UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
ENG. DE CONTROLE E AUTOMAÇÃO

JONNAS PIRES ESPINDULA - 193808

ESCANEAMENTO DE MODELO TRIDIMENSIONAL DE UMA PEÇA ATRAVÉS DE FOTOGRAMETRIA

Porto Alegre
2022

JONNAS PIRES ESPINDULA - 193808

ESCANEAMENTO DE MODELO TRIDIMENSIONAL DE UMA PEÇA ATRAVÉS DE FOTOGRAMETRIA

Trabalho de Conclusão de Curso (TCC-CCA) apresentado à COMGRAD-CCA da Universidade Federal do Rio Grande do Sul como parte dos requisitos para a obtenção do título de *Bacharel em Eng. de Controle e Automação* .

ORIENTADOR:
Prof. Dr. Heraldo José de Amorim

CO-ORIENTADOR(A):
Me. Yachel Rogério Mileski

Porto Alegre
2022

JONNAS PIRES ESPINDULA - 193808

ESCANEAMENTO DE MODELO TRIDIMENSIONAL DE UMA PEÇA ATRAVÉS DE FOTOGRAMETRIA

Este Trabalho de Conclusão de Curso foi julgado adequado para a obtenção dos créditos da Disciplina de TCC do curso *Eng. de Controle e Automação* e aprovado em sua forma final pelo Orientador e pela Banca Examinadora.

Orientador: _____
Prof. Dr. Heraldo José de Amorim, UFRGS
Doutor pela Universidade Federal do Rio Grande do Sul – Porto Alegre, Brasil

Banca Examinadora:

Prof. Dr. Heraldo José de Amorim, UFRGS
Doutor pela Universidade Federal do Rio Grande do Sul – Porto Alegre, Brasil

Prof. Dr. Marcelo Götz, UFRGS
Doutor pela Universität Paderborn – Paderborn, Alemanha

Prof. Dr. Edson Cordeiro do Valle, UFRGS
Doutor pela Universidade Federal da Bahia – Salvador, Brasil

_____
Prof. Dr. Mário Roland Sobczyk Sobrinho
Coordenador de Curso
Eng. de Controle e Automação

Porto Alegre, outubro de 2022.

RESUMO

Fotogrametria busca retomar informações físicas e geométricas de um objeto no espaço tridimensional através de imagens do objeto, ou seja, uma representação bidimensional. Este trabalho procura obter modelos tridimensionais de objetos através de imagens obtidas usando uma câmera cujos parâmetros são conhecidos e fixos. Para isso, é realizada a calibração da câmera para obtenção de seus parâmetros e então objetos de geometria simples são analisados usando imagens obtidas por essa câmera. Um algoritmo baseado na intersecção de volumes obtidos através das imagens analisadas é testado para verificar sua validade para obter as geometrias desejadas. O método não pode ser validado devido a problemas na obtenção de parte dos casos de intersecções de sólidos geradas, portanto alternativas para prosseguir na implementação e validação do método são sugeridas.

Palavras-chave: Visão Computacional, Processamento de Sinais, Automação e Controle, Fotogrametria.

ABSTRACT

Photogrammetry can be described as the set of methods and theory used in order to retrieve physical and geometric characteristics of an tridimensional object from bidimensional images. We propose an algorithm to obtain tridimensional models of objects with simple geometry by using images taken with a camera with fixed and known parameters. Camera calibration is performed in order to characterize the camera used in order to obtain the images used in the photogrammetry process. The volume intersection method could not be correctly evaluated due to issues in intersection generation for some of the cases tested. Therefore, alternative methods are proposed in order to proceed with the method implementation and evaluation.

Keywords: Computer Vision, Signal Processing, Automation and Control, Photogrammetry.

SUMÁRIO

LISTA DE ILUSTRAÇÕES 11

LISTA DE TABELAS 15

LISTA DE SíMBOLOS 17

1 INTRODUÇÃO 19

2 REVISÃO DA LITERATURA 21
2.1 Fotogrametria, visão computacional e estado da arte 21
2.2 Calibração de uma câmera 21
2.3 Reconhecimento de características descritivas de imagens 25

3 METODOLOGIA APLICADA 29
3.1 Solução proposta e considerações iniciais 29
3.2 Ferramentas e técnicas avaliadas 30
3.2.1 Visão computacional e OpenCV 31
3.2.2 Formatos de visualização e ferramentas relacionadas 31
3.3 Calibração da câmera 33

4 RESULTADOS DOS ENSAIOS 37

5 DISCUSSÃO DOS RESULTADOS 45

APÊNDICE A - DESCRIÇÃO DA IMPLEMENTAÇÃO PROPOSTA . . 47
A.1 Captura de imagens e calibração da câmera 47
A.2 Geração dos volumes descritivos 47

APÊNDICE B - CALIBRAÇÃO DA CÂMERA 75

APÊNDICE C - IMAGENS DOS ENSAIOS E VOLUMES GERADOS . 79

REFERÊNCIAS 85

LISTA DE ILUSTRAÇÕES

1 Sistema de referência utilizados na calibração da câmera 22
2 Representação da projeção de um ponto P_w no plano da imagem P. . . 24
3 Aplicação do método de Harris para identificação de pontos chave da imagem. 27
4 Configuração proposta para realização dos ensaios. 30
5 Representação visual da solução proposta. Um sólido é visto a partir de 2 pontos diferentes e dois volumes representativos são gerados . . 31
6 Imagem de ensaio com pontos de intersecções e fronteira do objeto identificados. O referencial do espaço é mostrado com linhas das cores vermelha, verde e azul e a fronteira do objeto é determinada por uma linha branca. 32
7 Envoltória convexa sobre duas formas geométricas. Na esquerda, a fronteira do paralelograma coincide com a envoltória convexa dos pontos da forma geométrica. Na direita, a envoltória convexa forma um semicírculo, não sendo coincidente com a fronteira do disco. . . . 33
8 Interface gráfica do software Meshmixer. 33
9 Interface gráfica do software Meshlab. 34
10 Exemplo de padrões utilizado para calibração da câmera utilizando OpenCV. Na esquerda há um padrão xadrez e a direita um padrão de pontos. 34
11 Padrão xadrez utilizado para calibração (arestas dos quadrados com 22,5 mm). 35
12 Posições e orientações da câmera durante a captura de imagens usadas para calibração obtidas a partir do processo de calibração. Escala em milímetros. 36
13 Objetos utilizados durante o teste: um cubo, um paralepípedo e dois cilindros de diferentes diâmetros. 37
14 Conjunto de imagens obtidas para o objeto de ensaio 1. 38
15 Aplicação da compensação das distorções. Na esquerda, imagem obtida pela câmera. No centro, imagem com compensação de distorções radiais e tangenciais aplicada. Na direita, diferença entre as imagens com contraste ajustado para realçe. 39
16 Seleção de pontos da imagem e envoltórias convexas geradas para as imagens. Na esquerda, imagem com detecção de cantos pelo método de Harris aplicada. No centro, envoltória convexa utilizando apenas os pontos detectados pelo método de Harris. Na direita, envoltória convexa utilizando pontos selecionados manualmente. 40

17 *Meshes* geradas através de imagens capturadas do objeto de ensaio 1. 40
18 *Meshes* geradas através de imagens capturadas do objeto de ensaio 2. 41
19 *Meshes* geradas através de imagens capturadas do objeto de ensaio 3. 42
20 *Meshes* geradas através de imagens capturadas do objeto de ensaio 4. 42
21 Exemplo de volume de intersecção gerado com sucesso a partir de outros dois volumes representativos utilizando o método proposto. Na esquerda estão os volumes descritivos originais e na direita o volume de intersecção gerado. 43
22 Exemplo de dois volumes de intersecção gerados a partir de outros dois volumes representativos utilizando o método proposto com resultado insatisfatório. 43
23 Visualização de dois volumes gerados a partir dos ensaios usando a biblioteca *PyVista* (topo e centro) e da intersecção gerada pela biblioteca (canto inferior). 44
24 Código Python usado para realizar a captura das imagens. 49
25 Código Python usado para obter os parâmetros de calibração da câmera - Verificação dos diretórios utilizados para salvar as imagens de saída. 50
26 Código Python usado para obter os parâmetros de calibração da câmera - Varredura das imagens de calibração, parte 1. 51
27 Código Python usado para obter os parâmetros de calibração da câmera - Varredura das imagens de calibração, parte 2. 52
28 Código Python usado para obter os parâmetros de calibração da câmera - Varredura das imagens de calibração, parte 3. 53
29 Código Python usado para obter os parâmetros de calibração da câmera - Varredura das imagens de calibração, parte 4. 54
30 Setup utilizado para gerar os volumes descritivos. 55
31 Código Python para gerar os volumes representativos das imagens do ensaio. 56
32 Código Python o método *GeneratePointDataOutput* - Parte 1. 57
33 Código Python o método *GeneratePointDataOutput* - Parte 2. 58
34 Código Python o método *GetPointsFromImage* - Parte 1. 59
35 Código Python o método *GetPointsFromImage* - Parte 2. 60
36 Código Python o método *GetFrameFromPatternPoints*. 61
37 Código Python o método *GetPointsInWorld* - Parte 1. 62
38 Código Python o método *GetPointsInWorld* - Parte 2. 63
39 Código Python o método *GeneratePointDataOutput* - Parte 3. 64
40 Código Python o método *GeneratePointDataOutput* - Parte 4. 65
41 Código Python o método *PlotAxesInImage* - Parte 1. 66
42 Código Python o método *PlotAxesInImage* - Parte 2. 67
43 Código Python o método *GeneratePointDataOutput* - Parte 5. 68
44 Código Python para o construtor da classe *VolumeDescriptor*. 69
45 Código Python para o método *CreateVolumeMeshWriteFile* - Parte 1. 70
46 Código Python para o método *CreateVolumeMeshWriteFile* - Parte 2. 71
47 Código Python para o método *CreateVolumeMeshWriteFile* - Parte 3. 72
48 Código Python para o método *CreateVolumeMeshWriteFile* - Parte 4. 73
49 Exemplo de arquivo *.ply* gerado a partir de uma imagem coma correspondência entre pontos no arquivo e na imagem representada. 74
50 Camera Microsoft LifeCam Cinema utilizada nos ensaios. 75

51 Imagens do padrão xadrez utilizadas para a calibração da câmera. . . 76
52 Posições e orientações da câmera durante a captura de imagens usadas para calibração obtidas a partir do processo de calibração. Escala em milímetros. Três orientações diferentes são mostradas. 77
53 Imagens capturadas para o objeto de ensaio 1. 80
54 Volumes gerados para as imagens do objeto de ensaio 1. 80
55 Imagens capturadas para o objeto de ensaio 2. 81
56 Volumes gerados para as imagens do objeto de ensaio 2. 81
57 Imagens capturadas para o objeto de ensaio 3. 82
58 Volumes gerados para as imagens do objeto de ensaio 3. 82
59 Imagens capturadas para o objeto de ensaio 4. 83
60 Volumes gerados para as imagens do objeto de ensaio 4. 83

LISTA DE TABELAS

1 Parâmetros da câmera obtidos pela calibração. 35
2 Dimensões dos paralelepídos utilizados nos ensaios. 37
3 Dimensões dos discos utilizados nos ensaios. 38
4 Relação entre ensaios, imagens capturadas e volumes gerados. 79

LISTA DE SÍMBOLOS

$\vec{x_i}$	Vetor de posição de um ponto no sistema de referência da imagem
H_i^w	Transformada entre o sistema de referência do mundo e sistema de referência da imagem
$\vec{X_w}$	Vetor de posição de um ponto no sistema de referência do mundo
x_i	Coordenada x da posição de um ponto no sistema de referência da imagem
y_i	Coordenada y da posição de um ponto no sistema de referência da imagem
X_w	Coordenada X da posição de um ponto no sistema de referência do mundo
Y_w	Coordenada Y da posição de um ponto no sistema de referência do mundo
Z_w	Coordenada Z da posição de um ponto no sistema de referência do mundo
S_w	Sistema de coordenadas do mundo
S_c	Sistema de coordenadas da câmera
S_p	Sistema de coordenadas do plano da imagem
S_i	Sistema de coordenadas da imagem
${}^w\vec{X}_P$	Coordenadas de um ponto arbitrário no sistema de referência do mundo S_w
${}^w\vec{O}_c$	Coordenadas da origem do sistema de referência da câmera S_c descrito no sistema de referência do mundo S_w
${}^wX_P$	Coordenada x de um ponto arbitrário no sistema de referência do mundo S_w
${}^wY_P$	Coordenada x de um ponto arbitrário no sistema de referência do mundo S_w
${}^wZ_P$	Coordenada x de um ponto arbitrário no sistema de referência do mundo S_w
${}^wO_{x,c}$	Coordenada x da origem do sistema de referência da câmera S_c no sistema de referência do mundo S_w
${}^wO_{y,c}$	Coordenada x da origem do sistema de referência da câmera S_c no sistema de referência do mundo S_w
${}^wO_{z,c}$	Coordenada x da origem do sistema de referência da câmera S_c no sistema de referência do mundo S_w
H_c^w	Transformada homogênea do sistema de referência da câmera S_c em relação ao sistema de referência do mundo S_w

R_c^w	Matriz de rotação para orientação do sistema de referência da câmera S_c em relação ao sistema de referência do mundo S_w
${}^w\vec{X}_w$	Ponto no sistema de referência do mundo S_w representado no sistema de referência do mundo S_w
${}^c\vec{x}_p$	Ponto no plano da imagem representado no sistema de referência da câmera S_c
${}^wP$	Ponto no sistema de referência do mundo S_w
H	Ponto principal do plano da imagem, coincidente com O_p
$\vec{O}_p$	Origem do sistema de referência do plano da imagem
${}^w\vec{O}_c$	Origem do sistema de referêcia da câmera S_c em relação ao sistema de referência do mundo S_w
c	Constante da câmera
H_c^p	Transformada homogênea do sistema de referencia do plano da imagem S_p em relação ao sistema de referência da câmera S_c
H_p^i	Transformada homogênenea do sistema de referência da imagem S_i em relação ao sistema de referência do plano da imagem S_p
s	Compensação de cisalhamento da imagem
m	Diferença de escala da imagem
x_H	Coordenada x do ponto principal H do plano de projeção da imagem no sistema de referência da imagem S_i
y_H	Coordenada y do ponto principal H do plano de projeção da imagem no sistema de referência da imagem S_i
$\vec{{}^i x}$	Posição de um ponto no sistema de referência da imagem S_i
$\Delta x(\vec{{}^i x}, \vec{q})$	Distorção na coordenada x no sistema de coordenadas da imagem S_i
$\Delta y(\vec{{}^i x}, \vec{q})$	Distorção na coordenada y no sistema de coordenadas da imagem S_i
$H_{i,dist}^i({}^i\vec{x})$	Transformada de compensação de erros não-lineares no sistema de referência da imagem S_i
r	Distância de um pixel na imagem em relação ao ponto principal H
q_i	Parâmetro da distorção parabólica da lente
$det(M)$	Determinante da matriz M
$tr(M)$	Traço da matrix M
λ_i	Autovalor i de uma matrix M

1 INTRODUÇÃO

Sistemas de visão computacional são utilizados em diversas áreas, como automação industrial, levantamentos topográficos, medicina, reconhecimento de padrões e sistemas de locomoção autonôma. A complexidade de aplicações na área varia com os objetivos desejados e restrições impostas pelo equipamento ou capacidade computacional disponíveis para aplicação, com diversas técnicas podendo ser aplicadas ao mesmo problema dependendo da combinação desses fatores. Assim, tanto atividades mais simples, como detecção de cantos e bordas, quanto atividades mais complexas, como reconhecimento de padrões independente de variações de escala e orientação, possuem uma gama de métodos diferentes que se adequam a diferentes limitações que o sistema pode apresentar.

Uma aplicação de sistemas de visão computacional é a de obter informações sobre o formato ou geometria de um objeto apenas utilizando imagens do mesmo, ou então utilizando imagens para complementar a informação espacial que pode ser obtida a partir de outros sensores, como radares e LIDARs (*Light Detection and Ranging*). Um método utilizado amplamente em aplicações topográficas, por exemplo, é a fotogrametria, que consiste na recomposição de informações sobre a geometria de um objeto a partir de imagens em diferentes ângulos do objeto.

A fotogrametria é utilizada para geração de mapas detalhados a partir de imagens de satélite, para realizar o escaneamento de estruturas grandes, como fachadas de prédios, de difícil acesso, como em sítios arqueológicos, ou onde outros métodos de medição não são possíveis ou a informação visual é complementar a outros sensores, como em carros autônomos.

Uma aplicação de fotogrametria em ambientes industriais é a realização de escaneamento de peças para obtenção de modelos tridimensionais que podem ser utilizados para design, análise ou manufatura de outras peças ou da própria peça analisada. Geralmente *scanners* 3D usam fotogrametria com alguma outra forma de sensoriamento, como sensores infravermelhos ou *laser*, ou então a aplicação de refletores específicos ao objeto para realizar um escaneamento.

O objetivo deste trabalho é avaliar a viabilidade de obter um modelo 3D a partir da intersecção de diversos volumes gerados a partir de diversas imagens utilizando um método puramente baseado em imagens com o conhecimento de pontos específicos de referência. Um método simples é proposto para o problema e então procura-se realizar a validação do mesmo após uma análise de ferramentas a serem utilizadas e realizar ensaios com corpos de prova de geometria simples.

2 REVISÃO DA LITERATURA

2.1 Fotogrametria, visão computacional e estado da arte

Fotogrametria é o grupo de técnicas envolvidas em retomar informações geométricas e físicas de um objeto através de imagens (ZHANG; YAO, 2005). Portanto, fotogrametria e visão computacional são campos de estudos que se entrelaçam, uma vez que que o principal objetivo da visão computacional é a extração de informação de imagens (MUNDY, 2005).

Entre os objetivos do estudo de visão computacional está a modelagem de objetos, na qual se almeja obter um modelo tridimensional completo e com precisão adequada de um objeto. Tal modelo pode ser usado em outros campos relacionados com visão computacional, como reconhecimento de objetos, mapeamento da movimentação executada por uma câmera em relação ao ambiente, e simulações que buscam gerar imagens realistas do objeto analisado de diversos pontos de vista(MUNDY, 2005). Esses resultados podem ser utilizados em outras atividades, como navegação autônoma ou escaneamento de estruturas de diferentes contextos, como usinagem, medicina ou arquitetura.

A fotogrametria tem seu desenvolvimento iniciado no século XIX juntamente com o próprio desenvolvimento da fotografia e tem como objetivo primário a precisão dos resultados obtidos por seus métodos (MUNDY, 2005). A principal aplicação da área de fotogrametria é a realização de mapeamentos topográficos a partir de fotografias obtidas através de aeronaves ou de satélites, nos quais os parâmetros das câmeras utilizadas, assim como o posicionamento de marcadores de referência no solo são conhecidos com altos níveis de precisão. O conhecimento dos parâmetros da câmera, assim como o posicionamento de pontos de referência no mundo são necessários e impactam nos resultados obtidos, uma vez que as principais fontes de erros no processo de fotogrametria são erros nas medições de pontos de controle ou erros nos modelos de projeção da câmera, como distorção radial ou então erros númericos na resolução das equações de projeção (MUNDY, 2005).

Para se obter informações de objetos e do espaço trimensional no qual o objeto está inserido, é necessário primeiro saber parâmetros que descrevem a câmera e as transformadas homogêneas utilizadas para descrever o objeto no espaço tridimensional como um imagem bidimensional.

2.2 Calibração de uma câmera

O objetivo de realizar a calibração de uma câmera é obter os parâmetros geométricos da câmera, como distância focal e distorções de lente, e a relação entre a posição e orientação da câmera em relação ao sistema de referência do mundo (STACHNISS, 2022). Ou seja, o

objetivo da calibração é encontrar a relação dada pela Equação 1:

$$\vec{x_i} = H_i^w \vec{X_w} \tag{1}$$

Onde $\vec{x_i}$ é a posição de um ponto no sistema de coordenadas da imagem, em *pixels*, $\vec{X_w}$ é a posição de um ponto no sistema de coordenadas do espaço (mundo) observado e H_i^w é a transformada que relaciona os sistemas de coordenadas entre si. Pode-se expandir a Equação 1 para exibir suas componentes conforme mostrado na Equação 2.

$$\begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = H_i^w \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \tag{2}$$

onde x_i e y_i são as coordenadas x e y, em *pixels*, de um ponto na imagem e X_w, Y_w e Z_w são as coordenadas do mesmo no sistema de referência do mundo.

Para determinação da transformada H_i^w, deve-se determinar as transformadas entre os sistemas de referências utilizados na composição da transformada H_i^w (Figura 1). Os sistemas de referência utilizados são:

- Sistema de coordenadas do mundo, referenciado como S_w,
- Sistema de coordenadas da câmera, referenciado como S_c,
- Sistema de coordenadas do plano da imagem, referenciado como S_p e,
- Sistema de coordenadas da imagem (ou do sensor da câmera), referenciado como S_i.

Usando esses sistemas de referência, podemos determinar duas categorias de parâmetros: *parâmetros extrínsecos* e *parâmetros intrínsecos* (STACHNISS, 2022). Os *parâmetros extrínsecos* descrevem o posicionamento da câmera em relação ao sistema de referência do mundo, enquanto os *parâmetros intrínsecos* descrevem as propriedades geométricas da câmera.

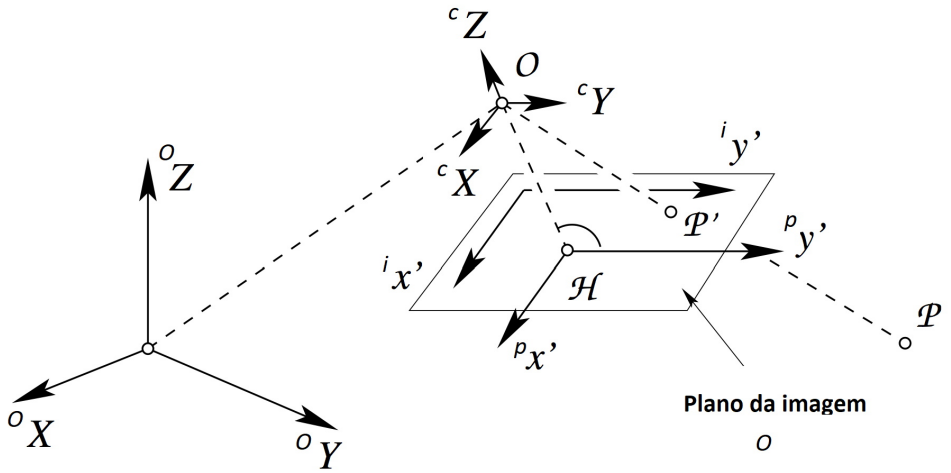

Figura 1: *Sistema de referência utilizados na calibração da câmera*

Fonte: Adaptado de (STACHNISS, 2022).

Para os parâmetros extrinsecos deve-se determinar a transformada que descreve a posição e orientação da camêra em relação ao sistema de referência do mundo, H_c^w. São necessários portanto seis parâmetros para descrever essa relação: três para a descrição do posicionamento e três para descrição da orientação. Assume-se que um ponto descrito no sistema de referência do mundo, ${}^w\vec{X}_P$, e a origem do sistema de coordenadas da câmera no mesmo sistema, ${}^w\vec{O}_c$, podem ser descritos, respectivamente, pelas Equações 3 e 4.

$$ {}^w\vec{X}_P = [{}^wX_P, {}^wY_P, {}^wZ_P, 1]^T \tag{3} $$

$$ {}^w\vec{O}_c = [{}^wO_{x,c}, {}^wO_{y,c}, {}^wO_{z,c}, 1]^T \tag{4} $$

Onde os parâmetros ${}^wX_P$, ${}^wY_P$, ${}^wZ_P$ são as coordenadas nos eixos x, y e z, respectivamente, no sistema de referência do mundo, S_w, de um ponto P arbitrário e os parâmetros ${}^wO_{x,c}$, ${}^wO_{y,c}$, ${}^wO_{z,c}$ são as coordenadas x, y e z, respectivamente, a origem do sistema de referência da câmera, ${}^w\vec{O}_c$, também no sistema de referência do mundo, S_w.

A transformada H_c^w, que representa a relação entre o sistema de referência da câmera ao sistema de referência tridimensional, no qual estão os objetos representados na imagem gerada pela câmera, pode ser dada pela Equação 5.

$$ H_c^w = \begin{bmatrix} R_c^w & -R_c^w\vec{X}_w \\ \vec{0}^T & 1 \end{bmatrix} \tag{5} $$

Onde R_c^w descreve a rotação entre os sistemas de referência do mundo, S_w, e o sistema de referência da câmera, S_c.

Quanto aos *parâmetros intrínsecos*, esses podem ser representados através da composição de transformadas que levam da representação do objeto no sistema de referência da câmera, S_c, até a representação no plano do sistema de referência da imagem, S_i.

A primeira transformada envolvida na descrição dos parâmetros intrínsecos é a *transformada de projeção ideal* (STACHNISS, 2022), H_c^p, a qual é responsável por projetar a imagem de um ponto $\vec{X}_w$ no sistema de referência do mundo no plano da imagem P, representado pelo sistema de referência S_p. A caracterização dessa transformada (descrita visualmente na Figura 2) é feita considerando-se que:

- não há distorção na representação no plano, ou seja, considera-se que a imagem é captada por uma lente sem distorção;
- todas as linhas entre um ponto no espaço e o plano da imagem passam pelo *centro de projeção*, ou seja, a origem do sistema de referência da câmera, ${}^w\vec{O}_c$;
- O *ponto principal*, H, do plano da imagem e o *ponto focal da câmera* encontram-se no mesmo eixo;
- A origem do sistema de referência da câmera ${}^w\vec{O}_c$ encontra-se a uma distância c do plano da imagem (e do ponto principal da imagem H).

Essa transformada que descreve a projeção ideal, e que pode ser chamada de *matriz de calibração da câmera ideal* (STACHNISS, 2022) é dada pela Equação 6.

$$ H_c^p = \begin{bmatrix} c & 0 & 0 & 0 \\ 0 & c & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{6} $$

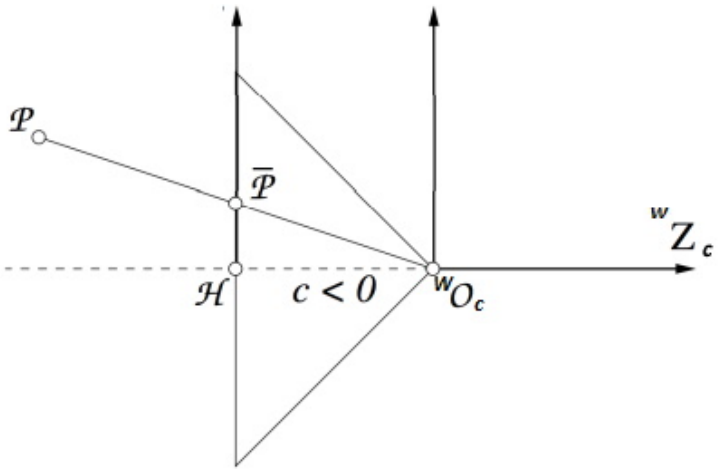

Figura 2: *Representação da projeção de um ponto P_w no plano da imagem P.*

Fonte: Adaptado de (STACHNISS, 2022).

Essa é uma transformada não inversível a qual acaba por causar perda de informação na captura da imagem. No caso, não é possível obter apenas a partir da imagem, sem nenhuma informação adicional que permita determinar a profundidade, a posição original de ^{w}P.

A próxima transformada relacionada aos parâmetros intrínsecos da câmera descreve a relação entre o o sistema de referência do plano de projeção da imagem S_p e o plano da imagem S_i e é dada pela Equação 7.

$$H_p^i = \begin{bmatrix} 1 & s & x_H \\ 0 & 1+m & y_H \\ 0 & 0 & 1 \end{bmatrix} \tag{7}$$

Onde s é uma compensação de cisalhamento da imagem, m é diferença de escala entre os eixos x e y da imagem, x_H e y_H são as coordenadas x e y, respectivamente, do ponto principal no plano de projeção da imagem (origem do sistema de referência S_p) no sistema de referência da imagem S_i (STACHNISS, 2022).

A última transformada de interesse na determinação do parâmetros intrínsecos relaciona os erros não-lineares presentes na captura da imagem, os quais podem ocorrer por diversos motivos, como lentes imperfeitas ou com distorções, características do sensor de captura da imagem, etc. Para um modelo geral, consideram-se as posições como representadas nas Equações 8 e 9.

$$^{i}x_{corrigido} = ^{i} x + \Delta x(^{i}\vec{x}, \vec{q}) \tag{8}$$

$$^{i}y_{corrigido} = ^{i} y + \Delta x(^{i}\vec{x}, \vec{q}) \tag{9}$$

Onde $\Delta x(^{i}\vec{x}, \vec{q})$ e $\Delta y(^{i}\vec{x}, \vec{q})$ são efeitos não-lineares que são dependentes da posição do *pixel*, portanto variam dependendo da posição no sistema de coordenadas da imagem S_i. Uma abordagem padrão para aplicar correções as imagens geradas pela câmera é modelar os erros como descrito nas Equações 10 e 11 para distorções radiais e como nas Equações 12 e 13 para distorções tangenciais.

$$^{real}x = x(1 + q_1 r^2 + q_2 r^4 + q_3 r^6) \tag{10}$$

$$^{real}y = y(1 + q_1 r^2 + q_2 r^4 + q_3 r^6) \tag{11}$$

$$^{real}y = x + [2p_1 xy + p_2(r^2 + 2x^2)] \tag{12}$$

$$^{real}y = y + [p_1(r^2 + 2y^2) + 2p_2 xy] \tag{13}$$

Onde r é a distância do *pixel* na imagem em relação ao ponto principal H, e os parâmetros q_1 e q_2 são parâmetros que devem ser encontrados para modelar a distorção parabólica da lente. Assim, a correção da lente pode ser aplicada através da transformada dada pela Equação 15.

$$^{i}_{i} = H_i^{i,dist}(^{i}\vec{x_{i,dist}}) \tag{14}$$

$$H_i^{i_{dist}} = \begin{bmatrix} 1 & 0 & \Delta x(\vec{x_{i,dist}}, \vec{q}) \\ 0 & 1 & \Delta y(\vec{x_{i,dist}}, \vec{q}) \\ 0 & 0 & 1 \end{bmatrix} \tag{15}$$

A transformada geral (Equação 17) relacionada aos parâmetros intrínsecos da câmera pode ser representada pela junção das transformadas dadas pelas Equações 7, 6 e 15.

$$H_i^c = H_i^{i_{dist}} H_i^p H_p^c \tag{16}$$

$$H_i^c(\vec{x_c}, \vec{q}) = \begin{bmatrix} c & cs & x_H + \Delta x(x,q) \\ 0 & c(1+m) & y_H + \Delta y(x,q) \\ 0 & 0 & 1 \end{bmatrix} \tag{17}$$

2.3 Reconhecimento de características descritivas de imagens

O reconhecimento de características que possam ser usadas para identificar objetos ou referenciar pontos entre diferentes imagens é um dos principais focos da visão computacional e possui diferentes métodos que podem ser aplicados dependendo das características e relações entre imagens demandados pela análise realizada. Diversos métodos foram desenvolvidos ao longo dos anos, com o aumento da complexidade das características observadas conforme o poder computacional cresceu. Entre os primeiros métodos desenvolvidos encontram-se os detectores de cantos e bordas, dois dos quais são bem conhecidos são os métodos de detecção de cantos de Harris (HARRIS; STEPHENS, 1988) ou de Shi-Tomasi (SHI; TOMASI, 1994). Métodos mais recentes focam em reconhecer pontos chave da imagem e encontrar descritores para pontos e então comparar imagens entre si com essas informações para encontrar relações, como mudanças de orientação ou posição de um objeto dentro da imagem. Entre os métodos dessa categoria encontram-se:

- SIFT (*Scale-Invariant Feature Transform*, Transformada Invariante em Escala de Características) (LOWE, 2004), focado em encontrar pontos chave e descritores entre imagens com mudanças de escala entre si,

- FAST (*Features from Accelerated Segment Test*, Características obtidas de Teste de Segmentação Acelerada) (ROSTEN; DRUMMOND, 2006), semelhante ao SIFT porém pensado para execução mais rápida, como em aplicações SLAM (*Simultaneous Localization and Mapping*, Localização e Mapeamentos Simultâneos),
- BRIEF (*Binary Robust Independent Elementary Features*) (CALONDER et al., 2010), focado em encontrar descritores com menor uso de memória e obter uma comparação mais rápida em relação ao SIFT,
- ORB (*Oriented FAST and Rotated BRIEF*) (RUBLEE et al., 2011), método que utiliza FAST para encontrar os pontos chave, aplica uma versão modificada dos descritores usados no método BRIEF para melhora de performance e aplica outras técnicas adicionais para lidar com a descrição de orientação

Entre os métodos citados, o método utilizado nesse trabalho é a detecção de cantos de Harris, uma vez que nosso escopo envolve a detecção de pontos chave, tanto para calibração quanto para detecção de cantos de um objeto. O método de detecção de Harris encontra uma diferença na intensidade dos *pixels* de uma imagem em tons de cinza em torno do *pixel* examinado. A expressão utilizada é dada pela Equação 18:

$$E(u,v) = \sum_{x,y} \underbrace{w(x,y)}_{\text{janela de observação}} \; [\; \underbrace{I(x+u,y+v)}_{\text{desvio de intensidade}} - \underbrace{I(x,y)}_{\text{intensidade}} \;]^2 \tag{18}$$

Onde a janela de observação utilizada pode ser tanto uma janela retangular ou uma janela Gaussiana e serve para dar pesos diferentes aos *pixels* observados. Os cantos são detectados para os valores máximos de $E(u,v)$.

A Equação 18 pode ser modificada para a Equação 19.

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix} \tag{19}$$

Onde M é dada pela Equação 20.

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x I_x & I_x I_y \\ I_x I_y & I_y I_y \end{bmatrix} \tag{20}$$

I_x e I_y são as derivadas da imagem nas direções x e y, respectivamente.

Nesse ponto, define-se uma função de graduação, dada pela Equação 21 usada para determinar se a janela observada contem um canto.

$$R = det(M) - k(tr(M))^2 \tag{21}$$

Sendo λ_1 e λ_2 os autovalores da matriz M, o determinante de M, $det(M)$, é igual a $\lambda_1 \lambda_2$ e o traço de M, $tr(M)$ é igual a $\lambda_1 + \lambda_2$. Assim, a função de graduação pode apresentar 3 resultados diferentes dependendo dos valores obtidos para λ_1 e λ_2:

- Em regiões sem grandes variações os valores λ_1 e λ_2 são pequenos, e portanto $abs(R)$ é pequeno,
- Em regiões com uma borda, temos $\lambda_1 >> \lambda_2$ ou $\lambda_2 >> \lambda_1$, o que implica em $R < 0$,
- Em regiões com um canto, tanto λ_1 quanto λ_2 são grandes e portanto R também é grande.

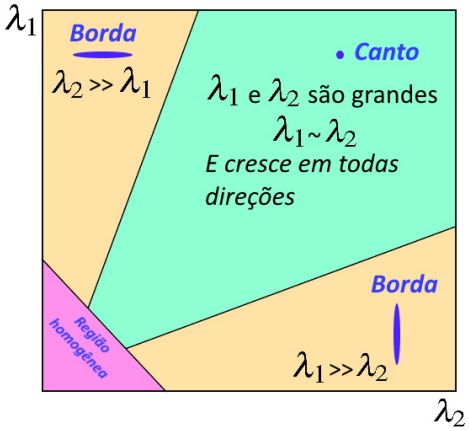

Figura 3: *Aplicação do método de Harris para identificação de pontos chave da imagem.*

Fonte: Adaptado de (OPENCV..., 2022).

Uma representação visual das regiões pode ser vista na Figura 3. A saída desse método é um mapa das intensidades R obtidas para cada pixel da imagem original, e os cantos podem ser detectados a partir da definição de um valor de comparação para R onde valores maiores de R indicam cantos.

3 METODOLOGIA APLICADA

O objetivo deste trabalho é a análise de viabilidade e a implementação de ferramentas que possibilitem obter um modelo tridimensional de um objeto através de fotogrametria. Uma proposta inicial de como obter esse resultado é delineada juntamente com o escopo esperado. Uma análise de ferramentas disponíveis para realização da tarefa também é descrita. O método proposto é descrito com os passos necessários para gerar os modelos tridimensionais desejados.

3.1 Solução proposta e considerações iniciais

A primeira etapa do processo é a realização da calibração da câmera. A partir da calibração, obtemos os parâmetros intrínsecos da câmera. A câmera utilizada terá seu foco fixo, assim a transformada que relaciona o sistema de referência da câmera ao sistema de referência da imagem, H_c^i, é constante. O processamento para cada imagem de um ensaio procura obter os parâmetros extrínsecos para imagem, tornando possível obter o posicionamento e orientação da câmera em relação ao mundo.

Utilizando uma superfície com um padrão xadrez de dimensões conhecidas, procura-se obter um modelo tridimensional de um objeto de dimensões que permitam o reconhecimento de, pelo menos, quatro pontos de intersecção diferentes do padrão xadrez da superfície (Figura 4). Esses pontos são necessários para obter o posicionamento da câmera em relação a superfície.

A proposta para obter um modelo tridimensional do objeto é gerar a descrição de um volume no espaço que contenha o objeto observado para cada imagem capturada e então realizar a intersecção desses volumes entre si, resultando num volume representativo do modelo no espaço (Figura 5). Para isso, é necessário realizar 3 etapas em cada imagem do ensaio para gerar um a descrição do volume descritivo da imagem.

A primeira etapa é aplicar um detector de cantos para identificar as intersecções entre os quadrados do padrão xadrez e identificar pelo menos quatro pontos com coordenadas conhecidas no sistema de referência no mundo. Dessa forma é possível determinar a posição e orientação da câmera no espaço, uma vez que os parâmetros intrínsecos da câmera já são conhecidos.

A segunda etapa é identificar pontos na imagem da borda do objeto observado de forma que possa se determinar uma fronteira para o objeto. Um método para obter a fronteira é determinar a envoltória convexa dos pontos da imagem que fazem parte da borda ou estão dentro o objeto. A Figura 6 mostra um resultado de ensaio no qual pontos do padrão xadrez foram utilizados para identificar o sistema de referência do mundo (as linhas de cores vermelha, verde e azul representam os eixos x, y e z, respectivamente) e a fronteira do objeto é determinada pela envoltória convexa dos pontos detectados na fronteira do

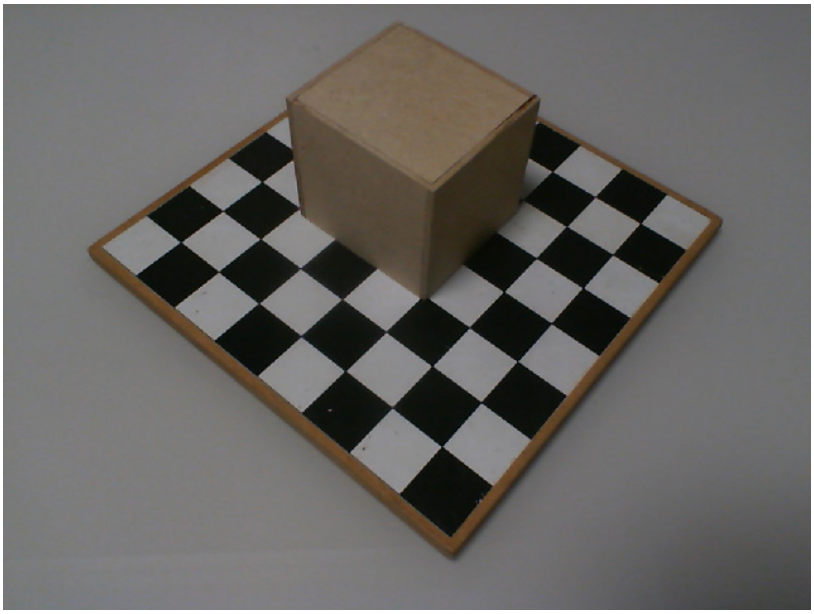

Figura 4: *Configuração proposta para realização dos ensaios.*

Fonte: O autor.

objeto. Deve-se atentar ao fato que esse método impõe limitações ao formato do objeto, pois caso a projeção dos seus pontos no plano da imagem gere uma forma com cavidades, a utilização da envoltória convexa acaba por mascarar a verdadeia área ocupada pelo objeto na imagem (caso que pode ser exemplificado pela Figura 7). Uma vez que o objetivo é uma análise de viabilidade da solução proposta, optou-se por descartar esses casos e utilizar apenas objetos cuja envoltória convexa dos seus pontos no plano da imagem não resulta-se em perda de informação sobre o objeto.

A terceira etapa consiste em determinar a projeção dos pontos que determinam a envoltória complexa do objeto no plano da imagem, o plano $x-y$ do sistema de referência S_p, sobre o plano da padrão xadrez no sistema de referência real de uma linha que passe pelo ponto no plano da imagem P_p e a origem do sistema de referência da câmera, O_c (o ponto no espaço onde a câmera se encontra). O volume que limita a zona onde o objeto se encontra delimitado a partir de faces triangulares geradas por arestas definidas pelo segmento de linha entre a origem do sistema de coordenadas da câmera S_c e os pontos projetados no plano $z_w = 0$ do sistema de referência do mundo, S_w. A base do volume é a área definida pela envoltória convexa dos pontos projetados.

Com as três etapas descritas executadas para cada imagem, pode-se então utilizar os volumes obtidos para obter um modelo do objeto através da intersecção dos volumes gerados. Uma descrição detalhada do processo, algoritmo utilizado e resultados de ensaios podem ser encontrados nos Apêndices A e C.

3.2 Ferramentas e técnicas avaliadas

Para realizar a implementação da proposta, procurou-se por soluções de visão computacional e ferramentas existentes de softwares *open-source*. Como não existem limitações em relação a performance ou arquitetura utilizada, uma vez que não se trata de uma aplicação embarcada ou uma solução que deva ser integrada de alguma forma a outra ferramenta, a principal consideração em termos de de linguagem escolhida para implementação foi a faci-

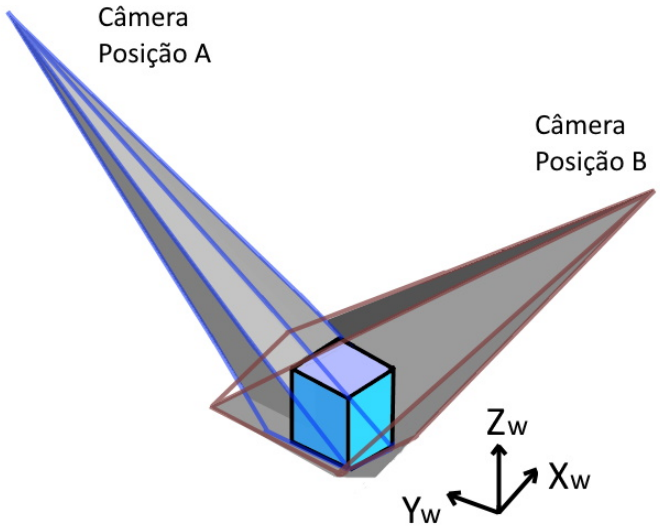

Figura 5: *Representação visual da solução proposta. Um sólido é visto a partir de 2 pontos diferentes e dois volumes representativos são gerados*

Fonte: O autor.

lidade de implementação e realização de testes de maneira iterativa. Para isso, escolheu-se como linguagem utilizada Python, devido a facilidade de prototipagem e disponibilidade de bibliotecas *open-source* para diversas aplicações com documentação extensiva.

3.2.1 Visão computacional e OpenCV

Uma vez que o intuito é verificar a validade do método da intersecção dos volumes para obter um modelo tridimensional do objeto, a implementação dos componentes relacionados a visão computacional é feita utilizando a biblioteca *OpenCV* (OPENCV..., 2022). *OpenCV* oferece implementações de diversos métodos como reconhecimento de padrões específicos utilizados para calibração, realização da calibração de uma câmera, correções de distorções da lente em imagens, detecção de características específicas da imagem e obtenção de transformadas entre sistemas de referência relacionados a imagem. Além disso, *OpenCV* é uma biblioteca de código aberto e extensamente documentada, o que facilita seu uso, uma vez que referências e exemplos de aplicações são encontradas na sua documentação.

Para realização das operações como transformadas em sistemas de referência, utiliza-se outra biblioteca de código aberto chamada *NumPy* (NUMPY..., 2022). *NumPy* possui diversas implementações relacionadas a sistemas lineares, manipulações de vetores e outras funcionalidades matemáticas com uma interface de fácil uso, além de acompanhar outras bibliotecas como a *matplotlib* para visualização de gráficos e imagens.

3.2.2 Formatos de visualização e ferramentas relacionadas

Existem diversos formatos disponíveis para exibição e armazenamento de malhas triangulares (*meshes*, em inglês) utilizadas para descrever objetos tridimensionais. Formatos como *Waveform .OBJ* (ALIAS/WAVEFRONT..., 2022) são mais comuns em utilizações relacionadas com computação gráfica, como desenvolvimento de jogos e animações, enquanto outros formatos como STL (STEREOLITHOGRAPHY..., 1989) são mais utilizados em aplicações como usinagem e impressão 3D. Há ainda formatos que são utilizados de

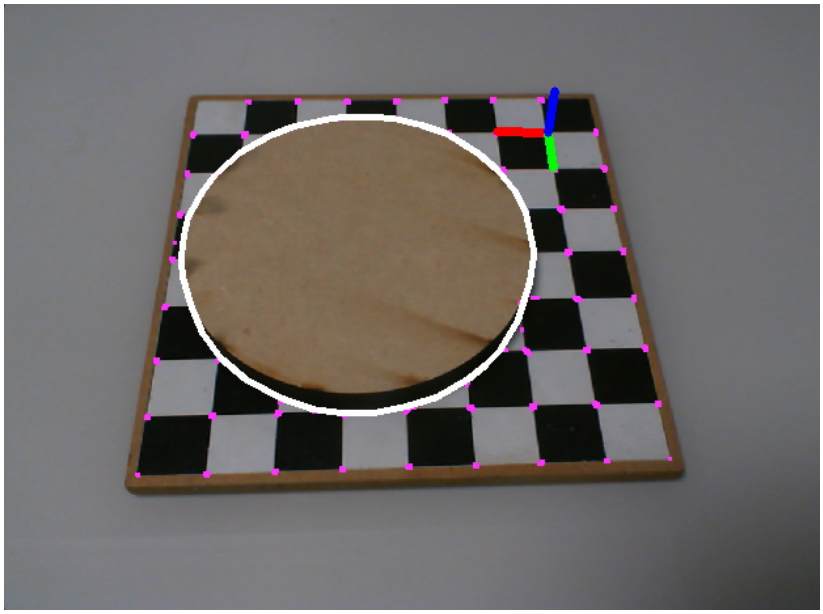

Figura 6: *Imagem de ensaio com pontos de intersecções e fronteira do objeto identificados. O referencial do espaço é mostrado com linhas das cores vermelha, verde e azul e a fronteira do objeto é determinada por uma linha branca.*

Fonte: O autor.

maneira mais abrangente, como o formato PLY (TURK, 1994), utilizado em modelagem 3D, jogos, usinagem e escaneamento tridimensional. Os formatos analisados para descrição dos modelos tridimensionais gerados foram o STL e PLY, pois ambos são comuns e possuem uma especificação com possibilidade de formatação em caracteres ASCII (além de uma formatação binária), o que facilita a verificação e apuração de erros nos arquivos de saída.

O formato STL é mais rígido em sua formatação do que o formato PLY, possuindo maiores restrições em relação a como ordenar os vértices que compõem cada triângulo, como especificar as normais relacionadas às faces e formatação do arquivo. O formato PLY é mais simples, consistindo basicamente de uma lista de vértices descritos por suas componentes e vetores normais e uma lista das faces da malha gerada especificando o número de vértices usados na face e sua ordem. A maior diferença é poder descrever objetos com polígonos que não são triângulos.

Para gerar os arquivos e poder manipular suas características, como vértices e vetores normais em relação aos vértices e faces, além de poder visualizar os resultados durante a geração dos arquivos, foi utilizada a biblioteca *PyVista* (PYVISTA..., s.d.). A biblioteca oferece interfaces para escrita e leitura de arquivos em vários formatos e permite realizar operações sobre descrições de objetos como modificação de malhas. A visualização dos modelos é feita através do *VTK* (*Visualization Toolkit*) (VTK..., s.d.), um sistema de software de operações gráficas para o qual *PyVista* serve como um *wrapper*, facilitando a utilização dentro de scripts Python.

Para visualização dos modelos gerados foram utilizadas duas ferramentas de código aberto e distribuídas gratuitamente: *Meshmixer* (AUTODESK..., 2022) e *Meshlab* (CIGNONI et al., 2008). Ambas ferramentas são usadas para visualizar e editar malhas 3D triangulares. *Meshmixer* pode ser utilizada para verificar a interseção de objetos, uma vez que oferece operações booleanas como diferença, união e intersecção de malhas, enquanto a *Meshlab*

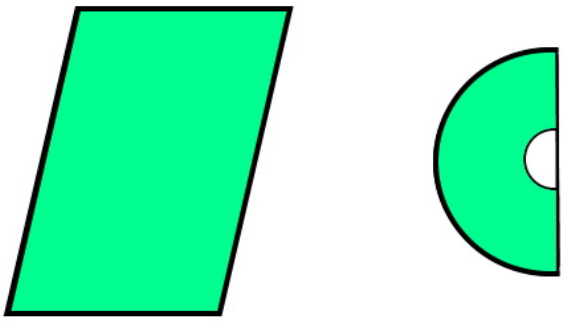

Figura 7: *Envoltória convexa sobre duas formas geométricas. Na esquerda, a fronteira do paralelograma coincide com a envoltória convexa dos pontos da forma geométrica. Na direita, a envoltória convexa forma um semicírculo, não sendo coincidente com a fronteira do disco.*

Fonte: O autor.

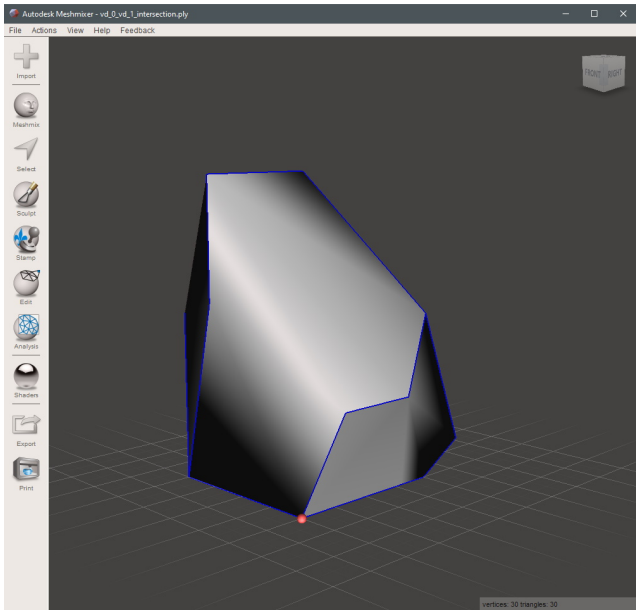

Figura 8: *Interface gráfica do software Meshmixer.*

pode ser utilizada para outras operações, como correções na malha e aplicações de filtros. No nosso contexto, *Meshlab* é utilizada para todas operações de correção das malhas geradas ou visualização, uma vez que possui uma interface mais leve e operação mais simples, enquanto a *Meshmixer* é usada apenas pela sua funcionalidade de intersecção de malhas que pode ser utilizada para verificar resultados obtidos durante simulações e ensaios.

3.3 Calibração da câmera

Para obtenção dos parâmetros intrínsecos, foram utilizados métodos presentes dentro da biblioteca *OpenCV* que permitem a calibração através da utilização de um padrão de imagem conhecido. O processo consiste em utilizar um padrão xadrez ou um padrão de pontos, como os mostrados na Figura 10, fixo em uma superfície plana e obter diversas

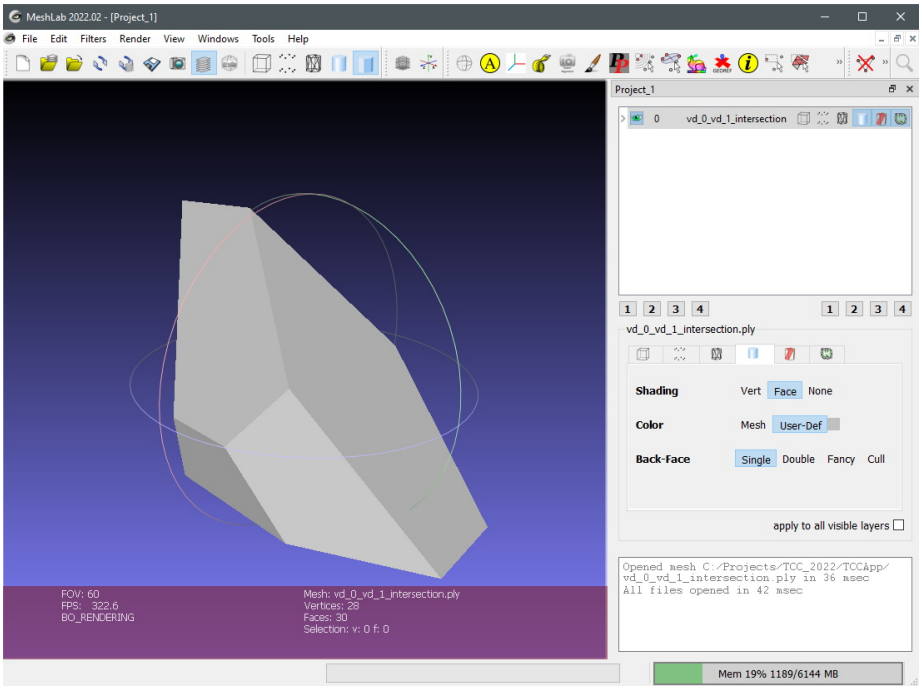

Figura 9: *Interface gráfica do software Meshlab.*

imagens com a câmera a ser calibrada. O processo considera que a câmera tem o foco fixo, e portanto a câmera deve ser configurada de acordo. *OpenCV* oferece métodos para comunicação, captura e visualização das imagens, facilitando o processo. Para realizar a calibração, foi desenvolvido um *script* em Python (listado no Apêncice A, seção A.1) para realizar a calibração a partir de um conjunto de imagens de um padrão xadrez.

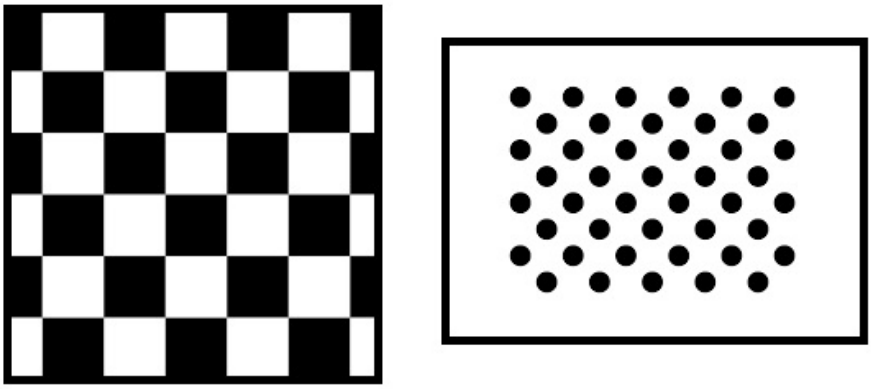

Figura 10: *Exemplo de padrões utilizado para calibração da câmera utilizando OpenCV. Na esquerda há um padrão xadrez e a direita um padrão de pontos.*

O padrão xadrez utilizado possui lados com oito quadrados, cada um com arestas de 22,5 mm (Figura 11) e foram obtidas 20 imagens de calibração com ângulos diferentes. A câmera utilizada teve suas configurações ajustadas para manter um foco fixo. Os métodos da biblioteca *OpenCV* utilizados e a ordem de chamada dos métodos foram:

- *findChessboardCorners*, responsável por encontrar as intersecções do tabuleiro e

ordená-las como uma lista de posições na imagem,

- *calibrateCamera*, a qual utiliza as informações obtidas com a função anterior e informações sobre o padrão de calibração utilizado (no nosso caso, padrão com 7 intersecções na vertical e 7 intersecções na horizontal, com arestas de 22,5 mm) para obter os parâmetros intrínsecos da câmera.

O resultado encontrado pelo processo de calibração resultou nos parâmetros intrínsecos mostrados na Tabela 1 e nas Equações 22, 23, 24, 25 e 26.

Símbolo	Parâmetro	Valor obtido
c	Distância focal	$615.2mm$
s	Compensação de cisalhamento	$0.0mm$
m	Diferença de escala	$0.4mm$
x_H	Coordenada x do ponto principal H	$328.0mm$
y_H	Coordenada y do ponto principal H	$234.8mm$
p_1	Parâmetro de distorção radial 1	$-0,0023mm^{-1}$
p_2	Parâmetro de distorção radial 2	$0,0017mm^{-1}$
q_1	Parâmetro de distorção tangencial 1	$0,0176mm^{-1}$
q_2	Parâmetro de distorção tangencial 2	$-0,1377mm^{-3}$
q_3	Parâmetro de distorção tangencial 3	$0,7881mm^{-5}$

Tabela 1: *Parâmetros da câmera obtidos pela calibração.*

Fonte: O autor.

$$H_i^c = \begin{bmatrix} c & s & x_H \\ 0 & c(1+m) & y_H \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 615.22 & 0.00 & 327.87 \\ 0.00 & 615.65 & 234.79 \\ 0.00 & 0.00 & 1.00 \end{bmatrix} \tag{22}$$

Figura 11: *Padrão xadrez utilizado para calibração (arestas dos quadrados com 22,5 mm).*

$$^{real}x = x(1+0,0176r^2+-0,1377r^4+0,7881r^6) \tag{23}$$

$$^{real}y = y(1+0,0176r^2+-0,1377r^4+0,7881r^6) \tag{24}$$

$$^{real}y = x + [-0,0046.xy + 0,0017(r^2 + 2x^2)] \tag{25}$$

$$^{real}y = y + [-0,0023(r^2 + 2y^2) + 0,0034.xy] \tag{26}$$

Uma vez com a calibração realizada, é possível utilizar os resultados obtidos para aplicar a correção das distorções nas imagens do ensaio e obter a posição e orientação da câmera a partir da equivalência entre pontos conhecidos do padrão, como mostrado na Figura 12, onde os resultados da calibração são utilizados para obter o posicionamento da câmera durante os ensaios de calibração. A configuração dos ensaios realizados pode ser encontrada no Apêndice B.

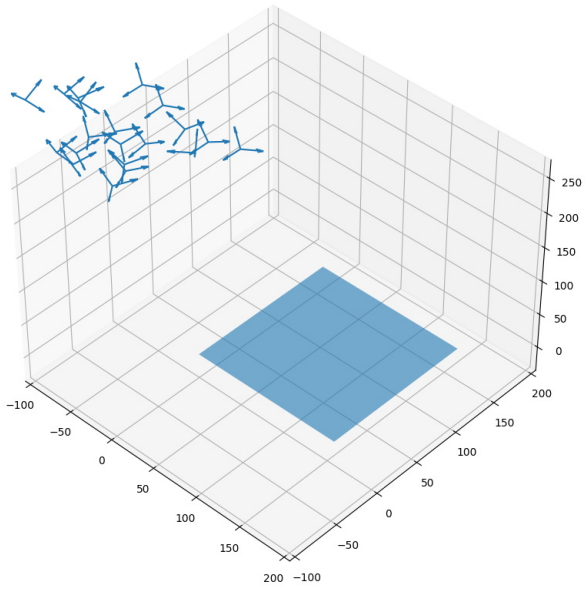

Figura 12: *Posições e orientações da câmera durante a captura de imagens usadas para calibração obtidas a partir do processo de calibração. Escala em milímetros.*

Fonte: O autor.

4 RESULTADOS DOS ENSAIOS

Para a validação da metodologia empregada com a câmera calibrada foram realizados testes com quatro objetos diferentes: dois paralelepípedos e dois discos com dimensões diferentes entre si (Figura 13). As dimensões dos objetos são listadas nas Tabelas 2 e 3.

Para cada objeto foram capturadas vinte imagens, girando o tabuleiro (sobre qual foi posicionado o objeto) no sentido horário em relação a câmera entre 30 a 35 graus entre cada imagem obtida. Após uma rotação completa da plataforma, o ângulo da câmera em relação ao plano tabuleiro foi mudado, de forma a observar o objeto de ensaio o mais próximo de um posicionamento perpendicular ao plano do tabuleiro. Enquanto as primeiras imagens encontravam-se com o eixo z na direção negativa da câmera em um ângulo de aproximadamente 45 graus em relação ao plano do tabuleiro, as últimas imagens do ensaio são tomadas com um ângulo de aproximadamente 60 graus de inclinação.

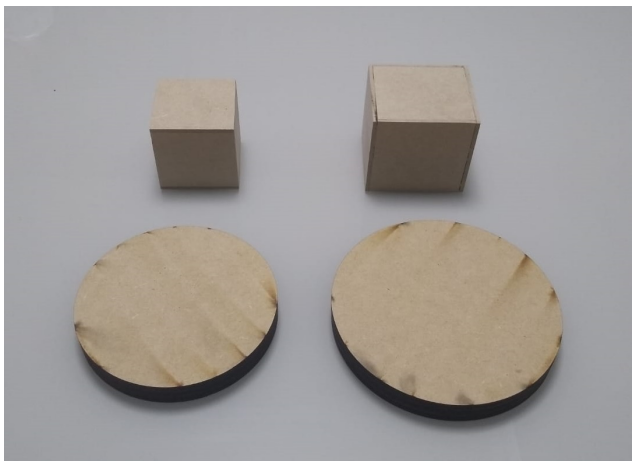

Figura 13: *Objetos utilizados durante o teste: um cubo, um paralepípedo e dois cilindros de diferentes diâmetros.*

Fonte: O autor.

Objeto de ensaio	Largura (mm)	Profundidade (mm)	Altura (mm)
1	59.43 ($\pm$0.01)	59.18 ($\pm$0.01)	59.50 ($\pm$0.01)
2	75.95 ($\pm$0.01)	69.14 ($\pm$0.01)	69.05 ($\pm$0.01)

Tabela 2: *Dimensões dos paralelepídos utilizados nos ensaios.*

Fonte: O autor.

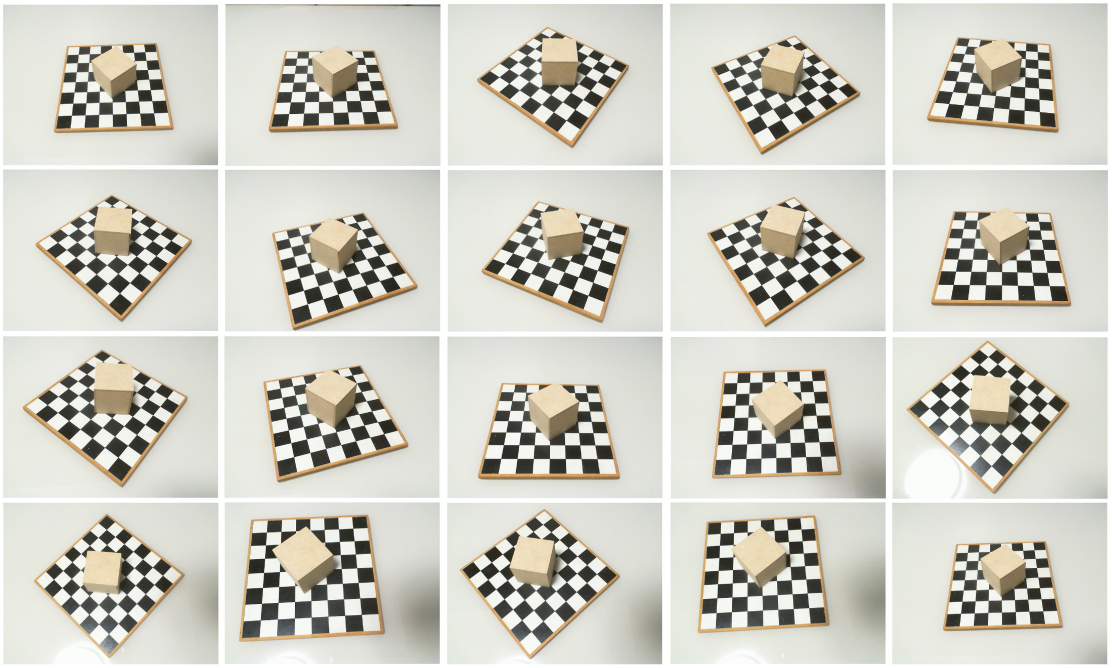

Figura 14: *Conjunto de imagens obtidas para o objeto de ensaio 1.*

Fonte: O autor.

Objeto de ensaio	Diâmetro (mm)	Altura (mm)
3	127.62 ($\pm$0.01)	18.33 ($\pm$0.01)
4	146.93 ($\pm$0.01)	18.36 ($\pm$0.01)

Tabela 3: *Dimensões dos discos utilizados nos ensaios.*

Fonte: O autor.

Após a aquisição das imagens para cada objeto, as correções das distorções radiais e tangenciais foram aplicadas utilizando funções da biblioteca *OpenCV* (Figura 15) utilizando os parâmetros encontrados na seção 3.3. Com as imagens corrigidas, o método de Harris para detecção de cantos foi aplicado às imagens utilizando *OpenCV* e modificando os parâmetros até se obter a detecção das intersecções do tabuleiro. Alguns pontos na borda do objeto também foram detectados pela aplicação da detecção de cantos. Como o objetivo do trabalho é verificar a validade do método proposto, pontos na borda do objeto foram selecionados manualmente, uma vez que apenas os pontos de intersecção do tabuleiro foram detectados corretamente em todos os casos, enquanto pontos na borda do objeto não foram detectados em todos os casos com os parâmetros utilizados. A Figura 16 apresenta os resultados das envoltórias convexas em torno dos pontos encontrados pelo método de Harris e dos pontos verificados manualmente.Observa-se que uma vez que os pontos detectados não são suficientes para que sua envoltória convexa no plano da imagem corresponda a área ocupada pela imagem, resultados obtidos por esses pontos não seriam representativos da forma observada. Portanto, para geração dos volumes descritivos de cada imagem, foram utilizados pontos selecionados manualmente.

O processo utilizado para se determinar o volume descritivo de cada imagem consiste em obter, a partir da posição da câmera em relação ao tabuleiro e da posição dos pontos na imagem, a projeção da imagem no plano do tabuleiro. Isso pode ser feito realizando as seguintes etapas:

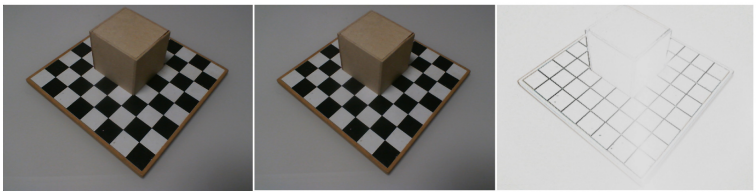

Figura 15: *Aplicação da compensação das distorções. Na esquerda, imagem obtida pela câmera. No centro, imagem com compensação de distorções radiais e tangenciais aplicada. Na direita, diferença entre as imagens com contraste ajustado para realçe.*

Fonte: O autor.

- Obter a posição do ponto P_i no plano da imagem no sistema de referência da câmera S_c,
- Determinar a intersecção da reta que passa pela origem do sistema de referência S_c e pelo ponto P_i pelo plano do tabuleiro, ou seja, $z_w = 0$,
- Repetir o processo para todos os pontos que definem a envoltória convexa dos pontos da borda do objeto na imagem,
- Gerar uma malha descritiva do volume utilizando triângulos cujas arestas são as retas entre a origem do sistema de referência da câmera O_c e dois pontos projetados no plano $z_w = 0$ e entre o par de pontos.

A seguir, pode-se usar a Equação 29, usando a Equação 30 para determinar o parâmetro t e encontrar a intersecção da reta que passa pela origem do sistema de referência da câmera S_c e o ponto P_i examinado com o plano $z_w = 0$ do sistema de referência real.

$$\vec{x_{P,c}} = H^i_c \vec{x_{P,i}} = \begin{bmatrix} R_i^{c-1} & -R_i^{c-1}\vec{O_i^c} \\ 0_{3x1} & 1 \end{bmatrix} \vec{x_{P,i}} \tag{27}$$

$$\vec{x_{P,w}} = H^c_w \vec{x_{P,c}} = \begin{bmatrix} R_c^{w-1} & -R_w^{c-1}\vec{O_c^w} \\ 0_{3x1} & 1 \end{bmatrix} \vec{P_{P,c}} \tag{28}$$

$$P_w(t) = O^c_w - t(O^c_w - x_{P,w}) \tag{29}$$

$$t = \frac{O^c_w}{O^c_w - x_{P,w}} \tag{30}$$

Com posse dos pontos referenciados ao sistema de refefência do mundo, podemos então criar as malhas triangulares. Para isso, foi utilizada a biblioteca *PyVista* mencionada anteriormente.

Os resultados são apresentados nas Figuras 17, 18, 19 e 20, que indicam qualitativamente que as malhas obtidas apresentam formatos semelhantes aos esperados pelo método proposto. Uma vez que as malhas foram criadas utilizando o formato *.ply*, foi possível verificar a visualização do seu formato utilizando ferramentas como *Meshlab*. Porém não foi possível verificar a aplicabilidade do método proposto devido a problemas no processo de intersecção de malhas.

Como pode ser visto na Figura 21, a intersecção entre dois volumes gerados de dois ensaios do mesmo objeto gera um volume de intersecção. Porém, para outras combinações

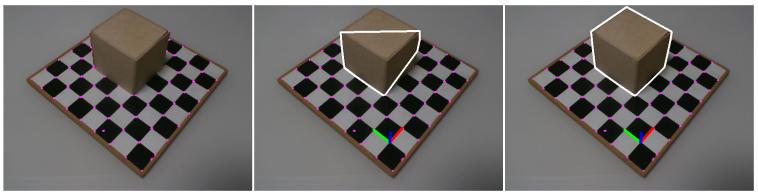

Figura 16: *Seleção de pontos da imagem e envoltórias convexas geradas para as imagens. Na esquerda, imagem com detecção de cantos pelo método de Harris aplicada. No centro, envoltória convexa utilizando apenas os pontos detectados pelo método de Harris. Na direita, envoltória convexa utilizando pontos selecionados manualmente.*

Fonte: O autor.

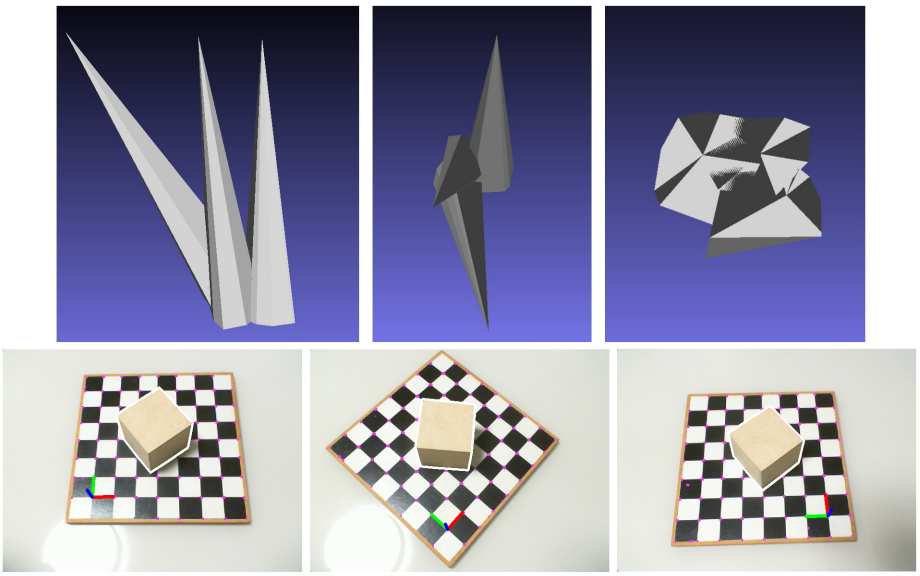

Figura 17: *Meshes geradas através de imagens capturadas do objeto de ensaio 1.*

Fonte: O autor.

de dois volumes, como pode ser visto na Figura 22, a intersecção falha, gerando um volume que parece a diferença entre a intersecção dos dois volumes e um dos volumes utilizados para gerar o volume de intersecção.

Entre os motivos considerados para essas falhas, uma das possibilidades analisadas são limitações na representação do sólido realizada pela biblioteca *PyVista* em relação à interpretação das malhas geradas. Para realizar uma intersecção booleana entre as malhas, é preciso um controle da orientação dos vetores normais de vértices e faces da malha, o que não pode ser gerado de maneira consistente pelo de maneira automática pelo*PyVista*. A implementação do método proposto (para detalhes dos algoritmos e códigos gerados, ver o Apêndice A) então gera os arquivos *.ply* manualmente, de forma a gerar *meshes* cujas normais das faces apontem para o lado externo do volume. Essa é uma das condições citadas na documentação da biblioteca *PyVista* pare realizar a intersecção booleana de *meshes*. Durante testes da aplicação da intersecção booleana foi verificado mesmo com a utilização dos arquivos *.ply* gerados manualmente, quando os modelos são carregados pelo *PyVista*, a intersecção não funciona para vários dos casos dos ensaios (Figura 23)

A ferramenta *Meshmixer* possui opções para realização da intersecção booleana entre

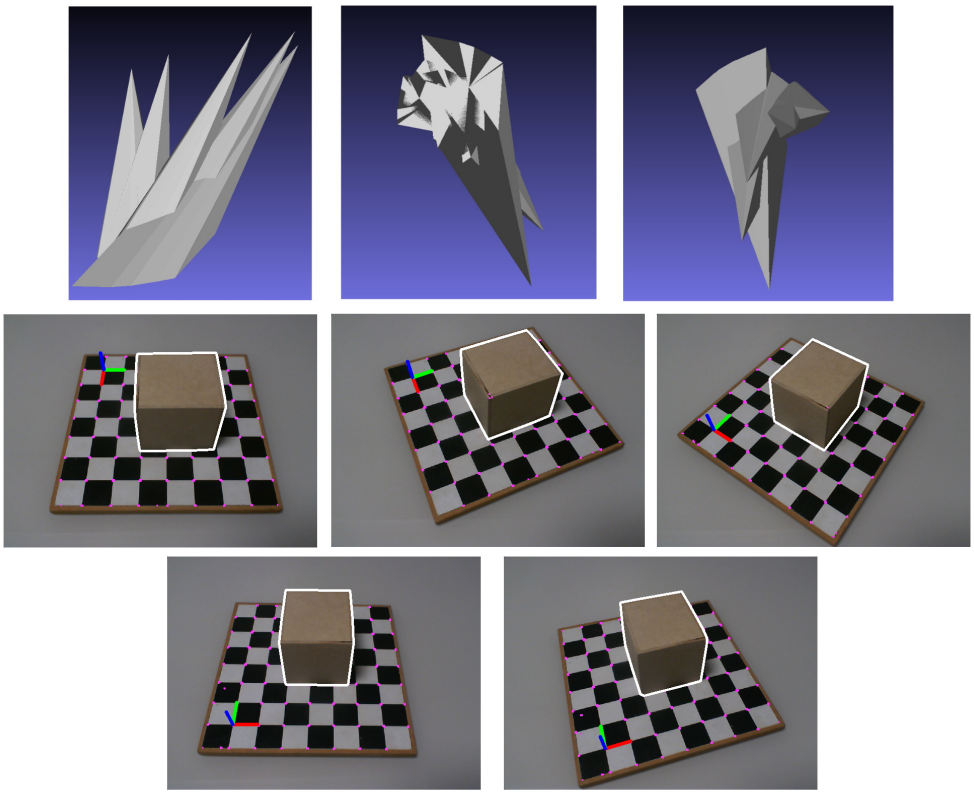

Figura 18: *Meshes geradas através de imagens capturadas do objeto de ensaio 2.*

Fonte: O autor.

sólidos, porém em nenhum dos testes utilizando a ferramenta foi possível realizar uma intersecção com sucesso. Logo, mesmo não utilizando o código desenvolvido para realizar a intersecção, não foi possível validar o funcionamento do método proposto utilizando a ferramenta *Meshmixer* também devido a erros no processo de intersecção.

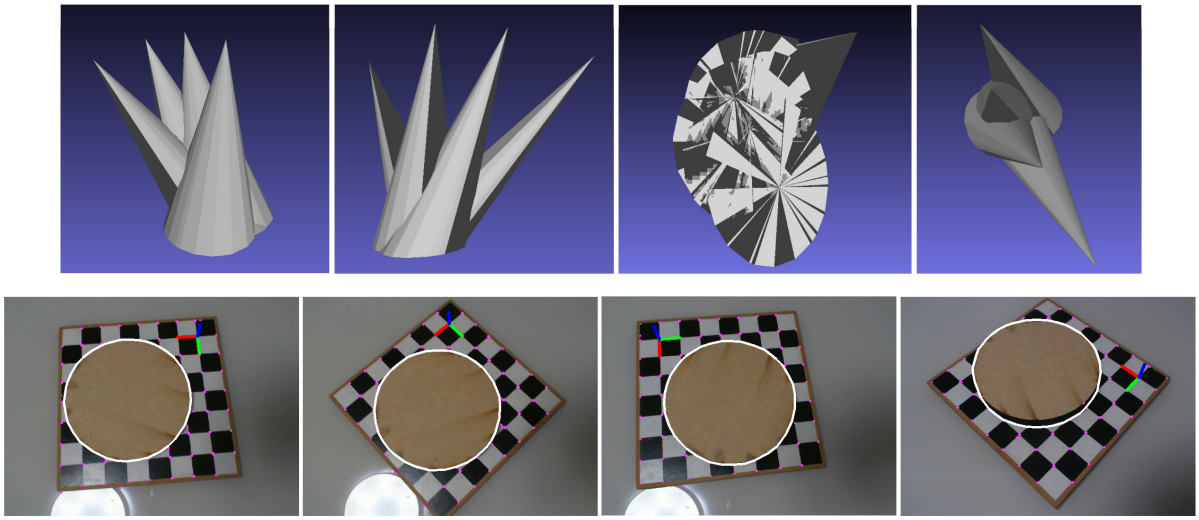

Figura 19: *Meshes geradas através de imagens capturadas do objeto de ensaio 3.*

Fonte: O autor.

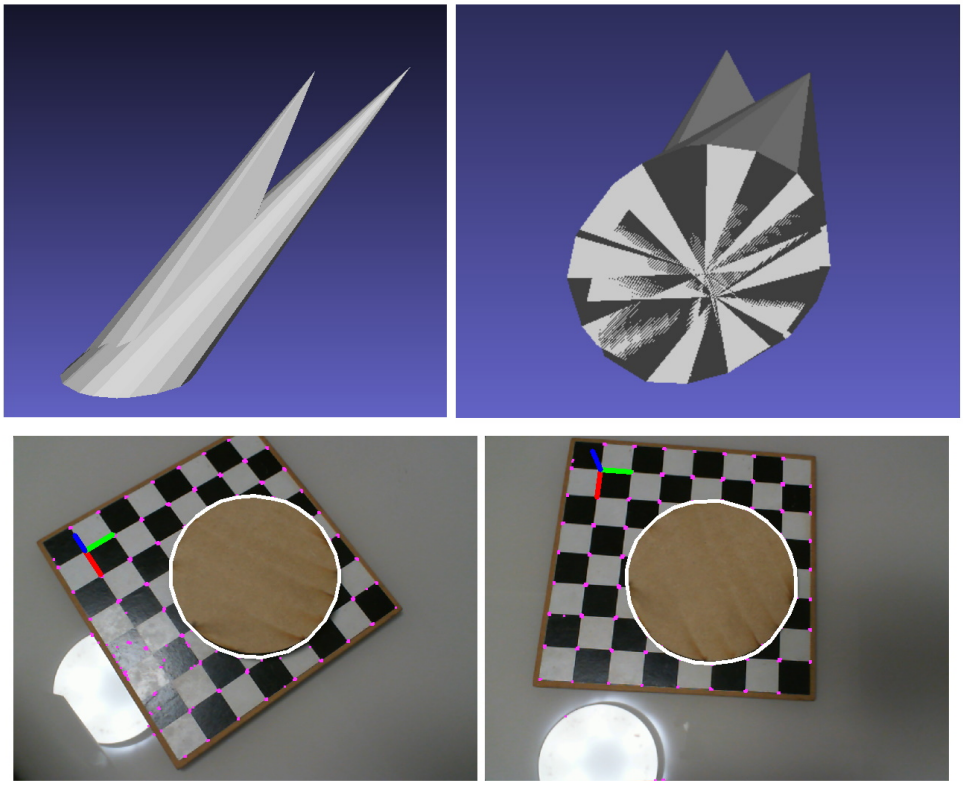

Figura 20: *Meshes geradas através de imagens capturadas do objeto de ensaio 4.*

Fonte: O autor.

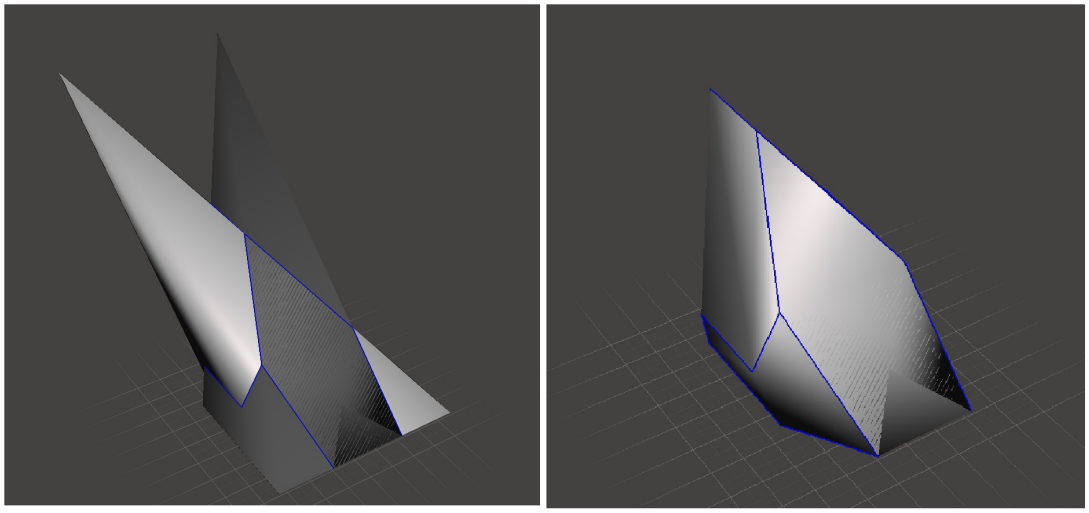

Figura 21: *Exemplo de volume de intersecção gerado com sucesso a partir de outros dois volumes representativos utilizando o método proposto. Na esquerda estão os volumes descritivos originais e na direita o volume de intersecção gerado.*

Fonte: O autor.

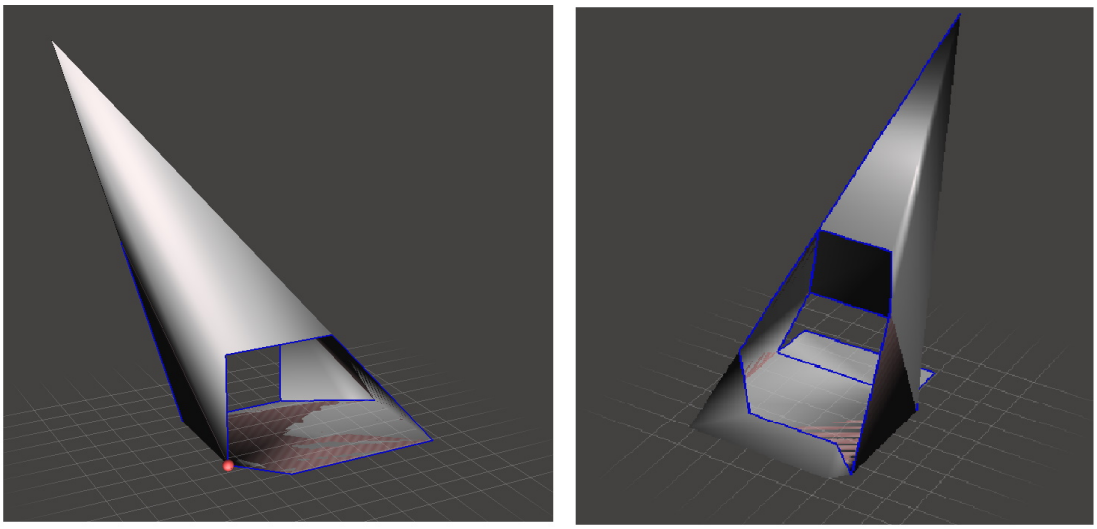

Figura 22: *Exemplo de dois volumes de intersecção gerados a partir de outros dois volumes representativos utilizando o método proposto com resultado insatisfatório.*

Fonte: O autor.

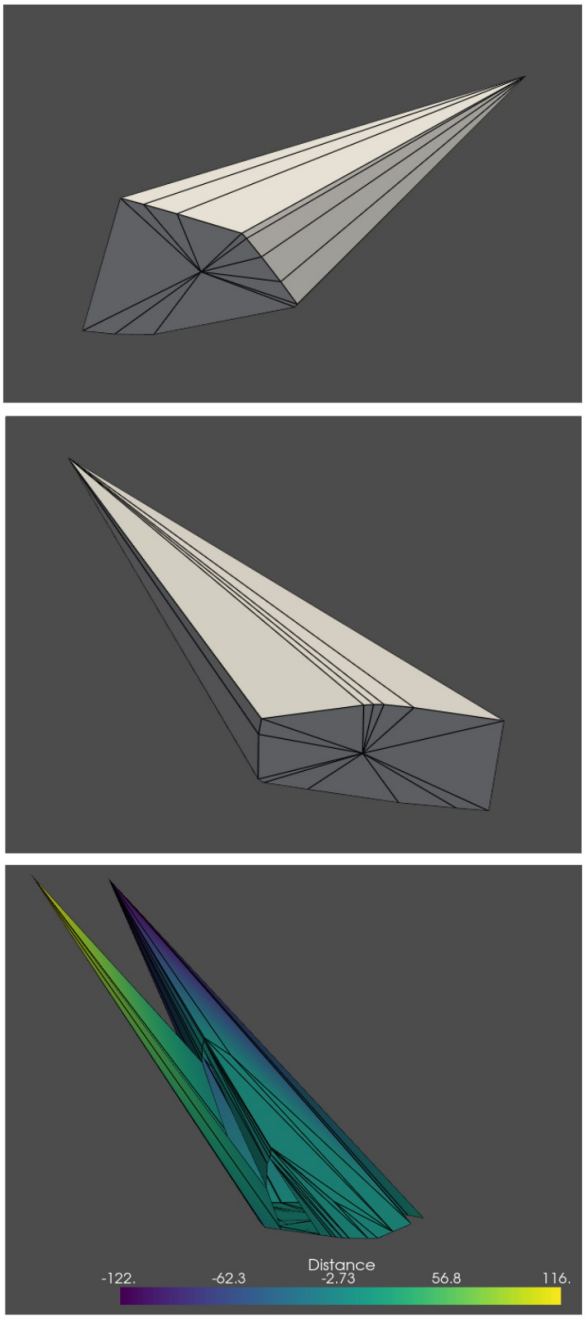

Figura 23: *Visualização de dois volumes gerados a partir dos ensaios usando a biblioteca PyVista (topo e centro) e da intersecção gerada pela biblioteca (canto inferior).*

Fonte: O autor.

5 DISCUSSÃO DOS RESULTADOS

O método sugerido não foi aplicado com sucesso nas condições testadas devido aos problemas na análise das malhas geradas. A validação da proposta não pode ser confirmada, embora os resultados obtidos através da visualização das malhas geradas sugira, por inspeção da sobreposição dos volumes representativos gerados, que a intersecção dos sólidos resulta em um volume que ao menos envolva o objeto de ensaio. Caso a intersecção de malhas possa ser realizada com sucesso, para ensaios futuros, imagens com ângulos de incidência do eixo do sistema de referência da câmera devem ser adicionadas para que o volume gerado seja mais representativo das formas observadas.

Quanto aos ensaios realizados, melhorias na metodologia de captura das imagens podem ser realizadas, como a utilização de um plano com padrão xadrez adicional, com posicionamento ortogonal ao plano usado como base, aplicação de iluminação uniforme ao objeto de ensaio e utilização de diferentes cores de luz aplicadas em direções diferentes. A utilização de um plano com padrão xadrez adicional permite a aquisição de imagens com a câmera paralela ao plano da superfície onde o objeto se encontra, garantindo assim uma descrição do objeto para ângulos entre a superície e o eixo da câmera baixos. A iluminação uniforme possibilita um melhor desempenho de métodos de detecção de cantos para detecção das bordas do objeto de ensaio. Como pode ser observado para alguns dos ensaios mostrados no Apêndice C, a detecção de cantos é prejudicada por regiões onde a borda do objeto encontra-se na sua sombra. A utilização de cores diferentes em diferentes direções possibilita a implementação de mecanismos que permitam a detecção da orientação do objeto sem a necessidade da determinação de 4 pontos de referência na imagem. Assim pode-se utilizar apenas um ponto conhecido como origem e usar as cores para determinar as direções dos eixos no sistema de referência do mundo.

Para correção do algoritmo proposto, uma solução é a criação de um método próprio para determinação dos volumes de intersecções ou a modificação dos métodos existentes dentro da biblioteca *PyVista* para remover a necessidade da orientação específica para as normais dos polígonos da malha tridimensional. Outra solução de menor custo é realizar testes utilizando outras funcões relacionadas a intersecção presentes na biblioteca *PyVista*, como funções de determinam as linhas nos sólidos utilizados na intersecção onde a intersecção ocorre. Utilizando essas linhas e as interseccionando com as linhas que compõem o sólido representativo, é possível encontrar os pontos representativos do volume interseccionado. Com esses pontos então podem ser geradas novas malhas e então repetir o processo iterativamente até todos volumes gerados nos ensaios serem utilizados.

APÊNDICE A - DESCRIÇÃO DA IMPLEMENTAÇÃO PROPOSTA

Para obter os volumes descritivos dos sólidos observados durante os ensaios, foi necessário desenvolver código para implementação dos algoritmos propostos para os processos de captura de imagens, calibração da câmera, localização da câmera nos ensaios e geração dos volumes descritivos utilizados para o processo de intersecção e geração do modelo tridimensional final. A descrição do código desenvolvido para cada etapa, com algoritmos aplicados e trechos relevantes do código necessários para implementação são descritos a seguir.

A.1 Captura de imagens e calibração da câmera

A captura de imagem pode ser feita através de um *script* Python responsável por realizar chamadas a biblioteca OpenCV que permitem a manipulação da câmera para captura de vídeo e configuração da câmera. Um dos *scripts* usados pode ser visto na Figura 24

Para realizar a calibração da câmera, o código mostrado nas Figuras 25, 26, 27, 28 e 29. é utilizado. A função *PerformCameraCalibration* verifica as imagens encontradas em um diretório que deve conter as imagens do ensaio de calibração e então encontra os parâmetros intrínsecos da câmera e os parâmetros de compensação de distorções radial e tangencial. A primeira parte do código (Figura 25), verifica se o usuário deseja salvar os resultados gerados e se o diretório informado para isso é valido. Em seguida (Figura 26), listas para armazenamento de pontos de interesse são geradas e o método varre as imagens presentes no diretório com o ensaio de calibração. Para encontrar o padrão xadrez na imagem, o método *findChessboardCorners* da biblioteca *OpenCV* é utilizado. Caso o padrão xadrez seja encontrado, os pontos do padrão são marcados na imagem para verificação posterior (Figura 27). A imagem com os pontos marcados pode ser salva ou apenas visualizada de acordo com os parâmetros passados a função *PerformCameraCalibration* pelo usuário (Figura 28). Finalmente, a obtenção dos parâmetros da câmera é realizada utilizando o método *calibrateCamera* da biblioteca *OpenCV* e a correção de distorção é aplicada as imagens do ensaio (Figura 29).

A.2 Geração dos volumes descritivos

Para gerar os volumes, foram criados dois *scripts*, *FeatureDescriptor* e *VolumeDescriptor*. *FeatureDescriptor* é um *script* com diversas funções para obter características das imagens de ensaio e *VolumeDescriptor* é um *script* com uma classe gerada para lidar

com a geração dos arquivos *.ply* e realizar as operações de intersecção nos sólidos. Na Figura 30 pode ser encontrado um trecho de código responsável por realizar um *setup* de parâmetros necessários para gerar as imagens, como cores do pontos usados como marcadores, parâmetros da câmera e comprimento da aresta do padrão xadrez utilizado. Após o *setup*, o código mostrado na Figura 31 pode ser usado para gerar os volumes descritivos. Esse trecho de código chama uma função responsável por obter as informações da imagem (função *GeneratePointDataOutput*, mostrada nas Figuras 32, 33, 39, 40 e 43) e então realizar chamadas a classe *VolumeDescriptor*. A função *GeneratePointDataOutput*, realiza a chamada de diversas outras funções, as quais podem ser encontradas nas Figuras 34, 35, 36, 37, 38, 41, 42

A classe *VolumeDescriptor* é responsável por gerar os volumes a partir das informações das imagens. Seu código pode ser visto nas Figuras 44, 45, 46, 47 e 48.

Um exemplo de arquivo gerado pela classe *VolumeDescriptor* pode ser visto na Figura 49, com a correspondências entre pontos da imagem e do código gerado ressaltadas.

Figura 24: *Código Python usado para realizar a captura das imagens.*

```
import cv2 as cv

CAMERA_ID = 1 #Identificacao da camera (no caso do teste,
              #esse era o ID da camera utilizada)
current_dir = "Diretorio_para_salvar_os_arquivos"
current_image = 1 #Indice para captura de imagem.

#Abrir a captura da camera
cap = cv.VideoCapture(CAMERA_ID)

if not cap.isOpened():
    print("Erro:␣acesso␣a␣camera␣nÃ£o␣obtido.")

else:
    #Leitura do frames a cada ciclo.
    while True:
        ret, frame = cap.read()

        if ret:
            #Mostra a imagem numa janela.
            cv.imshow("Imagem", frame)

        #Verifica a tecla pressionada
        key_pressed = cv.waitKey(1)
        if key_pressed == ord('i'):
            print("Info:")
            print("Diretorio␣do␣ensaio␣atual:␣%s" %(current_dir))

        #Captura da imagem
        elif key_pressed == ord('c'):
            cv.imwrite((current_dir+"/orig/capture_"+("%02d" %(
   current_image))+".png"), frame)
            gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
            cv.imwrite((current_dir+"/gray/capture_"+("%02d" %(
   current_image))+".png"), gray)

            current_image += 1
            print("%2d␣imagens␣capturadas!" %current_image)

        elif key_pressed == ord('q'):

            print("Captura␣encerrada.")
            print("%2d␣imagens␣capturadas" %current_image)
            break

    #Liberando captura e fechando janelas do OpenCV.
    cap.release()
    cv.destroyAllWindows()
```

Fonte: O autor.

Figura 25: *Código Python usado para obter os parâmetros de calibração da câmera - Verificação dos diretórios utilizados para salvar as imagens de saída.*

```
import cv2 as cv
import numpy as np
import os

def PerformCameraCalibration(self, save_calibrated_images=True,
    calib_image_names_list = [], overwrite_calibrated_images=True,
    calibrated_images_folderpath = "", grid_pattern = [7,7], side_length =
    22.5, folder_path =[], use_opencv_image_output = True):
    save_output = False

    #Verifica se sera necessario salvar as imagens apos
    #os parametros de calibracao da camera.
    if save_calibrated_images:

        #Verificacao dos diretorios fornecidos.
        if (calibrated_images_folderpath != ""):
            if not os.path.exists(calibrated_images_folderpath):
                try:
                    os.makedirs(calibrated_images_folderpath)
                    save_output = True

                except Exception:
                    print("CameraCalibration:␣PerformCameraCalibration␣
   method:␣output␣images␣could␣not␣be␣created.␣Images␣will␣not␣be␣saved.")

            else:
                save_output = True

    if save_output:
        print("CameraCalibration:␣PerformCameraCalibration␣method:␣output␣
   images␣will␣be␣saved.")
```

Fonte: O autor.

Figura 26: *Código Python usado para obter os parâmetros de calibração da câmera - Varredura das imagens de calibração, parte 1.*

```
#Estruturas usadas pelo OpenCV para realizar a calibracao.
#Listas usadas para os pontos de interesse no padrao xadrez
#na imagem e no sistema de referÃªncia do mundo.
obj_points = [] #Pontos no espaco tridimensional (real) (3D)
img_points = [] #Localizacao dos pontos na imagem (2D)

#Arrays usados durante processamento.
process_obj_points = np.zeros((grid_pattern[0]*grid_pattern[1],3), np.
    float32)
process_obj_points[:,:2] = (np.mgrid[0:grid_pattern[0], 0:grid_pattern[1]].
    T.reshape(-1,2))*side_length

#Avaliacao das imagens.
for image in calib_image_names_list:
    image_name = image.replace(folder_path, "")
    print("CameraCalibration:␣PerformCameraCalibration␣method:␣Processing␣
    file:␣%s" %(image))
    image_data = cv.imread(image)
    image_format = image_data.shape[::-1]
    image_format =[image_format[1], image_format[2]]
    print("\tCameraCalibration:␣PerformCameraCalibration␣method:␣%r" %(
    image_format))


    process_image = cv.cvtColor(image_data, cv.COLOR_BGR2GRAY)

    # Metodo "findChessboardCorners" e usado para encontrar os pontos de
    interseccao
    # no padrao xadrez da imagem.
    ret, intersects = cv.findChessboardCorners(process_image, (grid_pattern
    [0], grid_pattern[1]), flags=cv.CALIB_CB_ADAPTIVE_THRESH)
```

Fonte: O autor.

Figura 27: *Código Python usado para obter os parâmetros de calibração da câmera - Varredura das imagens de calibração, parte 2.*

```
    # Caso o padrao tenha sido identificado, o processo
    # a seguir e executado.
    if ret == True:
        print("\tCameraCalibration:␣PerformCameraCalibration␣method:␣
    Chessboard␣pattern␣found")

        obj_points.append(process_obj_points)

        # Metodo usado para refinar a determinacao do posicionamento dos
    pontos.
        refined_intersects = cv.cornerSubPix(process_image, intersects, ((
    grid_pattern[0]*2 + 1), (grid_pattern[1]*2 + 1)), (-1, -1), (cv.
    TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 30, 0.001))

        img_points.append(refined_intersects)

        # Mostra os pontos encontrados na imagem.
        cv.drawChessboardCorners(image_data, (grid_pattern[0], grid_pattern
    [1]), refined_intersects, ret)
```

Fonte: O autor.

Figura 28: *Código Python usado para obter os parâmetros de calibração da câmera - Varredura das imagens de calibração, parte 3.*

```
    filename_output = ""

    # Caso tenha sido escolhida anteriormente a opcao
    # de salvar as imagens de saida, esse trecho e executado.
    if save_output:
        if image_name in os.listdir(calibrated_images_folderpath):
            print("CameraCalibration:␣PerformCameraCalibration␣method:␣
%s␣already␣exists" %(image))
            if overwrite_calibrated_images:
                print("CameraCalibration:␣PerformCameraCalibration␣
method:␣%s␣overwritten" %(image))
                filename_output = calibrated_images_folderpath+"\\"+
image_name
        else:
            filename_output = calibrated_images_folderpath+"\\"+
image_name

        print("\t\tCameraCalibration:␣PerformCameraCalibration␣method:␣
file␣output␣path:␣%s" %(filename_output))

        if filename_output != "":
            cv.imwrite(filename_output, image_data)
            print("\t\tCameraCalibration:␣PerformCameraCalibration␣
method:␣Image␣saved.")

        # Caso se escolha mostrar a imagem de saida,
        # o codigo a seguir e executado.
        if use_opencv_image_output:
            if save_output:
                cv.imshow(filename_output, image_data)

            else:
                cv.imshow(image, image_data)

            cv.waitKey(100)

#Fecha a janela usada para mostrar a imagem.
if use_opencv_image_output:
    cv.destroyAllWindows()
```

Fonte: O autor.

Figura 29: *Código Python usado para obter os parâmetros de calibração da câmera - Varredura das imagens de calibração, parte 4.*

```
# Depois da varredura das imagens, os parametros de calibracao
# sao avaliados a partir dos dados das imagens.
ret, camera_matrix, distortion_coeffs, rotation_vectors,
    translation_vectors = cv.calibrateCamera(obj_points, img_points,
    image_format, None, None)

# O metodo "getOptimalNewCameraMatrix" e usado para melhorar o resultado
# da etapa anterior.
new_camera_matrix, roi = cv.getOptimalNewCameraMatrix(camera_matrix,
    distortion_coeffs, image_format, 1, image_format)


# Caso tenha sido escolhida anteriormente a opcao
# de salvar as imagens de saida, esse trecho e executado.
if save_output:
    if not os.path.exists((calibrated_images_folderpath+"\\undistort")):
        os.makedirs((calibrated_images_folderpath+"\\undistort"))

#Aplicacao da remocao de distorcao nas imagens do ensaio.
for image in self.calib_image_names_list:
    image_data = cv.imread(image)
    image_name = image.replace(self.folder_path, "")

    undistort_image = cv.undistort(image_data, camera_matrix,
    distortion_coeffs, None, new_camera_matrix)

    x, y, w, h = roi
    undistort_image = undistort_image[y:y+h, x:x+w]


    # Caso tenha sido escolhida anteriormente a opcao
    # de salvar as imagens de saida, esse trecho e executado.
    if save_output:
        cv.imwrite(calibrated_images_folderpath+"\\undistort\\"+image_name,
     undistort_image)

    # Caso se escolha mostrar a imagem de saida,
    # o codigo a seguir e executado.
    if use_opencv_image_output:
        cv.imshow(calibrated_images_folderpath+"\\undistort\\"+image_name,
    undistort_image)
        cv.waitKey(100)

#Fecha a janela usada para mostrar a imagem.
if use_opencv_image_output:
    cv.destroyAllWindows()
```

Fonte: O autor.

Figura 30: *Setup utilizado para gerar os volumes descritivos.*

```
from FeatureDescriptor import *
from VolumeDescriptor import *

new_camera_matrix = np.array([[615.22167969, 0.0, 327.87046662],
                              [0.0, 615.65362549, 234.79087398],
                              [0.0, 0.0, 1.0]], dtype=np.float32)
dist_coeffs = np.array([(0.01760001, -0.13768357, -0.00235607, 0.00165986,
    0.7880542)], dtype=np.float32)

camera_info = {"intrinsic_matrix": new_camera_matrix,
               "distortion_coeffs": dist_coeffs}

color_points = {"origin": np.array([0,155,155], dtype=np.uint8),
                "x_axis": np.array([100,155,155], dtype=np.uint8),
                "y_axis": np.array([155,100,155], dtype=np.uint8),
                "xy_axis": np.array([120,200,120], dtype=np.uint8),
                "border": np.array([25,200,200], dtype=np.uint8),
                "border_extended": np.array([75,200,200], dtype=np.uint8)}


axes_colors = {"x_axis": np.array([0,0,255], dtype=np.uint8),
               "y_axis": np.array([0,255,0], dtype=np.uint8),
               "z_axis": np.array([255,0,0], dtype=np.uint8),
               "lines": np.array([255,255,255], dtype=np.uint8)}

color_points={"points": color_points,
              "axes": axes_colors}

pattern_info={"side_length": 22.5}
```

Fonte: O autor.

Figura 31: *Código Python para gerar os volumes representativos das imagens do ensaio.*

```
#Imports necessarios.
from ast import pattern
from email.mime import image
from this import d
import cv2 as cv
import numpy as np
import os
from scipy.spatial import ConvexHull

def GetDataFromImages(folder_path, output_path, color_points, camera_info,
    pattern_info):

    # Obter pontos das imagens.
    point_data = GeneratePointDataOutput(folder_path, output_path,
    color_points, camera_info, pattern_info)

    #L ista usada para os volumes descritivos.
    vol_descriptors=[]

    # Criacao dos volumes descritos para cada imagem.
    print("Creating␣the␣volume␣descriptor␣for␣each␣image.")
    for image_data in point_data.keys():

        #Pontos da imagem.
        current_image_data = point_data[image_data]
        #Pontos da borda da envoltoria convexa.
        border_points = current_image_data["border_points"]
        #Informacao sobre o sistema de referencia da camera.
        frame_info = current_image_data["frame_info"]
        #Determinacao da posicao da camera.
        camera_position = -np.matmul(frame_info["rvec_matrix"].transpose(),
     frame_info["tvec"].reshape(3,1))

        #Cria o diretorio de saida para os arquivos .ply.
        if not os.path.exists(output_path+"\\ply_files"):
            os.makedirs(output_path+"\\ply_files")

        #Cria o descritor da imagem e escreve o arquivo .ply de saida.
        vol_desc = VolumeDescriptor(image_data, camera_position,
    border_points)
        vol_desc.CreateVolumeMeshWriteFile(output_path+"\\ply_files",
    image_data)

        vol_descriptors.append(vol_desc)
```

Fonte: O autor.

Figura 32: *Código Python o método GeneratePointDataOutput - Parte 1.*

```
def GeneratePointDataOutput(image_folder="", output_folder="",
    color_patterns=None, camera_info=None, pattern_info=None, save_output=
    True):

    return_value = None

    # Verifica se o diretorio indicado existe.
    if os.path.exists(image_folder):

        # Verificacao dos inputs passados.
        if isinstance(color_patterns, dict):
            if ("points" in color_patterns.keys()) and ("axes" in
    color_patterns.keys()):
                save_data_to_files = False

                # Verifica se os arquivos de saida devem ser criados
                # e se o diretorio de saida ja existe ou deve ser criado.
                if output_folder != "" and save_output:
                    if os.path.exists(output_folder):
                        save_data_to_files = True
                    else:
                        os.makedirs(output_folder)
                        save_data_to_files = True

                # Dicionario usado para armazenar os dados da imagem.
                image_data=dict()

                # Varredura das imagens no diretorio fornecido.
                print("GeneratePointDataOutput␣method:␣Parsing␣images␣for␣
    data␣points...")
                print("\tImage␣folder:␣%s" %(image_folder))
```

Fonte: O autor.

Figura 33: *Código Python o método GeneratePointDataOutput - Parte 2.*

```
            #Varedura das imagens.
            for item in os.listdir(image_folder):
                path_string = (image_folder+"\\"+item)

                # Verifica se o "item" nao e um diretorio e sim uma
imagem.
                if not os.path.isdir(path_string):
                    aux_image_data = dict()

                    print("\tGeneratePointDataOutput␣method:␣Parsing␣
image:␣%s" %(item))

                    # "GetPointsFromImage" e usado para obter os pontos
                    # de interesse na imagem.
                    output_points = GetPointsFromImage(image_folder,
item, color_patterns["points"], (output_folder+"\\points"),
save_data_to_files)

                    # Estruturacao dos dados obtidos para saida.
                    print("\tGeneratePointDataOutput␣method:␣Getting␣
axes␣info,␣rotation␣and␣translation␣vector␣for␣the␣camera␣frame␣for␣
image:␣%s" %(item))
                    axes_points = np.array([output_points["origin"],
                        output_points["x_axis"], output_points["y_axis"
],
                        output_points["xy_axis"]])

                    frame_info = GetFrameFromPatternPoints(axes_points,
                        camera_info["intrinsic_matrix"],
                        camera_info["distortion_coeffs"],
                        pattern_info["side_length"])

                    border_points = GetPointsInWorld(output_points["
border"],
                        camera_info["intrinsic_matrix"], frame_info["
rvec"],
                        frame_info["tvec"])

                    border_points_extended = GetPointsInWorld(
                        output_points["border_extended"],
                        camera_info["intrinsic_matrix"],
                        frame_info["rvec"], frame_info["tvec"])

                    aux_image_data["img_points"]=output_points
                    aux_image_data["frame_info"]=frame_info
                    aux_image_data["border_points"]=border_points
                    aux_image_data["border_points_extended"]=
border_points_extended

                    image_data[item]=aux_image_data
```

Fonte: O autor.

Figura 34: *Código Python o método GetPointsFromImage - Parte 1.*

```
def GetPointsFromImage(folder_path, image_name, points_color, output_folder
    ="", save_output=True):

    # Listas utilizadas.
    return_value = []
    origin = []
    x_axis = []
    y_axis = []
    xy_axis = []
    border_points = []
    border_points_extended = []

    print("GetPointsFromImage␣method:")
    image_path = folder_path + "\\" + image_name

    # Verifica se o caminho fornecido nao e um diretorio.
    if not os.path.isdir(image_path):
        print("\tImage␣path:␣%s" %(image_path))

        image_data = cv.imread(image_path)

        # Procura pelos pixels de referencia.
        for i in range(image_data.shape[0]):
            for j in range(image_data.shape[1]):
                if (image_data[i,j] == points_color["origin"]).all():
                    origin = np.array([j,i], dtype=np.float32)
                # float32 necessario devido ao usdo do metodo cv.solvePnP

                if (image_data[i,j] == points_color["x_axis"]).all():
                    x_axis = np.array([j,i], dtype=np.float32)
                # float32 necessario devido ao usdo do metodo cv.solvePnP

                if (image_data[i,j] == points_color["y_axis"]).all():
                    y_axis = np.array([j,i], dtype=np.float32)
                # float32 necessario devido ao usdo do metodo cv.solvePnP

                if (image_data[i,j] == points_color["xy_axis"]).all():
                    xy_axis = np.array([j,i], dtype=np.float32)
                # float32 necessario devido ao usdo do metodo cv.solvePnP
```

Fonte: O autor.

Figura 35: *Código Python o método GetPointsFromImage - Parte 2.*

```
            # Essa secao do codigo e usada para verificar a borda
            # de acordo com os pontos detectados pelo metodo de Harris
            # aplicado e com os pontos marcados manualmente.
            if (image_data[i,j] == points_color["border"]).all():
                border_points.append(np.array([j,i], dtype=np.float32))
            # float32 necessario devido ao usdo do metodo cv.solvePnP
                border_points_extended.append(np.array([j,i], dtype=np.
float32))
            # float32 necessario devido ao usdo do metodo cv.solvePnP
            if(image_data[i,j] == points_color["border_extended"]).all
():
                border_points_extended.append(np.array([j,i], dtype=np.
float32))
            # float32 necessario devido ao usdo do metodo cv.solvePnP

    # Cria o dicionario utilizado como saida.
    return_value={"origin": origin,
                  "x_axis": x_axis,
                  "y_axis": y_axis,
                  "xy_axis": xy_axis,
                  "border": border_points,
                  "border_extended": border_points_extended}

    # Salva os dados para arquivo.
    if output_folder !="" and save_output:
        if not os.path.exists(output_folder):
            os.makedirs(output_folder)

        output_file_name= output_folder+"\\"+image_name.replace(".png",
"")+"_points.txt"
        with open(output_file_name, "w") as f:
            f.write("name:%s\n" %(image_name))
            f.write("origin:[%d,%d]\n" %(origin[0], origin[1]))
            f.write("x_axis:[%d,%d]\n" %(x_axis[0], x_axis[1]))
            f.write("y_axis:[%d,%d]\n" %(y_axis[0], x_axis[1]))
            f.write("xy_axis:[%d,%d]\n" %(xy_axis[0], xy_axis[1]))
            f.write("borders:%d\n" %(len(border_points)))
            for entry in border_points:
                f.write("[%d,%d]\n" %(entry[0],entry[1]))

return return_value
```

Fonte: O autor.

Figura 36: *Código Python o método GetFrameFromPatternPoints.*

```
def GetFrameFromPatternPoints(axes_points, camera_matrix,
    distortion_coeffs, side_length=22.5):

    # Inicializacao de valores.
    return_value = []

    # Pontos para definicao do sistema
    # de referencia na imagem.
    origin_img = (0.0, 0.0, 0.0)
    x_axis = (side_length, 0.0, 0.0)
    y_axis = (0.0, side_length, 0.0)
    xy_axis = (side_length, side_length, 0.0)

    obj_points=np.array([origin_img, x_axis, y_axis, xy_axis], dtype=np.
    float32)

    # Obtencao do frame de referencia da imagem usando "solvePnP".
    retval, rvec, tvec = cv.solvePnP(obj_points, axes_points, camera_matrix
    ,
                                     distortion_coeffs, flags=0)
    # Rotacao "rvec" e um vetor, metodo "Rodrigues" recebe
    # esse vetor e gera a matriz de rotacao equivalente.
    rvec_matrix = cv.Rodrigues(rvec)[0]

    print("GetFrameFromPatternPoints␣method:")
    print("\trvec:␣%r" %(rvec))
    print("\trvec_matrix:␣%r" %(rvec_matrix))
    print("\ttvec:␣%r" %(tvec))

    return_value = {"rvec": rvec,
                    "rvec_matrix": rvec_matrix,
                    "tvec": tvec}

    return return_value
```

Fonte: O autor.

Figura 37: *Código Python o método GetPointsInWorld - Parte 1.*

```
def GetPointsInWorld(image_points, camera_matrix, rvec, tvec):

    # Lista de valores de saida.
    output_points = []

    print("GetPointsInWorld␣method:␣Initializing␣values...")

    # Obtem a matriz de rotacao inversa.
    rotation_matrix = cv.Rodrigues(rvec)[0].transpose()

    print("\trvec:\n\t%r\n\n\trotation_matrix:\n\t%r\n\n\ttvec:\n\n\t%r"
            %(rvec, rotation_matrix,-np.matmul(rotation_matrix,tvec)))

    transform_matrix=np.array([[camera_matrix[0,0], 0.0],[0.0,
    camera_matrix[0,0]]])
    transform_matrix=np.linalg.inv(transform_matrix)

    print("\n\tcamera_matrix:\n\t%r"%(camera_matrix))

    homogeneous_transform=np.concatenate((rotation_matrix,
                                          -np.matmul(rotation_matrix,tvec)),
                                          axis=1)

    aux_line = np.array([0.0, 0.0, 0.0, 1.0]).reshape(1,4)
    homogeneous_transform=np.concatenate((homogeneous_transform, aux_line),
     axis=0)
    print("\n\tHomogeneous␣transform:")
    print(homogeneous_transform)

    inv_homogeneous_transform=np.linalg.inv(homogeneous_transform)
    print("\n\tInverse␣homogeneous␣transform:")
    print(inv_homogeneous_transform)
    fx=camera_matrix[0][0]
```

Fonte: O autor.

Figura 38: *Código Python o método GetPointsInWorld - Parte 2.*

```
    # Varredura dos pontos da imagem.
    print("GetPointsInWorld␣method:␣Parsing␣the␣image␣points...")
    for point in image_points:
        # Processamento do ponto na imagem para
        # projeÃ§Ã£o no plano do tabuleiro xadrez.
        point_used = np.array([point[0], point[1], 1.0], dtype=np.float32).
    reshape(3,1)

        plane_point = np.matmul(np.linalg.inv(camera_matrix), point_used)*
    fx

        plane_point_real = np.concatenate((plane_point, [[1.0]]),axis=0).
    reshape(4,1)
        plane_point_real = np.matmul(homogeneous_transform,plane_point_real
    )
        plane_point_real = plane_point_real[0:3,0].reshape(3,1)

        t_vec = homogeneous_transform[0:3,3].reshape(3,1)

        t_param = t_vec[2]/(t_vec[2]-plane_point_real[2])

        #Definicao do ponto no plano do tabuleiro.
        world_point=t_vec-t_param[0]*(t_vec - plane_point_real)

        world_point = np.array([np.float32(world_point[0]),
                                np.float32(world_point[1]),
                                0.0], dtype=np.float32)

        output_points.append(world_point)

    return output_points
```

Fonte: O autor.

Figura 39: *Código Python o método GeneratePointDataOutput - Parte 3.*

```
    # Trecho executado caso as saidas do processo devam ser salvas.
                if save_data_to_files:
                    if not os.path.exists(output_folder+"\\image_outputs"):
                        os.makedirs(output_folder+"\\image_outputs")

                    #Parametros utilizados para gerar imagens.
                    line_length = pattern_info["side_length"]
                    axes_points = np.array([[line_length, 0.0, 0.0],
                        [0.0, line_length, 0.0],[0.0, 0.0, line_length]],
                        dtype=np.float32).reshape(-1,3)

                    img_points, jac = cv.projectPoints(axes_points,
                        frame_info["rvec"], frame_info["tvec"],
                        camera_info["intrinsic_matrix"],
                        camera_info["distortion_coeffs"])

                    print("\tGeneratePointDataOutput␣method:␣Plotting␣axes␣
    and␣the␣points'␣convex␣hull...")

                    axes_points = {"origin": np.array(
                                                [output_points["origin"][0],
                                                 output_points["origin"][1]],
                                                 dtype=np.uint16),

                                   "x_axis": np.array(
                                       [img_points[0][0][0],
                                       img_points[0][0][1]],
                                       dtype=np.uint16),

                                   "y_axis": np.array(
                                       [img_points[1][0][0],
                                       img_points[1][0][1]],
                                       dtype=np.uint16),

                                   "z_axis": np.array(
                                       [img_points[2][0][0],
                                       img_points[2][0][1]],
                                       dtype=np.uint16)}

    # Ordenamento dos pontos da borda obtidos pelo metodo de Harris.
                    border_points=[]
                    for border_point in output_points["border"]:
                        border_points.append(np.array([border_point[0],
                            border_point[1]], dtype=np.uint16))
```

Fonte: O autor.

Figura 40: *Código Python o método GeneratePointDataOutput - Parte 4.*

```
            # "PlotAxesInImage" e usado para inserir os eixos coloridos
 na imagem.
            PlotAxesInImage(path_string, axes_points,
                            color_patterns["axes"],
                            (output_folder+"\\image_outputs\\detected")
,
                            item, border_points)
            # Ordenamento dos pontos de borda selecionados manualmente.
                border_points_extended=[]
                for border_point in output_points["border_extended"]:
                    border_points_extended.append(np.array([
border_point[0],
                                        border_point[1]], dtype=np.uint16))

            # "PlotAxesInImage" e usado para inserir os eixos
            # coloridos na imagem.
            PlotAxesInImage(path_string, axes_points,
                            color_patterns["axes"],
                            (output_folder+"\\image_outputs\\manual"),
                            item, border_points_extended)
```

Fonte: O autor.

Figura 41: *Código Python o método PlotAxesInImage - Parte 1.*

```
def PlotAxesInImage(image_path, axes_inputs, color_inputs, output_folder=""
    , output_image_name="", border_points=None):

    # Verifica se o diretorio fornecido existe, e caso
    # nao exista, ele e criado.
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)

    # Carrega imagem e exibe os eixos dados como entrada.
    image_data = cv.imread(image_path)
    print("\tInputs:")
    print("\tOrigin:␣%r" %(axes_inputs["origin"]))
    print("\tX␣axis:␣%r" %(axes_inputs["x_axis"]))
    print("\tY␣axis:␣%r" %(axes_inputs["y_axis"]))
    print("\tZ␣axis:␣%r" %(axes_inputs["z_axis"]))

    # Cores usadas para os eixos.
    print("\tColors:")
    color_values = [(int(color_inputs["x_axis"][0]),
                     int(color_inputs["x_axis"][1]),
                     int(color_inputs["x_axis"][2])),

                    (int(color_inputs["y_axis"][0]),
                     int(color_inputs["y_axis"][1]),
                     int(color_inputs["y_axis"][2])),

                    (int(color_inputs["z_axis"][0]),
                     int(color_inputs["z_axis"][1]),
                     int(color_inputs["z_axis"][2])),

                    (int(color_inputs["lines"][0]),
                     int(color_inputs["lines"][1]),
                     int(color_inputs["lines"][2]))]
```

Fonte: O autor.

Figura 42: *Código Python o método PlotAxesInImage - Parte 2.*

```
print("\tX color: (%d, %d, %d)" %(color_values[0][0], color_values
[0][1],
                                  color_values[0][2]))
print("\tY color: (%d, %d, %d)" %(color_values[1][0], color_values
[1][1],
                                  color_values[1][2]))
print("\tZ color: (%d, %d, %d)" %(color_values[2][0], color_values
[2][1],
                                  color_values[2][2]))
print("\tLines color: (%d, %d, %d)" %(color_values[3][0], color_values
[3][1],
                                      color_values[3][2]))

# Plot das linhas representativas dos eixos.
cv.line(image_data, axes_inputs["origin"], axes_inputs["x_axis"],
color_values[0], 5)
cv.line(image_data, axes_inputs["origin"], axes_inputs["y_axis"],
color_values[1], 5)
cv.line(image_data, axes_inputs["origin"], axes_inputs["z_axis"],
color_values[2], 5)

# Verifica se os pontos de borda fornecidos sao validos.
if border_points is not None:
    # Selecao dos pontos que formam a envoltoria convexa.
    hull = ConvexHull(border_points)

    # Plot das linhas que formam a envoltoria convexa.
    for simplex in hull.simplices:
        cv.line(image_data, border_points[simplex[0]],
                border_points[simplex[1]], color_values[3], 3)

#Salva ou exibe a imagem com as linhas adicionadas.
if (output_folder!="") and (output_image_name!=""):
    cv.imwrite(output_folder+"\\"+output_image_name, image_data)

else:
    cv.imshow(image_path, image_data)
```

Fonte: O autor.

Figura 43: *Código Python o método GeneratePointDataOutput - Parte 5.*

```
            # Impressao na tela dos resultados.
            print("GeneratePointDataOutput␣method:␣Image␣points␣parsing␣
    completed.\n")
            print("GeneratePointDataOutput␣method:␣Printing␣outputs...")
            for image in image_data.keys():
                print("\tImage:␣%s" %(image))
                for entry in image_data[image].keys():
                    print("\t\tData:␣%s\n\t\t%r" %(entry, image_data[image
    ][entry]))

            print("GeneratePointDataOutput␣method:␣Done!")

            # Valor de retorno definido como dicionario com todos os
            # dados relevantes.
            return_value = image_data

        else:
            #Mensagem de erro caso alguma das entradas nao seja valida.
            print("GeneratePointDataOutput␣method:␣\"color_patterns\"␣
    should␣be␣a␣dictionary␣with␣2␣keys,␣\"points\"␣and␣\"axes\".")

    else:
        #Mensagem de erro caso alguma das entradas nao seja valida.
        print("GeneratePointDataOutput␣method:␣\"image_folder\"␣path␣not␣
    valid.")

return return_value
```

Fonte: O autor.

Figura 44: *Código Python para o construtor da classe VolumeDescriptor.*

```
from itertools import combinations
import pymesh as msh
from stl import mesh
import numpy as np
import pyvista as pv

from scipy.spatial import ConvexHull


class VolumeDescriptor(object):

    def __init__(self, volume_name = "vd", camera_position = [],
    points_in_plane = []):

        # Nome usado para identificar o volume.
        self.volume_name = volume_name

        type_verification = type(np.array([0.0, 0.0, 0.0]))

        # Determinacao da posicao da camera
        if isinstance(camera_position, type(list())) and len(
    camera_position) == 3:
            self.camera_position = np.array([camera_position[0],
    camera_position[1], camera_position[2]], dtype=np.float32).reshape(-1)
            print("VolumeDescriptor␣%s:␣Camera␣position␣set␣successfully!␣
    (%d,␣%d,␣%d)" %(self.volume_name, self.camera_position[0], self.
    camera_position[1], self.camera_position[2]))
        elif isinstance(camera_position, type_verification):
            self.camera_position = np.array([camera_position[0],
    camera_position[1], camera_position[2]], dtype=np.float32).reshape(-1)
            print("VolumeDescriptor␣%s:␣Camera␣position␣set␣successfully!␣
    (%d,␣%d,␣%d)" %(self.volume_name, self.camera_position[0], self.
    camera_position[1], self.camera_position[2]))
        else:
            self.camera_position = None
            print("VolumeDescriptor␣%s:␣Camera␣position␣was␣not␣set␣
    correctly!" %(self.volume_name))

        #Parsing dos pontos projetados no plano (volume descritivo).
        for entry in points_in_plane:
            valid_entries = True

            if not isinstance(entry, type_verification):
                valid_entries = False
                print("VolumeDescriptor␣%s:␣Error␣with␣entry␣%r" %(self.
    volume_name, entry))
            if valid_entries:
                self.points_in_plane = points_in_plane
                self.n_points = len(points_in_plane)

            else:
                self.points_in_plane = None
                self.n_points = None
```

Figura 45: *Código Python para o método CreateVolumeMeshWriteFile - Parte 1.*

```
def CreateVolumeMeshWriteFile(self, folder_path = "", filename = ""):

    #Inicializacao das strings com os diretorios usados.
    if folder_path == "":
        save_path = ""
    else:
        save_path = folder_path + "\\"

    if filename == "":
        save_path = save_path + self.volume_name
    else:
        save_path = save_path + filename

    print("CreateVolumeMeshWriteFile␣method:␣save␣file␣path:␣%s" %(
save_path))

    # Lista dos vertices encontrados no plano de projecao z=0.
    vertices_plane = []
    for point in self.points_in_plane:
        vertices_plane.append([point[0], point[1]])

    # Determinacao da envoltoria convexa.
    hull = ConvexHull(np.array(vertices_plane))

    # Centroide da area da envoltoria convexa.
    # Usada para gerar os triangulos.
    hull_points = []
    for simplex in hull.simplices:
        hull_points.append(vertices_plane[simplex[0]])
        hull_points.append(vertices_plane[simplex[1]])

    hull_points = np.array(hull_points)

    x_pos = sum(hull_points[:,0])/hull_points.shape[0]
    y_pos = sum(hull_points[:,1])/hull_points.shape[0]

    centroid = np.array([x_pos, y_pos, 0.0])
```

Fonte: O autor.

Figura 46: *Código Python para o método CreateVolumeMeshWriteFile - Parte 2.*

```
        # Os vertices sao ordenados, e suas normais
        # sao colocados apontando para o lado externo
        # do volume gerado.
        vertices = [self.camera_position.reshape(-1)] + self.
    points_in_plane + [centroid]
        vertices = np.array(vertices)

        n_vertices = vertices.shape[0] - 1

        center_of_volume = ((self.camera_position - centroid)/2)
        center_of_volume = center_of_volume[0]

        centroid_normal = (centroid - center_of_volume)
        centroid_normal = centroid_normal/np.linalg.norm(centroid_normal)

        camera_normal = (self.camera_position.reshape(-1) -
    center_of_volume)
        camera_normal = camera_normal/np.linalg.norm(camera_normal)

        # Listas para os vertices que formam as faces e normais
        # das faces.
        faces_list = []
        normals_list = []

        for vertex in vertices:
            vertex_normal = (vertex - center_of_volume)
            vertex_normal = vertex_normal/np.linalg.norm(vertex_normal)
            normals_list.append(vertex_normal)
```

Fonte: O autor.

Figura 47: *Código Python para o método CreateVolumeMeshWriteFile - Parte 3.*

```
    # Formam-se as as arestas dos triangulos e suas
    # normais sao calculadas.
    for simplex in hull.simplices:
        vertex1_normal=normals_list[simplex[0]+1]
        vertex2_normal=normals_list[simplex[1]+1]

        edge1_camera_normal=(vertices[simplex[0]+1] - self.
camera_position)
        edge1_camera_normal=edge1_camera_normal/np.linalg.norm(
edge1_camera_normal)

        edge2_camera_normal=(vertices[simplex[1]+1] - vertices[simplex
[0]+1])
        edge2_camera_normal=edge2_camera_normal/np.linalg.norm(
edge2_camera_normal)

        edge3_camera_normal=(self.camera_position - vertices[simplex
[1]+1])
        edge3_camera_normal=edge3_camera_normal/np.linalg.norm(
edge3_camera_normal)

        face_condition_1 = np.dot(np.cross(edge1_camera_normal,
edge2_camera_normal), vertex1_normal)
        face_condition_2 = np.dot(np.cross(edge2_camera_normal,
edge3_camera_normal), vertex2_normal)

        edge1_centroid_normal=(vertices[simplex[0]+1] - centroid)
        edge1_centroid_normal=edge1_centroid_normal/np.linalg.norm(
edge1_centroid_normal)

        edge2_centroid_normal= (vertices[simplex[1]+1] - vertices[
simplex[0]+1])
        edge2_centroid_normal=edge2_centroid_normal/np.linalg.norm(
edge2_centroid_normal)

        edge3_centroid_normal= (centroid - vertices[simplex[1]+1])
        edge3_centroid_normal=edge3_centroid_normal/np.linalg.norm(
edge3_centroid_normal)

        base_condition_1 = np.dot(np.cross(edge1_camera_normal,
edge2_camera_normal), vertex1_normal)
        base_condition_2 = np.dot(np.cross(edge2_camera_normal,
edge3_camera_normal), vertex2_normal)

        # Faces sao criadas (ordenadas de forma que suas
        # normais apontem para fora do volume).
        if (face_condition_1 < 0.0) and (face_condition_2 < 0.0):
            faces_list.append([3, 0, (simplex[1]+1), (simplex[0]+1)])
        else:
            faces_list.append([3, 0, (simplex[0]+1), (simplex[1]+1)])

        if (base_condition_1 < 0.0) and (base_condition_2 < 0.0):
            faces_list.append([3, n_vertices, (simplex[1]+1), (simplex
```

Figura 48: *Código Python para o método CreateVolumeMeshWriteFile - Parte 4.*

```
    try:

        # PyVista e usado para criar o volume e entao escrever
manualmente
        # o arquivo .ply para o volume descritivo.
        faces_polydata_cells = faces_list
        faces_polydata_cells = np.hstack(faces_polydata_cells)
        surface = pv.PolyData(var_inp=vertices, faces=
faces_polydata_cells)

        with open(save_path+"_written.ply", "wb") as file:
            file.write(b"ply\n")
            file.write(b"format␣ascii␣1.0\n")
            file.write(b"comment␣Manually␣generated␣PLY␣File\n")
            file.write(b"obj_info␣vtkPolyData␣points␣and␣polygons:␣vtk4
.0\n")
            file.write(b"element␣vertex␣%d\n" %(len(vertices)))
            file.write(b"property␣float␣x\n")
            file.write(b"property␣float␣y\n")
            file.write(b"property␣float␣z\n")
            file.write(b"property␣float␣nx\n")
            file.write(b"property␣float␣ny\n")
            file.write(b"property␣float␣nz\n")
            file.write(b"element␣face␣%d\n" %(len(faces_list)))
            file.write(b"property␣list␣uchar␣int␣vertex_indices\n")
            file.write(b"end_header\n")

            for index, vertex in enumerate(vertices):
                file.write(b"%f␣%f␣%f␣%f␣%f␣%f\n" %(vertex[0], vertex
[1], vertex[2], normals_list[index][0], normals_list[index][1],
normals_list[index][2]))

            for face in faces_list:
                file.write(b"%d␣%d␣%d␣%d\n" %(face[0], face[1], face
[2], face[3]))

        # Salva o volume dentro da instancia da classe
        # para o uso posterior.
        self.surface_polydata=pv.read(save_path+"_written.ply")

    except Exception:
        print("Error␣in␣creating␣the␣volume!")
```

Fonte: O autor.

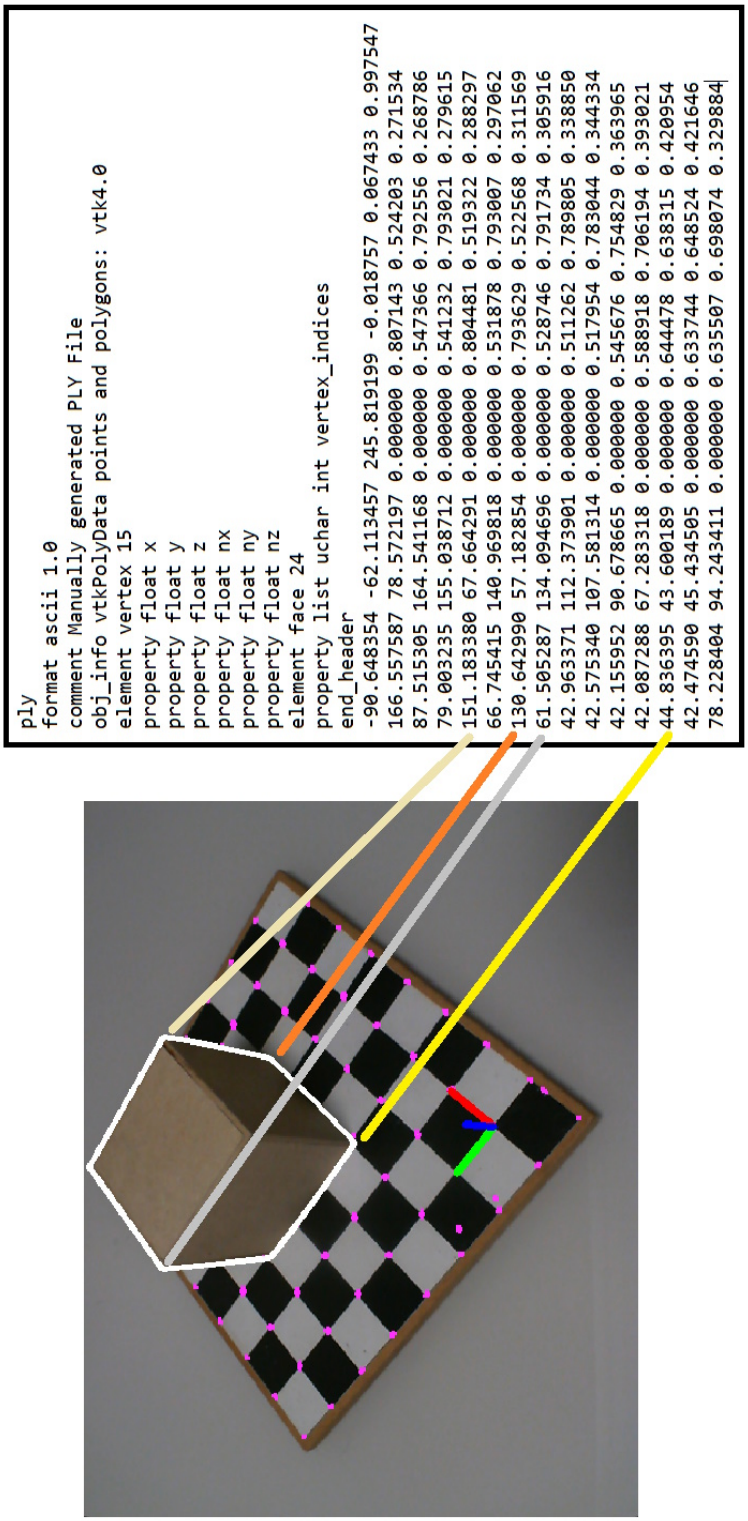

Figura 49: *Exemplo de arquivo .ply gerado a partir de uma imagem coma correspondência entre pontos no arquivo e na imagem representada.*

Fonte: O autor.

APÊNDICE B - CALIBRAÇÃO DA CÂMERA

A calibração da câmera foi realizada utilizando os *scripts* Python apresentados no Apêndice A. A câmera utilizada foi uma câmera Microsoft LifeCam Cinema (Figura 50). Essa câmera permite aquisição com resolução HD (proporção de 1280 por 720 *pixels*) e 30 quadros por segundo. Para os ensaios, a câmera foi configurada para captura de imagens com dimensões de 640 (largura) por 480 (altura) *pixels* e foco fixo. O foco fixo é necessário para que os mesmo parâmetros intrínsecos da câmera, assim como as mesmas correções de distorção, possam ser aplicadas a todas imagens dos ensaios subsequentes se a necessidade de recalcular os parâmetros (o que seria necessário caso o foco mudasse).

A calibração foi realizada utilizando um padrão xadrez com uma configuração de $8x8$ quadrados, cada um com arestas de $22,5$ milímetros. Para calibração foram utilizadas 20 imagens, sem um posicionamento preciso da câmera, rotacionando o tabuleiro ou mudando o ângulo entre o eixo da câmera e o plano da superfície onde o tabuleiro se encontra. As orientações do tabuleiro utilizadas podem ser vistas na Figura 51. Na orientação apresentada no canto inferior da Figura 51 pode ser verificado que os sistemas de referência apresentados, representando o sistema de referência da câmera S_c da câmera para cada cenário apresentado na Figura 51 possuem um de seus eixos apontando na direção positiva do eixo z do sistema de referência do mundo, S_w. Esse eixo, em cada ensaio, representa a direção do eixo z do sistema de referência da câmera S_c, onde o sentido negativo desse eixo aponta para o tabuleiro usado na calibração.

Pode-se observar na Figura 51 que a posição das câmeras entre ensaios encontra-se concentrada em uma região com valores para as coordenadas x e y negativos. Porém os ensaios foram realizados rotacionando o tabuleiro em relação à câmera, logo os resultados deveriam ser apresentados circulando a área representativa do tabuleiro. Isso não ocorre

Figura 50: *Camera Microsoft LifeCam Cinema utilizada nos ensaios.*

Fonte: (LIFECAM..., 2022)

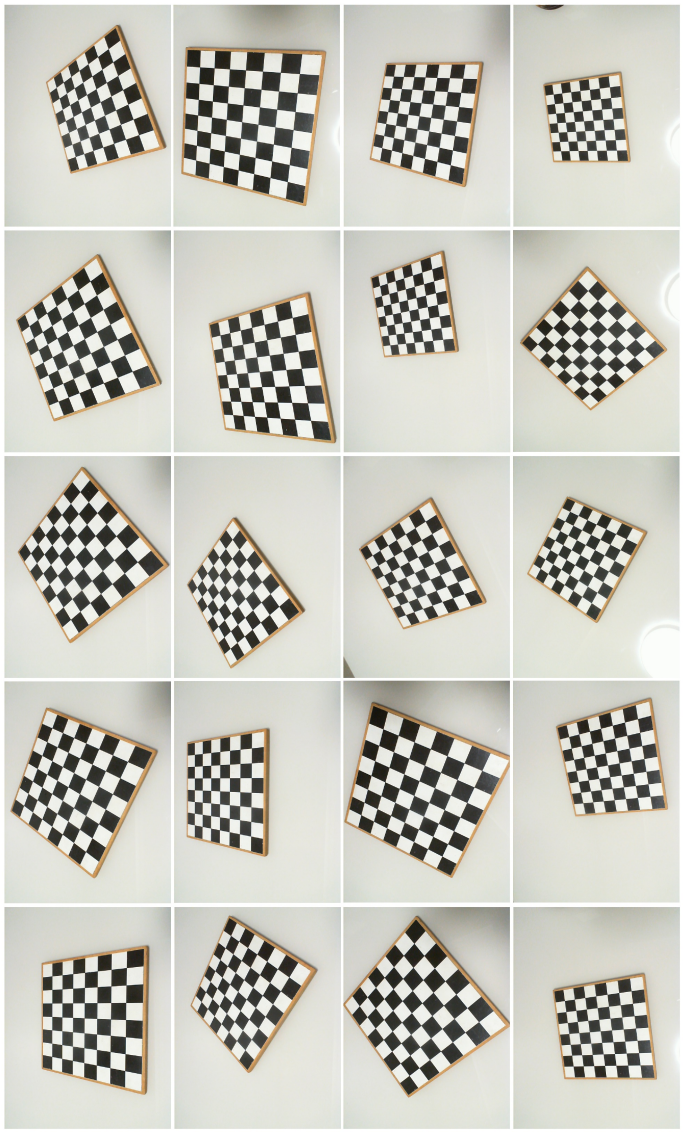

Figura 51: *Imagens do padrão xadrez utilizadas para a calibração da câmera.*

Fonte: O autor.

devido a simetria do tabuleiro, pois os métodos utilizados interpretam que os posicionamentos da câmera durante a calibração encontram-se na mesma região do espaço, alocando as posições no região com eixos x e y do sistema de referência do mundo S_w negativos e z positivo. Portanto a posição real de alguns desses resultados na verdade é localizada 180 graus em relação ao eixo z do sistema de referência do mundo S_w. Isso não ocorre durante os ensaios com os objetos de teste pois neles são determinadas as posições de pontos de referência. Tal agrupamento pode representar problemas em casos onde o tabuleiro não tenha quadrados com arestas com comprimento homogêneo.

Os resultados da calibração foram apresentados anteriormente na seção 3.3.

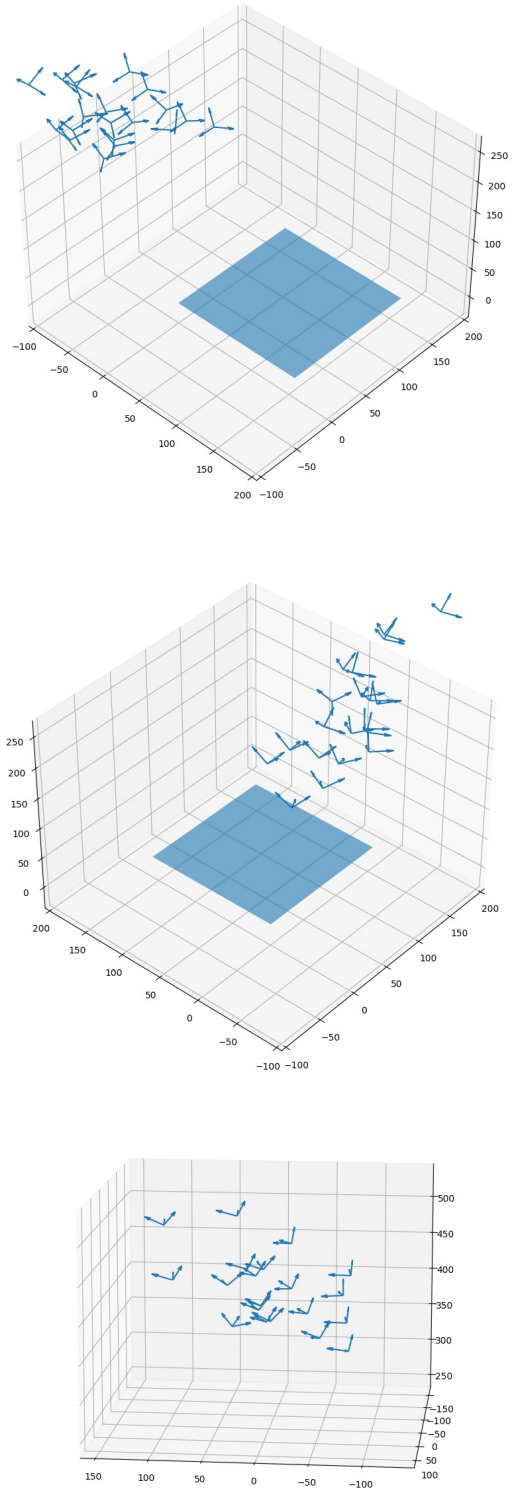

Figura 52: *Posições e orientações da câmera durante a captura de imagens usadas para calibração obtidas a partir do processo de calibração. Escala em milímetros. Três orientações diferentes são mostradas.*

Fonte: O autor.

APÊNDICE C - IMAGENS DOS ENSAIOS E VOLUMES GERADOS

Para os objetos de ensaios utilizados, mostrados na Figura 13 da seção 4, cujas dimensões são listadas nas Tabelas 2 e 3, também na seção 4, foram realizados ensaios com a captura de 20 imagens com diferentes posicionamentos e relações entre o a superfície com padrão xadrez e a câmera. Para cada imagem desses ensaios foi gerado um volume descritivo. Na Tabela 4 podem ser encontradas relação entre as figuras com os conjuntos de imagens obtidos para cada ensaio e os volumes gerados sobrepostos de acordo com os arquivos *.ply* gerados.

Objeto de ensaio	Imagens capturadas	Volumes gerados
1	Figura 53	Figura 54
2	Figura 55	Figura 56
3	Figura 57	Figura 58
4	Figura 59	Figura 60

Tabela 4: *Relação entre ensaios, imagens capturadas e volumes gerados.*

Fonte: O autor.

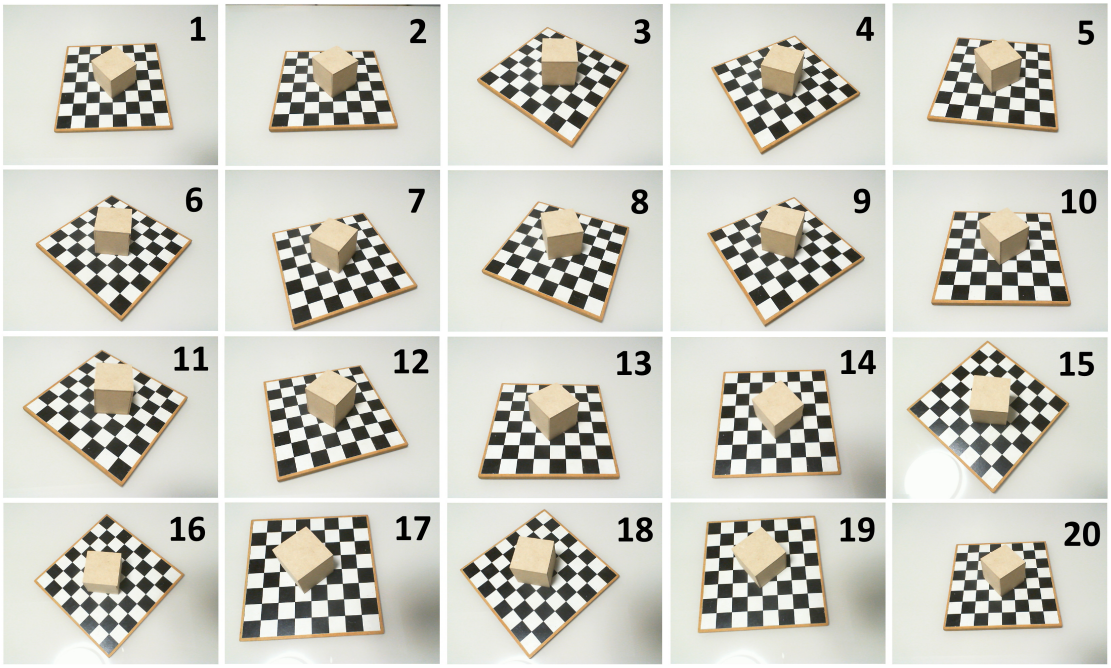

Figura 53: *Imagens capturadas para o objeto de ensaio 1.*

Fonte: O autor.

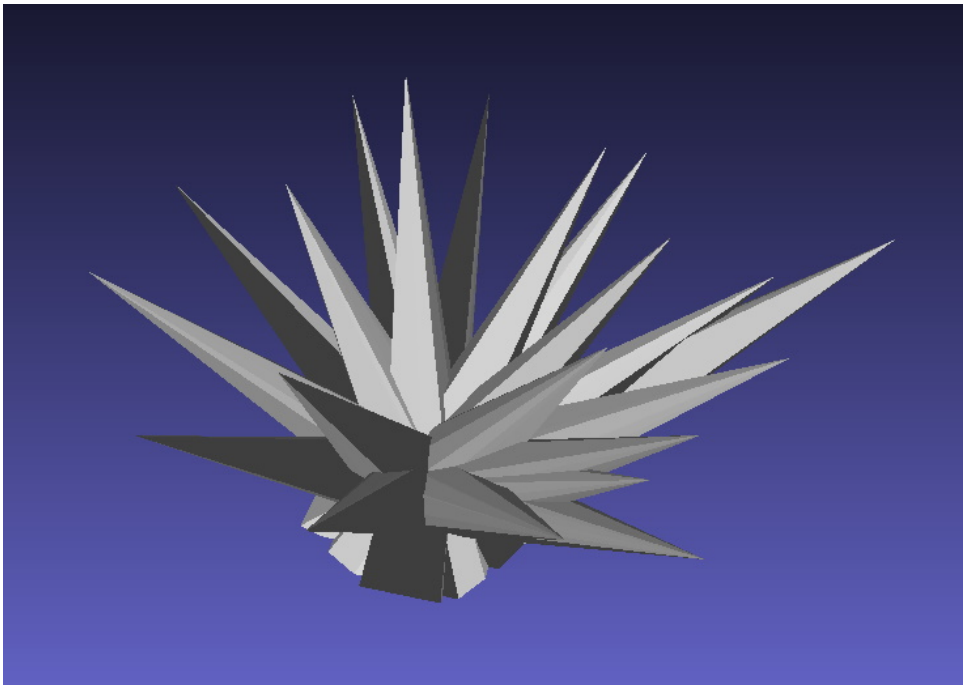

Figura 54: *Volumes gerados para as imagens do objeto de ensaio 1.*

Fonte: O autor.

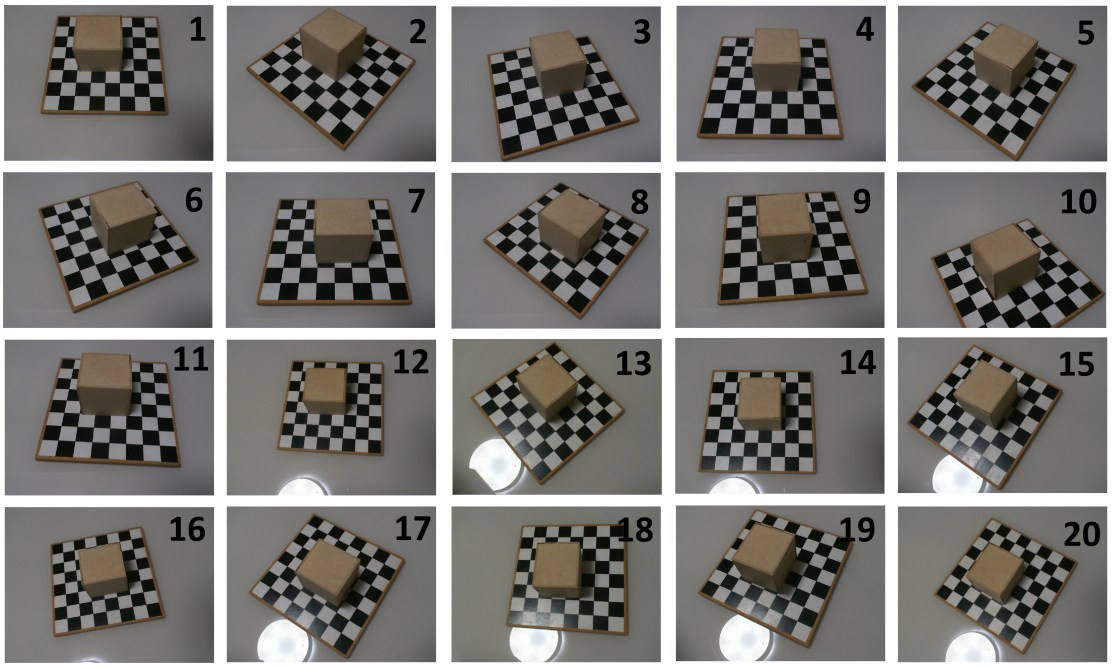

Figura 55: *Imagens capturadas para o objeto de ensaio 2.*

Fonte: O autor.

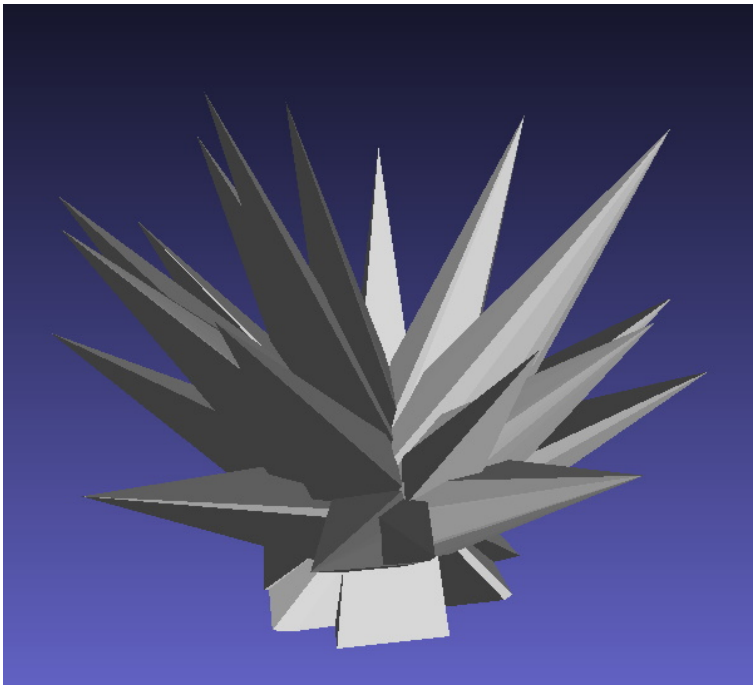

Figura 56: *Volumes gerados para as imagens do objeto de ensaio 2.*

Fonte: O autor.

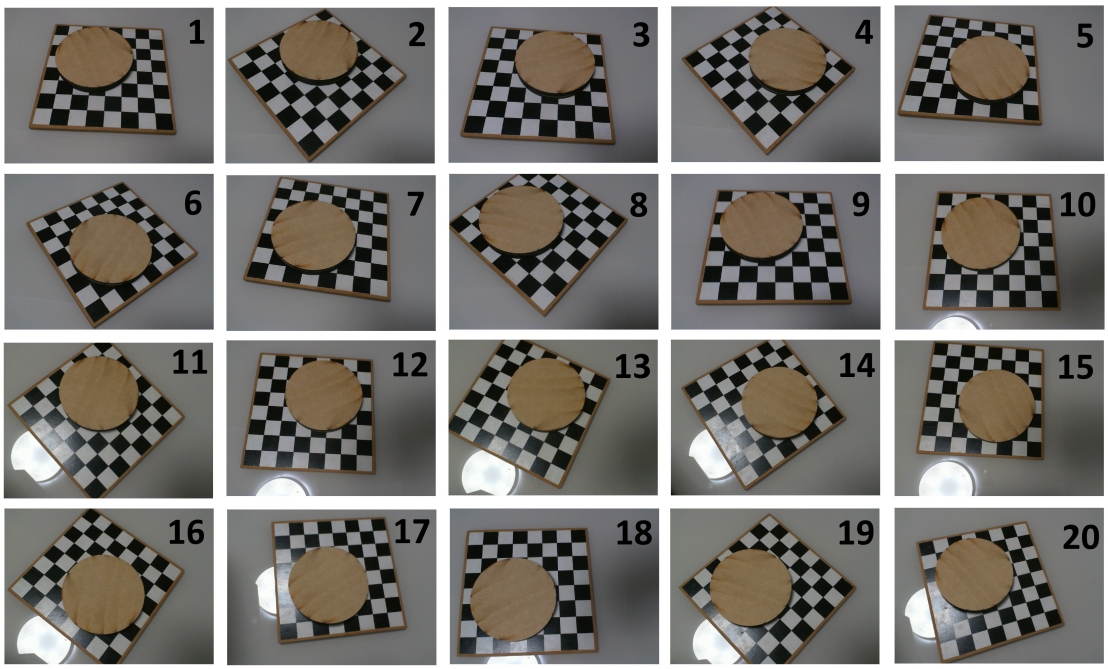

Figura 57: *Imagens capturadas para o objeto de ensaio 3.*

Fonte: O autor.

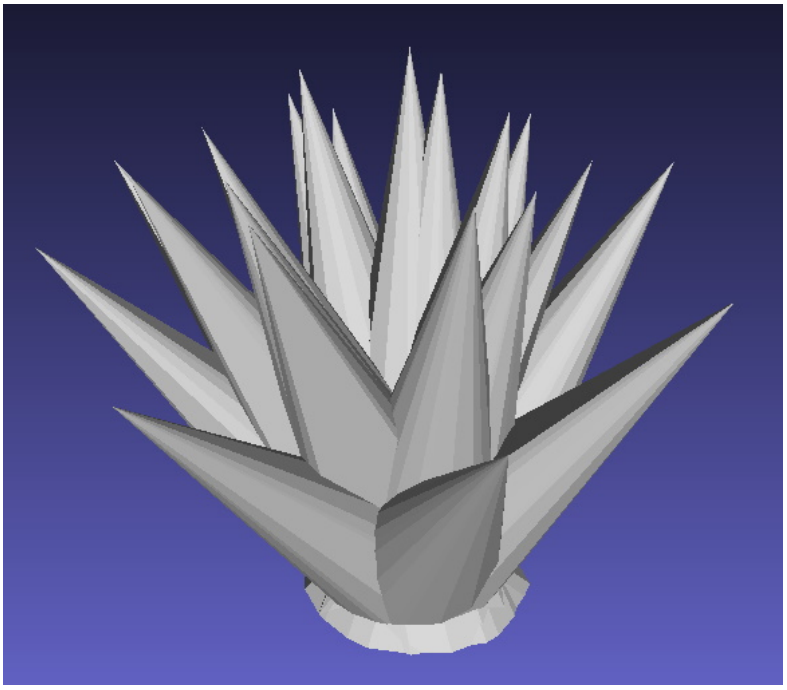

Figura 58: *Volumes gerados para as imagens do objeto de ensaio 3.*

Fonte: O autor.

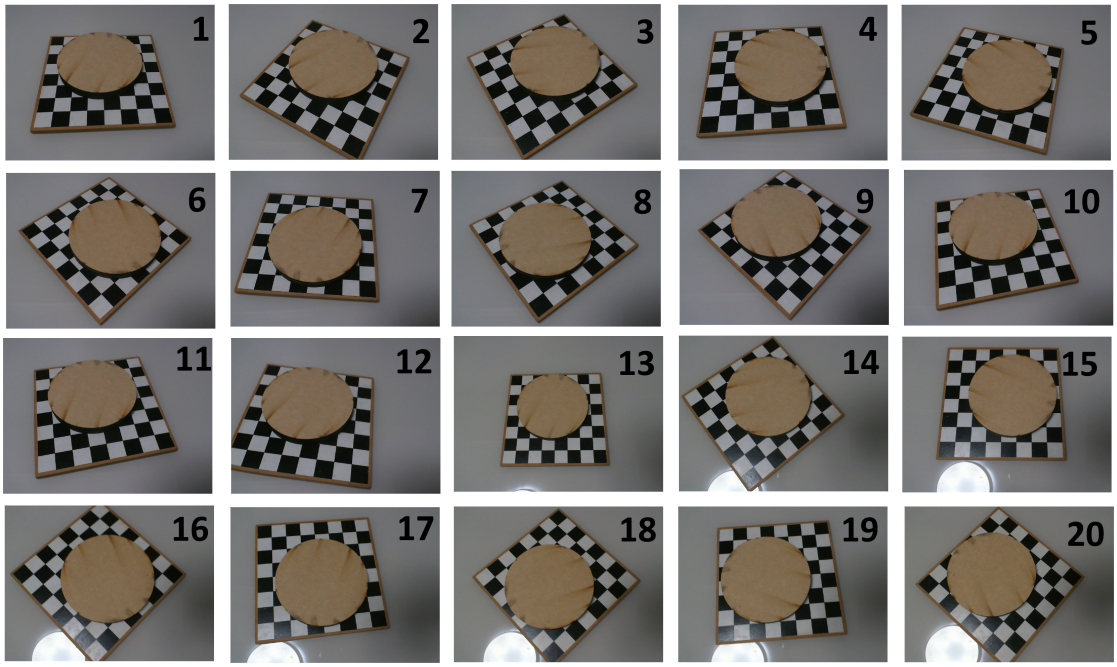

Figura 59: *Imagens capturadas para o objeto de ensaio 4.*

Fonte: O autor.

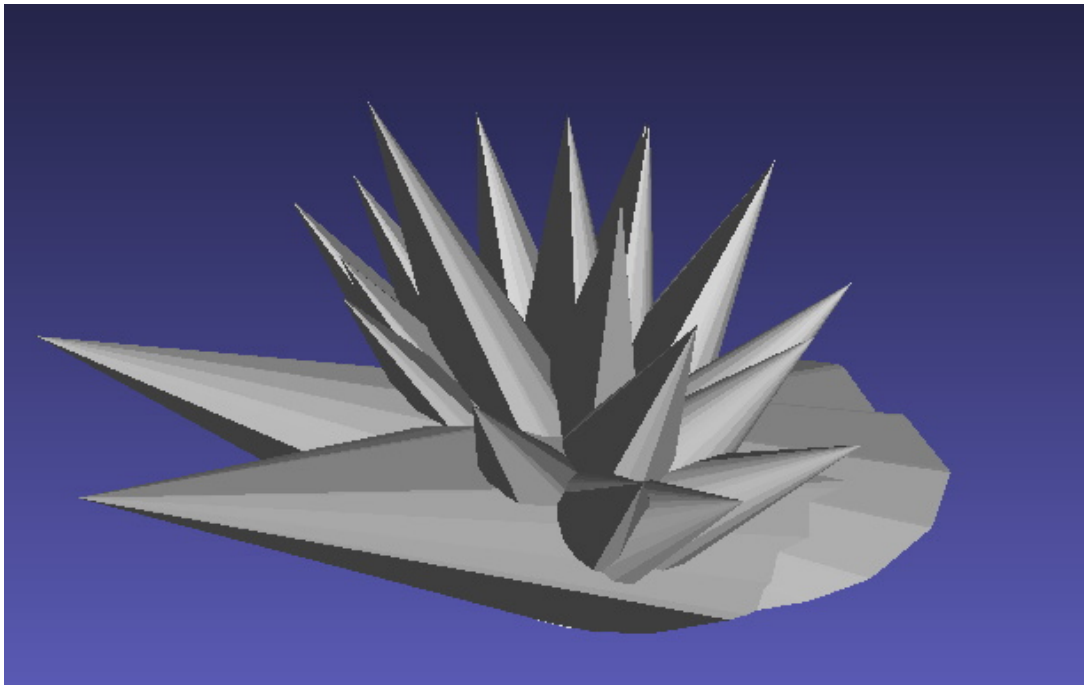

Figura 60: *Volumes gerados para as imagens do objeto de ensaio 4.*

Fonte: O autor.

REFERÊNCIAS

3D SYSTEMS, INC. *StereoLithography Interface Specification*. [S.l.], 1989.

AUTODESK, INC. *Autodesk Meshmixer*. [S.l.]. Disponível em: <https://www.meshmixer.com/>. Acesso em: 1 set. 2022.

CALONDER, M. et al. BRIEF: Binary Robust Independent Elementary Features. *11th European Conference on Computer Vision (ECCV)*, LNCS Springer, Creta, Grécia, 2010.

CIGNONI, P. et al. MeshLab: an Open-Source Mesh Processing Tool. In: SCARANO, V.; CHIARA, R. D.; ERRA, U. (Ed.). *Eurographics Italian Chapter Conference*. [S.l.]: The Eurographics Association, 2008. ISBN 978-3-905673-68-5. DOI: 10.2312/LocalChapterEvents/ItalChap/ItalianChapConf2008/129-136.

HARRIS, C.; STEPHENS, M. A Combined Corner and Edge Detector. *Alvey Vision Conference*, v. 15, 1988.

KITWARE INC. *VTK (Visualizatio Toolkit) - Documentation*. [S.l.]. Disponível em: <https://vtk.org/>.

LOWE, D. G. Distintive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, v. 60, p. 91–110, 2004. DOI: https://doi.org/10.1023/B:VISI.0000029664.99615.94.

LIFECAM Cinema - Microsoft. Microsoft Corporation. Disponível em: <https://www.microsoft.com/pt-br/accessories/products/webcams/lifecam-cinema?activetab=pivot:vis%C3%A3ogeraltab>. Acesso em: 13 out. 2022.

MUNDY, J. The relationship between photogrammetry and computer vision. Schenectady, EUA, 2005.

NUMPY. *NumPy documentation*. [S.l.]. Disponível em: <https://numpy.org/doc/stable/>. Acesso em: 4 ago. 2022.

OPENCV. *OpenCV documentation*. [S.l.]. Disponível em: <https://docs.opencv.org/4.x/index.html>. Acesso em: 4 ago. 2022.

PYVISTA DEVELOPERS. *PyVista - Documentation Reference*. [S.l.]. Disponível em: <https://docs.pyvista.org/>.

ROSTEN, E.; DRUMMOND, T. Machine learning for high speed corner detection. *9th European Conference on Computer Vision*, v. 1, p. 430–443, 2006.

RUBLEE, E. et al. ORB: An efficient alternative to SIFT or SURF. *International Conference on Computer Vision 2011*, p. 2564–2571, 2011.

SHI, J.; TOMASI, C. Good Features to Track. *9th IEEE Conference on Computer Vision and Pattern Recognition*, p. 593–600, 1994.

STACHNISS, C. Camera Extrinsics and Intrinsics. Bonn, Alemanha. Disponível em: <https://www.ipb.uni-bonn.de/html/teaching/msr2-2020/sse2-12-camera-params.pdf>. Acesso em: 3 ago. 2022.

TURK, G. *The PLY Polygon File Format*. [S.l.], 1994. Disponível em: <https://web.archive.org/web/20161204152348/http://www.dcs.ed.ac.uk/teaching/cs4/www/graphics/Web/ply.html>. Acesso em: 20 set. 2022.

WAVEFRONT, INC. *Alias/WaveFront Object (.obj) File Format*. [S.l.]. Disponível em: <https://people.cs.clemson.edu/~dhouse/courses/405/docs/brief-obj-file-format.html>. Acesso em: 20 set. 2022.

ZHANG, C.; YAO, W. The comparisons of 3D analysis between photogrammetry and computer vision. Yantazhonglu, China, 2005.