Universidade Federal de Juiz de Fora

Faculdade de Engenharia

Programa de Pós-Graduação em Engenharia Elétrica - Sistemas de Energia Elétrica

Luiz Augusto Zillmann da Silva

Reconstrução Tridimensional com Múltiplas Câmeras e Sistema Distribuído sob o Paradigma *Fog*

Juiz de Fora

2019

Luiz Augusto Zillmann da Silva

Reconstrução Tridimensional com Múltiplas Câmeras e Sistema Distribuído sob o Paradigma *Fog*

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica - Sistemas de Energia Elétrica da Universidade Federal de Juiz de Fora, na área de concentração em Sistemas de Energia Elétrica, como requisito parcial para obtenção do título de Mestre em Engenharia Elétrica.

Orientador: Professor Dr. Leonardo de Mello Honório

Coorientador: Professor Dr. Mario Antonio Ribeiro Dantas

Juiz de Fora

2019

Ficha catalográfica elaborada através do programa de geração
automática da Biblioteca Universitária da UFJF,
com os dados fornecidos pelo(a) autor(a)

Silva, Luiz Augusto Zillmann.
Reconstrução Tridimensional com Múltiplas Câmeras e Sistema Distribuído sob o Paradigma Fog / Luiz Augusto Zillmann Silva. -- 2019.
93 f. : il.

Orientador: Leonardo de Mello Honório
Coorientador: Mario Antonio Ribeiro Dantas
Dissertação (mestrado acadêmico) - Universidade Federal de Juiz de Fora, Faculdade de Engenharia. Programa de Pós Graduação em Engenharia Elétrica, 2019.

1. Realidade virtual. 2. Reconstrução 3D. 3. SLAM denso. I. Honório, Leonardo de Mello, orient. II. Dantas, Mario Antonio Ribeiro, coorient. III. Título.

Luiz Augusto Zillmann da Silva

Reconstrução Tridimensional com Múltiplas Câmeras e Sistema Distribuído sob o Paradigma *Fog*

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica - Sistemas de Energia da Universidade Federal de Juiz de Fora, na área de concentração em Sistemas de Energia, como requisito parcial para obtenção do título de Mestre em Engenharia Elétrica.

Aprovada em:

BANCA EXAMINADORA

Professor Dr. Leonardo de Mello Honório - Orientador
Universidade Federal de Juiz de Fora

Professor Dr. Mario Antonio Ribeiro Dantas - Coorientador
Universidade Federal de Juiz de Fora

Professor Dr. André Luís Marques Marcato
Universidade Federal de Juiz de Fora

Professor Dr. Marcelo Roberto Petry
Universidade Federal de Santa Catarina

AGRADECIMENTOS

Agradeço à minha família, em especial aos meus pais, Anelise Zillmann da Silva e Antonio Alves da Silva por todo trabalho e investimento em minha educação, e minha esposa Raïssa Ainsworth Rustichelli Teixeira, por todo o apoio nos melhores e piores momentos. Aos meus irmãos, nossa convivência desde a infância até os dias atuais ajudaram na construção de quem sou hoje.

Aos meus orientadores Professor Leonardo M. Honório, por me aceitar em seu grupo de pesquisa e acreditar em meu potencial me fornecendo todo o suporte para desenvolvimento dos trabalhos ao longo do mestrado, e Professor Mario A. R. Dantas, pelas incontáveis reuniões de trabalho, sempre construtivas e produtivas, bem como toda a motivação e confiança depositadas em mim. Os senhores têm meus agradecimentos e espero que nossa amizade permaneça após a conclusão deste trabalho.

Aos meus colegas de trabalho e estudos do Grupo de Robótica Inteligente (GRIn), em especial Vinicius F. Vidal, João M. S. Ribeiro, Felipe M. Dias, Guilherme Marins e Lucas C. N. Machado que sempre contribuíram de forma ativa para o sucesso deste e de muitos outros trabalhos do grupo.

Agradeço também à Transmissoras Brasileiras de Energia (TBE) e Enerpeixe, pelo suporte ao desenvolvimento deste trabalho através do projeto P&D "VIRTUAL". À Universidade Federal de Juiz de Fora (UFJF), por toda a estrutura e apoio durante a graduação e mestrado, uma parceria de longos anos.

RESUMO

Seja para geração de jogos eletrônicos ou para ferramentas de auxílio a profissionais, o estudo e desenvolvimento de sistemas de realidade virtual e/ou aumentada vem atingindo grandes proporções. Existem alternativas para geração de objetos e cenários tridimensionais, dentre elas os softwares padrão de design, com maior custo em tempo e mão-de-obra especializada, e os métodos de reconstrução 3D, mais rápidos e menos onerosos. Existem diversas formas de realizar a reconstrução 3D de objetos apresentadas na literatura, cada qual com sua características e peculiaridades, como através de visão estéreo, SLAM monocular, ou mesmo com câmeras RGB-D. Esta dissertação propõe uma abordagem híbrida, utilizando as vantagens de diferentes métodos, como a qualidade da odometria obtida com uma câmera estéreo aliada ao mapa bem definido obtido com o sensor RGB-D, utilizando, para isto, um sistema distribuído no paradigma FOG. A comunicação é realizada por meio do modelo publicador-subscritor, utilizando o *framework* ROS para a troca de mensagens no sistema. A utilização desta composição permite uma divisão do custo computacional, além de aumentar a escalabilidade, fazendo com que novos sensores e métodos possam ser agregados ao sistema de forma facilitada. Os modelos 3D obtidos, em forma de nuvens de pontos, se mostraram compatíveis com os melhores softwares apresentados na literatura, principalmente quando utilizado para reconstrução de ambientes e objetos de maior escala. Além disso, a utilização da visão estéreo para odometria se apresentou mais eficiente que estes métodos nas trajetórias testadas com relação às nuvens de pontos, justificando a utilização de múltiplos sensores. Os erros médios obtidos com as comparações entre a odometria visual e o braço robótico utilizado como *ground truth* são de ordem milimétrica. A implementação do sistema distribuído foi realizada com exito, porém obtendo um maior tempo de processamento devido à troca de mensagens via rede LAN. Os resultados obtidos sugerem a possibilidade da utilização da etapa de aquisição em sistemas embarcados. Através das nuvens obtidas, é possível aplicar algoritmos de geração de malhas, obtendo o objeto ou cenário a ser utilizado em um ambiente virtual ou em realidade aumentada.

Palavras-chave: Realidade virtual. Reconstrução 3D. SLAM denso.

ABSTRACT

Whether being for generating electronic games or tools of assistance to professional operators, the study and development of virtual and augmented reality systems has reached great proportions. As ways of reproducing three-dimensional objects and scenarios in virtual environments, there is reconstruction software as a faster and less costly alternative in specialized labor when compared with the standard form of generating these objects, through graphic design with specific software. There are several ways to perform 3D reconstruction presented in the literature, each with its characteristics and peculiarities, such as stereo vision, monocular SLAM, or even with RGB-D cameras. Therefore, in this work is proposed a mixed approach, using advantages of different methods, such as the accurate odometry from the stereo cameras allied with the well-defined map from the RGB-D sensor, through a distributed FOG system. The communication between processes and machines is done through the publisher-subscriber model, using the framework ROS for the message exchange. This composition allows the division of the computational costs, as well as increasing the scalability, which makes possible that new sensors and methods could be added in the future in an easy way. The obtained 3D models, in point cloud format, were comparable to the presented by the best software within the literature, mainly when used for large objects or environments reconstruction. Moreover, the stereo vision applied to obtain the odometry has shown to be more effective in the tested paths concerning the point clouds, justifying the application with multiple sensors. The odometry error, when compared to the robotic arm as the ground truth, was approximately 5 millimeters in the worst result. The implementation of the distributed system was realized successfully, having, however, a greater processing time due to the message exchange through a LAN network. The obtained results suggest the possibility of applying the acquisition step with embedded systems. Through these clouds, it is possible to apply mesh generation algorithms, obtaining the objects or scenarios to be used in virtual or augmented reality applications.

Key-words: Virtual reality. 3D reconstruction. Dense SLAM.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

UFJF	Universidade Federal de Juiz de Fora
SfM	Estrutura a partir do Movimento
SLAM	Localização e Mapeamento Simultâneos (*Simultaneous Localization and Mapping*)
SIFT	Transformação de Parâmetros Invariante a Escala
RGB	*Red, Green and Blue*
DLT	*Direct linear transformation*
ROS	*Robot Operating System*
OSRF	*Open-Source Robotic Foundation*
P2P	*Peer-to-Peer*
XML	*Extended Markup Language*
RPC	*Remote Procedure Calling*
GPU	*Graphics Processing Unit*
ICP	*Iterative Closest Point*
TSDF	*Truncated Signed Distance Function*
MVS	*Multi-View Stereo*
RGB-D	*Red, Green, Blue and Depth*
LAN	*Local Area Network*
IoT	*Internet of Things*
PCL	*Point Cloud Library*

SUMÁRIO

1 INTRODUÇÃO

A manutenção de equipamentos faz parte de diversos ambientes de engenharia, desde usinas de geração de energia até linhas de produção industriais. Para execução desse tipo de serviço, são necessários profissionais capacitados, sejam eles pertencentes ao quadro de funcionários ou terceirizados. Dessa forma, diversos cursos de capacitação e/ou treinamento são realizados para conferir ao responsável a devida capacitação.

Uma forma de conferir a esses profissionais um ambiente de treinamento rico em detalhes e livre de riscos a sua integridade é através da realidade virtual. Como sugerido pelo nome, e definido pela Sociedade de Realidade Virtual[1], trata-se de uma emulação da própria realidade, comumente utilizada em jogos e simulações, porém, recentemente, tendo foco em manutenção de equipamentos [2][3].

Existem diferentes formas de criar um ambiente virtual e os objetos tridimensionais que o compõem. Dentre elas, destacam-se softwares tradicionais de design como AutoCad e SolidWorks, que demandam mão-de-obra especializada de designers ou artistas, além de um maior tempo de desenvolvimento. Atualmente, entretanto, existem métodos mais sofisticados, que utilizam imagens capturadas do objeto a ser virtualizado, chamados de reconstrução tridimensional ou simplesmente reconstrução 3D [4]. Observa-se na literatura a utilização desse tipo de técnica em diversos sentidos, desde aplicações médicas e recriação de peças históricas à robótica e jogos 3D [5][6][7][8].

Dentre os métodos de reconstrução tridimensional, destacam-se os baseados em múltiplos pontos de vista capturados de um objeto ou cena, métodos *offline* denominados MVS (*Multi View Stereo*), e a reconstrução densa incremental em tempo real [9]. O primeiro método, apesar de ter produzido resultados empolgantes, não foi desenvolvido para aplicações em tempo real. Por outro lado, o segundo método, baseado em SLAM (*Simultaneous Localization and Mapping*), propõe um rastreamento da movimentação em tempo real, ao mesmo tempo em que promove a reconstrução do ambiente ou objeto desejado a partir dos *frames* obtidos sequencialmente [10].

Ambos os métodos citados possuem suas vantagens e desvantagens, porém, uma desvantagem que compartilham é o alto custo computacional. Em diversos trabalhos, são necessárias placas gráficas também chamadas *Graphics Processing Unit* (GPU), e processamento paralelo para mitigar esse efeito, e, principalmente no MVS, o processamento ainda é bastante demorado quando o objetivo é uma reconstrução densa [11][10][12][13].

Dessa forma, define-se a seguinte pergunta de pesquisa: **É possível desenvolver um sistema distribuído de reconstrução 3D sem grande prejuízo no desempenho visando futuras aplicações embarcadas?** Como resposta a esta pergunta, é sugerida a utilização de um sistema distribuído seguindo o paradigma *fog*, através da separação das etapas em nós *farmers* e *workers*, onde o sistema que contém os sensores representa o *worker* localizados na periferia do sistema que processa as imagens adquiridas e envia nuvens de pontos para serem reprocessados pelo *farmer*. Além disso, propõe-se a utilização de diferentes tipos de câmeras, fazendo uso das melhores características de cada uma delas. São definidos então os objetivos desta dissertação, bem como a metodologia utilizada para atingi-los, apresentados a seguir.

1.1 OBJETIVOS GERAIS

Elaborar o estudo sobre métodos de reconstrução tridimensional, avaliando suas principais características, principalmente o que os torna mais eficientes e o que prejudica seus resultados, utilizando estas informações para desenvolvimento de um sistema novo e eficaz. Além disso, estudar métodos de comunicação para sistemas distribuídos e o paradigma *fog*, visando futuras aplicações embarcadas.

1.2 OBJETIVOS ESPECÍFICOS

De forma a atender o objetivo geral proposto anteriormente, foram definidos os seguintes objetivos específicos:

- Identificação de melhores métodos de reconstrução 3D;

- Identificar dentre os métodos anteriores, seus resultados quanto a localização do sensor e o modelo tridimensional gerado;
- Propor a utilização de múltiplos sensores, fazendo uso das vantagens de cada um deles, para aquisição de informação do ambiente *in door* com baixa interferência luminosa;
- Propor sistema distribuído para reconstrução 3D, visando divisão do custo computacional entre máquinas do sistema;
- Avaliar o sistema em diversos cenários, sujeito a diferentes níveis de complexidade, e a possibilidade de aplicação de aquisição em sistema embarcado.

1.3 METODOLOGIA

Para atingir os objetivos propostos, foi utilizada a seguinte metodologia:

- Estudo da literatura, tanto a mais recente como os livros clássicos mais utilizados, buscando diferentes métodos de reconstrução 3D e os sensores utilizados por cada um deles;
- Implementação de sistema distribuído proposto, utilizando diferentes câmeras;
- Realização de experimentos para validação do sistema;
- Análise dos resultados obtidos em diferentes cenários;
- Comparação com métodos propostos na literatura.

1.4 ESTRUTURA DO TRABALHO

Esta dissertação propõe a utilização de múltiplos sensores através de um sistema distribuído, visando manter a qualidade da reconstrução e dividir o custo computacional entre as máquinas do sistema. As aquisições são limitadas a ambientes fechados e com pouca interferência de iluminação solar, além das limitações de distância características dos sensores utilizados. Este trabalho está dividido em 5 capítulos. O Capítulo 2 apresenta uma breve descrição de conceitos fundamentais

para entendimento da proposta deste trabalho. Além disso, apresenta uma revisão da literatura, contendo os principais trabalhos que influenciaram esta dissertação. No Capítulo 3, é apresentada a proposta de forma mais detalhada, descrevendo os tipos de sensores utilizados, bem como a composição do sistema distribuído. No Capítulo 4, é apresentado o ambiente experimental e os resultados obtidos com o método aplicado, bem como problemas encontrados ao longo do seu desenvolvimento. Finalizando, no Capítulo 5 são apresentadas as conclusões obtidas sobre a eficácia do método, onde foi obtido sucesso e o que precisa de melhoramentos. Além disso, são apresentadas as possibilidades para continuação do trabalho em relação a comparação dos resultados observados com a literatura.

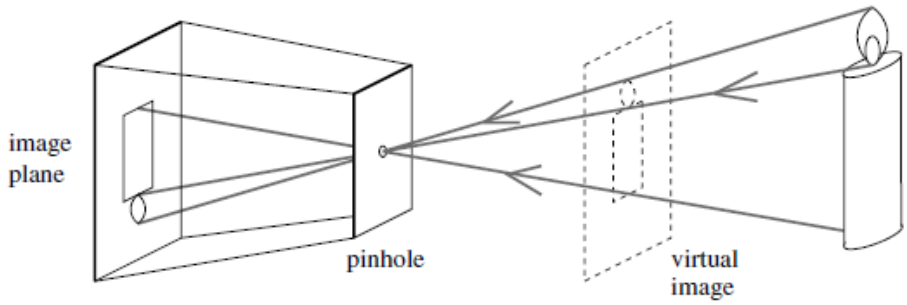
2 REFERENCIAL TEÓRICO E REVISÃO BIBLIOGRÁFICA

A presente seção tem por objetivo fornecer fundamentações teóricas, bem como uma revisão da literatura recente e relevante, para o bom entendimento da abordagem adotada. São discutidos conceitos de Visão Computacional, desde a aquisição da imagem até o processamento de nuvens de pontos, métodos de reconstrução tridimensional e comunicação entre os processos.

2.1 FORMAÇÃO DAS IMAGENS

Existem formas diferentes de representar o mundo real através de imagens, de pinturas até câmeras digitais modernas. Câmeras são dispositivos óticos utilizados para captura de imagens. Historicamente, a primeira câmera utilizada é a *Pinhole* que consiste em uma câmara escura com um furo em lugar de uma lente, o que dá origem ao nome. Por este furo passam os raios de luz, formando uma imagem invertida no interior da câmara [14][15][16]. A ilustração da câmera descrita é observada na Figura 1.

Figura 1 – Ilustração para camera *Pinhole*.



Fonte: [14]

Posteriormente, as câmera escuras foram substituídas por versões mais sofisticadas baseadas em lentes. A atual versão de câmera, a digital, simplificadamente consiste em lentes para o orifício de entrada de luz e uma câmara escura com uma superfície composta de material semicondutor capaz de gravar a imagem.

Define-se imagem, portanto, como sendo uma função bidimensional $f(x, y)$, onde x e y são coordenadas da imagem no plano de formação, e a amplitude f representa a intensidade naquela posição. Quando estas grandezas são definidas de forma finita e discreta, a imagem é dita digital [17].

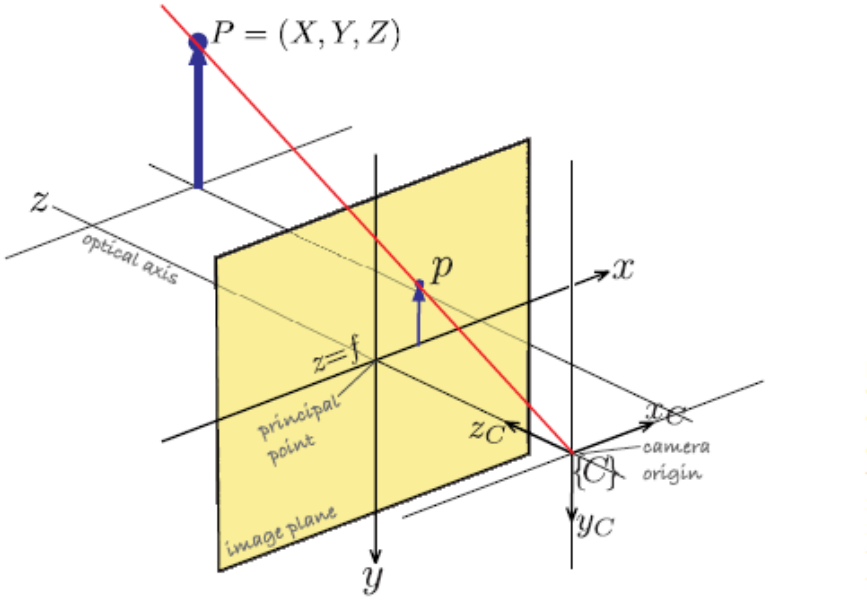
Uma imagem digital é composta por um número finito de elementos, cada qual com sua intensidade e posição. Estes elementos são chamados *pixels*, ou *picture elements*. Este termo é utilizado amplamente na literatura ao se tratar de imagens digitais. Com relação a intensidade, esta pode ser em escala de cinza, variando de 0 a 255, ou em cores, por exemplo RGB, onde cada cor tem sua escala de 0 a 255. [16][18][17][19].

2.2 VISÃO COMPUTACIONAL

Nesta seção, apresenta-se a definição e conceitos de Visão Computacional. Em seu livro [16], Peter Corke a define como sendo um grande campo de estudo onde o foco é o processamento de imagens para melhorá-las em benefício humano, interpretar o conteúdo de uma cena, ou mesmo a criação de modelos tridimensionais da cena em questão. Além disso, Solem diz que Visão se utiliza de métodos estatísticos para separar dados, fazendo uso de informações geométricas, físicas e teorias de aprendizado [19].

Partindo destas definições e tendo em vista o modelo de câmera citado anteriormente, é comum utilizar-se da perspectiva apresentada pela Figura 2, onde a imagem é formada em um plano a frente do centro de coordenadas que representa a posição da câmera. Nota-se que, desta forma, a imagem não é mais representada de forma invertida e o plano onde ocorre a projeção é conhecido como Plano da Imagem. Este plano dista $z = f$ em relação a origem do sistema de coordenadas da câmera, também chamado de *frame* da câmera. A distância f é chamada foco ou distância focal [16].

Figura 2 – Formação da imagem no plano frontal.



Fonte: [16].

Por semelhança de triângulos, é possível demonstrar que o ponto real $\mathbf{P} = (X, Y, Z)$, com relação ao *frame* da câmera, é projetado no plano da imagem como $\mathbf{p} = (x, y)$, seguindo as Equações 2.1 e 2.2. Trata-se, então, de uma projeção em perspectiva de $\mathbb{R}^3 \rightarrow \mathbb{R}^2$ [16].

$$x = f\frac{X}{Z} \tag{2.1}$$

$$y = f\frac{Y}{Z} \tag{2.2}$$

De forma a facilitar o desenvolvimento matemático matricial, em vez da utilização de sistemas de coordenadas Euclidianas são utilizadas coordenadas homogêneas. A diferença enter os sistemas está no número de dimensões aumentado, ou seja, o que no Euclidiano possui dimensão n, no sistema homogêneo possui $n + 1$. Dessa forma, o ponto pertencente ao plano da imagem, $\mathbf{p} = (x, y)$ passa a

ser representado como $\tilde{\mathbf{p}} = (x', y', z')$, onde

$$x' = f\frac{X}{z'}, \quad y' = f\frac{Y}{z'}, \quad z' = Z \tag{2.3}$$

Passando para a forma matricial compacta, tem-se a Equação 2.4

$$\tilde{\mathbf{p}} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{2.4}$$

Seguindo a mesma linha, o ponto tridimensional P em coordenadas homogêneas pode ser escrito como $\tilde{\mathbf{P}} = (X, Y, Z, 1)$, fazendo com que a perspectiva, com respeito ao *frame* da câmera, possa ser descrita pela Equação 2.5 a seguir.

$$\tilde{\mathbf{p}} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tilde{\mathbf{P}} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tilde{\mathbf{P}} \tag{2.5}$$

Ou

$$\tilde{\mathbf{p}} = \mathbf{K}\tilde{\mathbf{P}}, \quad para \quad \mathbf{K} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.6}$$

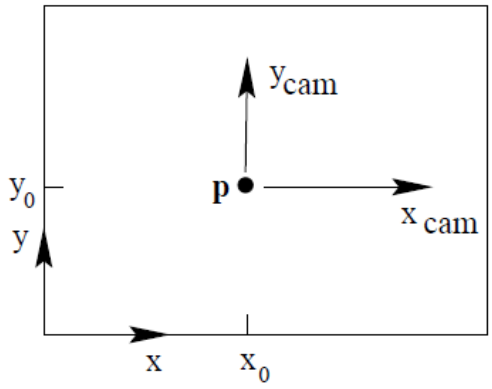
Onde $\mathbf{K}$, apresentada em 2.6, é chamada matriz intrínseca da câmera, ou simplesmente matriz da câmera, que será mais discutida na próxima seção. Mais detalhes a respeito da descrição matemática feita acima podem ser encontrados em [16], [14] e [20].

2.3 TRANSFORMAÇÃO DE CORPO RÍGIDO E CALIBRAÇÃO DE CÂMERAS

A partir das equações descritas anteriormente, é possível definir que os parâmetros intrínsecos são aqueles relacionados com a fabricação da câmera, ou seja, os parâmetros que compõem a matriz $\mathbf{K}$. As equações anteriores foram

descritas assumindo que a origem do sistema de coordenadas no plano da imagem está sobre o ponto principal, o que pode não ser verdade na maioria das situações. Dessa forma, é realizado um novo mapeamento e uma alteração para generalização na matriz **K** [20]. A Figura 3 ilustra o mapeamento.

Figura 3 – Mapeamento do plano da imagem relativo ao sistema de coordenadas da câmera.



Fonte: [20].

De forma a representar esse mapeamento, altera-se a Equação 2.5 da seguinte forma.

$$\tilde{\mathbf{p}} = \begin{bmatrix} f & 0 & x_0 & 0 \\ 0 & f & y_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tilde{\mathbf{P}} \tag{2.7}$$

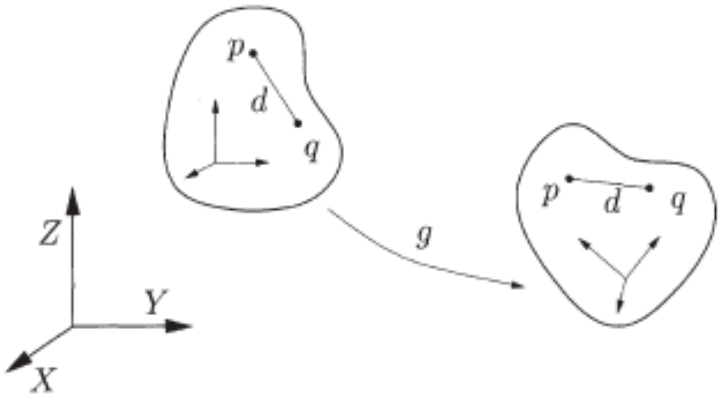
Já os parâmetros extrínsecos são os que relacionam o sistema de coordenadas da câmera com um sistema de coordenadas global fixo. A forma de relacionar estes *frames* é através de transformações de corpos rígidos.

2.3.1 Transformações de Corpo Rígido

Ma *et al.* [21], em seu livro, afirma que o estudo da projeção de perspectiva juntamente com transformações de corpo rígido, também chamadas de Movimento Euclidiano, são fundamentais para o estudo entre cenas tridimensionais e suas imagens bidimensionais. Para definir este tipo de transformação, utiliza-se da

propriedade de corpos rígidos de que para descrever o movimento de todos os pontos do corpo, é necessário somente especificar o movimento de um ponto, ou seja, o movimento do sistema de coordenadas anexo a este ponto. Isto somente pode ser garantido porque, por definição, a distância entre os pontos de um corpo rígido devem ser preservadas ao longo de movimentos. A Figura 4 exemplifica o movimento de um corpo rígido, enaltecendo os *frames* coordenados do corpo e o global, bem como a conservação da distância entre os pontos p e q.

Figura 4 – Distância preservada no movimento de um corpo rígido.

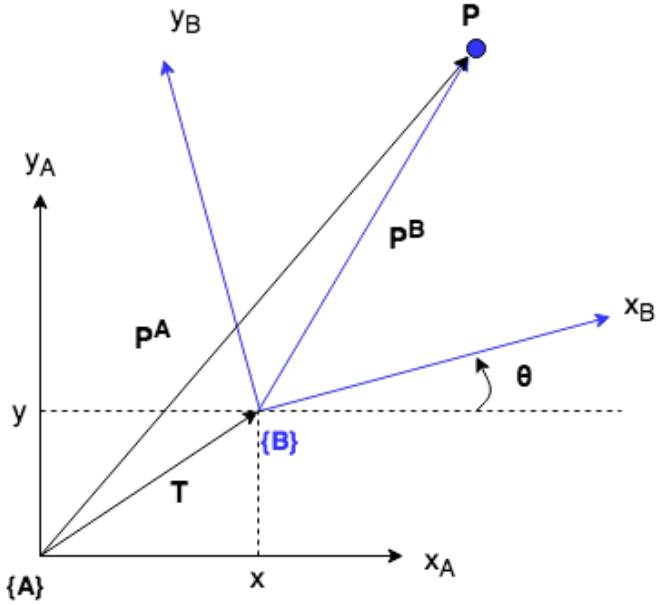


Fonte: [21].

Nesta dissertação, as transformações de corpos rígidos são utilizadas para relacionar os eixos coordenados dos sistemas. As transformações são compostas por dois tipos de movimentos relativos, translações e rotações. Translação corresponde a movimentar todos os pontos de um corpo rígido para uma mesma direção. Rotações indicam a rotação de um corpo em torno de um ponto ou eixo.

As Equações 2.8, 2.9 e 2.10 a seguir descrevem uma transformação para representação de um ponto em sistemas de coordenadas genéricos apresentados na Figura 5 [16][21].

Figura 5 – Translação e rotação em duas dimensões.



Fonte: Elaborada pelo autor.

Seja o ponto $\mathbf{p}^B = x^B\hat{x}_B + y^B\hat{y}_B$, onde $\hat{x}_B$ e $\hat{y}_B$ representam os vetores unitários paralelos aos eixos do sistema de coordenadas B, a representação do ponto P no *frame* B. A forma de determinar a posição do ponto P em relação ao *frame* A, ou seja, P^A, pode ser feita aplicando a seguinte transformação.

$$\mathbf{P}^A = \mathbf{T}_B^A\mathbf{P}^B \quad com \quad \mathbf{T}_B^A \in \mathbb{R}^{4x4} \tag{2.8}$$

Na Equação 2.8, $\mathbf{T}_B^A$ representa a transformação que descreve o ponto P em relação ao *frame* A, e pode ser descrita por uma rotação em torno da origem do sistema de coordenadas B e uma translação até o sistema A, como na Equação 2.9 mostra na forma compacta.

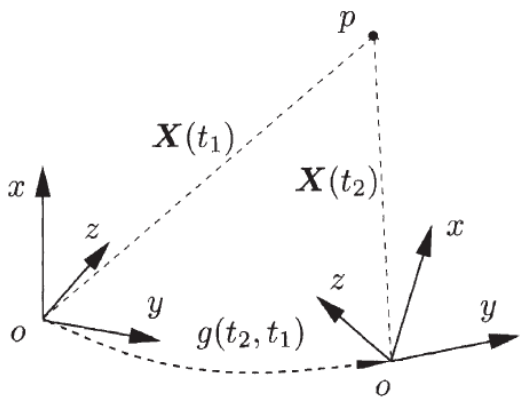
$$\mathbf{P}^A = \begin{bmatrix} \mathbf{R}_B^A & \mathbf{T} \\ 0_{1x2} & 1 \end{bmatrix} \mathbf{P}^B \tag{2.9}$$

Onde

$$\mathbf{R}_B^A = \begin{bmatrix} cos\theta & -sen\theta \\ sen\theta & cos\theta \end{bmatrix} \quad E \quad \mathbf{T} = \begin{bmatrix} x \\ y \end{bmatrix} \tag{2.10}$$

Trazendo para um exemplo onde há uma câmera em movimento, é possível expressar esse deslocamento através de uma transformação homogênea agora em três dimensões. A Figura 6 ilustra o problema.

Figura 6 – Translação e rotação em exemplo de movimento de câmera.



Fonte: [21] Editada.

Na Figura 6 anterior, ocorre o movimento da câmera, representada pelos sistemas de coordenadas em tempos diferentes, t_1 e t_2, bem como a posição de um ponto P em relação a cada sistema, $\mathbf{X}(t_1)$ e $\mathbf{X}(t_2)$. A função $g(t_1, t_2)$ representa o movimento entre os sistemas de coordenadas, ou seja, uma transformação de corpo rígido. A Equação 2.11 mostra a representação do ponto P em relação ao tempo t_2 a partir de $\mathbf{X}(t_1)$.

$$\mathbf{X}(t_2) = g(t_1, t_2)\mathbf{X}(t_1), \quad \forall t_1, t_2 \in \mathbb{R} \tag{2.11}$$

Onde

$$g(t_1, t_2) = \begin{bmatrix} \mathbf{R}(t_1, t_2) & \mathbf{T}(t_1, t_2) \\ 0_{1x3} & 1 \end{bmatrix} \tag{2.12}$$

Mais detalhes sobre transformações de corpo rígido podem ser encontrados em [21] e [16].

2.3.2 **Calibração**

Calibração é processo de obtenção dos parâmetros intrínsecos da câmera, ou seja, dos elementos da matriz **K**, além de parâmetros de distorção que podem ser utilizados para remover imperfeições na imagem, como distorções radiais oriundas do formato das lentes, ou tangenciais originadas no processo de montagem da câmera [20][22]. Para obter estes elementos, é preciso primeiramente obter a matriz de projeção, que carrega as informações intrínsecas e extrínsecas, como visto na Equação 2.13.

$$\mathbf{P} = \mathbf{K}[\mathbf{R}|\mathbf{T}] \tag{2.13}$$

Onde **P** é a matriz de projeção que mapeia um ponto homogêneo 3D em um ponto homogêneo 2D. **K** é a matriz de calibração, **R** a matriz de rotação que juntamente com **T**, matriz de translação, compõe a transformação do corpo rígido.

Os parâmetros que se deseja encontrar são distância focal, f_x e f_y, e centro ótico da câmera, x_0 e y_0, em *pixels*. Para isso é utilizado o algoritmo DLT, melhor descrito em [20].

Com relação às distorções, começando pelas radiais, seu efeito é mais evidente próximo a arestas contidas na imagem, gerando um efeito abaulado. A distorção radial é zero sobre o centro óptico da imagem e aumenta conforme se aproxima da periferia. Este tipo de distorção é pequeno e pode ser descrito pelos primeiros termos da expansão em série de Taylor em torno do centro (r = 0). Para câmeras padrão, os dois primeiros termos da série são utilizados, chamados k_1 e k_2, por convenção. Entretanto, para lentes com maior grau de distorção, como olho de peixe por exemplo, utiliza-se até o terceiro termo k_3 da série [22].

A seguir são apresentadas as Equações 2.14 e 2.15, utilizadas para correção das distorções do tipo radial. Onde (x, y) são as coordenadas originais da imagem para um ponto distorcido. Já $(x_{corrigido}, y_{corrigido})$ são as coordenadas novas corrigidas daquele ponto.

$$x_{corrigido} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{2.14}$$

$$y_{corrigido} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{2.15}$$

Onde $r^2 = x^2 + y^2$.

As distorções do tipo tangencial, como mencionado anteriormente, são oriundas de problemas de montagem e podem acarretar em um desalinhamento entre a lente e o plano de formação da imagem. Desta forma, a correção para tais distorções é apresentada pelas Equações 2.16 e 2.17, onde são adicionados os parâmetros p_1 e p_2, totalizando quatro (ou cinco, dependendo da câmera) coeficientes de distorção. Existem outros tipos de distorções, porém seus efeitos podem ser desprezados quando comparados às distorções radiais e tangenciais.

$$x_{corrigido} = x + [2p_1 y + p_2(r^2 + 2x^2)] \tag{2.16}$$

$$y_{corrigido} = y + [p_1(r^2 + 2y^2) + 2p_2 x] \tag{2.17}$$

Onde $r^2 = x^2 + y^2$.

A etapa de calibração é de suma importância para este trabalho, visto que é ela que possibilita a utilização da topologia de câmeras proposta. Mais detalhes serão discutidos no Capítulo 3.

2.4 VISÃO ESTÉREO

Trata-se de uma forma de emular a capacidade que os olhos humanos nos dão através de um par de câmeras, ou de um par de pontos de vista quando aplicado de forma monocular. A forma como os computadores fazem isso é através

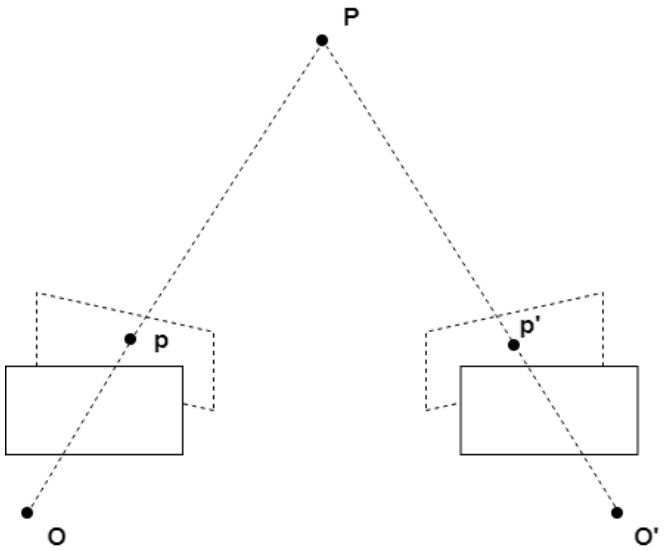
de correspondências entre as imagens obtidas por cada câmera. Utilizando estas correspondências e a linha base de separação entre as duas câmeras, é possível determinar a posição tridimensional dos pontos das imagens. O processo de busca por correspondências é computacionalmente custoso, fazendo com que seja necessário o uso de alguns conhecimentos de geometria para reduzir o espaço de busca, facilitando o processo [22].

Pode-se resumir a visão estéreo, quando usando duas câmeras, em quatro etapas: Remoção das distorções, retificação, encontro de correspondências e reprojeção. A remoção das distorções é obtida através da utilização dos parâmetros de calibração citados anteriormente, considerando distorção radial e tangencial.

2.4.1 **Retificação**

No mundo real, as câmeras raramente estarão exatamente alinhadas da forma necessária. Assim, deve-se utilizar os parâmetros de calibração e remoção de distorções citadas para posicionar os planos de forma retificada. Câmeras alinhadas fisicamente facilitam o trato matemático, fazendo com que o design do sistema estéreo possua grande importância. Portanto, utiliza-se a retificação, processo de ajuste de posição entre as duas câmeras, fazendo com que as imagens fiquem alinhadas e retificadas. A Figura 7 ilustra a retificação, começando com os planos representados pelos paralelogramos em linha pontilhada, e terminando com os retângulos em linha cheia.

Figura 7 – Retificação para visão estéreo.

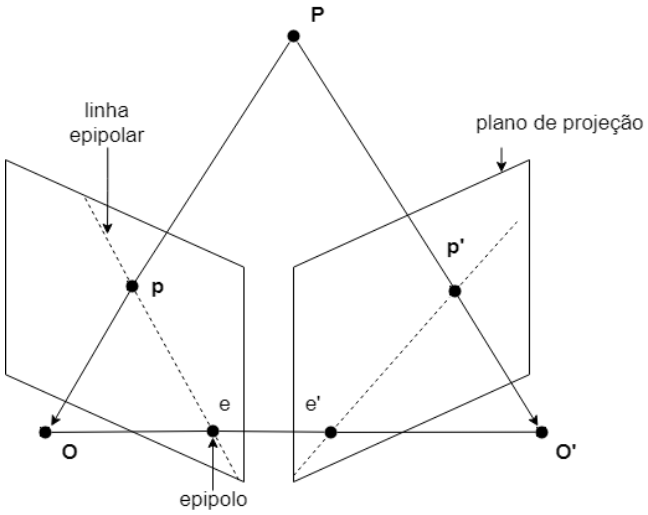


Fonte: Elaborada pelo autor.

2.4.1.1 *Geometria Epipolar*

Como base para a descrição do processo de retificação, aplica-se a Geometria Epipolar. Essa geometria, dependente dos parâmetros internos das câmeras, se utiliza de dois modelos *pinhole* e pontos de interesse nas imagens, chamados epipolos. Um epipolo, em um plano de projeção, é definido como a imagem do centro de projeção da câmera referente ao outro plano de projeção [20][22]. Para facilitar o entendimento, observa-se a Figura 8.

Figura 8 – Retificação para visão estéreo.



Fonte: Elaborada pelo autor.

Os pontos gerados pela interseção da linha que conecta os centros de projeção da câmeras e os planos de projeção, são chamados epipolos. As linhas que conectam os epipolos aos pontos **p** e **p'**, projeções do ponto **P**, no respectivo plano, são chamadas linhas epipolares. Para o caso da imagem da câmera à esquerda na Figura 8 acima, esta linha é definida por **p** e *e*. Além disso, define-se o plano formado entre o ponto real **P** e os centros de projeção **O** e **O'** como plano epipolar.

Para entender a utilidade das definições feitas acima, toma-se como referência uma das câmeras, a da esquerda por exemplo. Apenas com a imagem **p**, não é possível determinar a localização completa do ponto real, visto que o mesmo pode estar em qualquer parte da reta que conecta o ponto real e o centro de projeção. Entretanto, a projeção desta reta no plano da outra imagem (a da direita neste exemplo), é equivalente a linha epipolar, fazendo-nos concluir que a imagem de qualquer localização possível de **P** está nesta linha.

Como conclusão tem-se que um ponto característico observado em uma imagem, seu par correspondente em outra imagem deve estar localizado em sua linha epipolar correspondente, restrição esta conhecida como Restrição Epipolar. Esta restrição facilita a busca por correspondências, visto que reduz o espaço de busca de um plano para uma linha, reduzindo custos computacionais e eliminando

erros de correspondência.

2.4.1.2 *Matrizes Essencial e Fundamental*

Além da geometria descrita anteriormente, são necessários mais conceitos para realizar o alinhamento das câmeras: a matriz Essencial e a matriz Fundamental. A primeira carrega em si a informação referente à transformação de corpo rígido que leva do *frame* de uma câmera a outra no espaço físico, ou seja, os parâmetros extrínsecos de rotação e translação. Já a segunda, carrega as informações da primeira, de pose relativa, acrescida de informações sobre os parâmetros intrínsecos de calibração das duas câmeras [22].

A principal propriedade da matriz essencial utilizada neste trabalho é descrita na Equação 2.18 a seguir, que referencia os pontos das imagens em coordenadas físicas (reais) ou relativas às câmeras, não em pixeis, visto que só carrega informações de rotação e translação [20][22].

$$\mathbf{p'}^T\mathbf{E}\mathbf{p} = 0 \tag{2.18}$$

A partir da Equação 2.18 descrita acima, aparenta-se conhecer todos os dados de um ponto desde que seja conhecido o ponto na câmera oposta. Porém, há de se notar que a matriz $\mathbf{E}$ é deficiente em *rank*, sendo 3x3 e possuindo *rank* 2. Tem-se por essa matriz três parâmetros de rotação e dois de translação, não sendo possuído o parâmetro de escala.

Através da propriedade acima, é possível incluir as informações intrínsecas, visto que $\mathbf{E}$ apenas contém informações geométricas que relacionam uma câmera a outra em unidades físicas, não em pixeis. Dessa forma, seja K a matriz de calibração da câmera da esquerda, K' para a câmera da direita, de modo que $\mathbf{q} = K\mathbf{p}$.Tem-se que

$$\mathbf{q'}^T(\mathbf{K'}^{-1})^T\mathbf{E}\mathbf{K}^{-1}\mathbf{q} = 0 \tag{2.19}$$

Onde a matriz fundamental é definida como

$$\mathbf{F} = (\mathbf{K'}^{-1})^T\mathbf{E}\mathbf{K}^{-1} \tag{2.20}$$

Chegando a propriedade

$$\mathbf{q'}^T\mathbf{Fq} = 0 \tag{2.21}$$

Em [20], é dito que a matriz fundamental é a representação algébrica da geometria epipolar, vista através da Equação 2.22, onde **p** é o ponto de projeção no plano da esquerda, l' é a linha epipolar no plano da direita na Figura 8, e **F** é a matriz fundamental.

$$\mathbf{l'} = \mathbf{Fp} \tag{2.22}$$

Algoritmos para determinar a matriz fundamental se baseiam em correspondência de pontos, como o algoritmo de 7 pontos ou o algoritmo de 8 pontos [20][22]. Um problema destes métodos é que são muito sensíveis a *outliers*, sendo necessário filtrá-los antes do processo.

Assim como a matriz essencial, **F** também é deficiente em *rank*, possuindo sete parâmetros, dois para cada epipolo e três para a homografia que relaciona os dois planos das imagens. O fator de escala continua em falta.

2.4.1.3 *Calibração Estéreo*

Calibração estéreo é o processo de obtenção da relação geométrica entre as duas câmeras. Difere da retificação, que busca alinhar os planos das imagens, como se as câmeras estivessem alinhadas na hora da captura.

Dessa forma, a calibração consiste em encontrar a matriz de rotação e o vetor de translação que relacionam as câmeras. Para isso, faz-se uso dos parâmetros de calibração individual de cada câmera juntamente com a matriz essencial, ou através da matriz fundamental.

Seja, então, **P** um ponto qualquer no sistema de coordenadas do objeto. Através da calibração simples, pode-se obter a transformação que mapeia o ponto em relação aos sistemas de coordenadas de cada câmera. As Equações 2.23 e 2.24

calculam o ponto **P** referente ao sistema de coordenadas da câmera da esquerda e ao da direita, respectivamente.

$$\mathbf{P}_{esquerda} = \mathbf{R}_{esquerda}\mathbf{P} + \mathbf{T}_{esquerda} \tag{2.23}$$

$$\mathbf{P}_{direita} = R_{direita}\mathbf{P} + \mathbf{T}_{direita} \tag{2.24}$$

Tem-se, então, a Equação 2.25 que relaciona os pontos referentes a cada sistema de coordenadas das câmeras, por meio da matriz de rotação **R** e vetor de translação **T**.

$$\mathbf{P}_{esquerda} = \mathbf{R}^T(\mathbf{P}_{direita} - \mathbf{T}) \tag{2.25}$$

Com as três equações acima, encontra-se as relações

$$\mathbf{R} = \mathbf{R}_{direita}(\mathbf{R}_{esquerda})^T \tag{2.26}$$

e

$$\mathbf{T} = \mathbf{T}_{direita} - \mathbf{R}\mathbf{T}_{esquerda} \tag{2.27}$$

2.4.1.4 *Algoritmos para Retificação*

A partir do ferramental apresentado anteriormente, é possível aplicar métodos diferentes para obtenção da retificação, lembrando de seu benefício ao reduzir o custo computacional na busca por correspondências. Existem diversos métodos para realizar esse processo, porém os mais comumente utilizados são os Algoritmo de Hartley [23][20] e o Algoritmo de Bouguet [22].

O algoritmo de Hartley tem por objetivo encontrar homografias que mapeiam os epipolos ao infinito, minimizando a disparidade entre as imagens das câmeras. Não necessita encontrar os parâmetros intrínsecos, sendo necessário, porém, a obtenção da matriz fundamental. Este algoritmo possui a vantagem de ser realizado

de forma *online*, simplesmente observando os pontos da cena, porém possui a desvantagem de não atribuir escala, ou seja, não é possível determinar se o objeto utilizado na calibração está perto da câmera ou se trata de um objeto grande, por exemplo.

Já o algoritmo de Bouguet se utiliza de parâmetros de rotação e translação de duas câmeras já calibradas. Trata-se de simplesmente minimizar o número de variações que ocorrem em reprojeções, minimizando, consequentemente, as distorções de reprojeções. Além disso, maximiza a área sobreposta, ou seja, a área comumente vista pelas duas câmeras.

2.4.2 **Correspondências Estéreo**

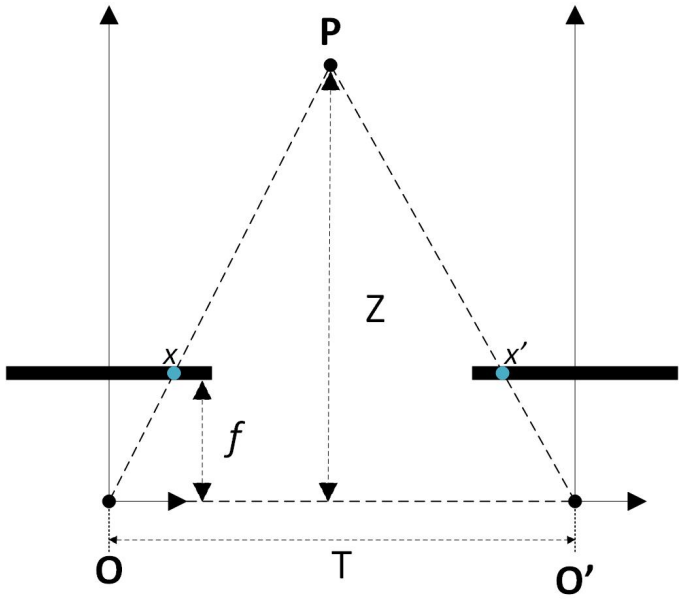
De forma lógica, as Correspondências Estéreo somente podem ser tomadas nas áreas sobrepostas, ou seja, áreas comuns entre as imagens das câmeras. Tendo conhecimento das coordenadas físicas da câmera, ou do tamanho de objetos em cena, é possível obter medidas de profundidade a partir da triangulação de disparidades. Sem esses dados, como visto no algoritmo de Hartley, só é possível obter a profundidade por um fator de escala.

As correspondências podem ser encontradas de diversas formas [22][24], lembrando sempre que, após a retificação, cada linha da imagem é agora uma linha epipolar. Então, o ponto de interesse na imagem da esquerda, por exemplo, tem seu correspondente na mesma linha na imagem da direita.

2.4.3 **Triangulação**

Após a remoção das distorções, alinhamento dos planos das imagens definindo a posição estéreo das câmeras, assumindo que as imagens estão alinhadas por linha, ou seja, toda linha de pixeis em uma das câmeras se alinha exatamente a sua linha correspondente na outra câmera, define-se este arranjo como paralelo frontal. Seja um ponto **P**, pertencente ao mundo real, com projeções nos planos das imagens **p** e **p'**, tem-se as coordenadas horizontais x e x', como observado na Figura 9.

Figura 9 – Componentes necessários para cálculo da disparidade.



Fonte: Elaborada pelo autor.

Neste caso, é possível mostrar que a profundidade é inversamente proporcional à disparidade entre essas vistas. Esta disparidade pode ser descrita por $d = x - x'$. Dessa forma, por semelhança de triângulos, é possível obter a profundidade Z do ponto **P**, descrita na Equação 2.28, onde T é a distância entre os centros de projeções das câmeras e f é a distância focal após a calibração.

$$Z = \frac{fT}{x - x'} \tag{2.28}$$

Devido a essa relação não-linear, é possível notar que quando a disparidade é aproximadamente 0, pequenas diferenças de disparidade causam grandes diferenças de profundidade. Já para o caso de grandes disparidades, pequenas variações de disparidade não geram diferença de profundidade tão demasiada. Como conclusão, tem-se que sistemas baseados em visão estéreo possuem alta resolução de profundidade para objetos relativamente perto das câmeras [22]. Este efeito será discutido mais detalhadamente no Capítulo 4.

2.5 SISTEMAS DISTRIBUÍDOS

Existem diversas definições para sistemas distribuídos, também chamados paralelos, porém, para esta dissertação, são utilizadas as definições de Tanenbaum [25] e Coulouris [26], concordando que sistemas distribuídos podem ser definidos como um conjunto de computadores independentes conectados em uma rede de dados, se apresentando aos usuários como um sistema único e coerente, se comunicando e coordenando ações por meio de troca de mensagens. Algumas formas de troca de mensagem, bem como a adotada por este trabalho, são discutidas a seguir.

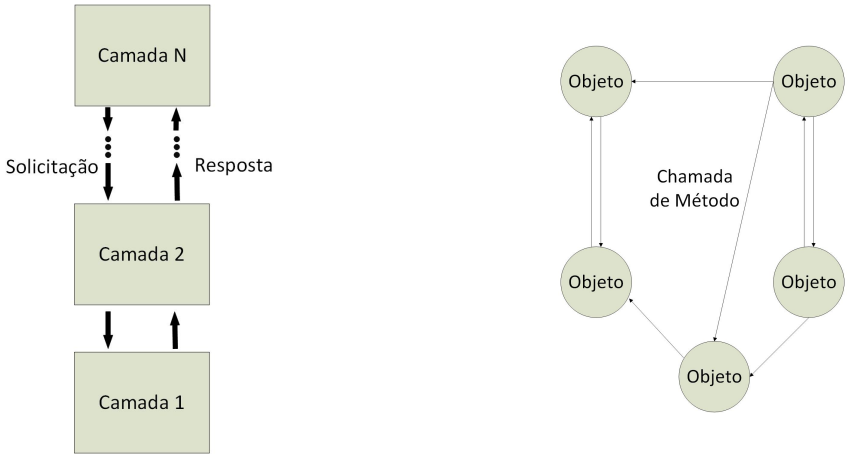
2.5.1 **Arquitetura de Sistemas Distribuídos**

A organização de sistemas distribuídos é, em sua maioria, relacionada a disposição dos *softwares* que compõe o sistema. A arquitetura diz como estes vários componentes são organizados e como devem interagir. A criação de um sistema distribuído requer que os *softwares* sejam instanciados e alocados em máquinas reais. Após a arquitetura de software, que corresponde à organização lógica entre os componentes, ser instanciada, esta passa a ser denominada arquitetura do sistema.

Para discutir arquiteturas de *software*, é importante também definir o estilo aplicado. Este estilo é a forma como os componentes são conectados entre si, como trocam informações e como fica configurado o sistema. Um componente de um sistema distribuído deve poder ser substituído, desde que respeite a norma de interface. Já um conector, é um mecanismo que faz o intermédio da comunicação, coordenação ou cooperação entre componentes. Através de componentes e conectores é possível definir diferentes estilos de arquiteturas.

Segundo Tanenbaum [25], os estilos mais importantes para sistemas distribuídos são os baseados em camadas, baseados em objetos, centrados em dados e baseado em eventos. Além disso, é dito que, dentre estes, os baseados em camadas e em objetos são os mais comumente utilizados e importantes, ilustrados pela Figura 10.

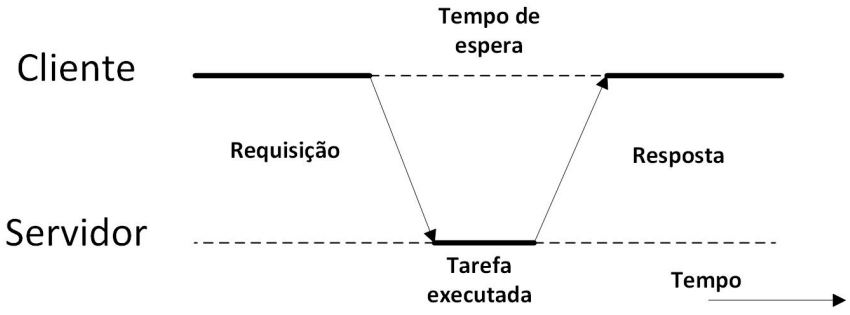
Figura 10 – Estilos de comunicação baseado em camadas, à esquerda, e baseado em objetos, à direita.



Após discutir os estilos de arquiteturas, deve-se considerar como os componentes estão organizados. Existem organizações de arquiteturas do tipo centralizada, onde um servidor implementa a maior parte dos componentes e funcionalidades, e a descentralizada, onde as máquinas possuem papéis equivalentes. Existem, também arquiteturas híbridas, onde ocorre a combinação das características das anteriores.

A arquitetura distribuída centralizada pode ser melhor entendida através do modelo cliente-servidor. Os processos nesse modelo são divididos em dois grupos: Clientes, que solicitam um serviço específico e aguardam uma resposta, e Servidores, que fornecem o serviço solicitado pelos clientes. Este tipo de interação também é chamada de pedido-resposta, do inglês *request-reply*, ilustrada na Figura 11.

Figura 11 – Interação genérica entre cliente e servidor.



Fonte: Elaborada pelo autor.

A forma mais simples de organizar um sistema no modelo cliente/servidor é tendo dois tipos de máquinas, um cliente contendo somente a implementação no nível de interface com o usuário, e uma máquina como servidor contendo o resto dos programas. Entretanto, sistemas tendo solução com base em múltiplos servidores estão mais comuns.

Em arquiteturas descentralizadas, as funções são espalhadas em múltiplas máquinas de forma tanto física como lógica. As máquinas possuem, então, um grupo muito específico de funções sendo, neste tipo de distribuição, cada parte responsável por seu próprio conjunto completo de dados, deixando a carga balanceada. Uma classe deste tipo de arquitetura é chamada *peer-to-peer* (P2P), ou par-a-par, onde cada processo se comporta como cliente e servidor. Essa classe evoluiu buscando organizar processos em nós e conexões, sendo os nós uma representação dos processos, e as conexões, como sendo possíveis canais de comunicação.

Arquiteturas híbridas, por sua vez, como já mencionado, combinam as características das arquiteturas anteriores, a exemplo do *superpeer* apresentado em [25]. Dentro deste tipo de arquitetura, encontra-se a classe de sistemas *edge-server*. Nesses sistemas, os servidores são posicionados nas periferias da rede, tendo como objetivo principal fornecer conteúdo após aplicar algum tipo de pré-processamento. Servidores *edge* podem ser utilizados para otimização de conteúdo e aplicações. Além deste exemplo, há também os sistemas híbridos colaborativos, como visto em [27], a ser melhor discutido na Seção 2.6.

2.5.2 **Comunicação em Processos Distribuídos**

Tanenbaum diz que a comunicação entre processos é o ponto mais importante em sistemas distribuídos [25]. Diferentemente dos não paralelos, que são baseados na ideia de memória compartilhada, os distribuídos dependem diretamente de trocas de mensagens entre processos, desde mais simples, até os mais complexos e modernos, como é o caso da internet.

Devido à falta da memória compartilhada, são utilizadas trocas de mensagens em baixo nível. Se um processo deseja se comunicar com outro, deve-se primeiramente construir uma mensagem e, então, realizar a chamada para envio

através da rede. Para isso, os processos devem "falar o mesmo idioma", para que a mensagem seja compreendida. Isto é, devem ser respeitados os protocolos de comunicação. Estes protocolos nada mais são que um conjunto de regras definindo como está disposta a informação naquela mensagem, que tipo de conteúdo está sendo enviado. O protocolo dita como será construída e como será lida a mensagem. Tratam-se de descrições em camadas, em diferentes níveis. Os protocolos de comunicação não serão discutidos a fundo nesta dissertação, que terá um foco maior nos tipos de comunicação.

Quanto ao tipo, é baseado em algumas definições. Existem tipos de comunicação persistente, onde a mensagem submetida para transmissão fica armazenada até que ocorra a entrega efetiva ao destinatário. Por outro lado, tem-se a transiente, onde a mensagem fica armazenada somente enquanto os processos de envio e recebimento estiverem em execução.

Além destas definições, a comunicação pode também ser síncrona ou assíncrona. Na assíncrona, o processo que envia a mensagem segue sua rotina, tendo sua mensagem armazenada imediatamente, por tempo limitado, após a submissão. Já na comunicação síncrona, o processo que envia a mensagem fica temporariamente bloqueado até que haja confirmação do envio, ou alguma forma de liberação.

Além disso, pode-se classificar a comunicação como discreta ou contínua (*stream*). Na contínua, múltiplas mensagens são enviadas de forma sequencial, sendo estas relacionadas por ordem ou marcação temporal, por *stamps* por exemplo. Na discreta, cada mensagem carrega em si um pacote completo de informação.

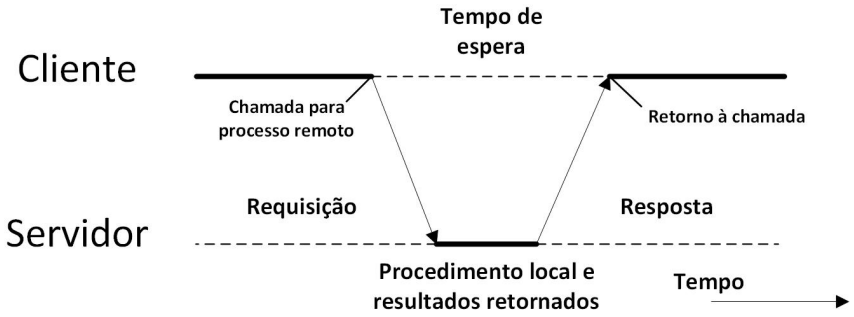
A partir destas definições, é possível estabelecer alguns tipos de comunicação, como *Remote Procedure Call* (RPC), ou Chamada de Procedimento Remoto, *Message-Oriented Communication*, traduzido livremente como Comunicação Orientada a Mensagens, Orientada a *Stream*, para *Stream-Oriented Communication*, além de *Multicast Communication*, todas bem discutidas em [25] e [26]. Nesta dissertação, porém, o foco será mantido na abordagem RPC.

Como já bem enfatizado, sistemas distribuídos se baseiam em troca de mensagens. Porém, este processo de envio e recebimento era muito visível para o programador. A solução para este problema veio com o RPC, onde é permitido a

um programa chamar procedimentos em outras máquinas.

A ideia principal é fazer a programação em sistemas paralelos parecer mais com a realizada nos convencionais, atingindo um alto nível de transparência. Em RPC, os procedimentos em maquinas diferentes podem ser chamados de forma similar a realizada localmente, sendo encobertas todas as etapas de troca de mensagem. A Figura 12 ilustra como RPC realiza chamadas por procedimentos em diferentes máquinas, do ponto de vista do modelo cliente/servidor.

Figura 12 – RPC em chamada remota para cliente e servidor.



Fonte: Elaborada pelo autor.

O processo passa a se portar como sistemas convencionais onde passam-se parâmetros ao procedimento e há a espera de um retorno ou resultado. Dessa forma, em vez de localmente solicitar ao sistema operacional os dados retornados, empacota os parâmetros em uma mensagem e solicita para ser enviada ao servidor. O cliente então fica bloqueado enquanto aguarda a resposta. Quando a mensagem chega ao servidor, seu sistema operacional próprio a transfere para um programa que transforma solicitações vindas pela rede em chamadas de procedimento. Neste código ocorre então o desempacotamento da mensagem e, então, é chamado o procedimento de forma padrão.

2.5.2.1 *Modelo Publicador-Subscritor*

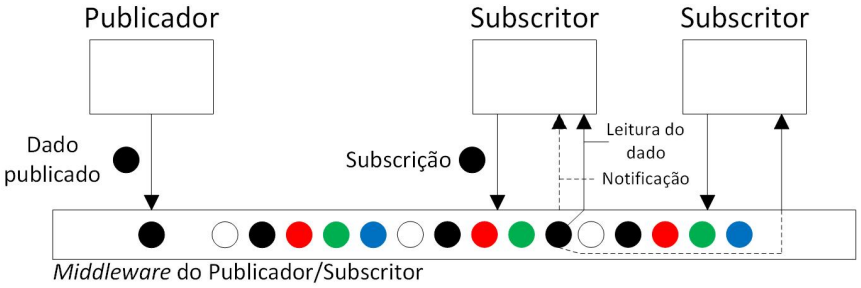
Até o momento, por uma questão didática, os conceitos foram apresentados pressupondo o modelo cliente-servidor. Entretanto, faz-se necessário também comentar a respeito do modelo publicador-subscritor. Trata-se de um modelo

baseado em eventos, processos podem subscrever mensagens de um conteúdo específico, os subscritores, enquanto que outros processos são responsáveis por publicar tais mensagens, os publicadores.

A dificuldade desse tipo de sistema está em correlacionar a subscrição ao evento publicado de forma correta, fazendo com que a maioria desses sistemas tenha a necessidade de manter ambos os processos ativos ao mesmo tempo.

Então, um conjunto de dados é dito publicado a partir do momento em que um processo o torna disponível para ser lido por outros processos. Para acontecer a leitura, uma chamada de subscrição deve ser passada ao *middleware* (ou interface de comunicação entre processos), contendo a descrição do tipo de dados que o subscritor deseja ter acesso. Dessa forma, a subscrição deve ser pareada, correspondida, ao pacote que deseja. Este processo é ilustrado na Figura 13.

Figura 13 – Troca de dados no modelo publicador-subscritor.



Fonte: Elaborada pelo autor.

Nessa situação, quando há a correspondência, ou o conjunto de dados é enviado ao subscritor ou é enviada uma notificação de quando os dados poderão ser lidos, indicado pelas linhas tracejadas na Figura 13. Para a primeira situação, o *middleware* não necessita de armazenamento para os dados. No segundo caso, entretanto, será necessário o armazenamento e operações adicionais de gerenciamento de dados. Descrições mais aprofundadas sobre o funcionamento deste modelo podem ser encontradas em [25] e [26].

2.5.3 **Paradigma *fog***

Para entender o conceito do paradigma *fog*, termo criado pela empresa CISCO, deve-se, primeiramente, definir computação *edge* [28][29]. Este conceito, que traduzido livremente significa aresta, extremidade ou periferia, trata-se de um novo modelo para analise e atuação em dados de IoT. Em *edge*, o processamento e armazenamento de dados são realizados na periferia da rede, mais próximo ao local de aquisição, o que dá origem ao nome.

Já a computação *fog* se refere a um padrão que permite associar múltiplas estruturas do tipo *edge*, o que garante escalabilidade ao sistema, ou seja, a possibilidade de adição de componentes e novos subsistemas *edge*. Suas principais características e funções são a análise e processamento de dados nos elementos *edge*, sendo enviados posteriormente para armazenamento e processamentos e análises futuros, e atuação em curtos espaços de tempo, normalmente em milissegundos.

2.5.4 **ROS**

Trata-se de um *framework*, desenvolvido na universidade de *Stanford*, que surgiu com o intuito de padronizar a pesquisa e distribuir ferramentas através de um sistema robusto e confiável de desenvolvimento. Em sua página oficial, é dito que o ROS foi construído do zero para encorajar colaboradores no desenvolvimento de softwares de robótica [30]. Por isso, atualmente, é controlado pela *Open-Source Robotic Foundation* (OSRF).

ROS consiste em um conjunto de ferramentas, bibliotecas e convenções utilizadas para desenvolvimento de aplicações em robótica. No artigo de apresentação do sistema [31], são resumidos seus principais objetivos:

- Comunicação *peer-to-peer*, ou ponto-a-ponto;
- Multi-linguagem;
- Baseado em *tools*, ou ferramentas;
- "Fino" (*Thin*);
- *Open-Source*.

A classe de comunicação P2P foi escolhida para satisfazer a comunicação tanto *onboard* como *offboard*, e entre os sistemas [31]. No ROS, essa comunicação pode ser realizada de duas formas: modelo cliente-servidor ou publicador-subscritor [32][33].

É dito Multi-linguagem, visto que pode ser desenvolvido em C++, Python, Octave e LISP, tendo cada uma sua implementação nativa [31]. A especificação do ROS está na camada de troca de mensagens, onde a comunicação P2P citada anteriormente acontece em XML-RPC. Trata-se de uma aplicação de RPC na linguagem estruturada XML, já mencionado anteriormente como sendo o protocolo que permite a comunicação entre procedimentos que ocorrem em diferentes aplicações ou máquinas, podendo até mesmo possuir sistemas operacionais diferentes como entre Mac e Windows [26] [34]. O ROS é implementado em Linux, porém está em desenvolvimento para outros sistemas operacionais. Mais sobre a comunicação em ROS será discutido no Capítulo 3.

Diz-se ser baseado em ferramentas devido à tentativa de controlar a complexidade através da utilização de várias pequenas ferramentas, cada uma delas resolvendo sua própria tarefa. É dito "fino" devido a sua fácil implementação e construção através do CMake, deixando toda a complexidade para bibliotecas e criando executáveis [9] [35]. Finalmente, é dito *Open-Source* visto que todo código fonte do ROS é de domínio público, facilitando contribuições e *debugs*.

Alguns conceitos devem ser definidos para melhor descrever a comunicação em ROS. Nesse *framework*, é possível criar processos de diversos tamanhos, e mesmo processos que contenham sub-processos. A esses blocos de programação atribui-se o nome de nó. O ROS se assemelha ao *framework* de *Object Request Broker* (ORB), onde este *broker* é responsável por registrar os nós e fazendo-os "conhecer" uns aos outros, gerenciando a comunicação entre eles, fazendo com que haja conexão e que as mensagens sejam corretamente enviadas e recebidas [26][25].

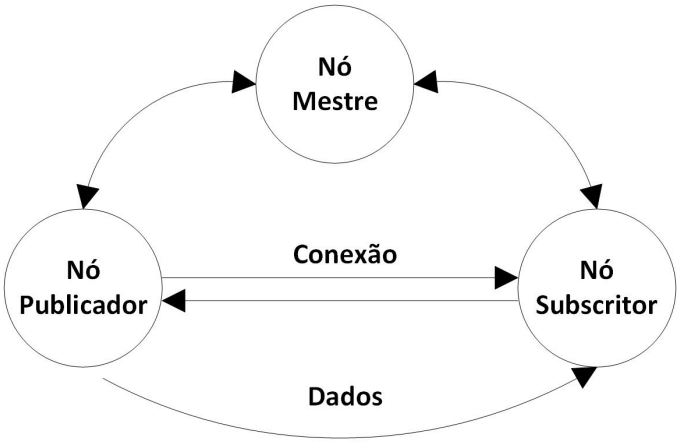
Dessa forma, existe em ROS um nó chamado mestre, responsável por atribuir nomes e registros aos nós. Além disso, atribui as conexões necessárias para comunicação seguindo o modelo desejado, seja cliente-servidor ou publicador-subscritor. Dessa forma, os nós conseguem se localizar para executar suas tarefas,

comunicando-se, como já citado, de forma P2P. Dessa forma, pode-se dizer que o nó mestra age como um *broker* para o ROS [36].

A troca de mensagens entre os nós acontece entre canais de comunicação denominados tópicos. Cada tópico é criado para transmitir um tipo específico de mensagem, o que facilita a organização da comunicação e evita o problema da correspondência no modelo publicador-subscritor descrito anteriormente em 2.5.2.1. As mensagens, propriamente ditas, podem ser estruturadas nos formatos mais convencionais de dados, como inteiros (*int*) ou ponto flutuante (*float*), etc; ou estruturas mais complexas como as nuvens de pontos utilizadas neste projeto (*PointCloud2*).

A partir das definições, é possível ilustrar como se realiza a comunicação entre nós em ROS, visto na Figura 14, onde os círculos representam os nós e as flechas, os tópicos. O nó mestre registra os nós existentes e estabelece a comunicação entre eles, fazendo com que o publicador possa publicar os dados no tópico a ser lido pelo subscritor.

Figura 14 – Troca de dados no modelo publicador-subscritor em ROS.



Fonte: Elaborada pelo autor.

2.6 TRABALHOS RELACIONADOS

Nesta seção vamos apresentar os métodos de reconstrução 3D que se caracterizam como trabalhos relacionados à presente proposta de dissertação.

Existem diferentes métodos para a realização de uma reconstrução tridimensional, porém seguindo princípios semelhantes, principalmente estimando a distância do sensor ao objeto de forma a obter sua posição no espaço, gerando uma nuvem de pontos. Além disso, deve-se estimar a localização dos *frames* subsequentes, de forma a enriquecer a nuvem inicial. Dessa forma, é possível classificar o método de reconstrução baseado na forma como obtém a posição tridimensional dos pontos, bem como na estratégia adotada para obtenção de todas as poses das câmeras relativas à nuvem principal. As subseções a seguir descrevem os métodos observados na literatura atualmente e as ferramentas necessárias para sua execução.

2.6.1 **Odometria Visual por Visão Estéreo**

Como discutido anteriormente, a visão estéreo consiste na utilização de duas câmeras com objetivo de emular a visão humana. A partir deste artifício, é possível determinar a posição relativa dos pontos no espaço com relação às câmeras, através da correspondência entre pontos característicos contidos nas imagens.

Dessa forma, a visão estéreo pode ser utilizada em diversas aplicações, como detecção de obstáculos na navegação de robôs móveis, ou detecção de faixas em rodovias. Com as entradas visuais obtidas, é possível também estimar o movimento dos sensores, processo conhecido como Odometria Visual. Nister *et al.*[37] propõe obter o movimento de um veículo terrestre por meio de sequência de vídeo monocular e estéreo, utilizando para isto um computador Pentium III com capacidade de processamento 1 GHz. Suas conclusões mostram que os resultados obtidos com a visão estéreo são mais confiáveis e acurados que os vindos do sistema monocular.

Trabalhos como este possibilitaram a aplicação da odometria visual em sistemas mais complexos, como o apresentado por Rawashdeh e Aladem [38], onde ocorre a aplicação à micro veículos aéreos (MAV). Sua intenção é atender a problemas como navegação em ambientes fechados, onde não é possível a utilização do GPS ou suas medidas não são confiáveis. Além disso, também utiliza o sistema visual para detectar obstáculos, como apontado anteriormente. Com um foco claro na aplicação em tempo real, este trabalho se propôs a reduzir os requisitos computacionais, visto que o veículo aéreo tem limitação para peso e baterias. Nota-se, então, a utilização de *features* do tipo ORB, para encontrar as correspondências,

apresentadas em [39] como uma alternativa eficiente e bastante veloz aos presentes na literatura. Sua aplicação foi realizada tanto em um computador Intel Core i7 com capacidade 2.4 GHz, como embarcado em uma NVIDIA Jetson TX1, mostrando redução significativa em tempo de processamento entre ambos, porém ainda aplicável para operações em tempo real. Destaca-se também que não foi utilizado nenhum tipo de aceleração por GPU.

Além destas aplicações, tem-se outras diretamente ligadas a reconstrução tridimensional como visto em [40], que aplica a visão estéreo para cenas estáticas também sem a utilização de GPU. Neste trabalho, também é utilizado o filtro de Kalman para refinar a velocidade estimada, obtida através da divisão entre os parâmetros de transformação e o tempo entre os *frames*. Outro fator interessante a se mencionar é a utilização do método de obtenção de mapas de disparidades densos chamado ELAS(*Efficient LArge-scale Stereo*), utilizado para realização de correspondências de forma rápida para imagens de alta resolução [41].

2.6.2 **Estrutura a Partir do Movimento**

Dentre os métodos para reconstrução, é importante destacar o *Struct from Motion* (SfM) [9]. Este procedimento estima a profundidade e, com isso, o modelo a partir de uma série de imagens obtidas de diversos pontos de vista, como a forma incremental realizada por Schönberger *et al.* [11]. Seu método é dito incremental visto que processa de forma sequencial o *pipeline* com reconstrução iterativa definido pelos autores. É importante destacar que em [22] o SfM é tratado como um caso muito semelhante ao obtido com visão estéreo, sendo porém, utilizadas imagens consecutivas de diferentes pontos de vista e em tempos também diferentes.

O primeiro passo consiste na definição de imagens de entrada e na extração de parâmetros. Devido a invariância a escala, localização e orientação, são utilizados descritores SIFT, contendo as características locais do ponto em análise [42]. Após a extração de parâmetros das imagens, são formados pares a partir das correspondências entre pontos chave. Além disso, é realizada a verificação geométrica das correspondências através da restrição da matriz fundamental.

Em sequência, dá-se início a reconstrução incremental, consistindo em

inicialização, registro da imagem, triangulação e *bundle adjustment*. A inicialização consiste na escolha do melhor par de imagens possível a ser utilizado como referência. Essa escolha deve ser realizada de forma criteriosa, visto que uma inicialização ruim pode arruinar o processo de reconstrução [11].

Já o registro das nuvens consiste no cálculo de novas poses de câmeras que capturaram novos *frames* através do método RANSAC [20]. Para isso, resolve-se um problema de perspectiva e ponto, ou PnP, usando novas correspondências com imagens já registradas. Com a adição de novas poses, parte-se para a triangulação, ou seja, adição dos novos pontos ao modelo global. *Bundle Adjustment* é o refinamento dado ao registro e à triangulação, a partir da minimização do erro de re-projeção, apresentado pela Equação 2.29 a seguir.

$$E = \sum_{j} \rho_j(||\pi(\mathbf{P}_C, \mathbf{X}_k) - x_j||_2^2) \tag{2.29}$$

Onde π função de re-projeção que mapeia pontos do modelo tridimensional no espaço das imagens. Nesta equação, ρ é a penalidade dada a um possível *outlier*, $\mathbf{P}_C$ é a pose da câmera, $\mathbf{X}_k$ o ponto do *frame* anterior a ser reprojetado e x_j é o ponto no *frame* atual.

Novos trabalhos ainda são recorrentes neste tema, como [43]. Neste caso, o *pipeline* do SfM é aplicado de forma modificada, utilizando uma câmera omnidirecional em vez de uma *pinhole*, alegando que o aumento do campo de visão traz robustez e precisão para processos de estimativa de posições. A utilização de uma câmera com estas características implica em um pré-processamento, convertendo as imagens adquiridas em mapas de cubos. O resultado final do pré-processamento são duas imagens cuja transformação relativa entre elas é puramente rotacional. O tipo de *feature* escolhida para aplicação do SfM foi ASIFT [44], uma variação do SIFT de Lowe [42] desenvolvida para ser invariante a transformações do tipo *affine*, obtendo correspondências mais robustas mesmo com capturas em ângulos variados.

2.6.3 SLAM Denso e Suas Variações

Apesar de métodos como SfM apresentarem resultados estimulantes [11] [9] [45] [46], essas abordagens não foram desenvolvidas para utilização em tempo

real. Na verdade, suas implementações são bastante custosas do ponto de vista do tempo de processamento quando realizando reconstruções densas, mesmo quando há a utilização de GPU e *clusters* como em [27].

Uma forma alternativa são os métodos de SLAM que são desenvolvidos com foco em implementações em tempo real. A princípio, as técnicas das mais variadas oferecem diversas funcionalidades, sendo, porém, baseadas numa mesma sequência de passos [9] [47]:

- Alinhamento espacial de imagens consecutivas;
- Detecção de fechamento de Loop local;
- Alinhamento de modelo global.

MonoSLAM [48] foi o primeiro algoritmo a realizar a localização e o mapeamento em tempo real, reconstruindo um modelo esparso, utilizando uma câmera. DTAM [10], por sua vez, também de forma monocular, realizou pela primeira vez uma reconstrução densa em tempo real. Trata-se de métodos que se utilizam da sobreposição entre imagens e minimizam funções de custo baseadas na Equação 2.29 alteradas ou abordadas de forma diferente em sua otimização.

Os avanços vindos em seguida utilizam um sensor do tipo RGB-D, que possuí esse nome visto que captura imagens RGB e realiza medições de profundidade (*Depth*), através de um emissor de laser e um receptor de infravermelho. São baseados em dois tipos de tecnologia, Sensor de Luz Estruturada, como Microsoft Kinect, Orbbec ASTRA e ASUS Xtion PRO Live, e tempo de voo para versões mais modernas de câmeras RGB-D como o Microsoft Kinect II [49][12][50].

Baseado neste tipo de sensores, Newcombe *et al.*[12] propõe o primeiro sistema de reconstrução densa e em tempo real utilizando um sensor Kinect. Seu *pipeline* é dividido em um primeiro passo para computar as leituras do sensor, gerando um mapa de vértices e normais a partir do mapa de profundidade. Na sequência, é realizada a estimativa de pose para o *frame* atual através do ICP (*Iterative Closest Point*)[51], sendo este em seguida integrado ao modelo global usando TSDF (*Truncated Signed Distance Function*)[52]. O último passo é a

utilização de TSDF da superfície gerada, produzindo um *frame* estimado, a ser usado como comparativo para a pose do próximo *frame*, dando sequência ao processo.

Para a etapa de estimativa de pose, são utilizadas as informações de vértices para o *frame* atual comparado ao anterior. Nesse cálculo, utiliza-se a mesma estratégia de minimização vista na Equação 2.29, ou seja, uma minimização de energia ou erro para obtenção da transformação de um *frame* para outro. A Equação 2.30 detalha o processo.

$$E(T_{g,k}) = \sum_{\mathbf{u}} (||(T_{g,k}\dot{V}_k(\mathbf{u}) - \hat{V}^g_{k-1}(\hat{\mathbf{u}})^T)\hat{N}^g_{k-1}(\hat{\mathbf{u}})||_2) \tag{2.30}$$

Dessa forma, o alinhamento ocorre entre as medidas atuais do sensor $(\mathbf{V}_k, \mathbf{N}_k)$, vértices e normais, contra o modelo previsto no tempo anterior, $(\hat{\mathbf{V}}_{k-1}, \hat{\mathbf{N}}_{k-1})$. O índice "g" indica a representação dos vértices no *frame* global.

2.6.4 **Elastic Fusion e Bundle Fusion**

Dentre os métodos de reconstrução presentes na literatura, o Elastic Fusion, desenvolvido por [47] *et al.*, foi escolhido como base de comparação devido ao seu foco principal estar presente no mapa e operação em tempo real. Além disso, [9] afirma que este método é robusto e acurado, tanto para localização quanto para estimativa da superfície, quando comparado a outros métodos de SLAM no estado-da-arte. Para isso, faz grande uso do processamento utilizando CUDA, para localização e cálculo de poses, bem como OpenGL para previsão e gerenciamento do mapa. O objetivo é ser capaz de gerar mapas densos compreensivos mesmo em ambientes complexos e com trajetórias irregulares.

O Elastic Fusion cria um mapa utilizando como unidade Elementos de Superfície, também chamadas de *Surfels*. Estes elementos carregam informações de posição tridimensional, normais referentes aos elementos, cores RGB, peso, raio do elemento e marcações de tempo para inicialização e última atualização [53] [54].

Já a estimativa de poses é realizada também por meio da minimização de

uma função de custos, como apresentado na Equação 2.31 a seguir.

$$E_{track} = E_{icp} + \omega_{rgb} E_{rgb} \tag{2.31}$$

Esta equação é composta por uma parcela dita geométrica, de forma análoga a Equação 2.30, como a vista em [12], com a parcela fotométrica, similar a vista em [10]. A variável ω_{rgb} tem como função atribuir peso a parcela fotométrica. São ditas diferentes pois visam minimizar em relação aos parâmetros de movimento ξ. Seguem as Equações 2.32 e 2.33, para os mapas de profundidade e cores respectivamente, que compõe a função de custo:

$$E_{icp} = \sum_{k} ((\mathbf{v}^k - exp(\hat{\xi})\mathbf{T}\mathbf{v}_t^k) \cdot \mathbf{n}^k)^2 \tag{2.32}$$

onde, $\mathbf{v}_t^k$ é a re-projeção do k-ésimo vértice no *frame* de profundidade atual, representado por t, $\mathbf{v}^k$ e $\mathbf{n}^k$ são os vértices e normais para o *frame* de coordenadas no tempo anterior, portanto, $t-1$; $\mathbf{T}$ é a estimativa atual da transformação do *frame* anterior para o atual; já $exp(\hat{\xi})$, é a matriz exponencial que mapeia um membro da álgebra de Lie $\mathfrak{se}_3$ em um membro do grupo de Lie $\mathbb{SE}_3$.

$$E_{rgb} = \sum_{\mathbf{u}\in\Omega} (I(\mathbf{u}, C_t^l) - I(\pi(\mathbf{K}exp(\hat{\xi})\mathbf{T}p(\mathbf{u}, D_t^l)), \hat{C}_{t-1}^a))^2 \tag{2.33}$$

Sendo $\mathbf{u}$ um pixel pertencente ao domínio da imagem $\Omega \subset \mathbb{N}^2$. A intensidade do pixel em uma imagem C, com cores $\mathbf{c}(\mathbf{u}) = [c_1, c_2, c_3]^T$, é definida como $I(\mathbf{u}, C) = (c_1 + c_2 + c_3)/3$. Portanto, C_t^l representa a imagem atual e $\hat{C}_{t-1}^a$ representa o modelo ativo de cores para o *frame* anterior. Informações mais detalhadas podem ser encontradas em [47][13][21].

A forma que é utilizada para atingir a alta consistência e qualidade no modelo gerado vem da deformação não-rígida aplicada a todas as *Surfels* em cada fechamento de *loop*. O fechamento local acontece quando ocorre um registro bem sucedido, já o global, quando ocorre uma correspondência entre o *frame* atual e uma base de dados que é criada no início do processamento.

Para essa abordagem é criado um grafo de deformação, composto por nós e conexões indicando as regiões do modelo a serem deformadas. A cada *frame* adquirido, um novo grafo é criado. Cada nó possui uma marcação de tempo, além de uma posição tridimensional, um conjunto de vizinhos próximos e uma transformação, contendo rotação e translação.

Na inicialização dos nós, estes recebem a mesma marcação de tempo de inicialização de seu *surfel* correspondente, bem como a mesma posição. Os nós são amostrados uniformemente dentro do conjunto de surfels, de forma à quantidade de nós ser muito menor que a de surfels, porém espelhando sua densidade espacial. Para cada *frame*, os nós são ordenados sequencialmente, fazendo com que um conjunto de vizinhos seja $N(G^n) = \{G_{t_0}^{n\pm1}, G_{t_0}^{n\pm2}, ..., G_{t_0}^{n\pm k}\}$, onde G^n é o n-ésimo nó e k é o número de vizinhos definidos, todos relacionados ao mesmo tempo t_0. Este método previne a interferência de áreas descorrelacionadas. Mais detalhes na deformação podem ser observados em [47][13][55].

Já o proposto em [56], segue uma linha parecida ao método anterior, utilizando, porém, correspondências de parâmetros em nuvem esparsa para o alinhamento das poses, além das características densas geométricas e fotométricas. Em [9], é dito que o *Bundle Fusion* propõe melhorar problemas de localização encontrados em métodos anteriores, o que é alcançado pela independência da coerência temporal vista anteriormente, fazendo com que haja um robusto método de relocalização, mesmo quando há oclusão ou movimentações repentinas do sensor.

Porém, [9] também atesta que, apesar de possuir desempenho superior aos outros algoritmos, principalmente no que se refere a falhas de localização, este método ainda possui problemas ao estimar poses em momentos de movimentação rápida e repentina do sensor, gerando problemas na acumulação.

2.6.5 **Sistemas Distribuídos para Reconstrução 3D**

Em [27], é observada a utilização de sistemas distribuídos em *clusteres* e GPU, mas também na chamada Computação Voluntária, melhor definida em [57] e [58]. Nesse tipo de sistema, voluntários fornecem recursos para o projeto.

Coro *et al.*[27], propõe uma reconstrução baseada em SfM, e utiliza 15

computadores com sistema Ubuntu 16.04, somando 16 núcleos virtuais com 32 GB de memória RAM e 100 GB de disco, e conta com mais uma máquina com 64 GB de RAM e uma GPU com capacidade 12GB. Além disso, para a etapa colaborativa de computação voluntária citada acima, foi utilizado o método *e-infrastructure*, um sistema desenvolvido na Europa que permite aos usuários colaborarem com pesquisas, trocas de informações e experimentos. Este sistema foi utilizado para geração de *datasets*, ou seja, os usuários podem compartilhar imagens e pastas criando um repositório para diversos objetos a serem reconstruídos.

2.6.6 **Referências Adicionais**

Nesta subseção, chama-se atenção para alguns trabalhos não encaixados nas subseções anteriores, porém com relevância para esta dissertação. Ibragimov e Afanasyev [59] realizam uma comparação entre o desempenho de sensores e métodos de SLAM baseados em ROS. Os autores comparam quatro sensores: Um sensor de distância (*rangefinder*) laser LIDAR, uma câmera RGB Basler de alta resolução, uma câmera estéreo ZED e um sensor RGB-D Kinect versão 2.0.

Para cada sensor, foram avaliados métodos de SLAM, como Hector SLAM para o sensor laser REF, ORB-SLAM e DPPTAM para SLAM monocular, SLAM estéreo para a câmera ZED, e RTAB-Map SLAM para o Kinect. Com a exceção do sensor a laser, a câmera estéreo ZED obteve o menor erro na estimação da trajetória, sendo seguida pelo ORB-SLAM. Este trabalho aponta a qualidade da odometria visual obtida pela câmera estéreo ZED. Além disso, aponta um método de SLAM em tempo real baseado em *features* do tipo ORB [39], o ORB-SLAM monocular [60], bem como sua versão ORB-SLAM2 [61], com suporte para diferentes tipos de câmera como estéreo e RGB-D, apresentando resultados bastante promissores.

Tem-se, também, o trabalho de Yilmaz e Karakus [62], onde é proposta a utilização de uma composição de sensores. Seu sistema contém um sensor Kinect versão 1.0 e uma câmera estéreo Bumblebee XB3. Seu método de fusão é uma derivação do Kinect Fusion [12], porém com o uso do mapa de profundidade obtido com a câmera estéreo, através do método ELAS [41].

2.7 CONSIDERAÇÕES DO CAPÍTULO

Neste capítulo foram discutidos primeiramente os conceitos fundamentais para entendimento do trabalho. É preciso enfatizar a importância de tópicos como as transformações de corpo rígido, fundamentais para entendimento e aplicação da união das nuvens de pontos na etapa de acumulação a ser discutida mais a frente. Além disso, o conceito de calibração também é de vital importância, como pode ser notado pela utilização dos parâmetros intrínsecos e extrínsecos em diversos momentos posteriores. O estudo acerca da visão estéreo também será utilizado, visto que é através dele que é obtida a odometria para o sistema. Com relação aos sistemas distribuídos, seu estudo permitiu ver no ROS uma ferramenta excelente para a implementação, fornecendo diversas bibliotecas de forma nativa, como o OpenCV e PCL, além de possibilitar um dos principais passos para o funcionamento do sistema, a troca de mensagens.

Quanto aos artigos revisados, chamamos atenção primeiramente ao de Ibragimov e Afanasyev [59], que apontou a odometria da câmera ZED como uma boa alternativa, obtendo resultados próximos ao sensor LIDAR superiores ao ORBS-LAM de Mur *et al.*[60]. Além disso, os autores utilizam como *ground truth*, medidas marcadas em fita como trajetória de um robô móvel, o que será adotado por esta dissertação em um dos cenários de avaliação do sistema. Apontamos também a utilização de uma disposição semelhante a que é aplicada nesta dissertação, apresentada por Yilmaz e Karakus em [62], porém de forma simplificada. Escolhemos para comparação qualitativa os resultados de reconstrução em forma de nuvem de pontos obtida pelo Elastic Fusion, desenvolvido por Whelan *et al.* e disponibilizada em código aberto em [63], devido a seu foco no modelo, e o Colmap [64] de Schonberger *et al.* [11] para métodos de SfM. Também para avaliação do sistema, desta vez com mais graus de liberdade, foi utilizado como *ground truth* a trajetória de um manipulador robótico, de forma similar a utilizada por Diaz *et al.* [65] ao analisar sensores RGB-D de primeira e segunda geração.

3 PROPOSTA

Como observado e já comentado anteriormente, existem diversas formas de realizar a reconstrução tridimensional de objetos e cenários, utilizando para isso diferentes tipos de sensores. Dentre estas formas, uma característica comumente encontrada nestes sistemas é o custo computacional para sua execução, sendo necessária muitas vezes a utilização de GPU. Além disso, os métodos estudados apresentam dificuldade em determinadas trajetórias, evidenciando problemas com a odometria visual utilizada, o que causa um pior resultado de nuvem de pontos. Assim sendo, esta dissertação apresenta uma proposta diferenciada, através da utilização das melhores características de cada sensor, a câmera estéreo para obtenção de uma odometria visual acurada e o sensor RGB-D para obtenção de nuvens de pontos de maior qualidade. É proposto, também, a separação da etapa de aquisição e acumulação em máquinas diferentes, visando dividir a carga computacional para cada computador, avaliando seu desempenho e a possibilidade de aplicação de forma embarcada. Com essa rede de computadores, é simulado um sistema *fog*, visto que há um processamento das imagens na etapa de aquisição, enviando para registro e armazenamento na etapa de acumulação.

3.1 APRESENTAÇÃO DO PROBLEMA

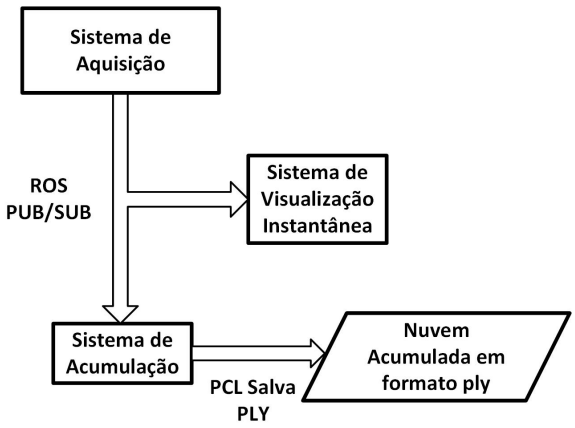
A reconstrução 3D é obtida através da solução de um problema de SLAM, sendo necessário para isso estimar a movimentação do sensor para então acumular os pontos obtidos em um mapa global. Cada *frame* adquirido gera uma nuvem de pontos que deve ser fundida ao modelo final de acordo com a posição rastreada.

Deseja-se realizar uma reconstrução tridimensional utilizando uma câmera Estéreo acoplada a uma câmera RGB-D, conectando-as a um computador, o que configura o sistema de aquisição. Os dados adquiridos são pré-processados e sincronizados por meio da marcação de tempo da mensagem recebida, para serem enviados ao sistema de acumulação.

Toda a comunicação será realizada por meio do modelo Publicador-Subscritor através do *framework* ROS. Dessa forma, a Figura 15 ilustra como é disposto o

sistema distribuído de modo geral. O processo começa com a Aquisição, sendo possível realizar a visualização instantânea do que será passado adiante ao sistema de Acumulação, onde são fundidas todas as nuvens de pontos enviadas pelo bloco anterior. O resultado é finalmente salvo em um formato *.ply*, muito utilizado para manipulação e visualização de nuvens de pontos.

Figura 15 – Modelo geral do sistema.

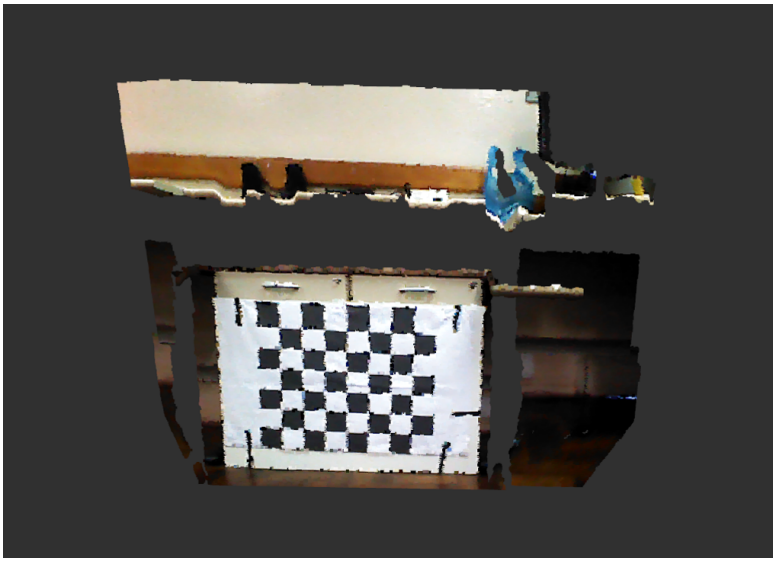


Fonte: Elaborada pelo autor.

3.2 PRÉ-PROCESSAMENTO

A princípio, o pacote astra_camera, além de fornecer a imagem RGB e mapa de profundidade, também disponibiliza a nuvem de pontos colorida já registrada. Porém, em avaliações primárias, foram notados problemas de registro entre o mapa de profundidade e as imagens RGB, como apresentado pela Figura 16. Na figura, é possível observar problemas de alinhamento entre os objetos, evidenciado pela distorção e separação do gaveteiro utilizado como modelo.

Figura 16 – Registro gerado de forma errada pelo pacote Astra.



Fonte: Elaborada pelo autor.

Para solucionar este problema, primeiramente foram calibradas as câmeras RGB e profundidade de forma estéreo, utilizando a *toolbox* nativa do *software* *Matlab*®. A partir desta calibração, foram obtidos os parâmetros intrínsecos das câmeras RGB e de profundidade, a transformação de corpo rígido que as relacionam, suas matrizes de projeção e a matriz Fundamental.

Com estas informações, foi criado um nó ROS que, ao invés de colher os pontos tridimensionais, recebe as imagens RGB e profundidade e publica a nuvem de pontos colorida. Isto é feito, primeiramente, levando os pontos do mapa de profundidade ao 3D. Esta operação é realizada da seguinte forma: um ponto $\mathbf{p} \in \mathbb{R}^3$ é obtido a partir da imagem da câmera de profundidade, através da Equação 3.1 a seguir.

$$\mathbf{p} = \mathbf{K}_d^{-1} \tilde{\mathbf{u}} d(\mathbf{u}) \tag{3.1}$$

onde $\mathbf{K}_d$ é a matriz que contém os parâmetros intrínsecos da câmera de profundidade, $\mathbf{u}$ é a posição do pixel no *frame* de profundidade, o que significa que $d(\mathbf{u}) \in \mathbb{R}$ é a profundidade medida naquele pixel. O $\tilde{\mathbf{u}}$ é a forma homogênea de $\mathbf{u}$.

No passo seguinte, estes pontos são novamente reprojetados ao 2D, porém através da matriz de projeção da câmera RGB, checando sempre a restrição epipolar através da matriz fundamental. O próximo passo é a des-homogenização, possibilitando correlacionar os pontos e atribuir as cores para cada medida de profundidade. O resultado é observado na Figura 17.

Figura 17 – Registro correto gerado pelo nó criado.

Fonte: Elaborada pelo autor.

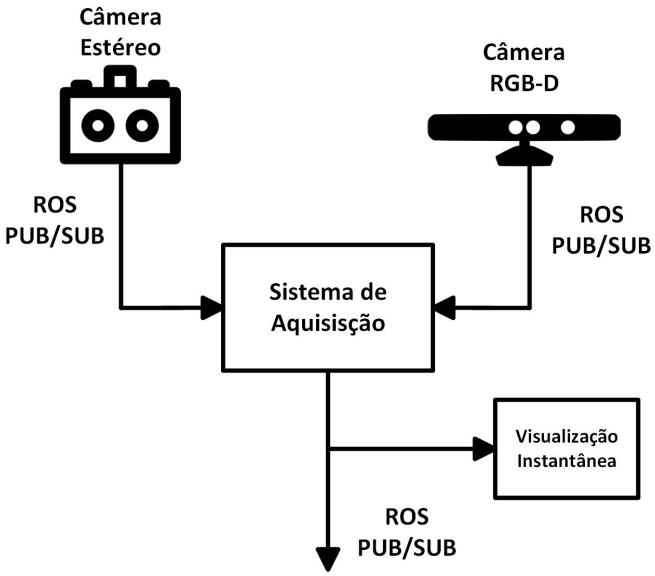
A utilização deste nó garantiu a entrega de uma nuvem de pontos registrada de forma correta. Porém, também trouxe uma redução na taxa de publicação de nuvens de pontos em relação ao realizado pelo pacote da câmera Astra. Antes as nuvens eram publicadas a aproximadamente 30 Hz, com o novo registro temos publicações em torno de 4 Hz.

3.3 SISTEMA DE AQUISIÇÃO

O Sistema de Aquisição possui por objetivo fazer a captura de imagens para constituição da nuvem de pontos e realizar a estimativa do posicionamento do conjunto de sensores. A Figura 18 mostra de forma simples de que é constituído

este sistema. Nela é mostrado que serão utilizadas dois tipos de sensores, uma câmera estéreo e um sensor RGB-D, composto por uma câmera RGB e um sensor de profundidade composto de um emissor e um receptor de infravermelho.

Figura 18 – Constituição do Sistema de Aquisição.

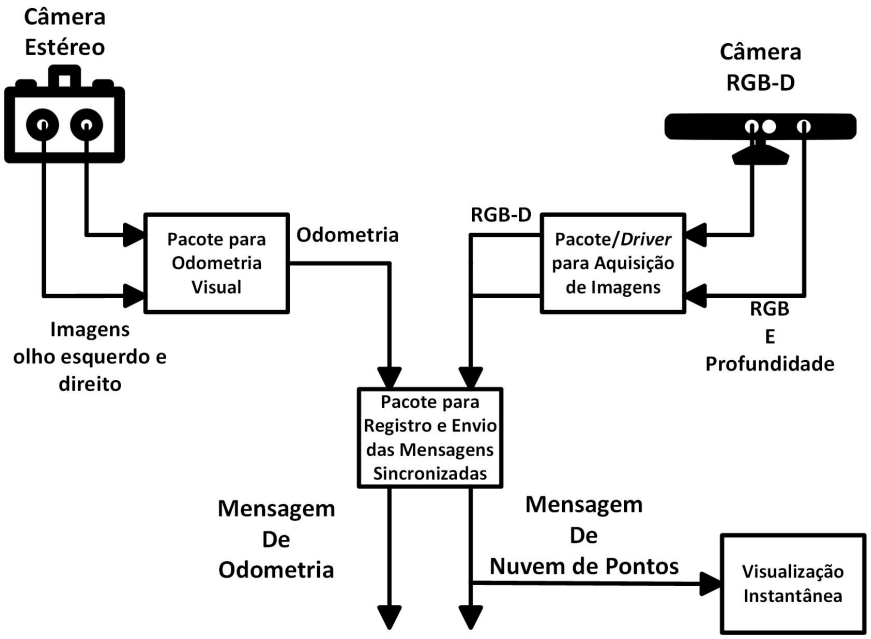


Fonte: Elaborada pelo autor.

Dissecando um pouco mais este sistema, definimos que a câmera estéreo será utilizada para estimação da movimentação do conjunto, através da odometria visual. As imagens geradas pela câmera são subscritas por um pacote ROS responsável por processá-las e passar adiante a informação de odometria da câmera estéreo.

Já a câmera RGB-D é responsável por publicar imagens do tipo RGB e seus respectivos mapas de profundidade. Além disso, foi desenvolvido um nó que, a partir destes dados, os registra e gera a nuvem de pontos colorida a ser acumulada. Este nó também subscreve a odometria publicada pela câmera estéreo com o objetivo de entregar ao sistema de acumulação os dados de nuvem de pontos e odometria de forma sincronizada pela marcação de tempo (*time stamp*) das mensagens. A Figura 19 melhor ilustra o processo.

Figura 19 – Sistema de Aquisição de forma detalhada.



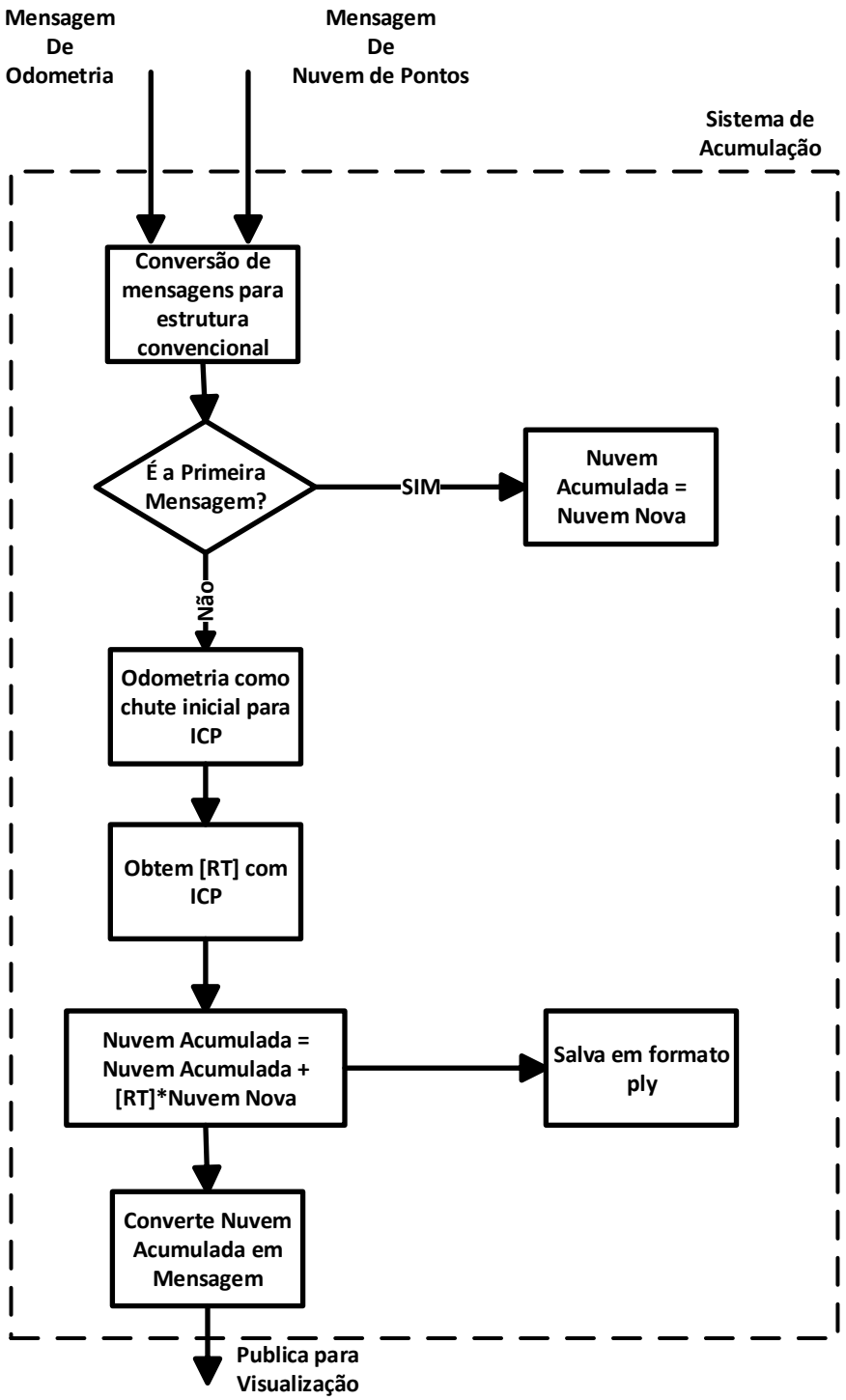
Fonte: Elaborada pelo autor.

Ao final do processo, a odometria e a nuvem de pontos são convertidas em duas mensagens de ROS, cada uma referente ao seu tipo, e publicadas em seus respectivos tópicos. Dessa forma, os dados ficam disponíveis para qualquer nó subscrever suas informações, como é o caso do visualizador instantâneo e do sistema de acumulação.

3.4 SISTEMA DE ACUMULAÇÃO

O Sistema de Acumulação é responsável por receber os dados fornecidos pela aquisição e utilizá-los para a construção de um modelo global. Este sistema segue o fluxograma ilustrado na Figura 20.

Figura 20 – Sistema de Acumulação.



Fonte: Elaborada pelo autor.

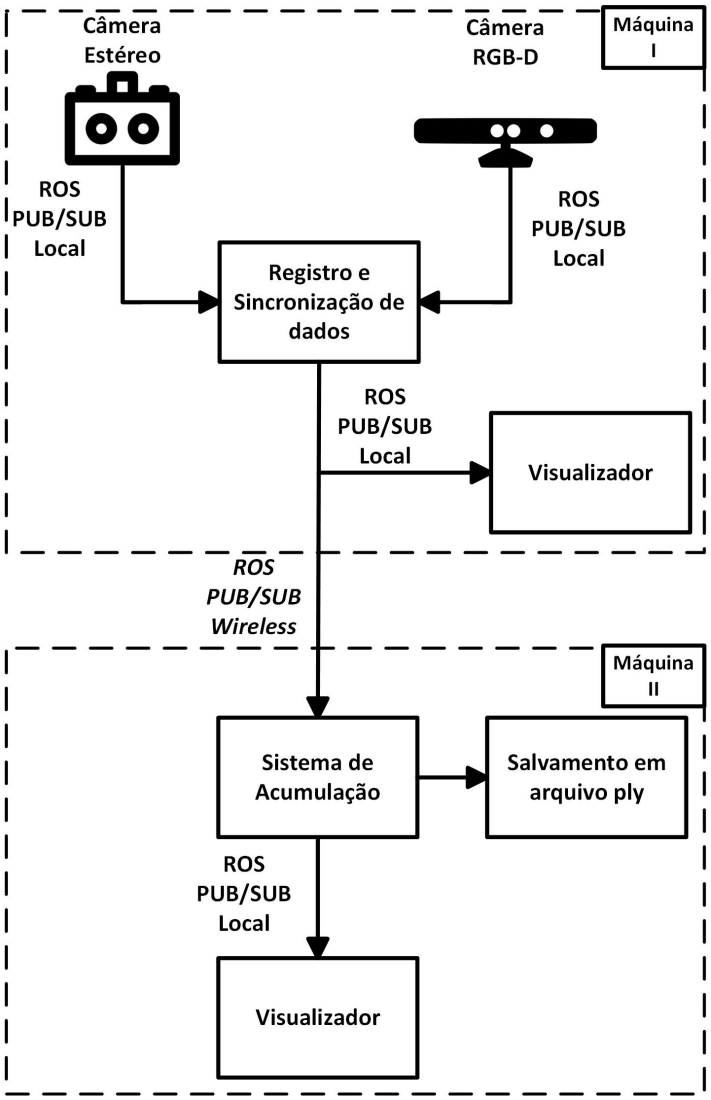
A primeira etapa consiste na conversão das mensagens de ROS em dados de odometria, na forma de *quaternion* para a rotação e vetor para a translação, e nuvem de pontos. A seguir, deve-se checar se a mensagem a chegar é a primeira, ou se a nuvem de pontos acumulada está vazia. Se é positiva a resposta, a primeira nuvem passa a ser referência para as próximas que vierem.

As mensagens que chegam na sequência devem sofrer uma transformação de corpo rígido para serem trazidas ao sistema de coordenadas de referência. Feito isso, a nuvem recém chegada pode ser adicionada ao modelo global.

3.5 ARQUITETURA DISTRIBUÍDA PROPOSTA

O resultado obtido a partir da união dos processos citados anteriormente, é o sistema distribuído ilustrado pela Figura 21. Na maquina I é aplicado o sistema de aquisição enquanto na máquina II temos o sistema de acumulação. As trocas de mensagens são realizadas por meio do modelo publicador-subscritor, disponibilizado pelo *framework* ROS. A aplicação deste sistema permite a utilização do par de sensores e confere um grau de escalabilidade, ou seja, permitindo que futuramente possam ser acrescentados novos componentes ao sistema.

Figura 21 – Sistema de Distribuído.



Fonte: Elaborada pelo autor.

Além disso, o custo computacional fica dividido entre as máquinas do sistema. A máquina de aquisição tem seus resultados enviados à de acumulação por meio do protocolo comunicação TCP/IP, com ou sem fio.

3.6 CONSIDERAÇÕES DO CAPÍTULO

Neste capítulo foi apresentada a proposta para serem atingidos os objetivos propostos nesta dissertação. De forma a dividir o esforço computacional, foram separados dois subsistemas independentes: um de aquisição, contendo as câmeras estéreo e RGB-D, responsável por calcular o movimento dos sensores, gerar a nuvem de pontos instantânea e transmitir estes dados; e outro de acumulação, responsável por receber os dados da aquisição e processá-los, acumulando as nuvens enviadas a cada mensagem e posicionadas de acordo com a odometria refinada com algoritmo ICP. O resultado é um modelo global da cena ou objeto reconstruído, que pode ser visualizado por interface ou manipulado como arquivo do tipo ".ply".

4 AMBIENTES EXPERIMENTAIS E RESULTADOS

Este capítulo tem por objetivo apresentar os testes desenvolvidos para avaliação do sistema proposto, bem como seus resultados. O sistema será avaliado de duas formas: uma delas será utilizando apenas uma máquina, com todo o processamento e comunicação realizados de forma local, porém seguindo o arranjo citado anteriormente. A outra será o arranjo de duas máquinas ilustrado pelos fluxogramas do capítulo anteriores. Essa possibilidade vem devido a utilização em ROS, fazendo com que todo o processo possa ser realizado uma ou múltiplas máquinas.

Além disso, alguns pré-processamentos precisaram ser realizados, melhor discutidos a frente. Toda a programação neste projeto foi realizada na linguagem C++, a exceção da calibração e controle do robô.

Para ambas as avaliações, são utilizados os seguintes materiais:

- Câmera StereoLabs ZED, com aquisição em 30 Hz e resolução de 640 x 480;
- Câmera Orbbec Astra, com aquisição em 30 Hz e resolução de 640 x 480;
- Computador Core i7 – 7700 CPU 3.60 GHz, 16 GB RAM, Sistema Operacional Ubuntu 16.04, Placa de Vídeo NVIDIA GeForce GTX 1050 Ti, ROS Kinetic;
- Computador Core i7 – 6820 CPU 2.70 GHz, 16 GB RAM, Sistema Operacional Ubuntu 16.04, Placa de Vídeo NVIDIA GeForce GTX 1070, ROS Kinetic;

Além disso, são utilizados alguns pacotes de ROS distribuídos de forma aberta. São estes:

- Pacote zed_ros_wrapper [66], responsável por realizar a interface com a câmera ZED e fornecer a odometria visual utilizada;
- Pacote astra_camera [67], realiza a interface com a câmera Astra;
- Pacote openni2_launch [68], pré-requisito para utilização do pacote astra citado no item anterior.

O arranjo de sensores utilizado é observado na Figura 22, chamando-se atenção que esta disposição teve de ser fixada para se manterem os parâmetros de calibração. Esta disposição é similar a encontrada em [62].

Figura 22 – Conjunto de sensores na disposição utilizada.

Fonte: Elaborada pelo autor.

Para a avaliação do sistema proposto, foram elaborados 5 cenários de testes. O cenário 0 (zero) apresenta os primeiros testes realizados de forma livre, avaliando possíveis problemas no sistema quanto ao alinhamento das nuvens, e permitindo que fossem realizadas alterações para os testes seguintes. No cenário 1, foi realizada o primeiro experimento para testes de odometria obtida através de visão estéreo, com o sistema implementado em uma única máquina. Como forma de comparação, foi utilizado um braço robótico como *ground truth*, verificando se a trajetória imposta ao robô era razoavelmente seguida pela odometria. Nesta avaliação também são apresentadas as nuvens de pontos acumuladas somente através da odometria visual.

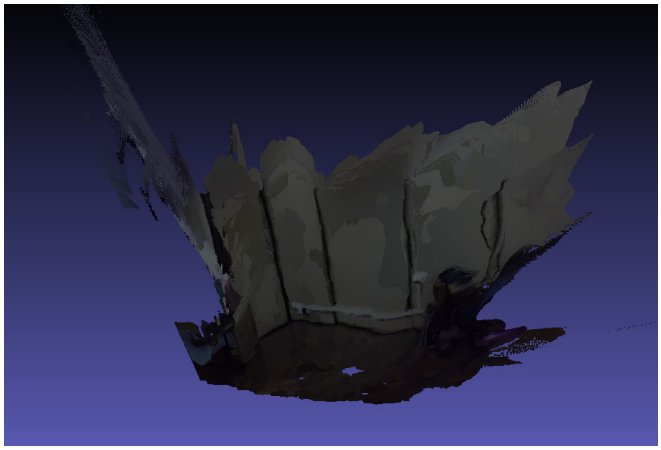
Já no cenário 2, novamente é utilizado o manipulador robótico, porém, desta vez, o objetivo é obter valores de erro de posicionamento do sensor quando comparado a posição do *end effector*, ou o final do braço robótico. O cenário 3 tem por objetivo avaliar a utilização da odometria visual como chute inicial para o método ICP de alinhamento de nuvens de pontos. No último cenário, o número 4, avalia-se o desempenho do sistema distribuído em duas máquinas, tendo como foco

os tempos de processamento e uso de memória. Será utilizado para comparação o tempo obtido quando executado o processo em apenas uma máquina. Como forma de melhor ilustrar os ensaios realizados, foram postados vídeos *online* nos links https://youtu.be/c1DaKlm_xBo e https://youtu.be/ITB38-T1MGI.

4.1 CENÁRIO DE AVALIAÇÃO 0

Neste cenário estão dispostos resultados pouco satisfatórios para a reconstrução 3D. Primeiramente, levantou-se a hipótese da utilização das nuvens vindas tanto da câmera RGB-D quanto da estéreo. Esta hipótese foi descartada, visto que a qualidade da nuvem estéreo, apesar de densa, apresentava muitas distorções em suas reconstruções quando comparada a RGB-D devido a falhas na medição da profundidade, como visto nas Figuras 23 e 24, obtidas na mesma cena.

Figura 23 – Nuvem de pontos produzida pela câmera estéreo ZED, mostrando suas distorções.



Fonte: Elaborada pelo autor.

Figura 24 – Nuvem de pontos produzida pela câmera RGB-D ASTRA, mostrando ser melhor comportada.

Fonte: Elaborada pelo autor.

4.2 CENÁRIO DE AVALIAÇÃO 1

Este cenário de experimento tem por objetivo validar a qualidade da odometria visual obtida através da visão estéreo. Para isso, foi utilizado como *ground truth* um braço robótico do robô Kuka YouBot, apresentado pela Figura 25. Como observado na figura, optamos por uma cena controlada, onde a maior fonte de características é o objeto a ser reconstruído. O algoritmo de Denavit-Hartenberg foi utilizado para a obtenção da cinemática direta do robô [16]. A partir deste modelo, foi possível determinar a cinemática inversa que permite enviar ao robô uma trajetória a ser seguida, obtida através do método do gradiente descendente.

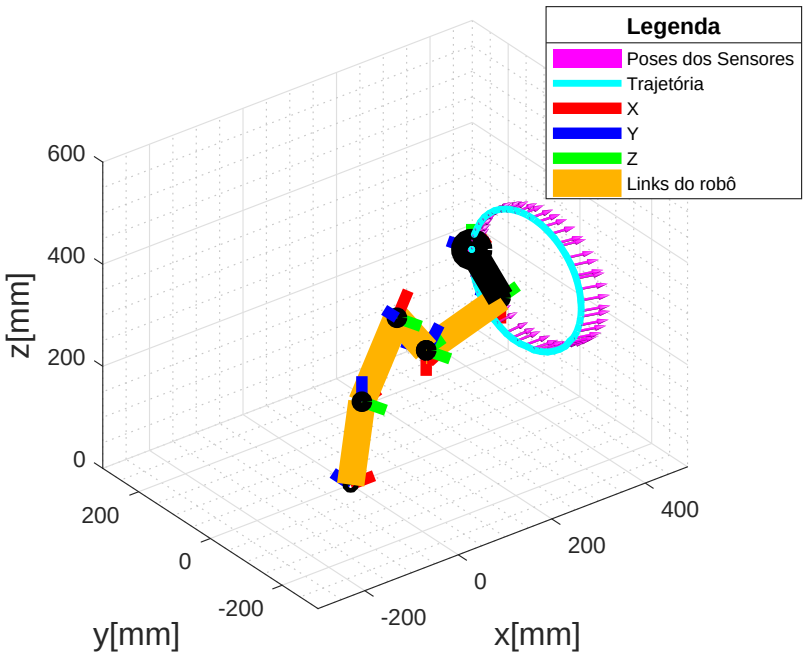
Figura 25 – Robô Kuka YouBot, utilizado com *ground truth*, juntamente com o cenário utilizado para testes.

Fonte: Elaborada pelo autor.

A primeira trajetória utilizada foi uma circunferência devido a dificuldade que apresenta para localização. O caminho é mostrado em ciano pelas Figuras 26 e 27. Têm-se: a posição da trajetória circular realizada pelo robô, na cor ciano; as flechas em magenta representam as poses dos sensores obtidas pela visão estéreo; os eixos em vermelho, azul e verde representam os eixos x, y e z para cada sistema de coordenadas de cada junta rotacional do robô; em laranja estão representados os elos do robô, ou seja, partes que compõem o braço e conectam as juntas.

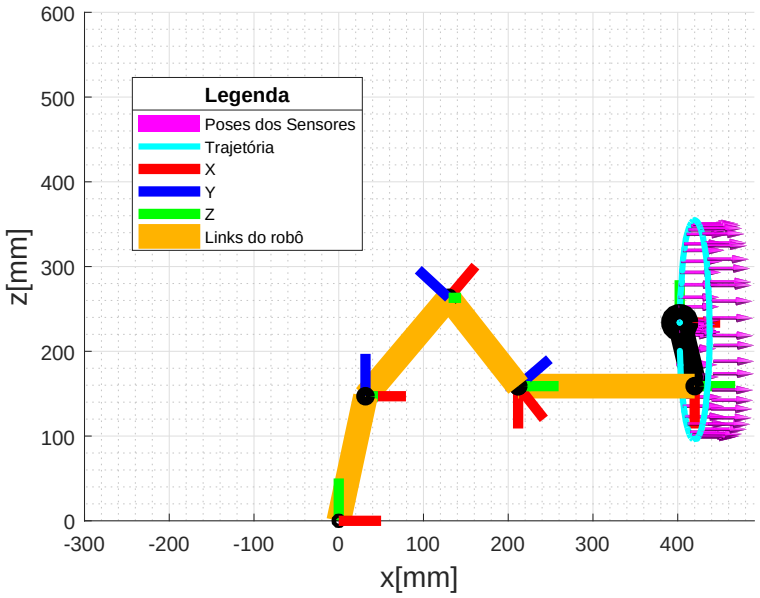
Como observado, a odometria visual apresentou uma trajetória circular, sugerindo que obteve exito na localização. Porém, existem alguns pontos que podem ser considerados errados, que prejudicariam a reconstrução 3D, que pode ser observada na Figura 28. Porém, estes são poucos, levantando a hipótese de que se retirados como *outliers* a reconstrução não seria prejudicada.

Figura 26 – Trajetória circular com seis pontos sincronizados para medição de erro.



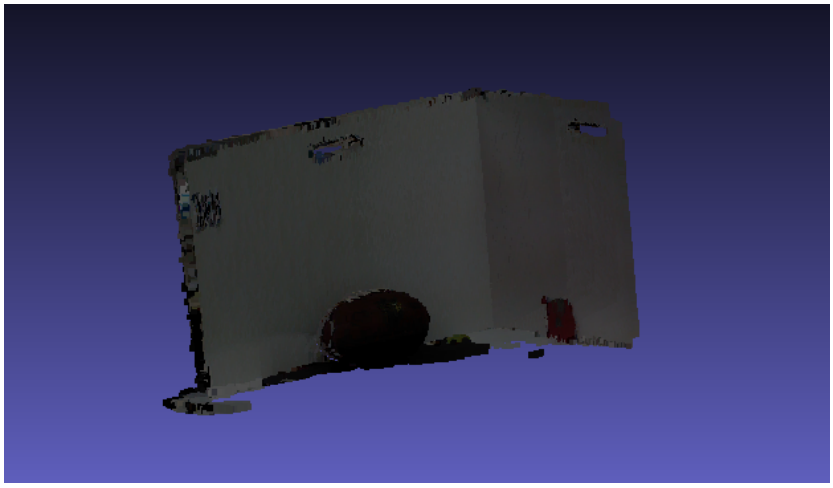
Fonte: Elaborada pelo autor.

Figura 27 – Trajetória circular com seis pontos sincronizados para medição de erro.



Fonte: Elaborada pelo autor.

Figura 28 – Nuvem de pontos obtida com trajetória circular.



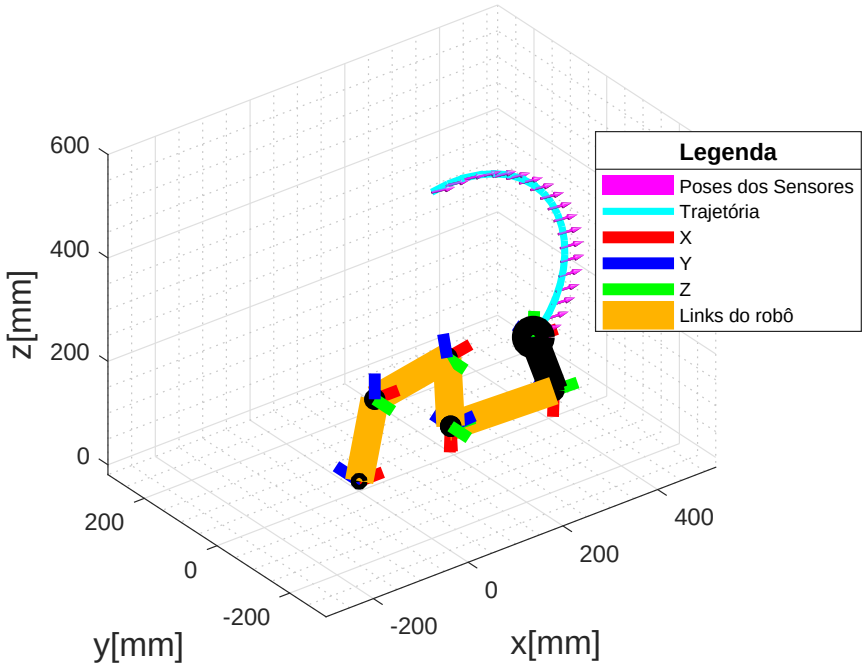
Fonte: Elaborada pelo autor.

Quanto aos demais pontos, é possível notar a característica circular da trajetória, porém seus erros em relação ao *ground truth* precisam ser determinados

de forma quantitativa. Para isso, deve haver um sincronismo entre as publicações de posições do robô recebidas e a odometria visual, uma tarefa não-trivial, porém solucionada no próximo cenário de experimentos.

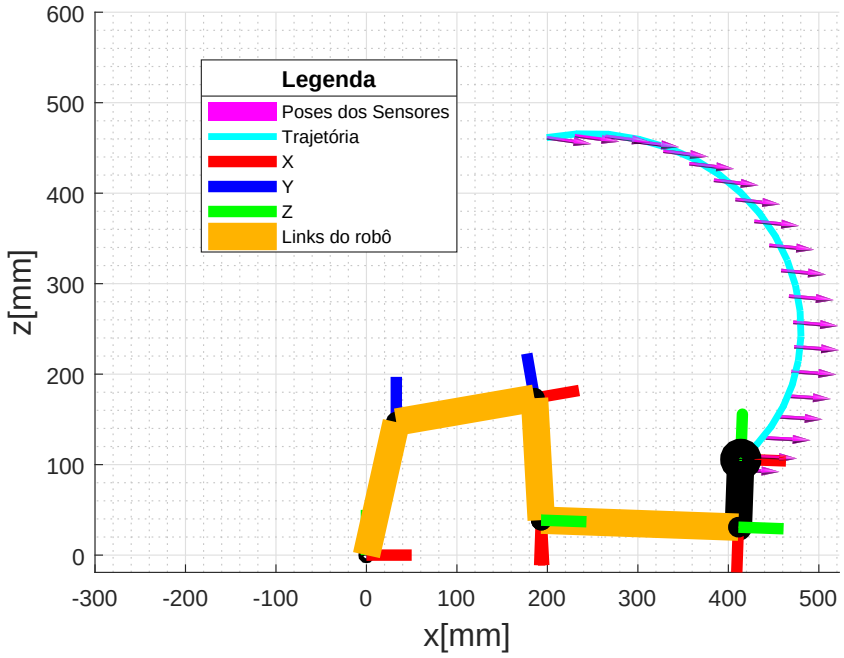
A segunda trajetória utilizada foi a apresentada pelas Figuras 29 e 30. O objetivo é avaliar a capacidade do sistema de localização de seguir a trajetória com variações de profundidade, complementando a trajetória circular anterior. Os resultados obtido foram semelhantes ao último caminho, onde o formato apresentado pela odometria visual é semelhante ao realizado pelo robô. A nuvem de pontos resultante pode ser observada na Figura 31.

Figura 29 – Trajetória alternativa, com mais variação de profundidade.



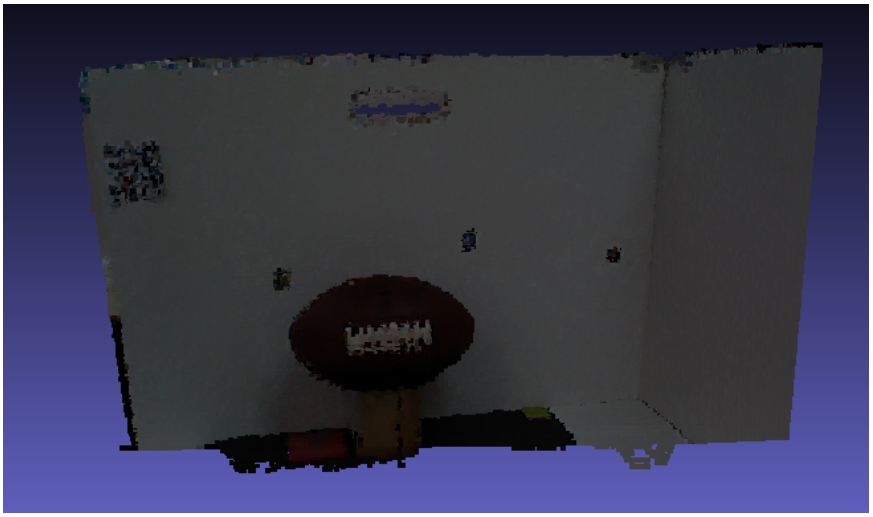
Fonte: Elaborada pelo autor.

Figura 30 – Outra vista para a segunda trajetória.



Fonte: Elaborada pelo autor.

Figura 31 – Nuvem de pontos obtida com a segunda trajetória.

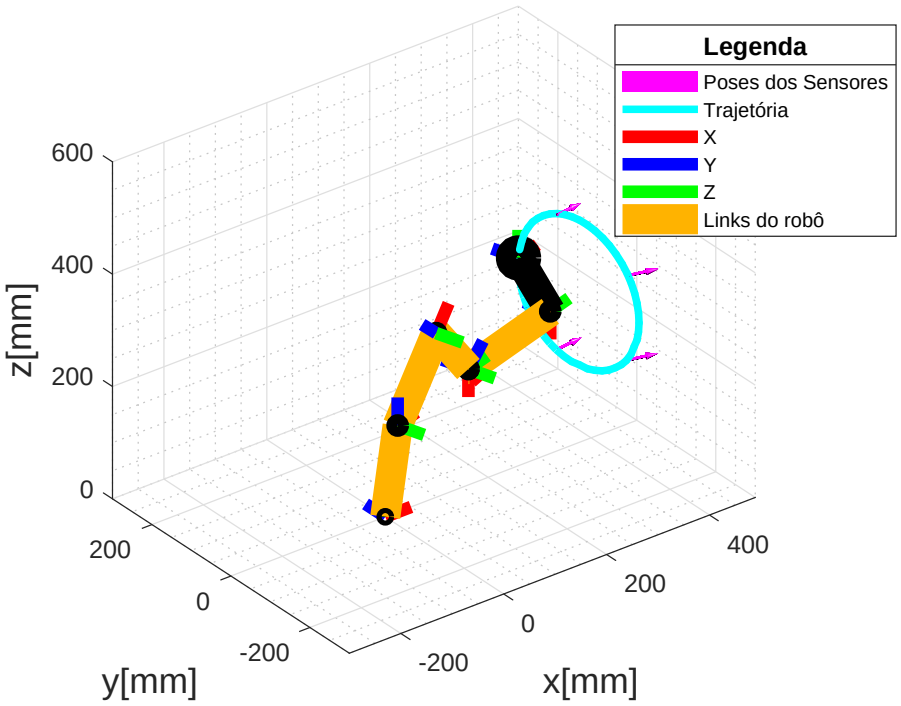


Fonte: Elaborada pelo autor.

4.3 CENÁRIO DE AVALIAÇÃO 2

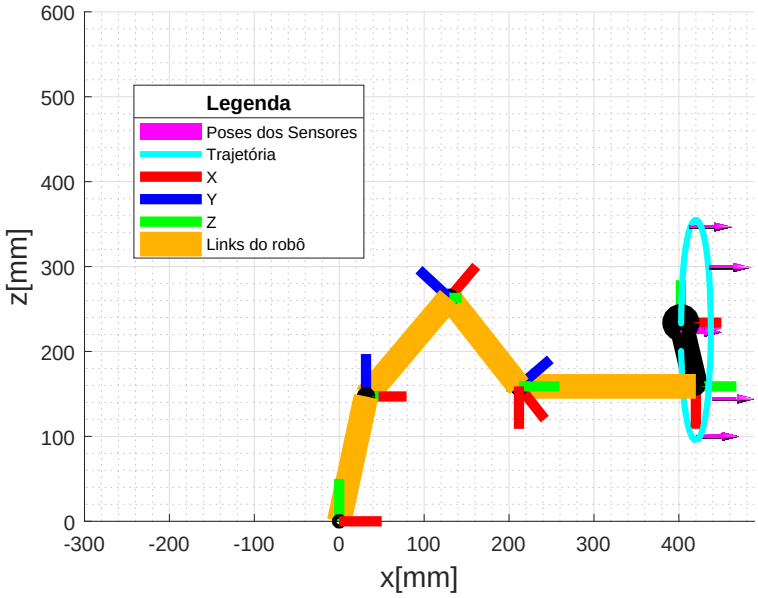
Neste cenário realizamos as mesmas trajetórias, circular e em arco com profundidade variada, realizadas pelo robô durante o ensaio anterior. Porém, foram sincronizados pontos a serem comparados para determinação do erro. Este sincronismo foi realizado por meio de publicações e subscrições controladas manualmente, ou seja, os dados somente eram publicados e subscritos quando desejado. As Figuras 32 e 33, ilustram novante a trajetória circular, sendo: a trajetória realizada pelo robô na cor ciano; as flechas em magenta representam as poses dos sensores obtidas pela visão estéreo; os eixos em vermelho, azul e verde representam os eixos x, y e z para cada sistema de coordenadas de cada junta rotacional do robô; em laranja estão representados os elos do robô, ou seja, suas partes compõem o braço e conectam as juntas.

Figura 32 – Trajetória circular com seis pontos sincronizados para medição de erro.



Fonte: Elaborada pelo autor.

Figura 33 – Trajetória circular com seis pontos sincronizados por uma vista diferente.



Fonte: Elaborada pelo autor.

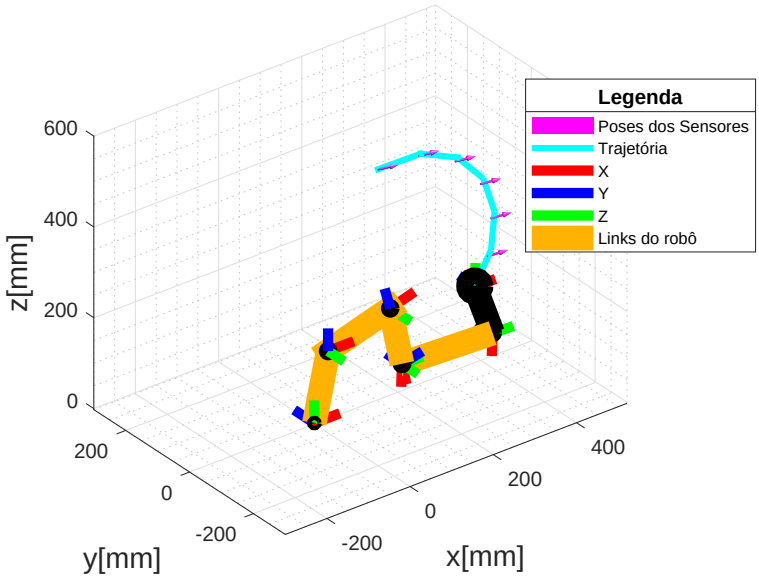
A Tabela 1 apresenta os valores dos erros obtidos através da diferença entre a posição do robô (*ground truth*) e a odometria visual, bem como a norma do erro, todos os valores apresentados em milímetros para a trajetória circular. A partir destes valores, chega-se a um erro quadrático médio de 3,8 mm.

Tabela 1 – Tabela contendo erros entre as poses do robô e as obtidas por visão estéreo para a trajetória circular.

Pontos	**P1**	**P2**	**P3**	**P4**	**P5**	**P6**
X [mm]	0,0002	1,3045	-1,4295	-6,0600	-7,7983	-0,4640
Y [mm]	-0.1291	1.3241	1.8884	-2,5900	-5,9095	-5,7577
Z [mm]	0,1530	0,6897	2,5990	2,3022	4,0891	7,2546
||Erro||	0,2002	1,9826	3,5163	6,9808	10,6045	9,2734

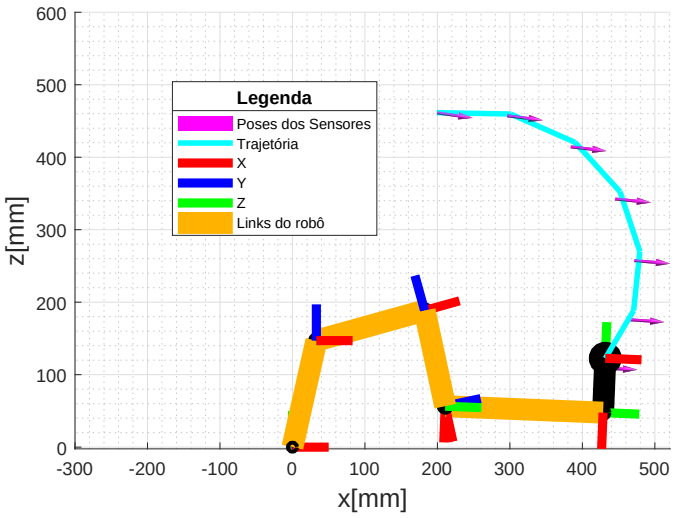
Quanto a segunda trajetória, as Figuras 34 e 35 ilustram os pontos sincronizados.

Figura 34 – Trajetória com profundidade variada, com sete pontos sincronizados para medição de erro.



Fonte: Elaborada pelo autor.

Figura 35 – Trajetória com profundidade, com sete pontos sincronizados, por um ângulo diferente.



Fonte: Elaborada pelo autor.

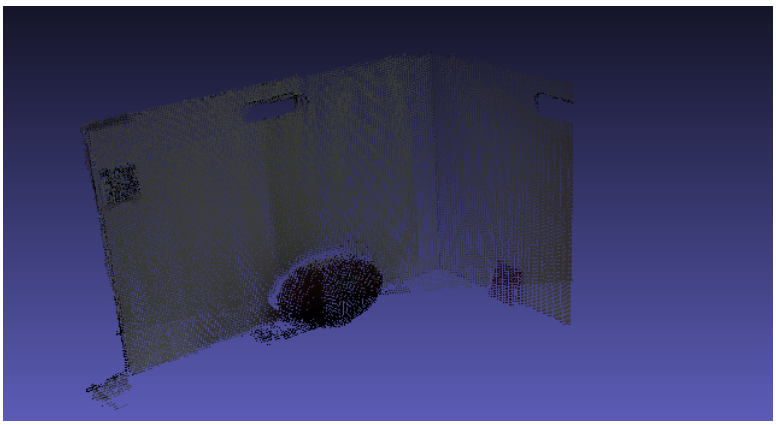
Como realizado anteriormente, a Tabela 2, a seguir, apresenta os valores de erros obtidos através da diferença entre a posição do robô e a odometria visual, também apresentando a norma do erro, tento todas as informações apresentadas em milímetros. O valor quadrático médio é agora 6,4 mm, superior ao observado na trajetória anterior.

Tabela 2 – Tabela contendo erros entre as poses do robô e as obtidas por visão estéreo para a trajetória de profundidade variada.

Pontos	**P1**	**P2**	**P3**	**P4**	**P5**	**P6**	**P7**
X [mm]	-0,2842	4,8284	5,8949	6,7364	7,5845	7,0788	5,4866
Y [mm]	0,0854	0,8135	1,0923	0,2270	-0,8132	-2,0117	-2,7256
Z [mm]	0,2930	2,4786	6,2703	10,0553	12,5111	12,6021	13,4857
||Erro||	0.4170	5.4880	8.6752	12.1054	14.6531	14.5935	14.8120

Neste cenário, a nuvem de pontos obtida na trajetória circular é a apresentada pela Figura 36. É possível notar que esta nuvem possui aparência mais esparsa que a anterior, devido a menor quantidade de posições de câmera utilizadas. Além disso, também percebe-se os efeitos do erro milimétrico na nuvem resultante, pequenas alterações no objeto a ser reconstruído. É válido lembrar que trata-se de uma nuvem de pontos "crua", sem nenhum tipo de pós-processamento, o que leva ao próximo cenário.

Figura 36 – Nuvem de pontos obtida com pontos sincronizados.



Fonte: Elaborada pelo autor.

4.4 CENÁRIO DE AVALIAÇÃO 3

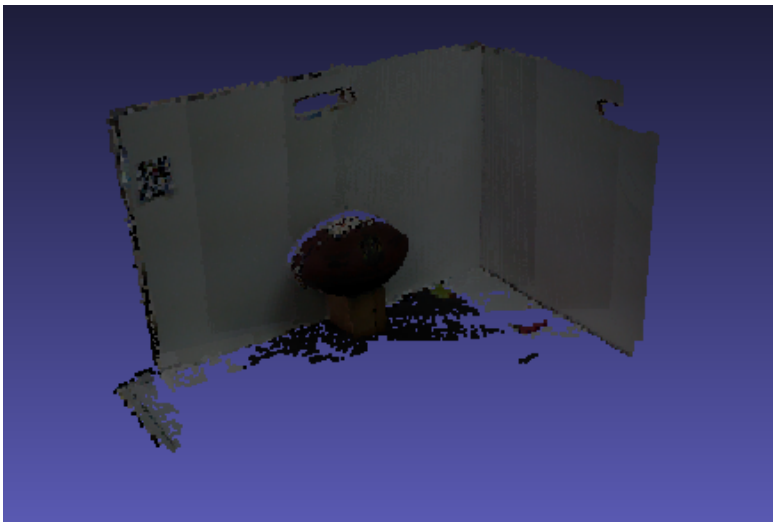
Como observado nos cenários anteriores, as nuvens obtidas não são alinhadas da melhor forma possível mesmo com um erro milimétrico. Isto se deve ao posicionamento dos pontos ser determinado pela odometria, sendo afetado mesmo por erros pequenos e gerando distorções, quando não duplicações de objetos em piores casos.

Como forma de reduzir estes efeitos na reconstrução, utilizou-se o algoritmo ICP implementado na *Point Cloud Library* (PCL). Este método, como mencionado em [69], obtém correspondências entre duas nuvens amostradas e minimiza a distância entre elas, obtendo uma nova transformação de corpo rígido que melhor ajusta as nuvens.

Para avaliar os resultados visuais, foram realizadas as duas trajetórias utilizadas anteriormente para o braço robótico: a circular e a trajetória em arco com variação de profundidade, que oferece maior grau de dificuldade para a odometria visual e consequentemente para o ICP.

A Figura 37 apresenta a nuvem de pontos obtida com o ajuste fino do ICP para a trajetória circular.

Figura 37 – Nuvem de pontos obtida com ICP para trajetória circular.



Fonte: Elaborada pelo autor.

Quando comparadas nuvens semelhantes, com e sem a utilização do ICP, é possível notar algumas diferenças, como a espessura do material usado para o fundo e os pontos deslocados na região do menor círculo vermelho, enaltecidas com os círculos vermelhos nas Figuras 38 e 39.

Figura 38 – Nuvem de pontos obtida com ICP, com marcações em círculos vermelhos.

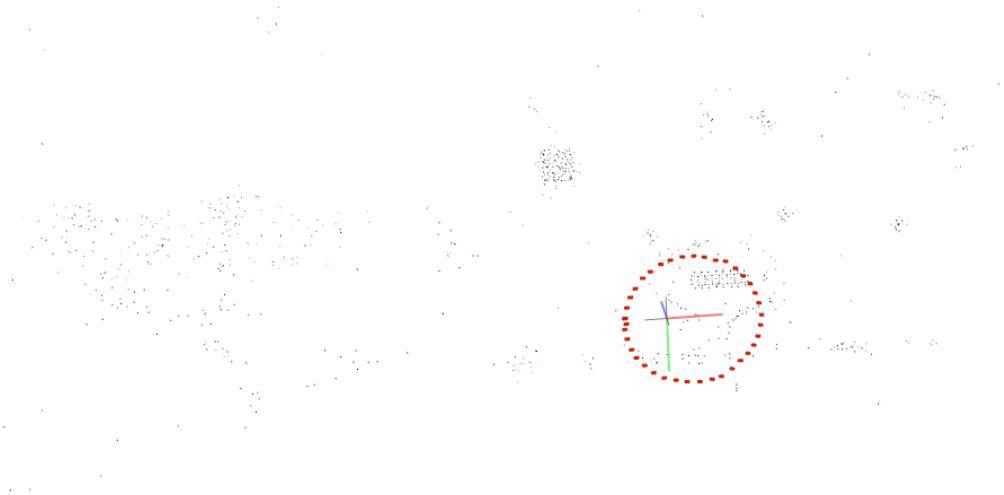
Fonte: Elaborada pelo autor.

Figura 39 – Nuvem de pontos obtida sem ICP para trajetória circular, com marcações vermelhas em problemas de alinhamento.

Fonte: Elaborada pelo autor.

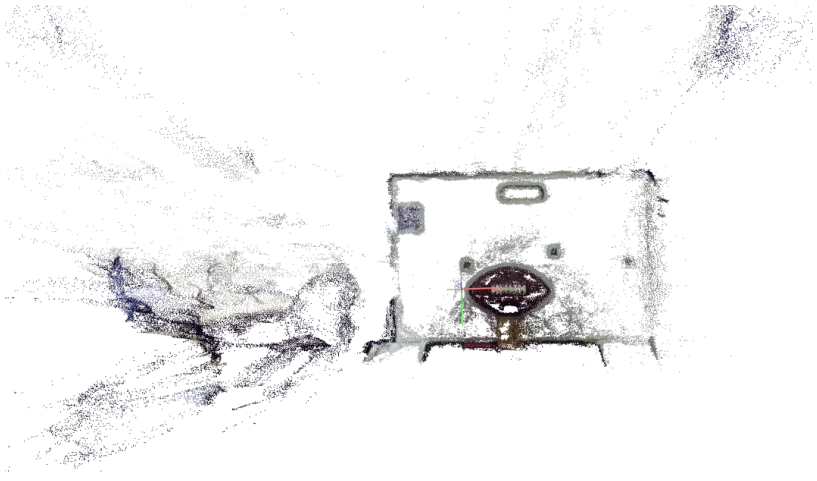
Como o objetivo deste cenário é a avaliação visual da nuvem de pontos, utilizou-se para comparação os resultados obtidos com os *software* abertos Colmap e Elastic Fusion. As Figuras 40, 41 e 42 apresentam os resultados de trajetória e nuvens de pontos obtidas com o Colmap para o caminho circular.

Figura 40 – Nuvem de pontos esparsa e trajetória obtidas com Colmap para o caminho circular.



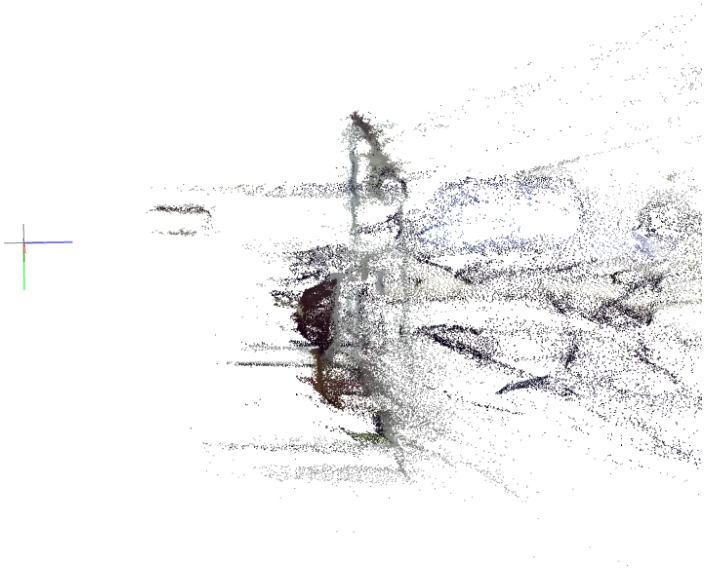
Fonte: Elaborada pelo autor.

Figura 41 – Nuvem de pontos densa obtida com Colmap para a trajetória circular.



Fonte: Elaborada pelo autor.

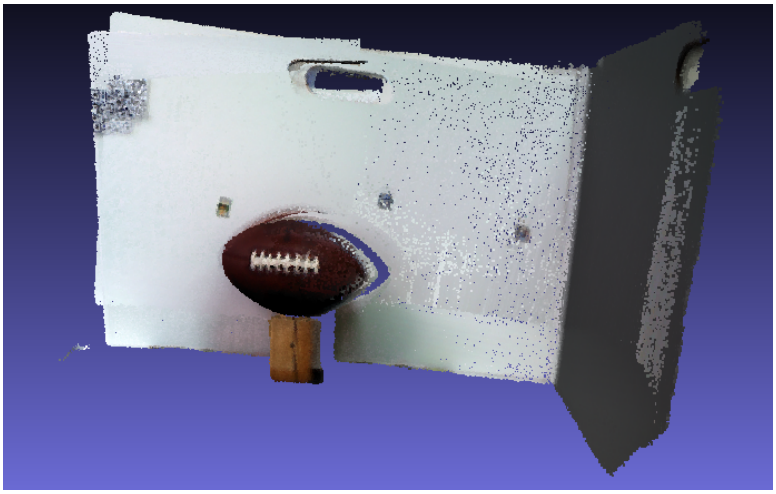
Figura 42 – Nuvem de pontos densa obtida com Colmap para a trajetória circular, por outro ângulo.



Fonte: Elaborada pelo autor.

A Figura 43 apresenta o resultado obtido com o Elastic Fusion para a trajetória circular. É possível notar problemas de acumulação, ao observar esta figura.

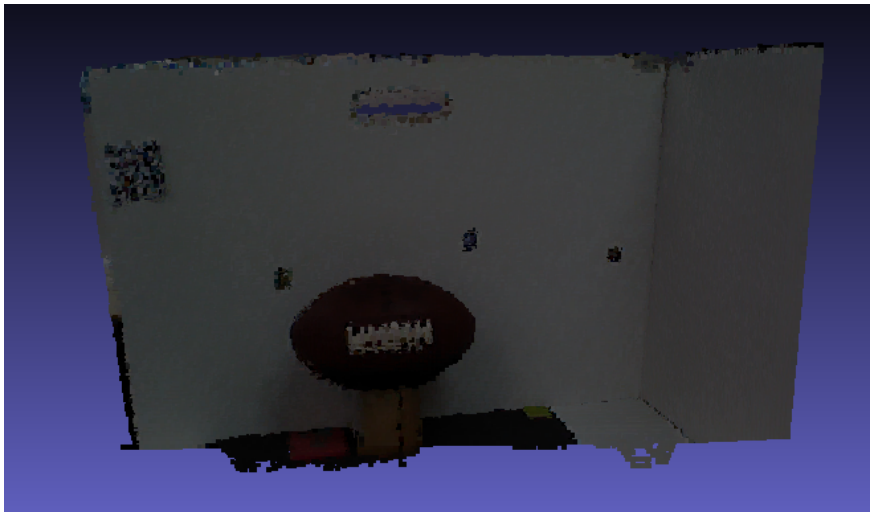
Figura 43 – Nuvem de pontos densa obtida com Elastic Fusion para a trajetória circular.



Fonte: Elaborada pelo autor.

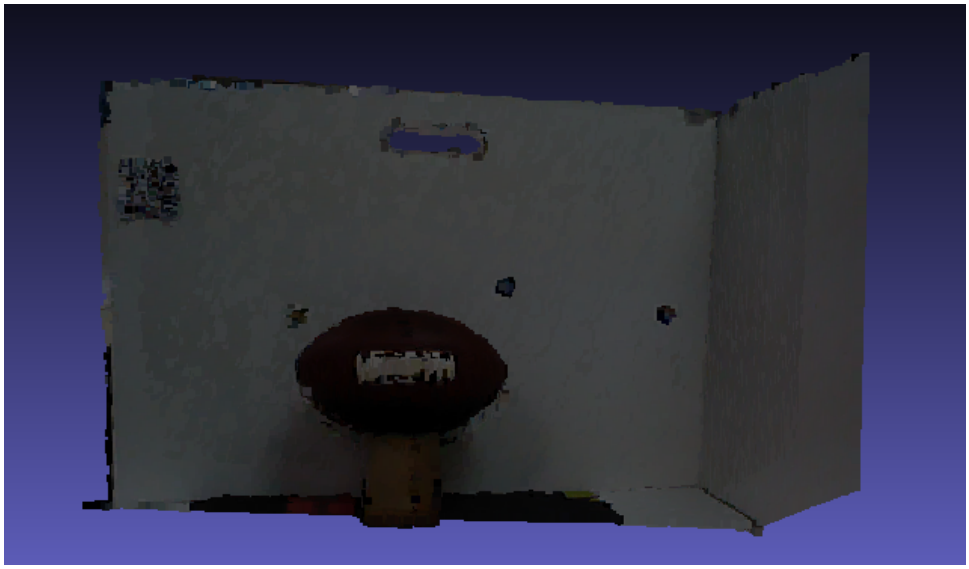
Com as Figuras 44 e 45 , tem-se o resultado em nuvens de pontos obtidos para a segunda trajetória, com e sem a aplicação do ICP, respectivamente.

Figura 44 – Nuvem de pontos obtida sem ICP para a segunda trajetória.



Fonte: Elaborada pelo autor.

Figura 45 – Nuvem de pontos obtida utilizando ICP para a segunda trajetória.

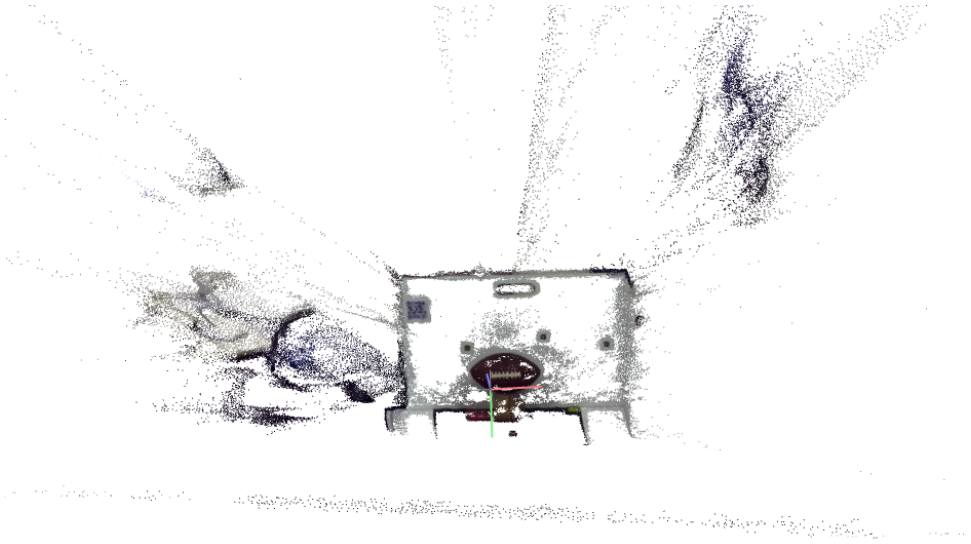


Fonte: Elaborada pelo autor.

Também para comparação, têm-se as Figuras 46 e 47, obtidas com Colmap,

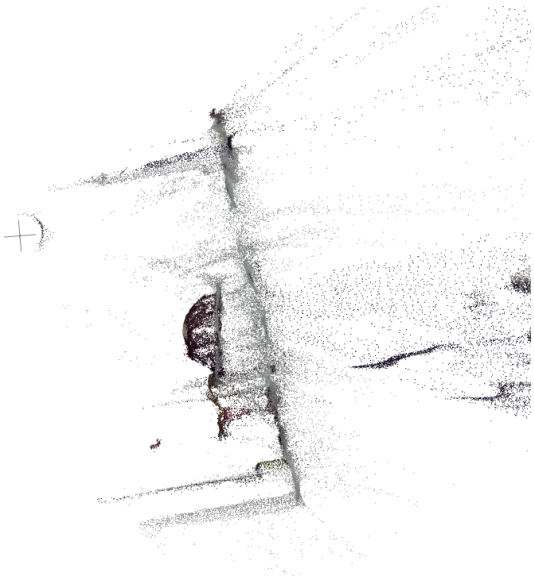
e Figuras 48 e 49, obtidas com Elastic Fusion. A aquisição seguiu a segunda trajetória proposta.

Figura 46 – Nuvem de pontos densa obtida com Colmap para a segunda trajetória.



Fonte: Elaborada pelo autor.

Figura 47 – Nuvem de pontos densa obtida com Colmap para a segunda trajetória, por um ângulo diferente.

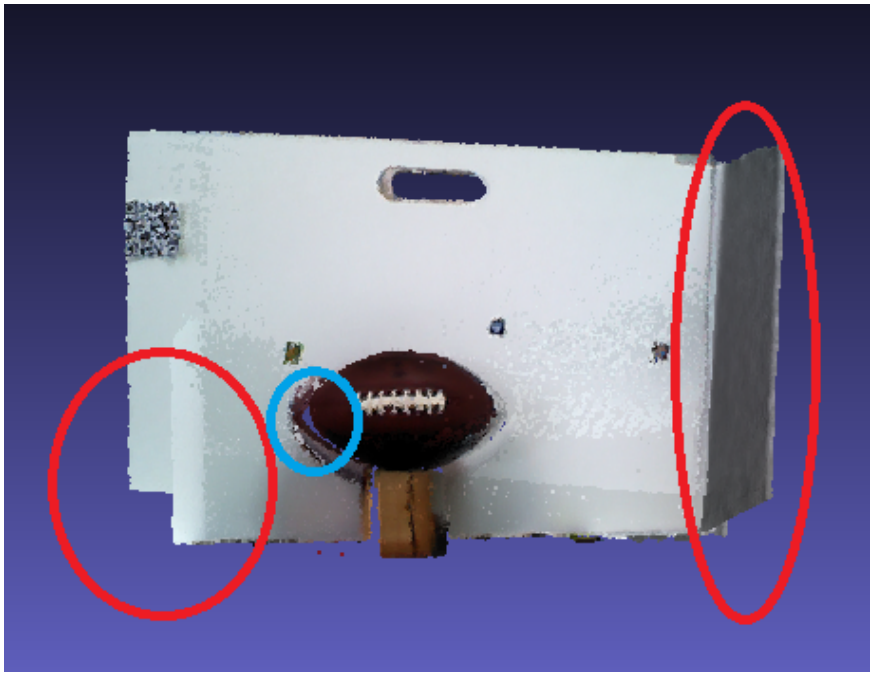


Fonte: Elaborada pelo autor.

Figura 48 – Nuvem de pontos densa obtida com Elastic Fusion para a segunda trajetória.

Fonte: Elaborada pelo autor.

Figura 49 – Nuvem de pontos densa obtida com Elastic Fusion para a segunda trajetória.



Fonte: Elaborada pelo autor.

Uma análise cautelosa mostra que, apesar de atingirem alta qualidade de reconstrução, estas nuvens apresentam alguns problemas. Nos resultados obtidos com o Colmap, é possível notar, principalmente pelas Figuras 42 e 47, erros de profundidade e deformação característicos de nuvens obtidas com visão estéreo, vista na Figura 40. Além disso, observa-se que a nuvem é menos densa quando comparada aos resultados do Elastic Fusion e desta dissertação.

Quanto ao Elastic Fusion, a qualidade de sua reconstrução deve ser reconhecida. Porém, é possível observar erros de acumulação provenientes de estimações errôneas da localização do sensor para ambas as trajetórias. A utilização da visão estéreo proposta nesta dissertação, aliada a nuvem RGB-D, obteve resultados superiores neste aspecto.

4.5 CENÁRIO DE AVALIAÇÃO 4

Neste cenário, o objetivo é avaliar o desempenho computacional, tomando como métrica de avaliação o tempo de processamento do sistema distribuído e local. São comparados os resultados do sistema em apenas um computador, com toda a comunicação local, contra o sistema FOG apresentado como proposta, onde duas máquinas se comunicam como sistema de aquisição e acumulação.

Para o sistema local, foram executados somente os processos de aquisição e acumulação, para evitar quaisquer interferências de processos que venham a prejudicar o processamento. O critério de parada utilizado foi a quantidade de pontos acumulados na nuvem, definido em 1 milhão de pontos. Para cada reconstrução, foram tomadas 10 medidas de tempo, de forma a calcular-se a média e o desvio padrão para a acumulação. Além disso, são também contabilizadas as frequências de subscrição da etapa de aquisição e a memória utilizada pela máquina de acumulação.

Para o sistema distribuído, foi realizada a mesma tarefa, para o mesmo cenário, porém com as duas máquinas, novamente executando somente os processos de aquisição e acumulação. Os resultados obtidos estão dispostos na Tabela 3, observando que o tempo para a reconstrução aumentou significativamente, o que indica o contrário da hipótese proposta. Dentre as razões para este aumento de

tempo, destacam-se ser atrasos na comunicação, realizada via cabo *ethernet* em rede LAN. Outro fator que dá suporte a esta sugestão é a diminuição da frequência de subscrição no tópico do sistema de aquisição para acumulação, de $0,98 \pm 0,03$ para o local e $0,70 \pm 0,01$ para o sistema distribuído.

Tabela 3 – Tabela com medidas de tempo para sistema local e distribuído.

Tempo (s)	**t1**	**t2**	**t3**	**t4**	**t5**	**t6**	**t7**	**t8**	**t9**	**t10**	**Média**	σ
Local	30	34	40	33	34	33	34	32	37	38	34,5	3,0
Distribuído	110	110	109	110	111	104	114	102	107	107	108,4	3,5

Entretanto, favorecendo a proposta desta dissertação, temos a redução da quantidade de memória RAM utilizada. Para o sistema local, o uso de memória variava em torno de 2 GB, enquanto que no sistema distribuído foi reduzido para 1,6 GB.

5 CONCLUSÕES E TRABALHOS FUTUROS

Esta dissertação apresentou um estudo e revisão bibliográfica acerca do tema de reconstrução tridimensional, popularmente chamada de reconstrução 3D. Ao longo deste processo, é clara a complexidade do tema, visto que cada etapa de um *pipeline* de reconstrução tem por si só grande complexidade e trabalho computacional, muitas vezes exigindo o uso de placas gráficas potentes.

Desta forma, foi proposto nessa dissertação a utilização de sistemas distribuídos para este tipo de aplicação, buscando afastar o processamento de maior carga da etapa de aquisição, avaliando a possibilidade de aplicação de sistemas de reconstrução de forma embarcada. Além disso, também foi alvo deste trabalho o aproveitamento das melhores qualidades de cada sensor, utilizando a câmera estéreo para localização aliado aos mapas bem definidos produzidos pelos sensores RGB-D.

Como resultado temos um sistema de reconstrução 3D que pode ser utilizado tanto localmente como distribuído, tendo sua comunicação gerenciada pelo *framework* de robótica ROS. A avaliação do sistema se deu primeiramente, verificando visualmente se a trajetória produzida pela odometria visual correspondia à realizada pelo *ground truth*, um braço robótico. Em seguida, foram avaliados os erros de posicionamento entre estes mesmos valores, concluindo que sua grandeza é dada em escala de milímetros. Porém, observamos que, mesmo diminuto, este erro causa variações visuais à nuvem de pontos, sendo necessária a aplicação do algoritmo ICP para ajuste fino, utilizando como chute inicial a odometria visual, superando, nos testes realizados, os métodos bem conceituados na literatura. É válido lembrar que os resultados foram obtidos em ambiente fechado e com exposição a luz solar limitada, além de iluminação similar para todos os ensaios.

Finalmente, avaliou-se o desempenho computacional do sistema, utilizando o tempo de processamento para a etapa de acumulação, a frequência de publicação da etapa de aquisição e memória RAM utilizada. O primeiro resultado apontou uma queda de desempenho do sistema distribuído com relação ao local. Este resultado se deve a menor velocidade de comunicação obtida com a rede LAN quando comparado à comunicação local. O que reforça essa ideia é a frequência de

subscrição, mesmo mantida a taxa de publicação da aquisição. Quanto a memória utilizada, foi obtida uma redução no consumo de 0,4 GB.

Os resultados obtidos favorecem a elaboração de trabalhos futuros. Destaca-se a aplicação embarcada em kit de desenvolvimento NVIDIA Jetson, aproveitando sua capacidade de processamento, GPU e sistema operacional Ubuntu compatíveis com ROS. Além disso, o ganho com a união dos sensores foi conclusivo. Dessa forma, uma boa proposta é a introdução de novos sensores, visando aumentar a acurácia da localização, fornecendo um ponto de partida melhor para o ICP. Outra forma de reduzir o tempo gasto no ICP e na aquisição seria através da diminuição na quantidade de pontos processados. Para alcançar este objetivo, uma alternativa é a aplicação de redes neurais convolucionais, com o objetivo de reconhecer regiões de interesse antes da geração da nuvem de pontos instantânea [70]. Este processo traria melhor qualidade à reconstrução e aumento do desempenho, necessárias para aplicações embarcadas onde há menor disponibilidade de recursos computacionais e inclusive em peso para caso a implementação seja feita em robôs móveis e até mesmo *drones* para inspeção.

Finalmente, realizar o estudo de métodos de geração de malhas, a etapa final da reconstrução tridimensional. Esta etapa conecta os pontos da nuvem em um objeto sólido, permitindo que o mesmo seja inserido em um ambiente virtual ou em realidade aumentada a ser utilizada em treinamentos e manutenção preventiva futuramente.

REFERÊNCIAS

[1] "*Virtual Reality Society*." `https://www.vrs.org.uk/virtual-reality/what-is-virtual-reality.html`. Acessado em 07/12/2018.

[2] P. Duan, L. Pang, Y. Jin, Q. Guo, and Z.-X. Jia, "Equipment maintenance system based on virtual reality technology," in *Quality, Reliability, Risk, Maintenance, and Safety Engineering (ICQR2MSE), 2012 International Conference on*, pp. 1396–1399, IEEE, 2012.

[3] H. Qing, "Research and application of virtual reality technology in mechanical maintenance," in *International Conference on Advanced Technology of Design and Manufacture (ATDM 2010)*, p. 256 – 258, IET, 2010.

[4] A. Harjoko, R. M. Hujja, and L. Awaludin, "Low-cost 3d surface reconstruction using stereo camera for small object," in *Signals and Systems (ICSigSys), 2017 International Conference on*, pp. 285–289, IEEE, 2017.

[5] M. Zhang, Z. Zhang, and W. Li, "3d model reconstruction based on plantar image's feature segmentation," in *Progress in Informatics and Computing (PIC), 2017 International Conference on*, pp. 164–168, IEEE, 2017.

[6] G. Kontogianni, C. Koutsaftis, M. Skamantzari, A. Georgopoulos, and C. Chrysanthopoulou, "Developing and exploiting 3d textured models for a serious game application," in *Games and Virtual Worlds for Serious Applications (VS-Games), 2016 8th International Conference on*, pp. 1–4, IEEE, 2016.

[7] D. Shcherbinin, "Virtual reconstruction and 3d visualization of vostok spacecraft equipment," in *Engineering Technologies and Computer Science (EnT), 2017 International Workshop on*, pp. 56–58, IEEE, 2017.

[8] L. von Stumberg, V. Usenko, J. Engel, J. Stückler, and D. Cremers, "From monocular slam to autonomous drone exploration," in *Mobile Robots (ECMR), 2017 European Conference on*, pp. 1–8, IEEE, 2017.

[9] R. Jamiruddin, A. O. Sari, J. Shabbir, and T. Anwer, "Rgb-depth slam review," *arXiv preprint arXiv:1805.07696*, 2018.

[10] R. A. Newcombe, S. J. Lovegrove, and A. J. Davison, "Dtam: Dense tracking and mapping in real-time," in *Computer Vision (ICCV), 2011 IEEE International Conference on*, pp. 2320–2327, IEEE, 2011.

[11] J. L. Schonberger and J.-M. Frahm, "Structure-from-motion revisited," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4104–4113, 2016.

[12] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon, "Kinectfusion: Real-time dense surface mapping and tracking," in *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on*, pp. 127–136, IEEE, 2011.

[13] T. Whelan, M. Kaess, H. Johannsson, M. Fallon, J. J. Leonard, and J. McDonald, "Real-time large-scale dense rgb-d slam with volumetric fusion," *The International Journal of Robotics Research*, vol. 34, no. 4-5, pp. 598–626, 2015.

[14] D. A. Forsyth and J. Ponce, "A modern approach," *Computer vision: a modern approach*, pp. 88–101, 2003.

[15] R. Szeliski, *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[16] P. Corke, *Robotics, Vision and Control: Fundamental Algorithms In MATLAB® Second, Completely Revised*, vol. 118. Springer, 2017.

[17] R. C. Gonzalez, R. E. Woods, *et al.*, *Digital image processing*. Prentice hall Upper Saddle River, NJ, 2002.

[18] A. Koschan and M. Abidi, *Digital color image processing*. John Wiley & Sons, 2008.

[19] J. E. Solem, *Programming Computer Vision with Python: Tools and algorithms for analyzing images*. "O'Reilly Media, Inc.", 2012.

[20] R. Hartley and A. Zisserman, *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[21] Y. Ma, S. Soatto, J. Kosecka, and S. S. Sastry, *An invitation to 3-d vision: from images to geometric models*, vol. 26. Springer Science & Business Media, 2012.

[22] G. Bradski and A. Kaehler, *Learning OpenCV: Computer vision with the OpenCV library*. "O'Reilly Media, Inc.", 2008.

[23] R. I. Hartley, "Theory and practice of projective rectification," *International Journal of Computer Vision*, vol. 35, no. 2, pp. 115–127, 1999.

[24] Z.-y. Li, L.-m. Song, J.-t. Xi, Q.-h. Guo, X.-j. Zhu, and M.-l. Chen, "A stereo matching algorithm based on sift feature and homography matrix," *Optoelectronics Letters*, vol. 11, no. 5, pp. 390–394, 2015.

[25] A. S. Tanenbaum and M. Van Steen, *Distributed systems: principles and paradigms*. Prentice-Hall, 2007.

[26] G. F. Coulouris, J. Dollimore, and T. Kindberg, *Distributed systems: concepts and design*. pearson education, 2005.

[27] G. Coro, M. Palma, A. Ellenbroek, G. Panichi, T. Nair, and P. Pagano, "Reconstructing 3d virtual environments within a collaborative e-infrastructure," *Concurrency and Computation: Practice and Experience*, p. e5028, 2018.

[28]

[29] "*Edge computing vs. fog computing*." `https://www.cisco.com/c/en/us/solutions/enterprise-networks/edge-computing.html`. Acessado em 07/12/2018.

[30] "About ros." `http://www.ros.org/about-ros/`. Acessado em 12/11/2018.

[31] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, p. 5, Kobe, Japan, 2009.

[32] J. P. C. d. Souza *et al.*, "Pouso autônomo de vants baseado em rede neural artificial supervisionada por lógica fuzzy," 2018.

[33] F. d. O. Coelho *et al.*, "Missões autônomas em robôs móveis com tração diferencial: planejamento de caminhos, localização e mapeamento," 2018.

[34] "Xml-rpc specification." `http://xmlrpc.scripting.com/spec.html`. Acessado em 12/11/2018.

[35] "Cmake." `https://cmake.org/`. Acessado em 12/11/2018.

[36] "About ros." `http://wiki.ros.org/Master`. Acessado em 24/12/2018.

[37] D. Nistér, O. Naroditsky, and J. Bergen, "Visual odometry for ground vehicle applications," *Journal of Field Robotics*, vol. 23, no. 1, pp. 3–20, 2006.

[38] S. A. Rawashdeh and M. Aladem, "Toward autonomous stereo-vision control of micro aerial vehicles," in *Aerospace and Electronics Conference (NAECON) and Ohio Innovation Summit (OIS), 2016 IEEE National*, pp. 151–155, IEEE, 2016.

[39] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *Computer Vision (ICCV), 2011 IEEE international conference on*, pp. 2564–2571, IEEE, 2011.

[40] A. Geiger, J. Ziegler, and C. Stiller, "Stereoscan: Dense 3d reconstruction in real-time," in *Intelligent Vehicles Symposium (IV), 2011 IEEE*, pp. 963–968, Ieee, 2011.

[41] A. Geiger, M. Roser, and R. Urtasun, "Efficient large-scale stereo matching," in *Computer Vision–ACCV 2010*, pp. 25–38, Springer, 2010.

[42] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[43] M. Song, H. Watanabe, and J. Hara, "Robust 3d reconstruction with omni-directional camera based on structure from motion," in *Advanced Image Technology (IWAIT), 2018 International Workshop on*, pp. 1–4, IEEE, 2018.

[44] G. Yu and J.-M. Morel, "Asift: An algorithm for fully affine invariant comparison," *Image Processing On Line*, vol. 1, pp. 11–38, 2011.

[45] N. Snavely, I. Simon, M. Goesele, R. Szeliski, and S. M. Seitz, "Scene reconstruction and visualization from community photo collections," *Proceedings of the IEEE*, vol. 98, no. 8, pp. 1370–1390, 2010.

[46] N. Snavely, "Scene reconstruction and visualization from internet photo collections: A survey," *IPSJ Transactions on Computer Vision and Applications*, vol. 3, pp. 44–66, 2011.

[47] T. Whelan, R. F. Salas-Moreno, B. Glocker, A. J. Davison, and S. Leutenegger, "Elasticfusion: Real-time dense slam and light source estimation," *The International Journal of Robotics Research*, vol. 35, no. 14, pp. 1697–1716, 2016.

[48] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse, "Monoslam: Real-time single camera slam," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, no. 6, pp. 1052–1067, 2007.

[49] S. B. Gokturk, H. Yalcin, and C. Bamji, "A time-of-flight depth sensor-system description, issues and solutions," in *Computer Vision and Pattern Recognition Workshop, 2004. CVPRW'04. Conference on*, pp. 35–35, IEEE, 2004.

[50] "Product-astra." `https://orbbec3d.com/product-astra/`. Acessado em 13/11/2018.

[51] P. J. Besl and N. D. McKay, "Method for registration of 3-d shapes," in *Sensor Fusion IV: Control Paradigms and Data Structures*, vol. 1611, pp. 586–607, International Society for Optics and Photonics, 1992.

[52] B. Curless and M. Levoy, "A volumetric method for building complex models from range images," in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pp. 303–312, ACM, 1996.

[53] M. Keller, D. Lefloch, M. Lambers, S. Izadi, T. Weyrich, and A. Kolb, "Real-time 3d reconstruction in dynamic scenes using point-based fusion," in *3D Vision-3DV 2013, 2013 International Conference on*, pp. 1–8, IEEE, 2013.

[54] H. Pfister, M. Zwicker, J. Van Baar, and M. Gross, "Surfels: Surface elements as rendering primitives," in *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pp. 335–342, ACM Press/Addison-Wesley Publishing Co., 2000.

[55] R. W. Sumner, J. Schmid, and M. Pauly, "Embedded deformation for shape manipulation," in *ACM Transactions on Graphics (TOG)*, vol. 26, p. 80, ACM, 2007.

[56] A. Dai, M. Nießner, M. Zollhöfer, S. Izadi, and C. Theobalt, "Bundlefusion: Real-time globally consistent 3d reconstruction using on-the-fly surface reintegration," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 4, p. 76a, 2017.

[57] "Computação voluntária ufcg." `http://www.dsc.ufcg.edu.br/~pet/jornal/julho2010/materias/informatica.html`. Acessado em 07/12/2018.

[58] S. Monteiro and G. Cesar, "Computação voluntária," Master's thesis, 2012.

[59] I. Z. Ibragimov and I. M. Afanasyev, "Comparison of ros-based visual slam methods in homogeneous indoor environment," in *Positioning, Navigation and Communications (WPNC), 2017 14th Workshop on*, pp. 1–6, IEEE, 2017.

[60] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "Orb-slam: a versatile and accurate monocular slam system," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[61] R. Mur-Artal and J. D. Tardós, "Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.

[62] O. Yilmaz and F. Karakus, "Stereo and kinect fusion for continuous 3d reconstruction and visual odometry," in *Electronics, Computer and Computation (ICECCO), 2013 International Conference on*, pp. 115–118, IEEE, 2013.

[63] "Código aberto elastic fusion no repositório github.." `https://github.com/mp3guy/ElasticFusion`. Acessado em 25/12/2018.

[64] "Colmap-documentation." `https://colmap.github.io/`. Acessado em 13/11/2018.

[65] M. G. Diaz, F. Tombari, P. Rodriguez-Gonzalvez, and D. Gonzalez-Aguilera, "Analysis and evaluation between the first and the second generation of rgb-d sensors," *IEEE Sensors journal*, vol. 15, no. 11, pp. 6507–6516, 2015.

[66] "Pacote zed-ros." `http://wiki.ros.org/zed-ros-wrapper`. Acessado em 13/11/2018.

[67] "Pacote astra." `http://wiki.ros.org/astra_camera/`. Acessado em 13/11/2018.

[68] "Pacote openni2 em ros." `http://wiki.ros.org/openni2_launch/`. Acessado em 13/11/2018.

[69] A. Segal, D. Haehnel, and S. Thrun, "Generalized-icp.," in *Robotics: science and systems*, vol. 2, p. 435, 2009.

[70] L. Silva, V. Vidal, M. Silva, M. Santos, A. Carvalho, A. Cerqueira, L. Honório, H. Rezende, J. Ribeiro, A. Pancoti, *et al.*, "Automatic recognition of electrical grid elements using convolutional neural networks," in *2018 22nd International Conference on System Theory, Control and Computing (ICSTCC)*, pp. 822–826, IEEE, 2018.

[71] M. Santos, M. Silva, V. Vidal, L. Honório, V. Lopes, L. Silva, H. Rezende, J. Ribeiro, A. Cerqueira, A. Pancoti, *et al.*, "Experimental validation of quadrotors angular stability in a gyroscopic test bench," in *2018 22nd International Conference on System Theory, Control and Computing (ICSTCC)*, pp. 783–788, IEEE, 2018.

[72] M. Santos, D. Silva, M. Silva, V. Vidal, L. Honório, V. Lopes, L. Silva, H. Rezende, J. Ribeiro, A. Cerqueira, *et al.*, "Project and design of multirate loop controllers for fixed-wings aircrafts," in *2018 22nd International Conference on System Theory, Control and Computing (ICSTCC)*, pp. 777–782, IEEE, 2018.

[73] H. Rezende, M. Silva, M. Santos, L. Honório, L. Silva, V. Vidal, J. Ribeiro, A. Cerqueira, A. Pancoti, and B. Regina, "Signal estimation for uav control loop identification using artificial immune systems," in *2018 22nd International Conference on System Theory, Control and Computing (ICSTCC)*, pp. 579–584, IEEE, 2018.

APÊNDICE A – Publicações

Neste apêndice estão apresentados os trabalhos produzidos e publicados ao longo da pesquisa realizada para esta dissertação, todos apresentados no evento 22^{nd} *International Conference on System Theory, Control and Computing* (ICSTCC), realizado em Sinaia, na Romênia. O primeiro artigo apresenta uma técnica que pode ser utilizada para seleção de alvos ou objetos de interesse a serem reconstruídos, como foi descrito nos trabalhos futuros apresentados nesta dissertação.

1. ***Automatic Recognition of Electrical Grid Elements using Convolutional Neural Networks*** [70].

   Aplicação da técnica de *Deep Learning*, Redes Neurais Convolucionais, para reconhecimento de objetos de interesse para inspeção e diagnóstico remoto para linhas férreas brasileiras. Todo material para treinamento, validação e testes foi obtido no formato de vídeo durante uma inspeção. Os resultados atingiram uma taxa de acerto maior que 93%.

2. ***Experimental Validation of Quadrotors Angular Stability in a Gyroscopic Test Bench*** [71].

   Apresentação de uma bancada giroscópica para testes de veículos aéreos não-tripulados (UAVs) sub ou sobre-atuados. Com isso, é possível realizar testes de estabilidade e diferentes estruturas para *loops* de controle evitando riscos como danos à equipamentos ou acidentes, porém mantendo os graus de liberdade para dinâmica de estabilidade angular e sem interferência da bancada de testes, visto que não influencia os momentos de inércia da aeronave. A bancada também pode ser usada de forma educacional, devido sua fácil construção e montagem.

3. ***Project and Design of Multi-Rate Loop Controllers for Fixed-Wings Aircrafts*** [72].

   Apresentação do modelo cinemático e dinâmico de um veículo aéreo não-tripulado (UAVs) com asa fixa. Além disso, são apresentados controladores Proporcional (P) e Proporciona, Integral e Derivativo para *loop* de controle

em cascata, um interno para controle de atitude angular e outro externo para controle de posição inercial. Os *loops* de controle são projetados para rolagem, arfagem, guinada, altitude e velocidade linear, no sentido positivo e negativo com relação ao nariz da aeronave. Este controle é realizado por meio da integração de diferentes níveis, através da técnica de Fechamento de *Loops* Sucessivos. Os resultados foram satisfatórios, levando em consideração que as simulações utilizaram parâmetros de uma aeronave já construída.

4. ***Signal Estimation for UAV Control Loop Identification Using Artificial Immune Systems*** [73].

   Utilização de um algoritmo seguindo a meta-heurística bio-inspirada Sistema Imunológico Artificial para estimar um sinal que melhor identifica o *loop* de controle angular para um quadrotor do tipo aéreo não-tripulado (UAVs). O *loop* de controle angular foi aproximado por um sistema de segunda ordem através do método dos mínimos quadrados. Os resultados foram satisfatórios, fornecendo um sinal rico o suficiente para fornecer uma estimativa correta do sistema.