espaço (x,y). A Figura 2.20 demonstra a representação de dois pontos.

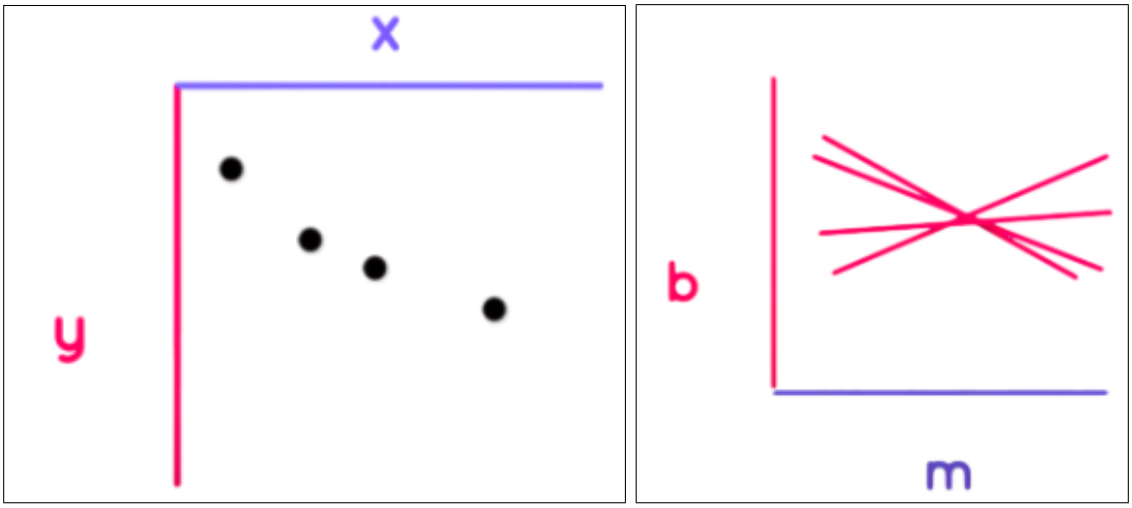


(a) Dois Pontos; (b) Representação de dois pontos no *Hough Space*;

Figura 2.20: Exemplo de representação de dois pontos no *Hough Space*;

Considerando o exemplo da Figura 2.20, existem várias retas que podem cruzar os pontos individualmente, porém apenas uma pode cruzar de forma consistente os dois pontos. Isso pode ser determinado pelo ponto de intersecção das retas no *Hough Space*, pois esse ponto representa os coeficientes m e b da reta que cruza os dois pontos no espaço (x,y). Para mais de dois pontos, porém, o processo não é o mesmo. Por exemplo, no caso de 4 pontos representados no espaço (x,y)

que não estão totalmente alinhados, ou seja, não existe uma reta que passe pelos quatro de forma consistente. Muitas vezes, é interessante encontrar uma reta que se aproxime de interseccionar os quatro pontos. Neste caso, uma aproximação deve ser feita. A Figura 2.21 demonstra um exemplo de representação de quatro pontos não totalmente alinhados.

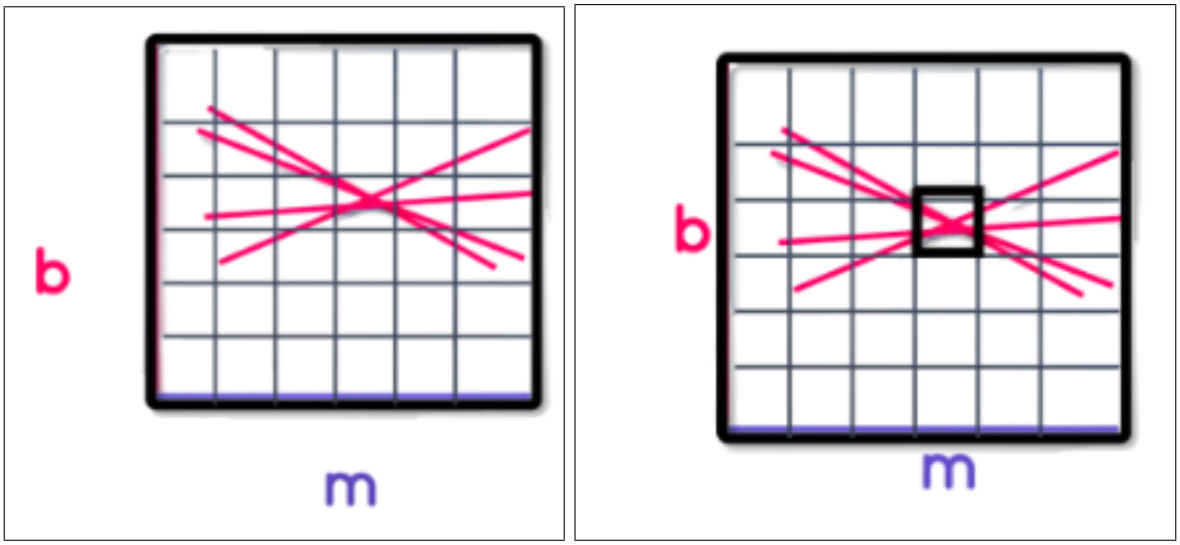


(a) Quatro Pontos;

(b) Representação de quatro pontos no *Hough Space*;

Figura 2.21: Representação de quatro pontos não totalmente alinhados no *Hough Space*.

O que pode ser feito nesse caso é a divisão do *Hough Space* em quadrados de tamanho único, formando um *grid*. Para encontrar a melhor opção de reta, deve-se considerar o quadrado onde ocorreram mais intersecções de reta. A Figura 2.3.5 demonstra esse processo.



(a) Divisão em grid do *Hough Space*;

(b) Região com mais interseções;

Figura 2.22: *Grid* para determinação de melhor reta de pontos não colineares

.

O valor do centro do quadrado é então definido como o melhor resultado para os coeficientes m e b da reta que passa pelos quatro pontos. O resultado para esse exemplo é mostrado na Figura 2.23.

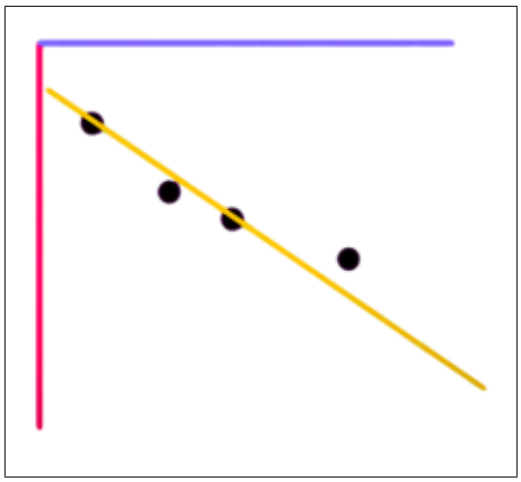


Figura 2.23: Representação de reta definida pelo processo de *grid* do *Hough Space*.

Porém, existe um problema nessa representação. Não é possível representar retas verticais do sistema de coordenadas (x,y) no *Hough Space*, pois o coeficiente m é infinito. Ou seja, obter retas a partir de pontos que estão alinhados verticalmente não seria possível. A solução do problema é converter o sistema de coordenadas cartesiano para o sistema de coordenadas polar (Duda e Hart, 1972). No sistema polar, as retas podem ser definidos como:

$$\rho = xcos\theta + ysin\theta. \tag{2.18}$$

A Figura 2.24 demonstra um exemplo de representação no sistema de coordenadas polar.

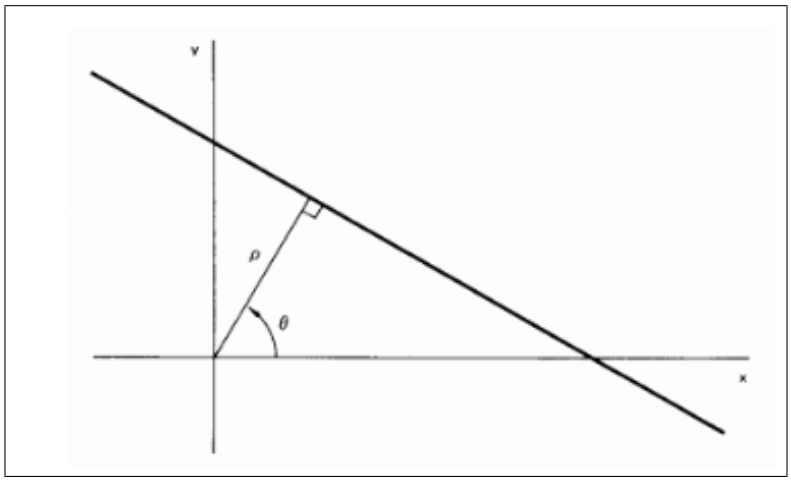
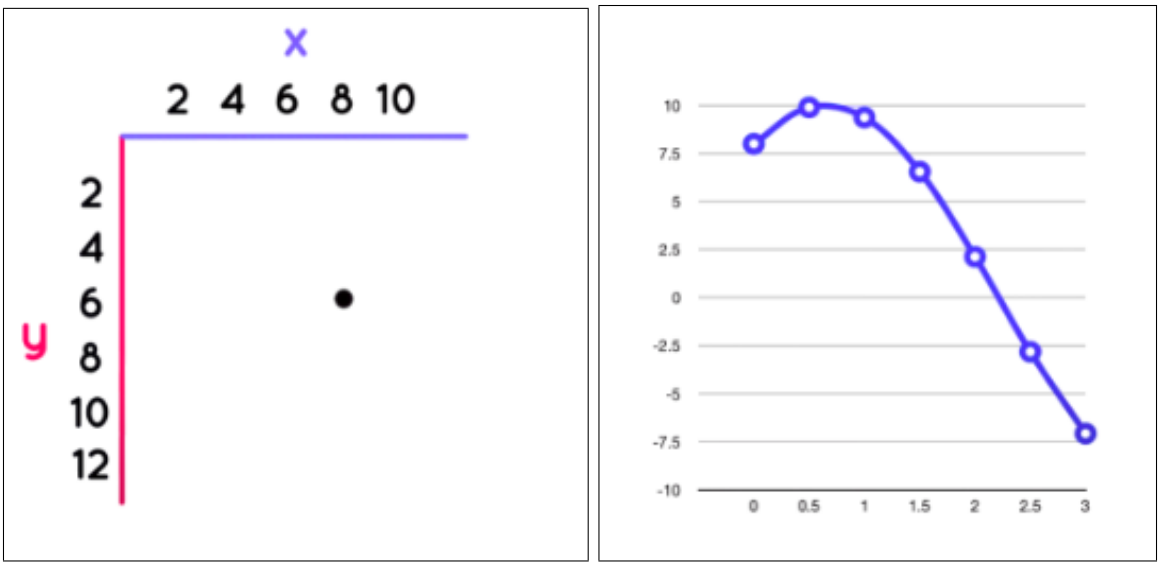


Figura 2.24: Sistema de coordenadas polar (Duda e Hart, 1972).

Como foi visto, no sistema de coordenadas cartesiano, as retas que cruzam um ponto são representadas por uma reta no *Hough Space*. No sistema de coordenadas polar, elas são representadas

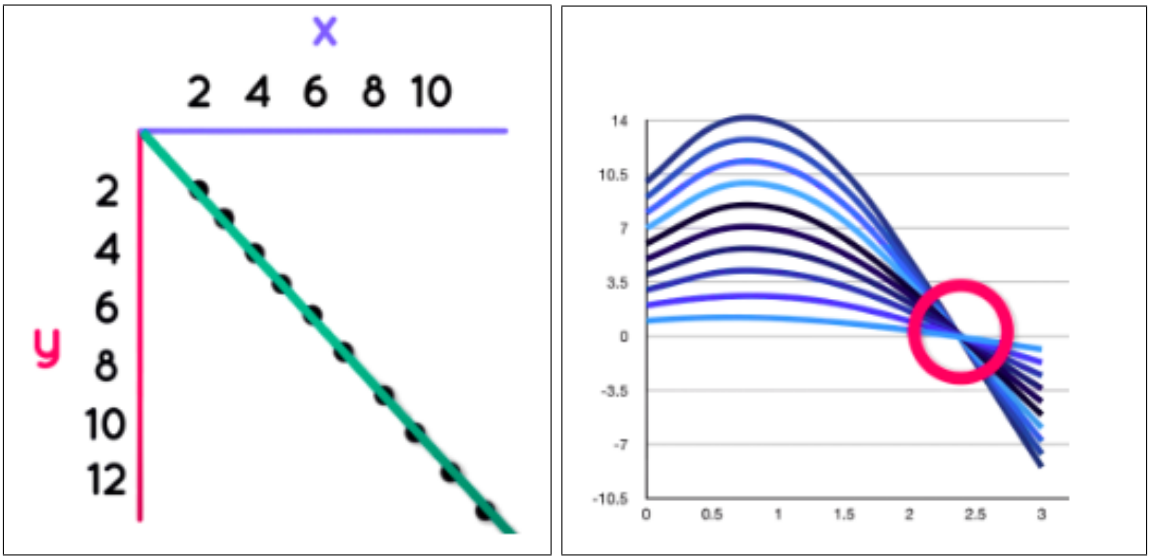
por uma curva senoidal, como pode ser visto na Figura 2.25.



(a) Ponto no sistema polar; (b) Hough Space no sistema polar;

Figura 2.25: Representação de ponto no Hough Space no sistema de coordenadas polares.

O conceito para obtenção das retas que cruzam mais de um ponto é basicamente o mesmo do sistema cartesiano. Pontos geram curvas senoidais, a intersecção dessas curvas definem os coeficientes ρ e θ que define a reta que cruza os pontos. Isso pode ser visto na Figura 2.26.



(a) Vários pontos, com a reta já definida; (b) Intersecção no *Hough Space*;

Figura 2.26: Obtenção da reta que cruza vários pontos a partir da interseção de curvas senoidais no *Hough* Space no sistema polar.

2.3.6 Biblioteca OpenCV

OpenCV é uma biblioteca de visão computacional de código aberto. É uma biblioteca escrita em C e C++ que pode ser executada no Linux, Windows e Mac OS X. Existe um desenvolvimento ativo de interfaces para Python, Ruby, Matlab e outras linguagens. A OpenCV foi projetada com o foco em aplicações em tempo real e para eficiência computacional (BRADSKI E KAEHLER, 2008).

A licença de código aberto do OpenCV foi estruturada para que quem usá-la possa produzir produtos comerciais sem a obrigação de tornar o código do produto aberto ou retornar otimizações da biblioteca para domínio público. O maior objetivo do OpenCV é fornecer uma infraestrutura de visão computacional capaz de desenvolver aplicações de visão sofisticadas de forma rápida e de fácil uso (BRADSKI E KAEHLER, 2008).

A biblioteca OpenCV possui mais de 500 funções que abrangem diferentes áreas na visão computacional, entre elas estão: inspeção de produtos, imagens médicas, segurança, interface do usuário, calibração de câmera, visão estéreo e robótica (BRADSKI E KAEHLER, 2008).

É uma excelente biblioteca para o presente trabalho, facilitando a otimização de imagens, assim como a reconstrução 3D a partir de uma nuvem de pontos.

2.3.7 Marching Cubes Algorithm

Marching Cubes é um algoritmo para renderizar isosuperfícies em dados volumétricos. Ou seja, é responsável por criar uma malha de triângulos a partir de um objeto digital ou nuvem de pontos.

O algoritmo divide o espaço tridimensional em um número arbitrário de cubos, definidos por 8 pixels. O algoritmo determina como que a nuvem de pontos intercepta um cubo, e então analisa o próximo cubo. Se um ou mais pixels de um cubo (vértices) tiverem valores menores que algum ponto das nuvens de pontos e um ou mais pixels tiverem valores maiores que esse ponto, então o cubo é importante para a criação da malha. Ao determinar quais cubos foram interceptados pela superfície da nuvem de pontos, pode-se criar triângulos que dividem o cubo em regiões dentro da nuvem de pontos (Figura 2.27) e fora da nuvem de pontos. Ao conectar esses triângulos, tem-se a representação da superfície do objeto (Lorensen e Cline, 1987).

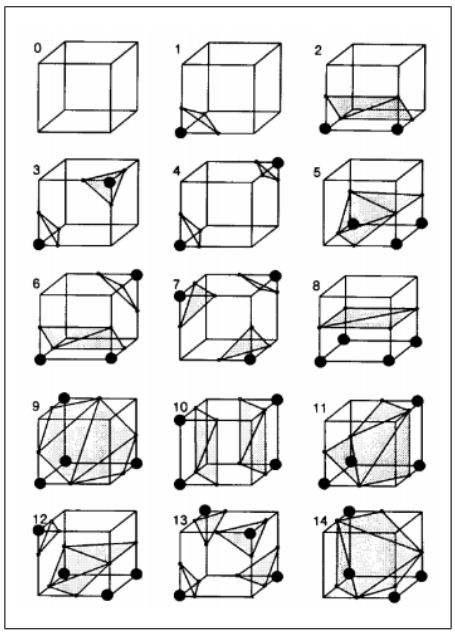


Figura 2.27: Cubos triangulados (Lorensen e Cline, 1987).

Para melhor entendimento do processo, pode-se imaginar como ele seria aplicado sobre uma nuvem de pontos no plano bidimensional. A Figura 2.28 apresenta uma representação de uma nuvem de pontos e do espaço bidimensional dividido por quadrados, que seriam os cubos no espaço tridimensional.

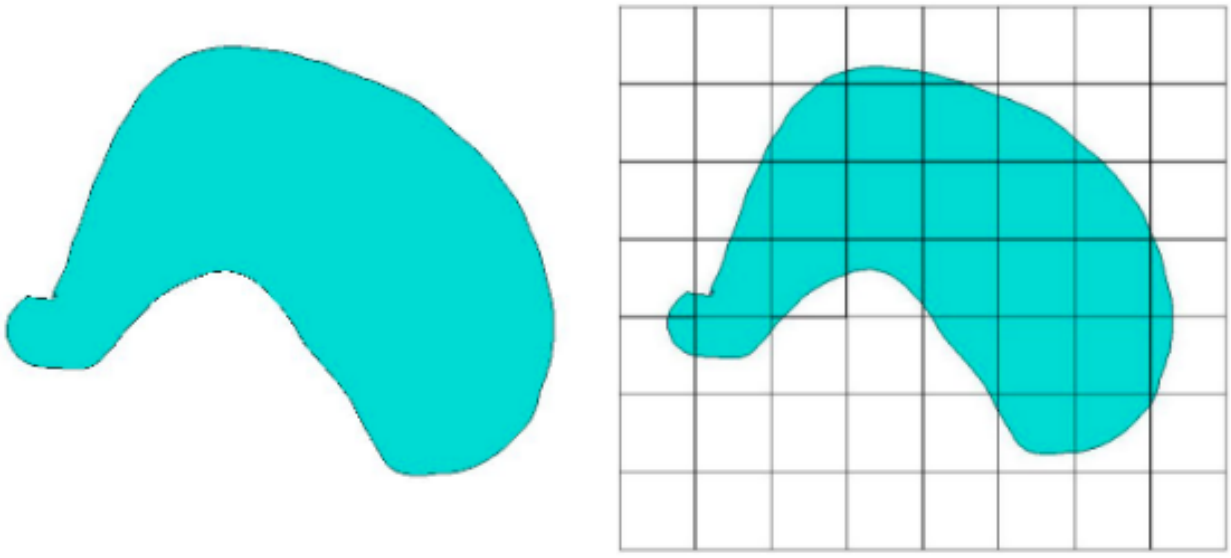


Figura 2.28: Nuvem de pontos na plano 2D e divisão do espaço em quadrados uniformes (Anderson, 2020).

Pela imagem, pode-se determinar facilmente as vértices que estão dentro da nuvem de pontos e as que estão fora. A Figura 2.29, então, apresenta os vértices que estão dentro da nuvem na cor

vermelha e os vértices fora na cor azul.

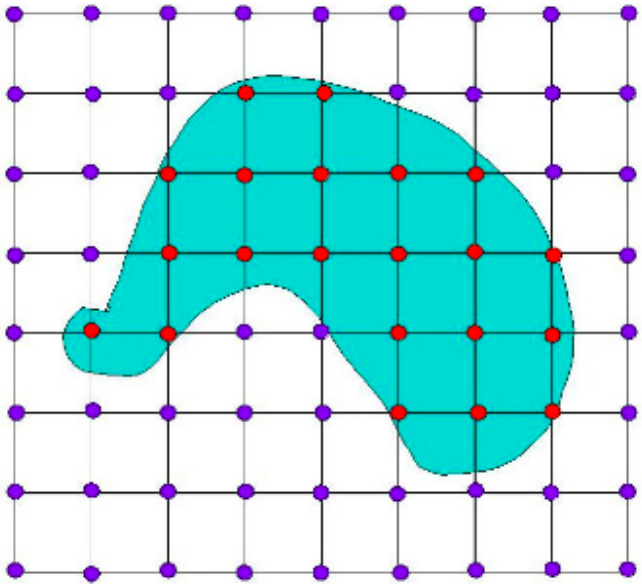


Figura 2.29: Vértices dos quadrados (Anderson, 2020).

Sabe-se que a superfície da nuvem de pontos interceptou algum lugar das arestas definidas por um vértice que está dentro e por um vértice que está fora da nuvem de pontos. Por isso, marcou-se o centro dessas arestas com um ponto na cor rosa (Figura 2.30).

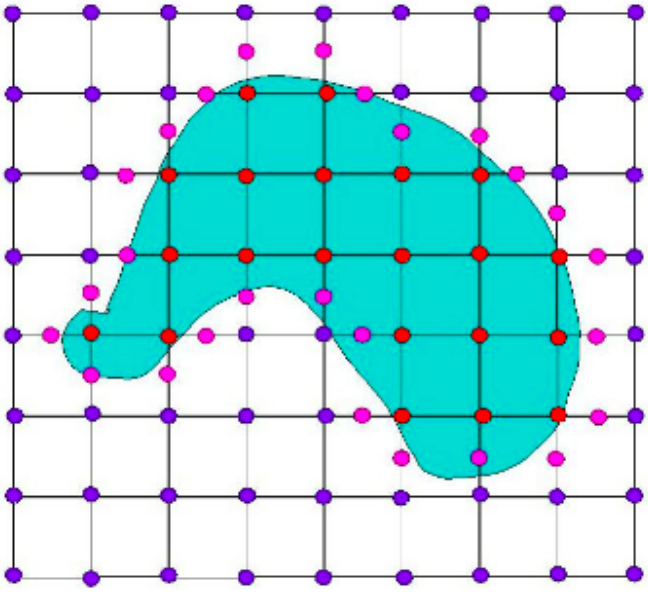


Figura 2.30: Pontos que representam a superfícies (Anderson, 2020).

Definidos os pontos, o que deve ser feito agora é ligá-los, como pode ser visto na Figura 2.31.

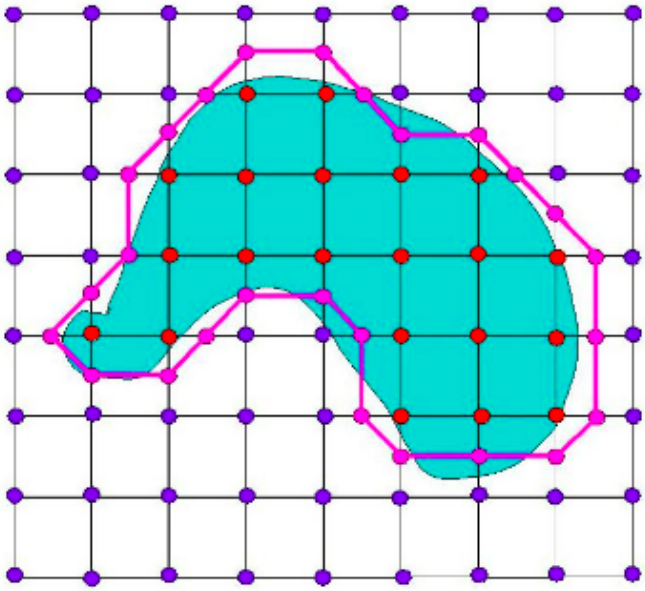


Figura 2.31: Representação aproximada da superfície (Anderson, 2020).

Essa conexão entre os pontos, que define uma aproximação da superfície, representa o processo de formação dos triângulos do algoritmo no espaço tridimensional. No espaço tridimensional, a definição dos pontos pode ser representada pelo exemplo da Figura 2.32. As 15 possíveis configurações de triangulação em um cubo estão disponíveis na Figura 2.27.

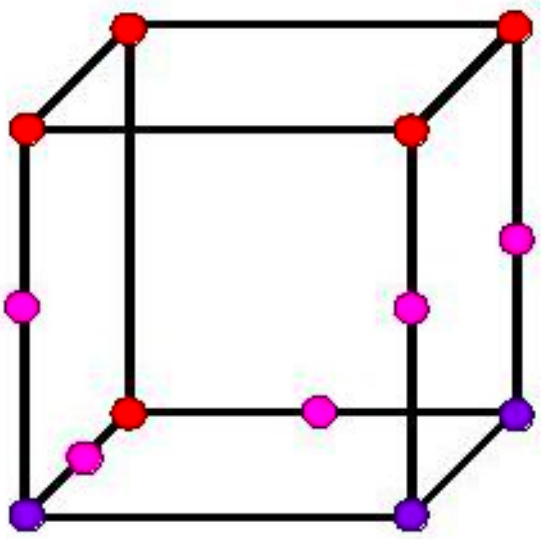


Figura 2.32: Definição de pontos da superfície em um cubo (Anderson, 2020).

Capítulo 3

Estrutura de Hardware

Este capítulo apresentará os materiais utilizados na construção do *scanner* de triangulação a laser, tanto sua parte mecânica, quanto os eletrônicos utilizados. O *hardware* foi projetado e desenvolvido por um projeto de graduação de engenharia mecânica da Universidade de Brasília (HIRANO, 2020).

3.1 Eletrônicos

O conjunto de eletrônicos que foi utilizado no *scanner* se encontra na Tabela 3.1

Laser	Laser LED 650nm
Câmera	C270 do fabricante Logitech
Microcontrolador	Arduino Mega 2560
Shield para microcontrolador	Ramps 1.4
Driver para o motor	Driver A4988
Motor para o eixo Z	JK42HS40-1004A(1.8°) do fabricante JKONGMOTOR
Fonte	Fonte genérica estabilizada de 12V e 30A

Tabela 3.1: Tabela de eletrônicos.

3.2 Estrutura Mecânica

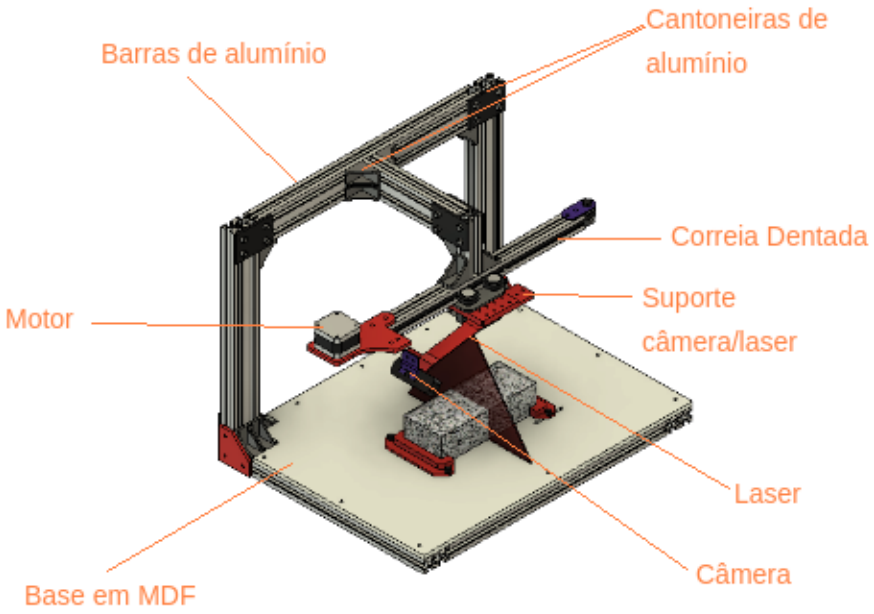


Figura 3.1: Projeto módulo de triangulação.

A Figura 3.1 representa a renderização gráfica da estrutura mecânica do sistema de módulo de triangulação projetado.

A estrutura mecânica do scanner desenvolvido é formada por perfis de alumínio de 20x20mm e 40x20mm (Figura 3.2). Fabricaram-se cantoneiras de alumínio para união dos perfis por parafusos, com o intuito de manter a estrutura estável e robusta (Figura 3.3).

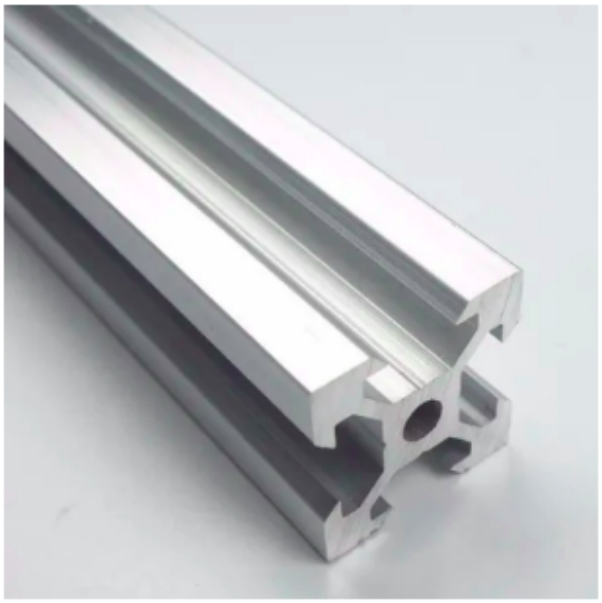


Figura 3.2: Perfil de alumínio estrutural V-slot 20x20.

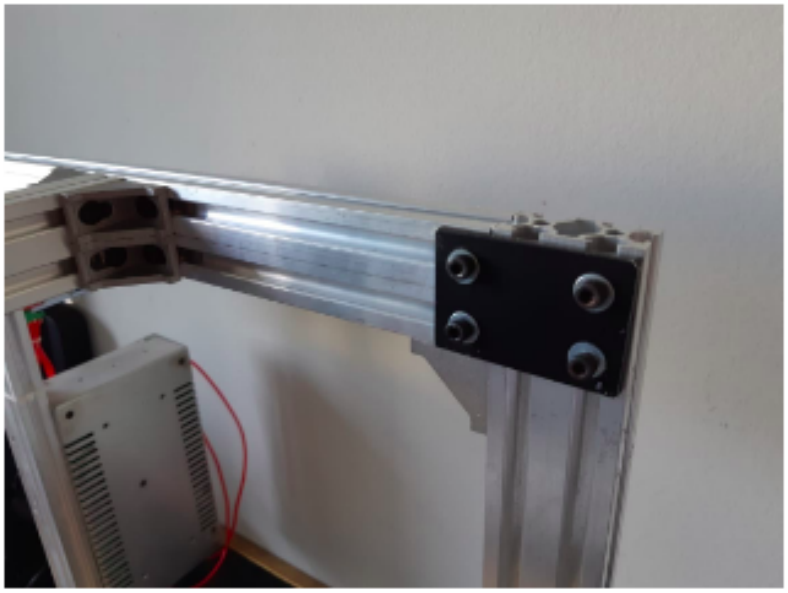


Figura 3.3: Cantoneiras de alumínio.

A base da estrutura foi construída com um painel de MDF cortado com máquina a laser. Foram projetados dois suportes para a estabilização do corpo de prova sobre a base da estrutura e para definir a área de trabalho do *scanner*. Ambos os suportes foram impressos em 3D no material PLA e fixados com parafusos sobre a base (Figura 3.4).

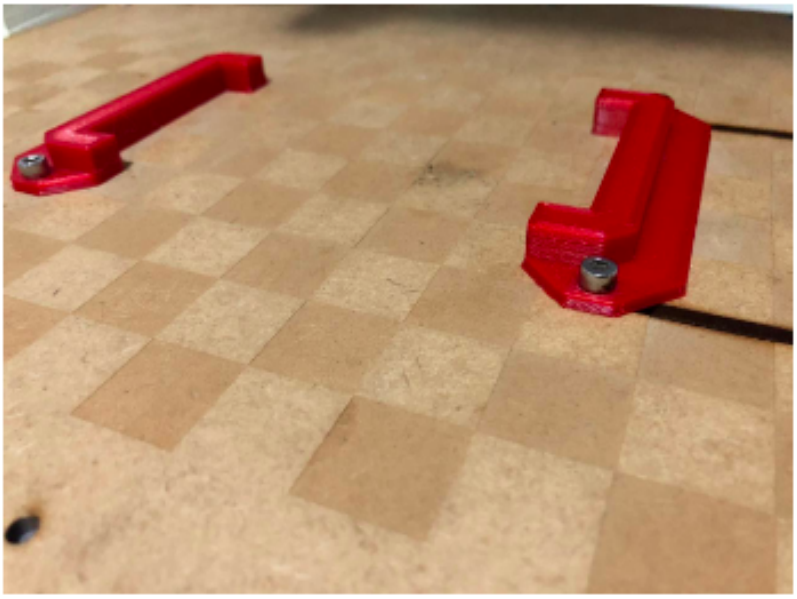


Figura 3.4: Base do *scanner*.

O suporte da câmera e do laser linha foram modelados em 3D e então impressos em 3D no material PLA (ácido poliláctico). O suporte foi projetado para que seja possível o ajuste do posicionamento de altura e ângulo da câmera, isso foi feito para que seja possível a calibração do módulo de triangulação (Figura 3.5).

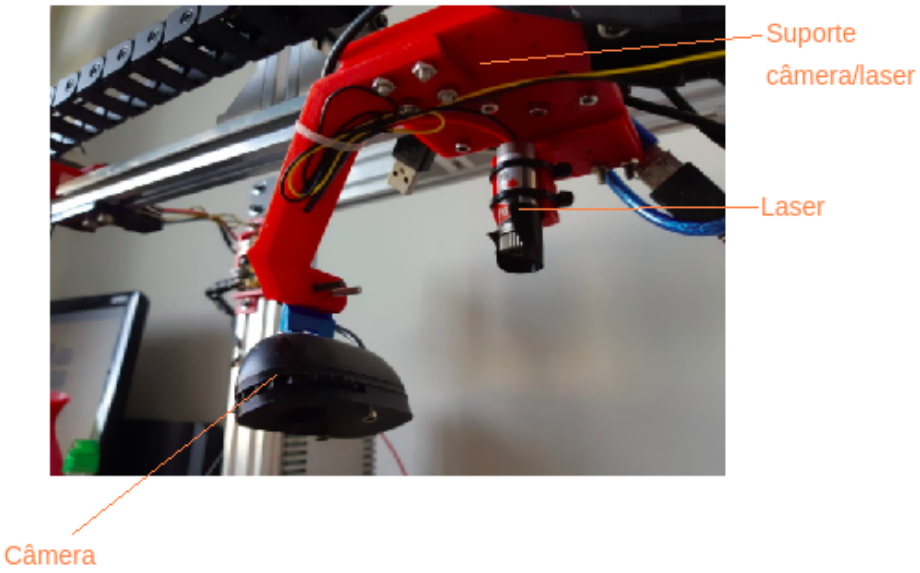


Figura 3.5: Estrutura câmera/laser

Foi acoplado à estrutura câmera/laser uma correia dentada para que a translação da estrutura por um motor fosse possível (Figura 3.6).

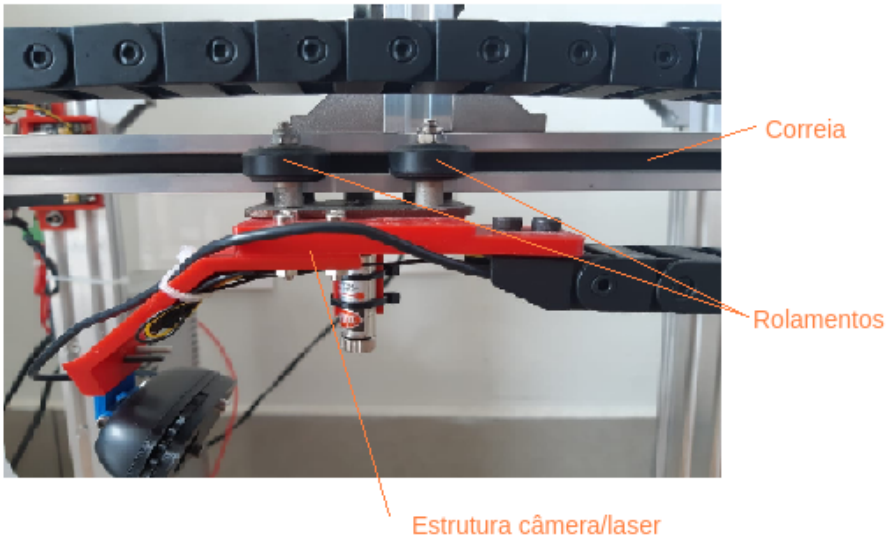


Figura 3.6: Correia dentada acoplada à estrutura câmera/laser.

A correia foi então acoplada ao motor de passo. O motor de passo foi fixado na estrutura por meio de um suporte modelado digitalmente e impresso em 3D no material PLA (Figura 3.7). O mesmo foi feito para a fixação do microcontrolador, do driver e da fonte (Figura 3.8).

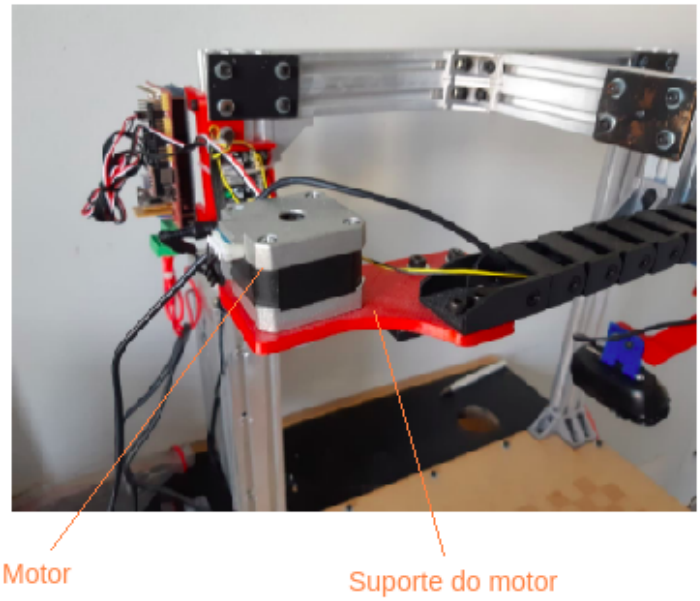


Figura 3.7: Suporte do motor.

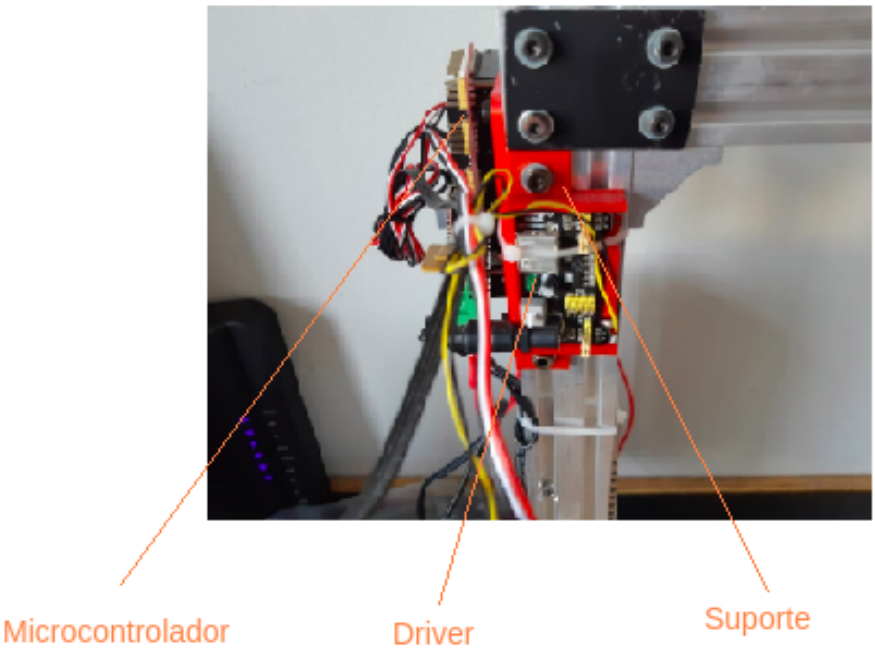


Figura 3.8: Fixação dos eletrônicos.

A Figura 3.9 mostra o projeto finalizado.

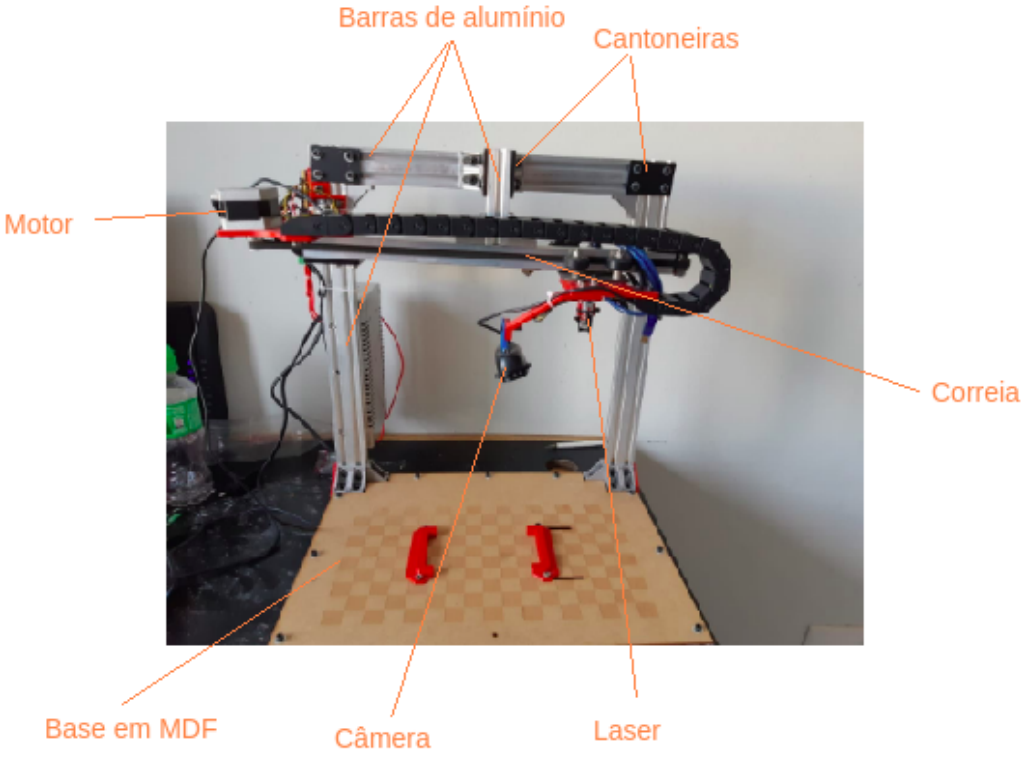


Figura 3.9: Estrutura de *hardware* do módulo de triangulação.

A máquina construída possui como precisão no eixo de translação da estrutura câmera/laser o valor de 0.3 mm. Em relação ao ângulo da câmera e do laser, fixou-se manualmente no valor de 20° aproximadamente. Este valor será definido precisamente no processo de calibração dos parâmetros do módulo (seção 5.3). Esses e outras medidas técnicas do *scanner* estão na tabela 3.2.

Precisão no deslocamento	0.3 mm
Velocidade	16.67 mm/s
Dimensões (X,Y,Z)	500x460x440 mm
Volume de trabalho (X,Y,Z)	60x150x130 mm^3
Ângulo câmera/laser	20°
Distância entre o laser e a base	280 mm

Tabela 3.2: Tabela de eletrônicos.

Capítulo 4

Otimização das Imagens de Profundidade

Neste capítulo, é realizada uma análise e o processamento das imagens captadas pela câmera. O objetivo aqui é remover distorções e ruídos das imagens para que a segmentação do laser seja precisa. O processo se inicia pela calibração da câmera, com o intuito de remover distorções provindas da lente da câmera. Todos os *frames* passam pelo processo de remoção de distorções. Depois disso, a segmentação por intensidade de pixels é realizada para obter apenas o laser nas imagens. Filtros e detecção de bordas são aplicados para que as imagens finais possuam menos ruídos. Ao final do capítulo, a reta de referência, que é a reta que representa o laser sobre a base do scanner sem influência de um objeto, é obtida como uma forma de calibração da máquina.

4.1 Calibração da Câmera

Como foi explicado na seção 2.3.1, os objetivos do código de calibração de uma câmera é obter a matriz de intrínsecos 3x3 K, as matrizes 3x3 de rotação R e o vetor 3x1 t usando pontos 3D conhecidos (X,Y,Z) e as coordenadas correspondentes na imagem (x, y) (equações 2.8 e 2.11). O processo pode ser definido pelos seguintes passos:

- Captar imagens do posicionamento em diferentes ângulos de um tabuleiro de xadrez;
- Encontrar as coordenadas dos pixels das vértices do tabuleiro na imagem;
- Obter a matriz 3x3 K, as matrizes 3x3 de rotação R e o vetor 3x1 t usando pontos 3D conhecidos (X,Y,Z) e as coordenadas correspondentes na imagem (x, y);.
- Salvar os dados;
- Ler os dados e remover as distorções de novas imagens;

Então, primeiramente, foram tiradas várias fotos de diferentes ângulos do tabuleiro com a câmera usada no módulo de triangulação (exemplos na Figura 4.1).

Figura 4.1: Imagens capturadas do tabuleiro de xadrez.

Por meio da biblioteca OpenCV para a linguagem de programação Python, utilizou-se a função *findChessboardCorners* para encontrar as coordenadas dos pixels referentes às vértices do tabuleiro nas imagens. Essa função tem como entrada a imagem do tabuleiro e a quantidade de vértices por linha e coluna do tabuleiro. Sua saída são as coordenadas em pixel das vértices encontradas (exemplo na Figura 4.2) e um retorno booleano que indica se foi possível a detecção.

Figura 4.2: Vértices do tabuleiro de xadrez definidas.

Então, foram passadas as coordenadas globais do tabuleiro de xadrez, as coordenadas dos vértices obtidos das imagens e o tamanho das imagens para a função *calibrateCamera* do OpenCV. A função retorna, então, a matriz intrínseca da câmera, os coeficientes de distorção, o vetor de rotação R e o vetor de translação t.

As seguintes informações foram obtidas:

$$K = \begin{bmatrix} 503.685 & 0 & 313.676 \\ 0 & 503.380 & 243.256 \\ 0 & 0 & 1 \end{bmatrix}; \tag{4.1}$$

$$\begin{bmatrix} 0.20835 & -0.46865 & 4.51082 & -0.00193 & 0.23759 \end{bmatrix}, \tag{4.2}$$

em que K é a matriz de intrínsecos e a matriz 4.2 representa os coeficientes de distorção. Foram obtidas também as matrizes de rotação e vetores de translação, que são muito grandes para representação neste documento.

Todos esses parâmetros foram salvos em um arquivo de texto separado.

O último passo, então, é a aplicação de todas essas informações sobre as novas imagens captadas pela câmera com o intuito de remover as distorções. Para isso, usou-se a função *undistort* do OpenCV, responsável pelo ajuste e remoção das distorções tendo como entrada os parâmetros fornecidos pela função *calibrateCamera*.

O código completo de calibração está no apêndice A.

4.2 Segmentação

Um exemplo de captação da câmera do módulo de triangulação sem nenhum processamento de imagem, apenas com a calibração da câmera, está representado na Figura 4.3.

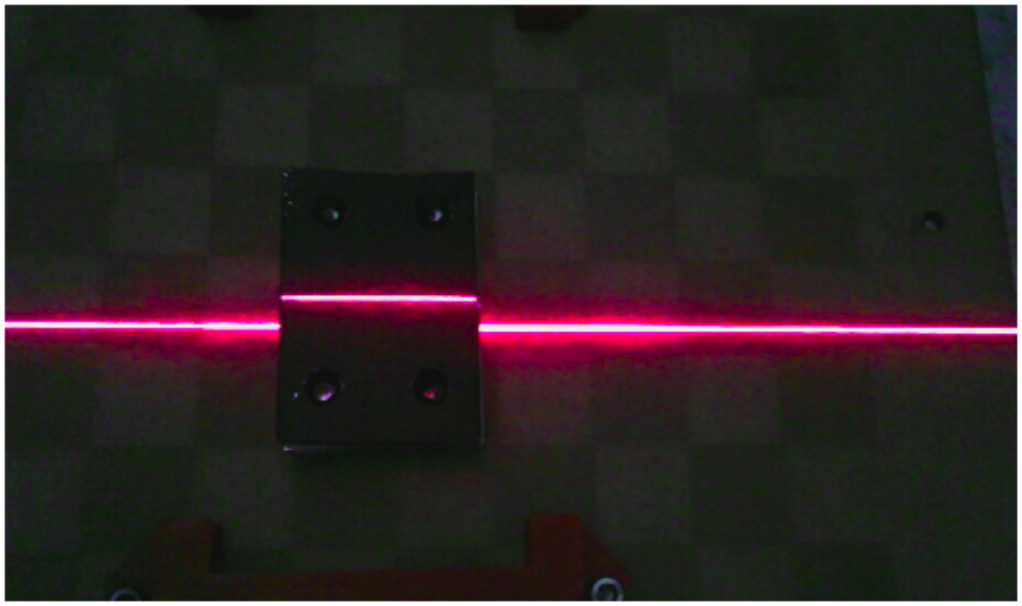


Figura 4.3: Imagem captada pela câmera sem nenhuma otimização (imagem original).

Para que o módulo de triangulação seja eficiente e preciso, é fundamental a segmentação da imagem para que apenas a luz do laser refletida pela superfície seja analisada e processada. Por isso, uma análise do espectro de luz, do comprimento de onda ou da intensidade de cores do laser que é refletida é interessante. Como foi explicado na seção 2.3.2 deste documento, a construção de histogramas da imagem é uma forma de analisar a distribuição da intensidade de pixel em uma imagem.

O primeiro passo então foi a construção de um histograma separado pelos canais RGB (vermelho, verde, azul) da Figura 4.3. A Figura 4.4 mostra o resultado do histograma.

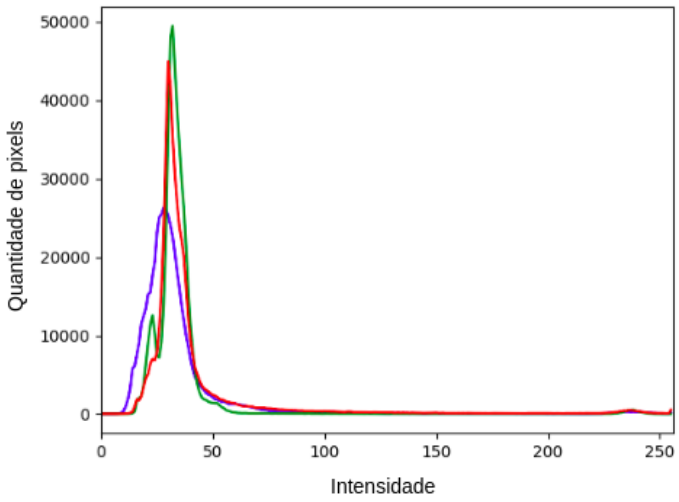


Figura 4.4: Histograma referente a imagem original. A cor vermelha representa o canal R, a cor verde o canal G e a cor azul o canal B.

Pelo histograma obtido, percebe-se que existe uma concentração muito grande de pixels com intensidade entre 0 e 50. Isso indica uma grande quantidade de pixels escuros, com baixa intensidade.

Para melhorar a análise, a imagem original foi cortada apenas na região em que a luz refletida do laser está presente (Figura 4.5).

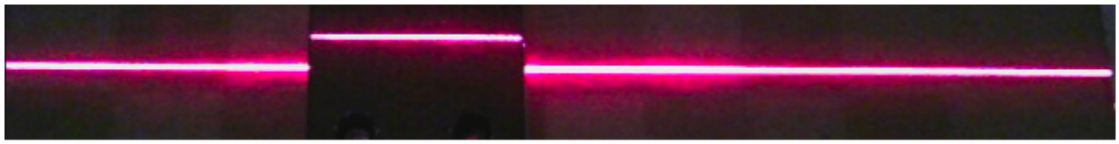


Figura 4.5: Imagem original cortada.

A Figura 4.6 representa o histograma da imagem original cortada.

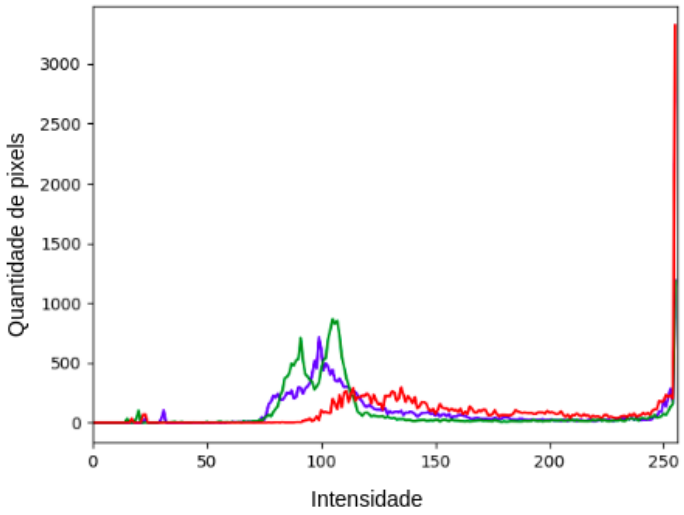


Figura 4.6: Histograma da imagem cortada.

Neste histograma percebe-se ainda uma quantidade considerável de pixels de baixa intensidade próximos a valor 100. Porém, os pixels de maior intensidade são os que interessam para a segmentação, pois são os que representam a reflexão do laser. Por isso, ampliou-se a representação gráfica para uma melhor análise dos pixels de alta intensidade (Figura 4.7).

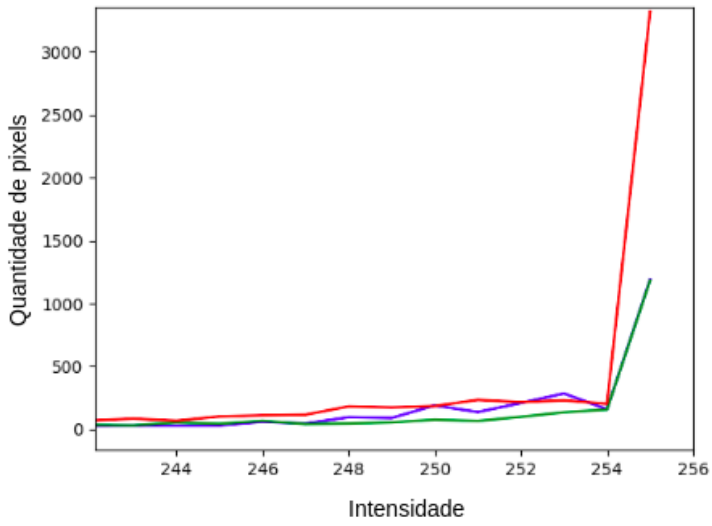
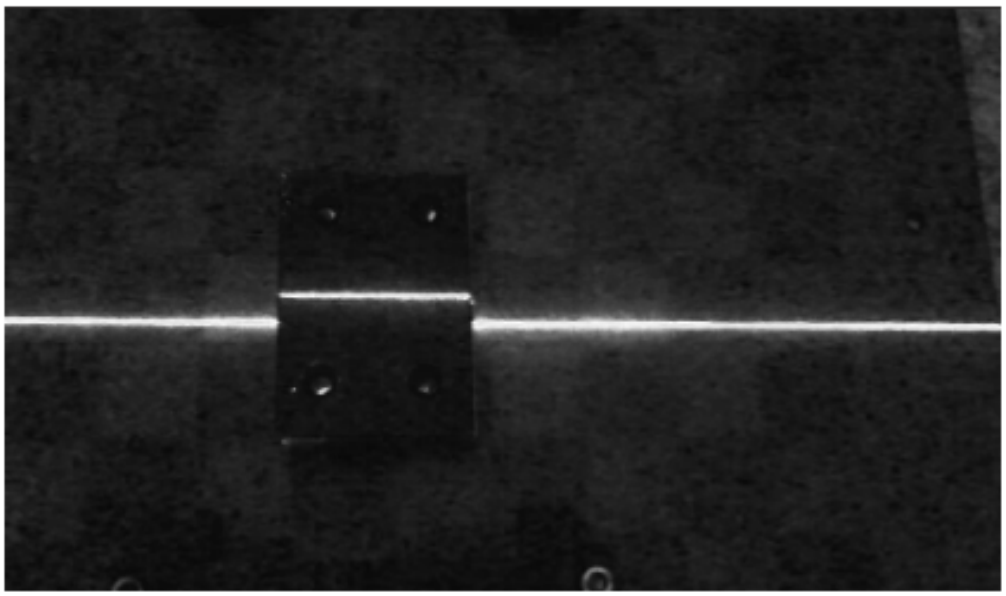
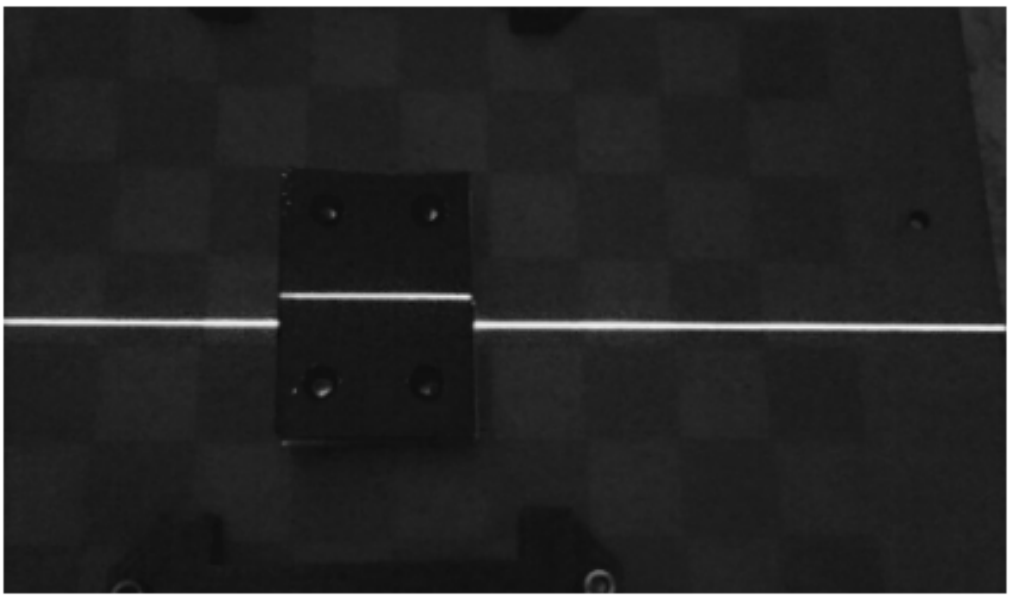


Figura 4.7: Histograma ampliado da imagem cortada.

Percebe-se que há um pico de pixels com intensidades próximas a 255, tanto para o canal R (vermelho), quanto para o canal G (verde). Segmentou-se assim, para meios de comparação, a imagem original tanto para o canal R, quanto para o canal G (Figura 4.8).



(a) Canal R da imagem original;



(b) Canal G da imagem original;

Figura 4.8: Segmentação por canal da imagem original.

Lembrando que o objetivo dessa análise é obter a região em que existe uma variação brusca da intensidade de pixel, ou seja, a região que representa o centro da reflexão do laser. Em volta do centro de reflexão existem, reflexões consideradas secundárias (ruídos) de intensidade minimamente inferiores, que podem atrapalhar consideravelmente a obtenção de dados precisos. Por isso, é necessário filtrar bem esses ruídos.

Pela Figura 4.8, é fácil perceber que o resultado da segmentação pelo canal G é melhor do que pelo canal R. Apesar do canal R ter um pico muito maior em torno do valor 255, analisando

o histograma da imagem ampliada (Figura 4.7), percebe-se que no canal G, a presença de pixels abaixo do valor de intensidade 254 é menor do que nos outros canais. Esses pixels representam em sua maioria a reflexão de luz secundária do laser. Isso pode explicar a presença menor de ruídos no canal G.

Aplicou-se então um *threshold* de valor mínimo 245 e valor máximo de 255 sobre a segmentação da imagem original pelo canal G. Ou seja, pixels com valores abaixo de 245 são redefinidos com o valor 0, enquanto pixels entre 245 e 255 são redefinidos com o valor 255. A Figura 4.9 representa o resultado desse processo.

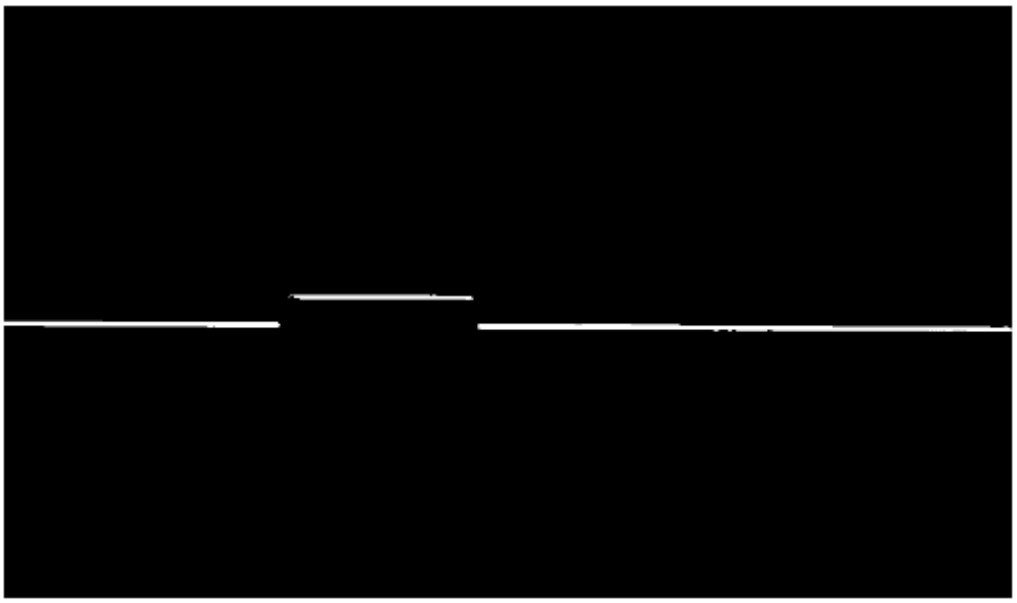


Figura 4.9: Segmentação por canal e threshold da imagem original.

A reflexão do laser torna-se então bem definida na imagem de resultado da segmentação.

Vale salientar neste documento que a reflexão da luz pode atuar de maneira desigual dependendo da superfície do objeto a ser escaneado. Objetos de superfícies diferentes refletem diferentes comprimentos de onda, por isso os histogramas de diferentes objetos podem apresentar pequenas variações. O ideal é criar e analisar histogramas para cada tipo de objeto.

O código completo da geração de histogramas e da segmentação por canais e *threshold* está presente no Apêndice B.

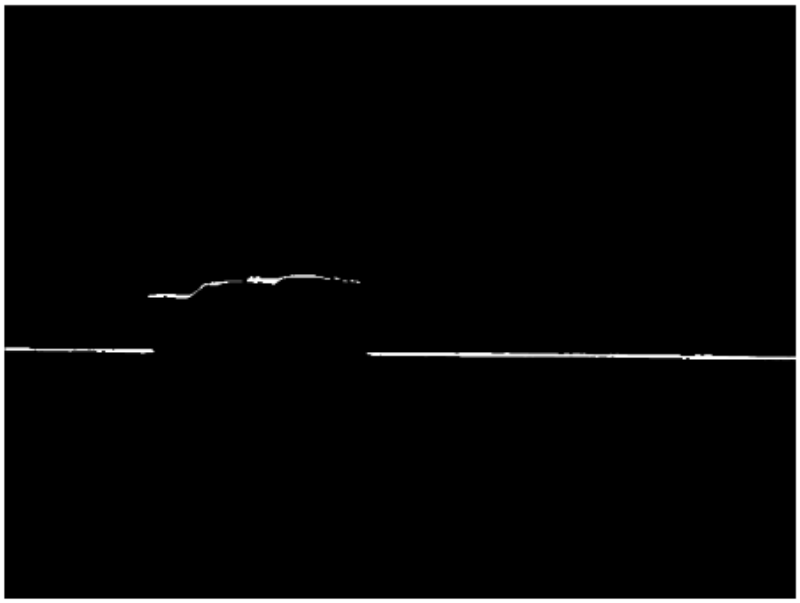
4.3 Filtros e Detecção de Borda

A imagem segmentada ainda pode apresentar ruídos consideráveis. É interessante otimizar ainda mais a imagem com a aplicação de filtros. Como foi visto na seção 2.3.3, o *Gaussian Filter* é um excelente instrumento para suavizar e remover ruídos de uma imagem, além de, se comparado aos filtros disponíveis, preserva bem os contornos na imagem. A suavização na imagem torna também o processo de detecção de borda mais eficiente, removendo possíveis variações bruscas na

detecção de superfícies, ocasionada principalmente pela presença de ruídos.

A detecção de bordas é necessária neste processo, pois a detecção de superfícies e retas se torna mais eficiente com bordas bem definidas. Para isso, utilizou-se o *Canny Filter* (seção 2.3.4). A biblioteca OpenCV possui a função *Canny*, que aplica o próprio *Canny Filter* e ainda inclui também a aplicação de um Gaussian Filter.

A Figura 4.10 apresenta o resultado da aplicação do *Canny Filter* sobre um exemplo de imagem segmentada gerada pelo scanner.



(a) Exemplo de imagem segmentada;

(b) Aplicação de Canny Filter sobre imagem segmentada;

Figura 4.10: Aplicação do Canny Filter.

4.4 Reta de Referência

Para obter a altura em pixels de cada ponto da superfície do objeto escaneado, é necessário fazer a relação de cada pixel na imagem com a reta de referência definida pela reflexão do laser sobre a base do scanner. Por isso, é importante que se defina a posição dessa reta na imagem e a sua inclinação.

Para se entender melhor este processo, o plano cartesiano de pixels da imagem deve ser explicado. O eixo de coordenadas na horizontal da imagem é definido com x e o eixo de coordenadas na vertical da imagem é definido como y. A Figura 4.11 demonstra o posicionamento do plano cartesiano nas imagens.

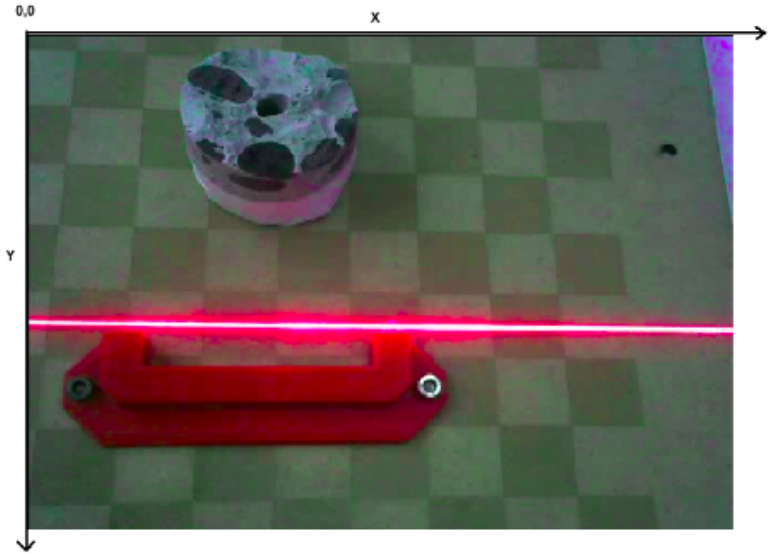


Figura 4.11: Plano cartesiano.

A primeira coisa a se fazer então é captar uma imagem pelo scanner sem a presença de um objeto sob a emissão do laser. Com essa imagem, deve-se definir a região de interesse, onde está a reflexão do laser, para análise.

A câmera usada no *scanner* tem como saída imagens na dimensão 480x640 pixels. Para determinar a região de interesse, colocou-se a imagem representada dentro do plano cartesiano. Com isso é possível definir em pixels a região de interesse como é mostrado na Figura 4.12.

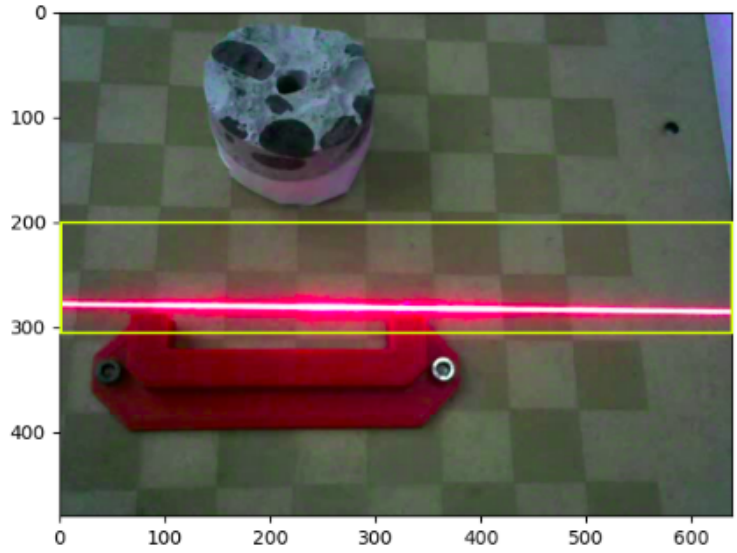


Figura 4.12: Região de interesse.

A região de interesse foi determinada como a área entre os pixels 200 e 300 no eixo y. Aplicou-se a segmentação apresentada na seção 4.2 e a detecção de bordas (seção 4.3) sobre a região de interesse. O resultado é representado na Figura 4.13.

Figura 4.13: Bordas da região de interesse.

A Figura 4.13 define então os pontos que representam a reflexão do laser. O que deve ser feito agora é obter a reta que cruza todos esses pontos ou que cruza a maioria dos pontos. Isso pode ser feito pela aplicação do *Hough Transform* sobre a imagem (Seção 2.3.5).

A biblioteca OpenCV fornece a função *HoughLinesP*, que aplica o *Hough Transform* sobre a

imagem para obter as retas que cruzam os pontos.

O primeiro argumento da função é a imagem de onde as retas devem ser detectadas.

O segundo e o terceiro parâmetros definem a resolução do acumulador *Hough*, que é um fator bidimensional que representa o tamanho do quadrado do *grid* no *Hough Space*. Como foi explicado na seção 2.3.5, o quadrado que possui uma maior concentração de intersecções representa os coeficientes ρ e θ da reta que passa de forma mais consistente sobre pontos. Quanto maior o quadrado, menor é a precisão em que as retas serão detectadas. Porém, quadrados muito pequenos podem gerar inacurácias e um longo tempo de processamento. Depois de alguns teste realizados, a dimensão do quadrado foi definida como: $\rho = 2$ e $\theta = 1$.

O quarto parâmetro define o *threshold*, ou seja, ele define o mínimo de intersecções que devem ocorrer dentro de um quadrado para que possa ser definida uma reta. Após alguns testes, o valor definido foi de $j = 20$.

A Figura 4.14 demonstra o resultado deste processo com os valores definidos e com outros valores de teste.

(a)

(b)

Figura 4.14: Resultados das retas criadas com imagem original escurecida ao fundo (apenas para visualização). (a) Retas criadas com os parâmetros: $\rho = 2$; $\theta = 1$;$j = 20$; (b) Retas criadas com os parâmetros: $\rho = 5$; $\theta = 2$;$j = 20$;

Observando atentamente a Figura 4.14, percebe-se a presença de várias retas na cor azul sobre o feixe do laser. Essas retas possuem diferentes tamanhos e coeficientes angulares. Por isso é necessário obter a média, para que apenas uma reta seja definida. Para isso, basta fazer a média dos coeficientes angulares das retas e a média de seus coeficientes lineares.

Aplicando a média tanto para o coeficiente angular quanto para o coeficiente linear, têm-se os

coeficientes da reta de referência.

O resultado está presente na Figura 4.15.

Figura 4.15: Reta de referência sobre imagem original escurecida (apenas para visualização).

O *scanner* foi construído para que a linha do laser ficasse paralela ao eixo horizontal x da imagem. Porém para garantir que a linha de referência esteja paralela, aplicou-se um algoritmo para rotacionar cada imagem gerada pela câmera usando como parâmetro o coeficiente angular da reta de referência. Para isso, utilizou-se a função *getRotationMatrix2D* da biblioteca OpenCV.

O código completo para obtenção da reta de referência está no Apêndice C. O processo de obtenção da reta de referência é realizado apenas para o primeiro frame do vídeo gerado pelo *scanner*. Considera-se que a linha de referência não é alterada com o passar da varredura sobre o objeto escaneado.

Capítulo 5

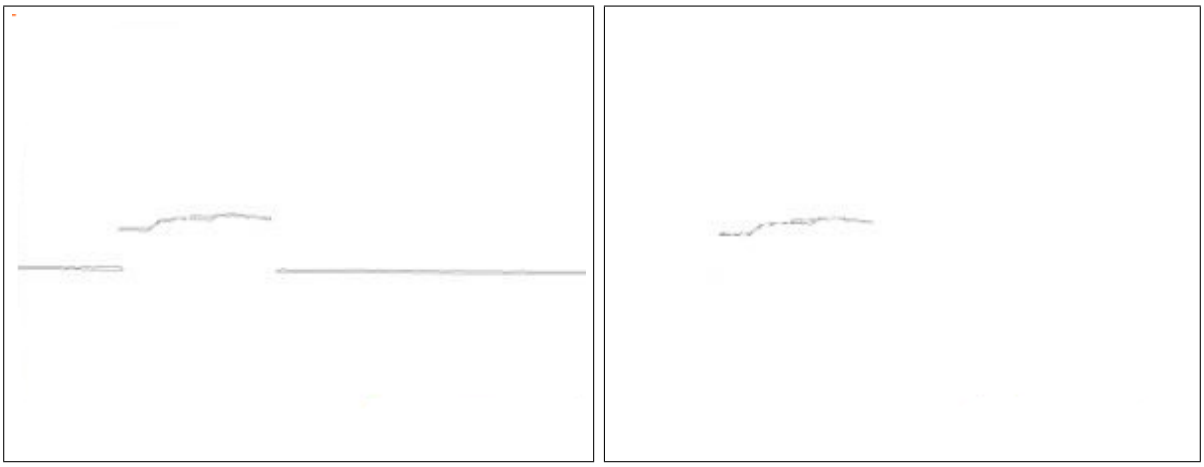
Reconstrução 3D

Neste capítulo, é explicado o processo de reconstrução 3D a partir das imagens segmentadas e filtradas geradas pela câmera do *scanner*. Primeiramente então, é mostrado como a nuvem de pontos é construída, considerando que com a reta de referência definada, uma nova região de interesse é formada. Demonstra-se aqui também a concatenação das imagens ao longo do eixo z, para que a nuvem esteja no espaço tridimensional. No último tópico do capítulo, é explicado o uso do *Marching Cubes Algorithm* para gerar faces e vértices a partir da nuvem de pontos e a obtenção do arquivo final do modelo digital.

5.1 Nuvem de pontos

É necessária a construção da nuvem de pontos no plano tridimensional para que a reconstrução digital 3D do objeto seja possível.

Obtida a reta de referência, explicada na seção anterior, pode-se definir que apenas os pontos acima dessa reta devem ser considerados, ou seja, pontos que estão abaixo no eixo y são, logicamente, considerados ruídos . Por isso, qualquer ponto da imagem de gradiente gerada pela segmentação da seção 4.2 que esteja abaixo da reta de referência é desprezado, como pode ser visto na Figura 5.1. Isso pode ser feito definindo uma nova região de interesse que está acima da reta de referência.



(a) (b)

Figura 5.1: Remoção de pontos abaixo da reta de referência. As cores das imagens foram invertidas para uma melhor visualização dos pontos. (a) Imagem segmentada original; (b) Remoção de pontos abaixo da reta de referência (fora da nova região de interesse).

Com isso, tem-se na imagem apenas os pontos referentes à superfície do objeto. Para que o modelo digital seja um modelo sólido é necessário o preenchimento da nuvem de pontos abaixo dos pontos da superfície. Ou seja, se existe um ponto em um coluna (eixo y da imagem) os pixels abaixo dele até a reta de referência devem ser preenchidos. Isso pode ser feito com uma varredura sobre as colunas da imagem, procurando o ponto de maior altura em relação à reta de referência. Encontrado esse ponto, os pontos abaixo à eles são definidos com o valor 255. A Figura 5.2 apresenta o resultado desse processo.

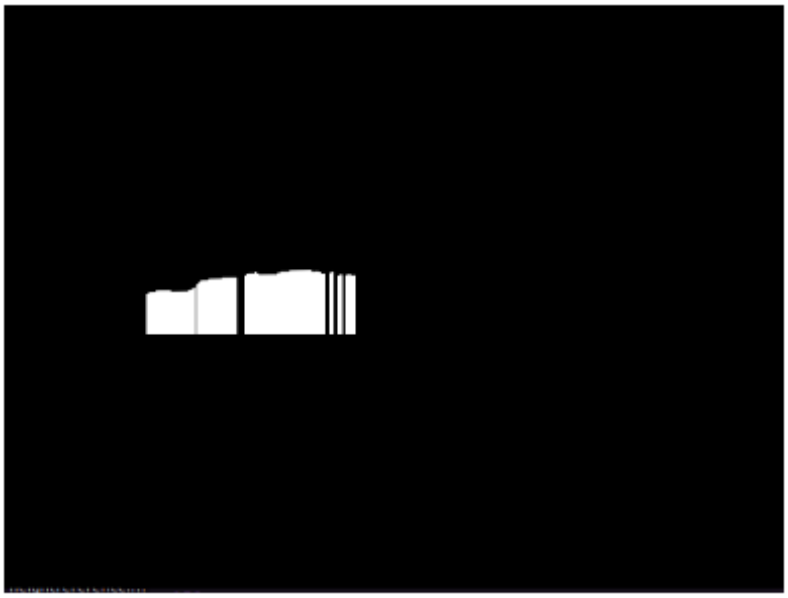


Figura 5.2: Preenchimento de pontos abaixo da superfície.

Realizado esse processo, tem-se então a imagem de profundidade para cada *frame* captado do

objeto.

Até aqui, foi vista apenas a obtenção de pontos no plano bidimensional (x,y), apenas no sistema de coordenadas cartesiano da imagem. Precisam-se definir também os pontos no espaço tridimensional (x,y,z) para a reconstrução 3D.

O *scanner* funciona com a captação da câmera, que está voltada para a reflexão do laser. Essa estrutura câmera/laser se desloca no eixo z para que o laser passe por toda a superfície do objeto. Durante esse deslocamento, a câmera capta diferentes frames. Ou seja, cada frame representa um posicionamento diferente no eixo z, e a partir de cada frame, uma imagem de profundidade é gerada.

O que deve ser feito agora é a concatenação no eixo z das imagens de profundidade geradas. A biblioteca Numpy da linguagem de programação Python fornece a função *dstack* que facilita muito essa concatenação. Ela tem como entrada matrizes bidimensionais, no caso deste trabalho são as imagens de profundidade, e os concatena a longo do eixo z, gerando como saída um parâmetro tridimensional. Ou seja, cada imagem de profundidade representa uma unidade no eixo z, neste caso 1 pixel. Este valor depende diretamente da velocidade de deslocamento da estrutura câmera/laser e da captação da câmera de frames por segundo. Por isso, este valor de 1 pixel será ajustado na calibração realizada na seção 5.3 deste documento.

5.2 Modelo Digital

Já com a nuvem de pontos definida tridimensionalmente, ainda há um problema a ser resolvido para que a reconstrução 3D seja criada. Os pontos da nuvem ainda não apresentam nenhuma conectividade. Para obter um modelo digital do objeto é necessário criar uma malha de triângulos que representam a superfície do objeto. A formação dessa malha pode ser obtida pelo *Marching Cubes Algorithm*, que por meio da nuvem de pontos cria faces e vértices que se conectam formando a malha. A lógica do *Marching Cubes Algorithm* foi explicada na seção 2.3.7 deste documento.

A biblioteca *scikit-image* para a linguagem de programação Python fornece a função measure.marching_cubes que aplica sobre uma nuvem de pontos o *Marching Cubes Algorithm* e retorna a malha da superfície.

Com isso tem-se agora os pontos interpolados em vértices e faces que definem a superfície digitalizada do objeto.

O último passo é então exportar essa superfície para um arquivo no formato STL[1], formato que pode ser renderizado na maioria de softwares de visualização 3D. Para isso foi utilizada a biblioteca numpy-stl da linguagem de programação Python, que é uma biblioteca simples capaz de trabalhar com arquivos STL e objetos tridimensionais de uma maneira rápida e fácil. O arquivo STL é formado passando os vértices e os triângulos obtidos pelo *Marching Cubes Algorithm* para a função *mesh* da biblioteca.

[1] STereoLithography - Formato de arquivo que descreve a geometria da superfície de um objeto 3D.

A Figura 5.3 apresenta um exemplo de renderização da reconstrução 3D de um objeto escaneado pelo módulo de triangulação desenvolvido neste trabalho.

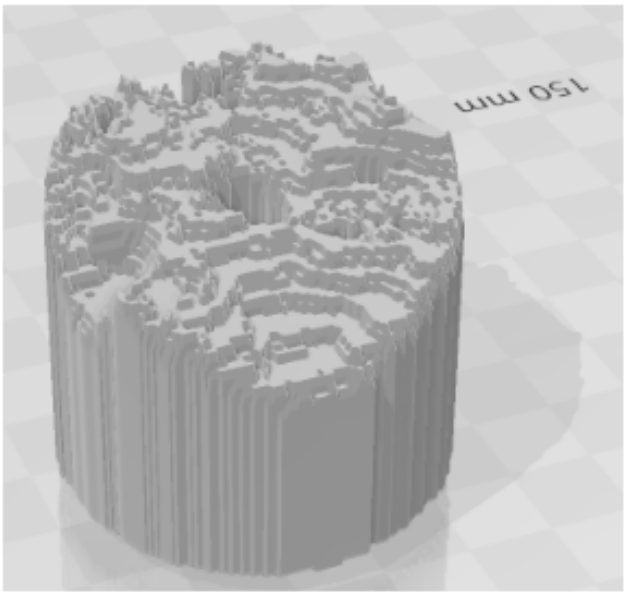


Figura 5.3: Exemplo do resultado da reconstrução 3D.

O código completo implementado para a reconstrução 3D está no Apêndice D.

5.3 Calibração dos Parâmetros do Módulo

O modelo gerado na seção anterior ainda possui um problema. Os valores das dimensões do modelo ainda estão em pixels, ou seja, a calibração para que o modelo final fique nas dimensões reais do objeto escaneado é necessária.

A conversão de pixels para milímetros pode ser feita pelas equações abaixo:

$$\Delta X = \frac{\Delta x}{n}; \tag{5.1}$$

$$\Delta Y = \frac{\Delta y}{nsin(\alpha)}; \tag{5.2}$$

$$\Delta Z = \frac{\Delta z v}{fps}, \tag{5.3}$$

em que, ΔX, ΔY, ΔZ são as dimensões reais em milímetros do objeto, n é a ampliação ótica das lentes da câmera (ou seja, é fator de conversão de pixels para milímetros), α é o ângulo entre o laser e a câmera, v é a velocidade do deslocamento da estrutura câmera/laser em mm/s, fps é a taxa de frames por segundo da câmera e Δx, Δy e Δz são as dimensões em pixels do objeto escaneado.

A equação 5.1 representa a conversão dos pontos do sistema de coordenadas da imagem em x para o sistema de coordenadas do objeto em X. Nesse processo não é necessário a aplicação de uma triangulação, por isso apenas a conversão de pixels para mílimetros, por meio do fator n, é necessária.

A equação 5.2 representa a conversão da localização dos pontos do sistema de coordenadas da imagem em y para o sistema de coordenadas do objeto em Y. Este processo envolve a triangulação para obtenção da altura do objeto. Isso foi explicado detalhamente na seção 2.2 deste documento.

A equação 5.3 representa a conversão da localização dos pontos do sistema de coordenadas da imagem em z para o sistema de coordenadas do objeto em Z. Cada imagem de profundidade obtida representa, originalmente, uma unidade no eixo z, porém é necessário obter o valor em milímetros dessa unidade. Para isso, tem-se que descobrir o quanto a máquina se desloca em Z para cada captação de $frame$ (uma unidade em z) na câmera. Os valores de fps e v são usados para essa obtenção. Com esses valores definidos, a equação 5.3 é capaz de realizar a conversão.

Para se ter uma conversão precisa, é necessário obter então todos os parâmetros das equações acima. Para isso, um objeto de dimensões conhecidas deve ser escaneado.

Um cubo de arestas de dimensões $50 \pm 0.02mm$ foi impresso em 3D para que pudesse ser usado na calibração. Esse cubo foi então escaneado e o processo de reconstrução 3D realizado. O modelo digital gerado teve como dimensões os valores finais: $\Delta x = 200$ pixels; $\Delta y = 57$ pixels; $\Delta z = 90$ pixels;

Substituindo os valores de ΔX e Δx na equação 5.1 tem-se que:

$$50 = \frac{200}{n}; \tag{5.4}$$

$$n = 4\ pixels/mm. \tag{5.5}$$

Substituindo os valores de ΔY, Δy e n na equação 5.2 tem-se que:

$$50 = \frac{57}{4sin(\alpha)}; \tag{5.6}$$

$$sin(\alpha) = 0.285; \tag{5.7}$$

$$\alpha = 16.56^{\circ}. \tag{5.8}$$

Por meio das especificações técnicas da câmera, sabe-se que $fps = 30frames/s$. Então, substituindo os valores de ΔZ, Δz e fps na equação 5.3 tem-se que:

$$50 = \frac{90v}{30}; \tag{5.9}$$

$$v = 16.67\ mm/s. \tag{5.10}$$

Um outro cubo, porém com arestas de dimensões $75 \pm 0.02\ mm$, também foi utilizado. Esse cubo foi então escaneado e o processo de reconstrução 3D realizado. O modelo digital gerado teve como dimensões os valores finais: $\Delta x = 300$ pixels; $\Delta y = 84$ pixels; $\Delta z = 135$ pixels;

Substituindo os valores de ΔX e Δx na equação 5.1 tem-se que:

$$75 = \frac{300}{n}; \tag{5.11}$$

$$n = 4\ pixels/mm. \tag{5.12}$$

Substituindo os valores de ΔY, Δy e n na equação 5.2 tem-se que:

$$75 = \frac{84}{4sin(\alpha)}; \tag{5.13}$$

$$sin(\alpha) = 0.280; \tag{5.14}$$

$$\alpha = 16.20°. \tag{5.15}$$

Então, substituindo os valores de ΔZ, Δz e fps na equação 5.3 tem-se que:

$$75 = \frac{135v}{30}; \tag{5.16}$$

$$v = 16.67\ mm/s. \tag{5.17}$$

Percebe-se que para os dois casos os valores de n e v possuíram os mesmos valores, porém o mesmo não ocorreu para o valor de α. Por isso, uma média aritmética foi aplicada sobre os dois valores de α. O valor obtido foi de $16,38°$.

O valor de v de $16.67mm/s$ encontrado para os dois cubos é o mesmo que foi fornecido pelas especificações técnicas da máquina na tabela 3.2, o que demonstra um bom processo de calibração no eixo Z e uma boa precisão do motor.

Como todos os valores dos parâmetros obtidos, foram aplicadas ao final do processo de reconstrução 3D as equações 5.1, 5.2 e 5.3 para mudança de escala do modelo digital gerado.

A calibração, aqui realizada, apenas obteve os parâmetros de conversão do modelo digital com dimensões em pixels para um modelo digital com dimensões em milímetros. Não foram levados em consideração os possíveis problemas de fabricação e montagem da máquina.

Capítulo 6

Resultados e Análise

Este capítulo mostrará os resultados do escaneamento e reconstrução de três corpos de prova. As dimensões do modelo digital e do modelo real são comparadas. E, ao final do capítulo, uma análise sobre os possíveis motivos dos erros encontrados é feita.

Com os parâmetros todos definidos pelo processo de calibração, os testes sobre objetos reais podem ser realizados.

O primeiro teste foi realizado com o escaneamento do corpo de prova demonstrado na Figura 6.1. Este corpo de prova é um objeto formado por alumínio de superfície regular e com furos passantes em suas extremidades.

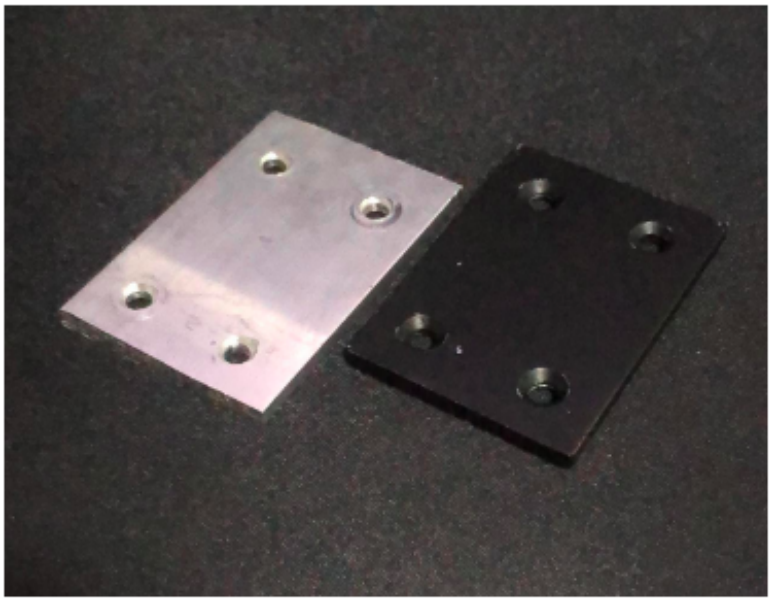


Figura 6.1: Corpo de prova 1.

Para simplificar o processo de análise, apenas se utilizou das maiores dimensões do objeto, em cada eixo no plano cartesiano tridimensional, como modo de comparação. O corpo de prova (objeto real) possui como dimensões máximas, em cada eixo, os valores: $\Delta X = 40.0 \pm 0.02\ mm$;

$\Delta Y = 1.8 \pm 0.02\ mm$; $\Delta Z = 60.0 \pm 0.02\ mm$.

O escaneamento e o processo de reconstrução 3D foram executados em 38 segundos. No processo não foram consideradas possíveis variações de dimensão devido a temperatura do objeto sob o laser. O modelo digital gerado pela reconstrução tridimensional do corpo de prova 1 está representado na Figura 6.2.

Figura 6.2: Modelo digital do corpo de prova 1.

O modelo digital foi então criado a partir da reconstrução do objeto. O modelo foi renderizado no software 3DBuilder da empresa Microsoft para a obtenção das suas dimensões, porém poderia ter sido renderizado em qualquer programa de visualização 3D de modelos digitais que suporta arquivos STL. Sistemas como esses apresentam modelos digitais de forma fiel às suas dimensões. A Tabela 6.1 apresenta os valores das dimensões do corpo de prova 1 e do modelo digital gerado, além do erro obtido no processo.

Dimensões	Corpo de Prova	Modelo Digital	Erro
ΔX	$40.0 \pm 0.02\ mm$	$39.0\ mm$	$-1.0\ mm$
ΔY	$1.8 \pm 0.02\ mm$	$2.6\ mm$	$+1.2\ mm$
ΔZ	$60.0 \pm 0.02\ mm$	$61.3\ mm$	$+1.3\ mm$

Tabela 6.1: Dimensões para o corpo de prova 1.

O segundo teste foi realizado com o escaneamento do corpo de prova demonstrado na Figura 6.3. Este corpo de prova é um objeto formado por concreto de superfície irregular e com o furo passante em seu centro.

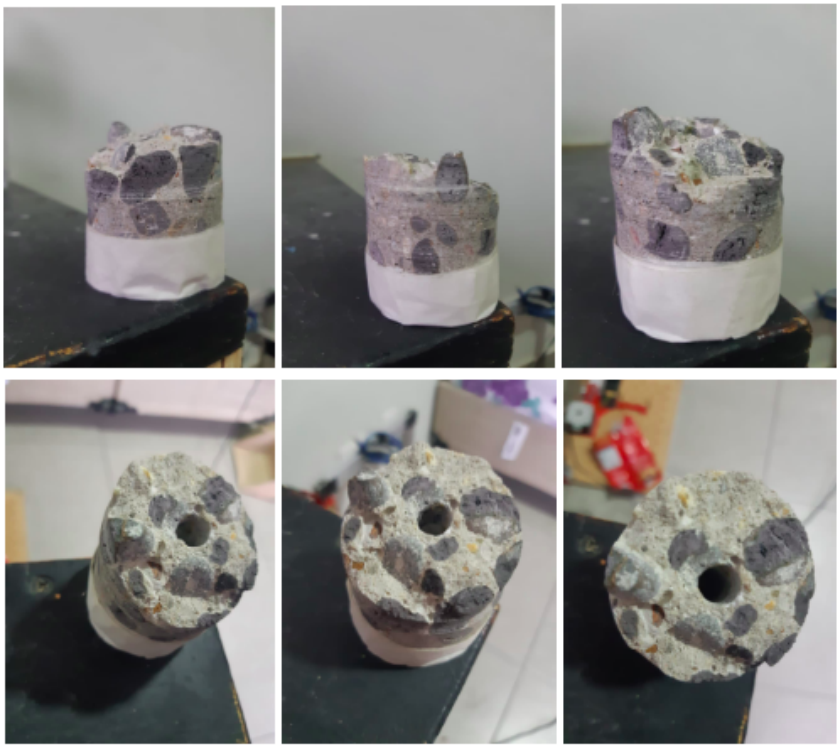


Figura 6.3: Corpo de prova 2.

O corpo de prova possui como dimensões máximas os valores: $\Delta X = 46.0 \pm 0.02\ mm$; $\Delta Y = 46.0 \pm 0.02\ mm$; $\Delta Z = 54.7 \pm 0.02\ mm$.

O escaneamento e o processo de reconstrução 3D foram executados em 44 segundos. No processo não foram consideradas possíveis variações de dimensão devido a temperatura do objeto sob o laser. O modelo digital gerado pela reconstrução tridimensional do corpo de prova 2 está representado na Figura 6.4.

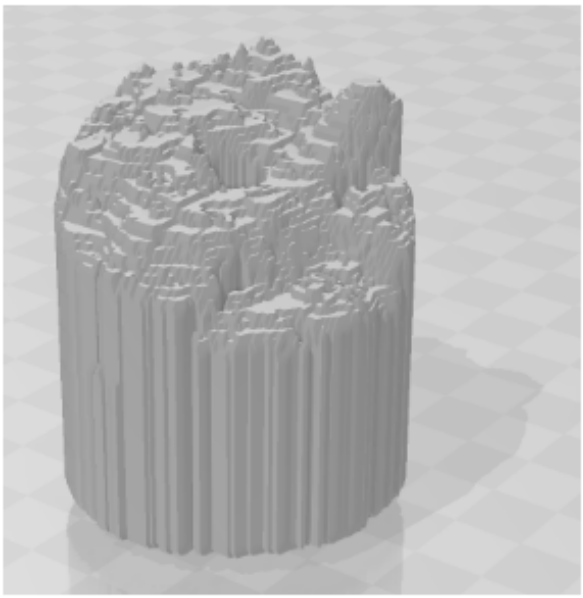


Figura 6.4: Modelo digital do corpo de prova 2.

A Tabela 6.2 apresenta os valores das dimensões do corpo de prova 2 e do modelo digital gerado, além do erro obtido no processo.

Dimensões	Corpo de Prova	Modelo Digital	Erro
ΔX	$46.0 \pm 0.02\ mm$	$44.6\ mm$	$-1.4\ mm$
ΔY	$46.0 \pm 0.02\ mm$	$47.6\ mm$	$+1.6\ mm$
ΔZ	$54.7 \pm 0.02\ mm$	$56.0\ mm$	$+1.3\ mm$

Tabela 6.2: Dimensões para o corpo de prova 2.

O terceiro teste foi realizado com o escaneamento do corpo de prova demonstrado na Figura 6.5. Este corpo de prova também é um objeto formado por concreto de superfície irregular e com o furo passante em seu centro.

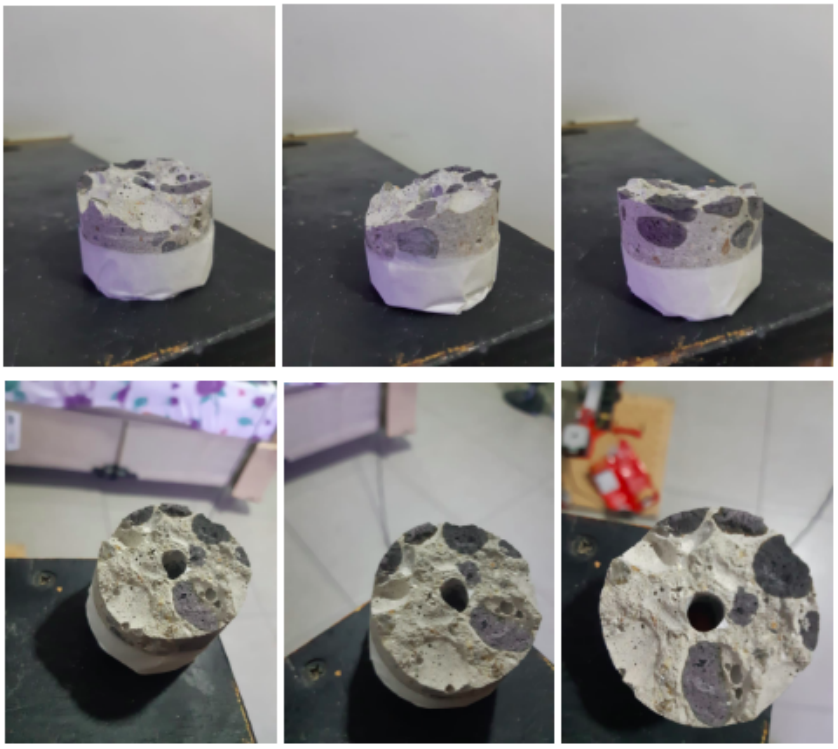


Figura 6.5: Corpo de prova 3.

O corpo de prova 3 possui como dimensões máximas os valores: $\Delta X = 46.2 \pm 0.02\ mm$; $\Delta Y = 46.2 \pm 0.02\ mm$; $\Delta Z = 37.0 \pm 0.02\ mm$.

O escaneamento e o processo de reconstrução 3D foram executados em 42 segundos. No processo não foram consideradas possíveis variações de dimensão devido a temperatura do objeto sob o laser. O modelo digital gerado pela reconstrução tridimensional do corpo de prova 3 está representado na Figura 6.6.

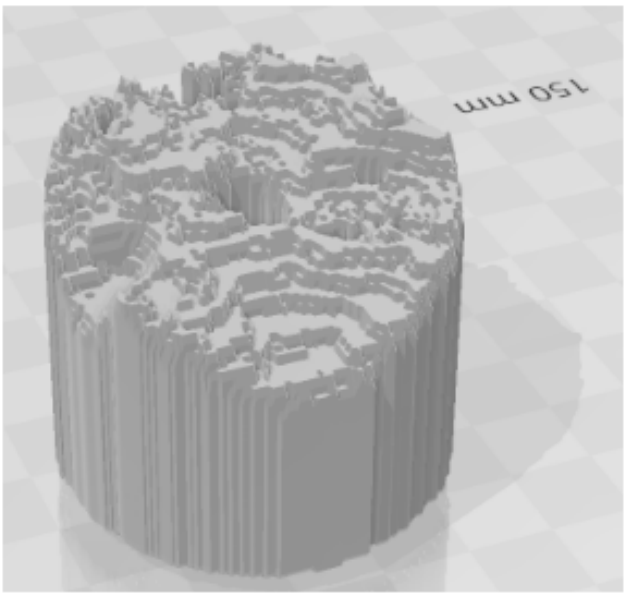


Figura 6.6: Modelo digital do corpo de prova 3.

A Tabela 6.3 apresenta os valores das dimensões do corpo de prova 3 e do modelo digital gerado, além do erro obtido no processo.

Dimensões	Corpo de Prova	Modelo Digital	Erro
ΔX	$46.2 \pm 0.02\ mm$	$44.7\ mm$	$-1.5\ mm$
ΔY	$46.2 \pm 0.02\ mm$	$47.7\ mm$	$+1.5\ mm$
ΔZ	$37.0 \pm 0.02\ mm$	$38.8\ mm$	$+1.8\ mm$

Tabela 6.3: Dimensões para o corpo de prova 2.

Percebe-se que para os três corpos de prova testados, o erro de dimensão, para as três dimensões, está entre $1mm$ e $2mm$. Um dos motivos que pode explicar esse erro é a espessura da linha formada pela reflexão do laser. Essa reflexão foi medida várias vezes nas imagens originais (sem aplicação de filtros e segmentação) e teve-se em média um valor de 5 pixels no eixo y da imagem. Se convertido para plano real em milímetros com o uso da equação 5.2, tem-se o valor de $4.4mm$. Esse é um valor considerável para o escaneamento. Os filtros e as segmentações aplicadas sobre o feixe do laser restringem mais esse valor de 5 pixels, porém podem não ter sido suficientes para obter o centro exato do feixe, em relação ao eixo y, da imagem.

A aplicação do *Canny Filter* sobre a o feixe do laser gera ainda a manutenção de múltiplos pontos em uma mesma coluna na imagem, ou seja, mais de um ponto em um mesmo valor no eixo x. Neste trabalho, foi considerado o ponto em cada coluna que esteja mais distante da reta de referência para representação da superfície do objeto. Isso não é otimizado e pode gerar erros consideráveis.

Outro problema é a diminuição de intensidade e espessura nas extremidades do objeto. Isso dificulta uma boa segmentação do laser nas bordas do objeto, o que pode explicar os resultados dos

corpos de prova, onde todos os modelos digitais obtidos tiveram no eixo x uma menor dimensão do que o objeto real.

Além disso, no processo de calibração não foi considerado possíveis erros de fabricação e montagem do *scanner*. Não foram feitos testes para detectar possíveis variações de precisão da máquina em relação à mudanças de posição na estrutura camera/laser, o que pode também gerar erros consideráveis.

Capítulo 7

Conclusão e Trabalhos Futuros

Este trabalho implementou um método de reconstrução 3D a partir de um módulo de triangulação a laser de baixo custo.

Com as imagens captadas pela câmera do scanner, primeiramente foi obtida a reta de referência, que representa a reflexão do laser sobre a base do scanner sem a interferência de objetos externos. Para isso, foi utilizado o *Hough Transform* para detecção de retas a partir de pontos posicionados sobre uma imagem.

Filtros e a segmentação por histograma foram usados para definir bem a reflexão do laser sobre o objeto. Todos os frames captados pela câmera foram segmentados, com isso foi possível obter os pontos que representam a superfície do objeto escaneado.

Obtidos os pontos em cada frame (as imagens de profundidade), aplicou-se então um método para concatenação das imagens de profundidade para que a nuvem de pontos no plano tridimensional fosse formada. Com nuvem de pontos definida, o *Marching Cubes Algorithm* foi usado para criar uma malha formada por triângulos e vértices que representam a superfície. Então essa malha foi exportada em um arquivo STL, que pode ser renderizado em quase todos os softwares de renderização 3D no mercado.

Por fim, os resultados desse trabalho apresentaram como erro apenas o valor entre $1mm$ e $2mm$. *Scanners* de triangulação a laser profissionais encontrados no mercado apresentam precisão entre 0.1 a 0.25 milímetros (Hirano, 2020). Os filtros apresentaram uma boa eficiência na remoção de ruídos das imagens captadas, e a segmentação conseguiu definir bem a reflexão do laser. A próxima seção deste documento apresentará possíveis trabalhos para melhorar a precisão do *scanner* aqui apresentado.

Foi feita uma análise simples sobre os modelos digitais de três corpos de prova, considerando as dimensões máximas de cada eixo.

7.1 Trabalhos Futuros

O complemento deste trabalho seria uma análise aprofundada sobre os filtros aplicados sobre a reflexão do laser, com o intuito de diminuir o erro obtido. Como foi explicado na análise dos resultados, a intensidade dos pixels próximos às extremidades do objeto diminui consideravelmente, o que dificulta uma boa segmentação. Para resolver esse problema, uma análise melhor sobre os processos de segmentação pode ser feita para que as extremidades da superfície do objeto não sejam perdidas.

Depois da aplicação do *Canny Filter*, mais de um ponto fica presente em uma mesma coluna. O que pode ser feito aqui é a aplicação de um algoritmo mais otimizado para encontrar apenas um ponto que representará melhor a superfície do objeto.

Este trabalho não apresentou no resultado final as cores do objeto. Um trabalho futuro interessante seria a obtenção das cores da superfície do objeto e sua representação no modelo digital final.

Outra aplicação interessante à esse trabalho, é o uso de filtros adaptativos sobre a nuvem de pontos para suavizar e diminuir ruídos da superfície final gerada.

A calibração de forma mais completa também pode ser aplicada sobre o projeto, com o intuito de amenizar possíveis erros de fabricação da máquina. A máquina pode ser também aperfeiçoada, com uma melhor e mais robusta estrutura mecânica, evitando possíveis desalinhamentos e imprecisões.

Além disso, a aplicação deste trabalho em algum ambiente real, como para a inspeção de superfícies de prédios, viadutos ou barragens, para detectar rachaduras, fissuras ou outras manifestações patológicas seria uma ótima área de estudo e de desenvolvimento.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] ALBUQUERQUE, L., A. (2006). Alinhamento de Imagens de Profundidade na Reconstrução 3D De Objetos de Forma Livre. Dissertação de Mestrado em Sistemas Mecatrônicos, Publicação ENM.DM-09A/06, Departamento de Engenharia Mecânica, Universidade de Brasília, Brasília, DF, 125p.

[2] Anderson, Ben. An Implementation of the Marching Cubes Algorithm. Carleton College Department of Computer Science, Northfield, USA. Disponível em: <`http://www.cs.carleton.edu/cs_comps/0405/shape/marching_cubes.html`>. Acesso em: 24 de nov. de 2020.

[3] An Introduction to Laser Triangulation Sensors. AZOSensors. Disponível em <https://www.azosensors.com/article.aspx?ArticleID=523>. Acesso em: jun. 2019.

[4] BRADSKI, G.; KAEHLER, A. Learning OpenCV. First Edition. United States of America. O'Reilly, September 2008.

[5] CHITYALA, R.; PUDIPEDDI, S.. Image Processing and Acquisition using Python. Chapman Hall/CRC Mathematical and Computational Imaging Sciences. CRC Press, 2014.

[6] CIPOLLA, R.; BATTIATO, S.; FARINELLA, G. M. Machine Learning for Computer Vision. [S.l.]: Springer Publishing Company, Incorporated, 2012.

[7] CONCI, A.; AZEVEDO, E.; LETA, F. R. Computação Gráfica: Teoria e Prática. Rio de Janeiro: Elsevier, v. 2, 2008. 407 p.

[8] DAVIES, E.R., Computer Vision: Principles, Algorithms, Applications, Learning. Fifth Edition. Royal Holloway, University of London, United Kingdom. Academic Press, 2018.

[9] DE ARAÚJO, S. A. Casamento de padrões em imagens digitais livre de segmentação e invariante sob transformações de similaridade. Universidade de São Paulo. [S.l.]. 2009.

[10] DESELAERS, T. Features for image retrieval. Rheinisch-Westfalische Technische Hochschule, Technical Report, Aachen, 2003.

[11] Duda, R. O. and P. E. Hart, "Use of the Hough Transformation to Detect Lines and Curves in Pictures,"Comm. ACM, Vol. 15, pp. 11–15 (1972).

[12] FREITAS, G. Metodologia e aplicabilidade da digitalização 3D a laser no desenvolvimento de moldes para calçados e componentes. 115 p. Dissertação (Mestr e em Engenharia) - Programa de Pós Graduação em Engenharia de Minas, Metalúrgica e Materiais, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2006.

[13] GHELLERE, Jhonattan Salvador. Detecção de Objetos em Imagens por meio da combinação de descritores locais e classificadores. Trabalho de Conclusão do Curso Superior de Ciência da Computação. Universidade Tecnológica Federal do Paraná. Medianeira, 2015, 90p.

[14] GONZALEZ, R. C.; WOODS, R. E. Digital Image Processing. 2. ed. [S.l.]: Prentice Hall, 2002.

[15] HIRANO, THIAGO Y., (2020). Projeto de Módulo de Trangulação/Scanner 3D de Baixo Custo com Visão Computacional. Projeto de Graduação em Engenharia Mecânica, Publicação FT.TG-n ◦022, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF, 57p.

[16] Hough, P.V.C. Method and means for recognizing complex patterns. U.S. Patent 3,069,654, Dec. 18, 1962.

[17] Ian Goodfellow, Yoshua Bengio, and Aaron Courville (2016). Deep Learning. MIT Press.

[18] LIBERMAN, F. Classificação de imagens digitais por textura usando redes Neurais. Rio Grande do Sul: Universidade Federal do Rio Grande do Sul. Instituto de Informática. Curso de Pós-Graduação em Ciência da Computação, 1997.

[19] LISIN, Dimitri A; MATTAR, Marwan A; BLASCHKO, Matthew B; LEARNED-Miller, Erik G; BENFIELD, Mark C. Combining local and global image features for object class recognition. Computer Vision and Pattern Recognition-Workshops, 2005. CVPR Workshops. IEEE Computer Society Conference on, 2005. 47-47.

[20] NIXON, M. S.; AGUADO, A. S. Feature Extraction and Image Processing. 1. ed.[S.l.]: Newnes, 2002.

[21] PAVI, S; BORDIN, F; VERONEZ, M., R.. O Uso do Laser Scanner Terrestre na Inspeção de Pontes e Viadutos de Concreto: uma Revisão Bibliográfica. Rio de Janeiro: VII Congresso Brasileiro de Pontes e Estruturas, 2014.

[22] Pears N., Liu Y., Bunting P.:“3D Imaging, Analysis and Applications”. London: Springer London, 2012.

[23] PEDRINI, H.; SCHWARTZ, W. R. Análise de Imagens Digitais: Princípios, Algoritmos e Aplicações. São Paulo: Thompson, 2008.

[24] PENATTI, O. A. B. Estudo comparativo de descritores para recuperação de imagens por conteúdo na web. Biblioteca Digital da Unicamp. [S.l.]. 2009.

[25] Poredoš, P.; Povšič, K.; Novak, B.; Jezeršek, M. Three-Dimensional Measurements of Bodies in Motion Based on Multiple-Laser-Plane Triangulation. Rev. Téc. Fac. Ing. Univ. Zulia 2015, 38, 53–61.

[26] RAOUI, Y.; BOUYAKHF, E. H. A. D. M.; REGRAGUI, F. Global and local image descriptors for Content Based Image Retrieval and object recognition. Applied Mathematical Sciences, 5, 2011. 2109-2136.

[27] Revisão de Conceitos em Projeção, Homografia, Calibração de Câmera, Geometria Epipolar, Mapas de Profundidade e Varredura de Planos, Maikon Cismoski dos Santos, 2012.

[28] Sansoni, G.; Trebeschi, M.; Docchio (2009), "F. State-of-The-Art and Applications of 3D Imaging Sensors in Industry, Cultural Heritage, Medicine, and Criminal Investigation". Sensors, vol 9, pág 568-601.

[29] SZELISKI, R. Computer vision: algorithms and applications. Springer, 2010.

[30] What is Laser Triangulation?. MoviMED. Disponível em <http://www.movimed.com/knowledgebase/what-is-laser-triangulation/>. Acesso em: jun. 2019.

[31] W.Lorensen, H.Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. Computer Graphics, 21 (4): 163-169, July 1987.

[32] ZÚÑIGA, L. D. O. Método de verificação do desempenho do scanners laser usando um artefato tridimensional. 99 p. Dissertação (Mestre em Sistemas Mecatrônicos) - Programa Pós - Graduação em Sistemas Mecatrônicos, Universidade de Brasília, Brasília, 2013.

APÊNDICES

I. A: CALIBRAÇÃO DA CÂMERA

```
#!/usr/bin/env python

import cv2
import numpy as np
import os
import glob

# Defining the dimensions of checkerboard
CHECKERBOARD = (6,9)
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)

# Creating vector to store vectors of 3D points for each checkerboard image
objpoints = []
# Creating vector to store vectors of 2D points for each checkerboard image
imgpoints = []


# Defining the world coordinates for 3D points
objp = np.zeros((1, CHECKERBOARD[0]*CHECKERBOARD[1], 3), np.float32)
objp[0,:,:2] = np.mgrid[0:CHECKERBOARD[0], 0:CHECKERBOARD[1]].T.reshape(-1, 2)
prev_img_shape = None

# Extracting path of individual image stored in a given directory
images = glob.glob('./images/*.jpg')
for fname in images:
    img = cv2.imread(fname)
    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    # Find the chess board corners
    # If desired number of corners are found in the image then ret = true
    ret, corners = cv2.findChessboardCorners(gray, CHECKERBOARD, cv2.CALIB_CB_ADAPTIVE_THRESH+
        cv2.CALIB_CB_FAST_CHECK+cv2.CALIB_CB_NORMALIZE_IMAGE)

    """
    If desired number of corner are detected,
    we refine the pixel coordinates and display
    them on the images of checker board
    """
    if ret == True:
        objpoints.append(objp)
        # refining pixel coordinates for given 2d points.
        corners2 = cv2.cornerSubPix(gray,corners,(11,11),(-1,-1),criteria)

        imgpoints.append(corners2)

        # Draw and display the corners
        img = cv2.drawChessboardCorners(img, CHECKERBOARD, corners2,ret)

    cv2.imshow('img',img)
    cv2.waitKey(0)

cv2.destroyAllWindows()

h,w = img.shape[:2]

"""
```

```
Performing camera calibration by
passing the value of known 3D points (objpoints)
and corresponding pixel coordinates of the
detected corners (imgpoints)
"""
ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints, gray.shape[::-1],None,None)

print("Camera matrix : \n")
print(mtx)
print("dist : \n")
print(dist)
print("rvecs : \n")
print(rvecs)
print("tvecs : \n")
print(tvecs)

# Using the derived camera parameters to undistort the image

img = cv2.imread(images[0])
# Refining the camera matrix using parameters obtained by calibration
newcameramtx, roi = cv2.getOptimalNewCameraMatrix(mtx, dist, (w,h), 1, (w,h))

# Method 1 to undistort the image
dst = cv2.undistort(img, mtx, dist, None, newcameramtx)

dst = cv2.remap(img,mapx,mapy,cv2.INTER_LINEAR)

# Displaying the undistorted image
cv2.imshow("undistorted image",dst)
cv2.waitKey(0)
```

II. B: HISTOGRAMA E SEGMENTAÇÃO

```
img = cv2.imread('imagembloco1.png')
#Carrega a imagem na variável img


color = ('b','g','r')
#Define os canais rgb

for i,col in enumerate(color): #percorre pelos canais definindo o histograma
    histr = cv2.calcHist([img],[i],None,[256],[0,256])
    plt.plot(histr,color = col)
    plt.xlim([0,256])
plt.show() #renderiza o histograma
height = image.shape[0] #quantidade de pixels no eixo y
width = image.shape[1] #quantidade de pixels no eixo x
green = image[:,:,1] #segmentacao por canal
threshlow=245 #valores de threshold
threshhigh=255
ret2,thresh2 = cv2.threshold(green,threshlow,threshhigh,cv2.THRESH_BINARY) #aplicação do threshold
```

III. C: RETA DE REFERÊNCIA

```
from mpl_toolkits.mplot3d.art3d import Poly3DCollection
from skimage import measure
from math import atan, sin
import cv2
import numpy as np
from stl import mesh

import matplotlib.pyplot as plt


def LoG(x, y, sigma):
    temp = (x ** 2 + y ** 2) / (2 * sigma ** 2)
    return -1 / (np.pi * sigma ** 4) * (1 - temp) * np.exp(-temp)

def make_coordinates(image, line_parameters):
        slope, intercept = line_parameters
        x1 = 0
        x2 = image.shape[1]
        y1 = int(slope*x1 + intercept)
        y2 = int(slope*x2 + intercept)
        return np.array([x1,y1,x2,y2])


def average_slope_intercept(image, lines):
        fit = []
        for line in lines:
                x1,y1,x2,y2 = line.reshape(4)
                parameters = np.polyfit((x1,x2), (y1,y2), 1)
                slope = parameters[0]
                intercept = parameters[1]
                fit.append((slope, intercept))
        fit_average = np.average(fit, axis=0)
        fit_line = make_coordinates(image, fit_average)
        return np.array([fit_line])

def canny(image):
        gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
        gray = cv2.GaussianBlur(gray, (5, 5), 0)
        thresh = cv2.threshold(gray, 45, 255, cv2.THRESH_BINARY)[1]
        thresh = cv2.erode(thresh, None, iterations=2)
        thresh = cv2.dilate(thresh, None, iterations=2)
        thresh = cv2.Canny(thresh,50,80)

        return thresh


def display_lines(image,lines):
        line_image = np.zeros_like(image)
        if lines is not None:
                i=0
                for line in lines:
                        i = i+100
                        x1,y1,x2,y2 = line.reshape(4)
```

```
                    cv2.line(line_image, (x1,y1), (x2,y2), (255,i,0), 1)

        return line_image

def region_of_interest(image, y1, y2):
  height = image.shape[0]
  width = image.shape[1]
  polygons = np.array([
              [(0,y1),(width,y1), (width, y2), (0, y2)]
        ])
  mask = np.zeros_like(image)
  cv2.fillPoly(mask,polygons, 255)
  masked_image = cv2.bitwise_and(image,mask)
  return masked_image


def LoG(x, y, sigma):
    temp = (x ** 2 + y ** 2) / (2 * sigma ** 2)
    return -1 / (np.pi * sigma ** 4) * (1 - temp) * np.exp(-temp)



#abertura da variável de leitura
cap = cv2.VideoCapture('bloco2.avi')
zeros=np.zeros((480,640))
zerosnovo=np.zeros((480,640))
threshlow=245
threshhigh=255
kernel = np.ones((5,5),np.uint8)
contador=0

ret, image = cap.read()

height = image.shape[0] #quantidade de pixels no eixo y
width = image.shape[1] #quantidade de pixels no eixo x
green = image[:,:,1] #segmentacao por canal
ret2,thresh2 = cv2.threshold(green,threshlow,threshhigh,cv2.THRESH_BINARY) #aplicação do threshold
thresh2 = cv2.Canny(thresh2,50,80)
cropped_image = region_of_interest(thresh2, 200, 300)
#cv2.imshow('red',cropped_image)
#cv2.waitKey(0)


lines = cv2.HoughLinesP(cropped_image, 2, np.pi/180, 20, np.array([]), minLineLength=20, maxLineGap = 300)
averaged_line = average_slope_intercept(thresh2,lines)

line_image = display_lines(image, averaged_line)
combo_image = cv2.addWeighted(image, 0.3, line_image, 2, 1)
```

IV. D: RECONSTRUÇÃO 3D

```
from mpl_toolkits.mplot3d.art3d import Poly3DCollection
from skimage import measure
from math import atan, sin
import cv2
import numpy as np
from stl import mesh

import matplotlib.pyplot as plt


def LoG(x, y, sigma):
    temp = (x ** 2 + y ** 2) / (2 * sigma ** 2)
    return -1 / (np.pi * sigma ** 4) * (1 - temp) * np.exp(-temp)

def make_coordinates(image, line_parameters):
        slope, intercept = line_parameters
        x1 = 0
        x2 = image.shape[1]
        y1 = int(slope*x1 + intercept)
        y2 = int(slope*x2 + intercept)
        return np.array([x1,y1,x2,y2])


def average_slope_intercept(image, lines):
        fit = []
        for line in lines:
                x1,y1,x2,y2 = line.reshape(4)
                parameters = np.polyfit((x1,x2), (y1,y2), 1)
                slope = parameters[0]
                intercept = parameters[1]
                fit.append((slope, intercept))
        fit_average = np.average(fit, axis=0)
        fit_line = make_coordinates(image, fit_average)
        return np.array([fit_line])

def canny(image):
        gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
        gray = cv2.GaussianBlur(gray, (5, 5), 0)
        thresh = cv2.threshold(gray, 45, 255, cv2.THRESH_BINARY)[1]
        thresh = cv2.erode(thresh, None, iterations=2)
        thresh = cv2.dilate(thresh, None, iterations=2)
        thresh = cv2.Canny(thresh,50,80)

        return thresh


def display_lines(image,lines):
        line_image = np.zeros_like(image)
        if lines is not None:
                i=0
                for line in lines:
                        i = i+100
                        x1,y1,x2,y2 = line.reshape(4)
```

```
                    cv2.line(line_image, (x1,y1), (x2,y2), (255,i,0), 1)

        return line_image

def region_of_interest(image, y1, y2):
  height = image.shape[0]
  width = image.shape[1]
  polygons = np.array([
              [(0,y1),(width,y1), (width, y2), (0, y2)]
        ])
  mask = np.zeros_like(image)
  cv2.fillPoly(mask,polygons, 255)
  masked_image = cv2.bitwise_and(image,mask)
  return masked_image


def LoG(x, y, sigma):
    temp = (x ** 2 + y ** 2) / (2 * sigma ** 2)
    return -1 / (np.pi * sigma ** 4) * (1 - temp) * np.exp(-temp)



#abertura da variável de leitura
cap = cv2.VideoCapture('bloco2.avi')
zeros=np.zeros((480,640))
zerosnovo=np.zeros((480,640))
threshlow=245
threshhigh=255
kernel = np.ones((5,5),np.uint8)
contador=0

ret, image = cap.read()

height = image.shape[0] #quantidade de pixels no eixo y
width = image.shape[1] #quantidade de pixels no eixo x
green = image[:,:,1] #segmentacao por canal
ret2,thresh2 = cv2.threshold(green,threshlow,threshhigh,cv2.THRESH_BINARY) #aplicação do threshold
thresh2 = cv2.Canny(thresh2,50,80)
cropped_image = region_of_interest(thresh2, 200, 300)
#cv2.imshow('red',cropped_image)
#cv2.waitKey(0)


lines = cv2.HoughLinesP(cropped_image, 2, np.pi/180, 20, np.array([]), minLineLength=20, maxLineGap = 300)
averaged_line = average_slope_intercept(thresh2,lines)

line_image = display_lines(image, averaged_line)
combo_image = cv2.addWeighted(image, 0.3, line_image, 2, 1)
```

```
while(cap.isOpened()):
    ret, image = cap.read()
    if ret == True:
      contador = contador+1
      print(contador)
      height = image.shape[0]
      width = image.shape[1]
      green = image[:,:,1]
      ret2,thresh2 = cv2.threshold(green,threshlow,threshhigh,cv2.THRESH_BINARY)
      thresh2 = cv2.Canny(thresh2,50,80)


      ################ Rotacionando o frame ######################

      x1,y1,x2,y2 = averaged_line.reshape(4)
      parameters = np.polyfit((x1,x2), (y1,y2), 1)
      rotation_matrix = cv2.getRotationMatrix2D((width/2, height/2), (atan(parameters[0])*180)/np.pi, 1)
      thresh2 = cv2.warpAffine(thresh2, rotation_matrix, (width, height))

      ########################################################


      referencia_height = int(parameters[1] - parameters[0]*(width/2))
      #print(referencia_height)
      thresh2 = region_of_interest(thresh2, 0, referencia_height - 5)



      i=0
      for i in range(0, width):
        j=0
        mais_alto = 0
        for j in range(0, referencia_height):
          if mais_alto != 0:
            thresh2[j][i] = 255
          elif thresh2[j][i] != 0:
            mais_alto = 1
      i=0

      cv2.imshow('frame',thresh2)
      zerosnovo = np.dstack((zerosnovo, thresh2))
      if cv2.waitKey(1) & 0xFF == ord('q'):
          break
    else:
      break
print("saiu")
verts, faces, normals, values = measure.marching_cubes(zerosnovo, 0)
cube = mesh.Mesh(np.zeros(faces.shape[0], dtype=mesh.Mesh.dtype))

for i, f in enumerate(verts):
    verts[i][1] = verts[i][1]*0.25 #horizontal
    verts[i][0] = verts[i][0]*0.8823 #altura



      verts[i][2] = verts[i][2]*0.5542 #movimento
      #vertices[i][2] = vertices[i][2]*2

  print(faces)
  print(enumerate(faces))
  for i, f in enumerate(faces):
      #print("a")
      for j in range(3):
          #print("b")
          cube.vectors[i][j] = verts[f[j],:]
  cube.save('5.stl')
  cap.release()
  cv2.destroyAllWindows()
```