Pontifícia Universidade Católica
do Rio de Janeiro

Marcelo de Mattos Nascimento

Utilizando reconstrução 3D densa para odometria visual baseada em técnicas de structure from motion

Dissertação de Mestrado

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós–graduação em Informática do Departamento de Informática da PUC-Rio

Orientador: Prof. Alberto Barbosa Raposo

Rio de Janeiro
Setembro de 2015

Pontifícia Universidade Católica
do Rio de Janeiro

Marcelo de Mattos Nascimento

Utilizando reconstrução 3D densa para odometria visual baseada em técnicas de structure from motion

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós–graduação em Informática do Departamento de Informática do Centro Técnico Científico da PUC-Rio.Aprovada pela Comissão Examinadora abaixo assinada.

Prof. Alberto Barbosa Raposo
Orientador
Departamento de Informática — PUC-Rio

Prof. Marcelo Gattass
Departamento de Informática — PUC-Rio

Manuel Eduardo Loaiza Fernandez
Departamento de Informática — PUC-Rio

Prof. José Eugênio Leal
Coordenador Setorial do Centro Técnico Científico — PUC-Rio

Rio de Janeiro, 17 de Setembro de 2015

Marcelo de Mattos Nascimento

Graduou-se em Engenharia de Computação na Pontifícia Universidade Católica do Rio de Janeiro em 2010.

Ficha Catalográfica

Nascimento, Marcelo de Mattos

Utilizando reconstrução 3D densa para odometria visual baseada em técnicas de structure from motion / Marcelo de Mattos Nascimento; orientador: Prof. Alberto Barbosa Raposo. — Rio de Janeiro : PUC-Rio, Departamento de Informática, 2015.

v., 59 f: il. ; 29,7 cm

1. Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática.

Inclui referências bibliográficas.

1. Informática – Dissertação. 2. Odometria visual. 3. Reconstrução 3D densa. 4. SFM. 5. Kinect Fusion. 6. Visão Computacional. 7. Realidade Aumentada. I. Raposo, Alberto Barbosa. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

Agradecimentos

Ao meu orientador e minha família. Sem a ajuda de Tia Helida e Felipe Martins esse trabalho não seria possível, Meu muito obrigado aos dois.

Resumo

Nascimento, Marcelo de Mattos; Raposo, Alberto Barbosa. **Utilizando reconstrução 3D densa para odometria visual baseada em técnicas de structure from motion**. Rio de Janeiro, 2015. 59p. Dissertação de Mestrado — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Alvo de intenso estudo da visão computacional, a reconstrução densa 3D teve um importante marco com os primeiros sistemas em tempo real a alcançarem precisão milimétrica com uso de câmeras RGBD e GPUs. Entretanto estes métodos não são aplicáveis a dispositivos de menor poder computacional. Tendo a limitação de recursos computacionais como requisito, o objetivo deste trabalho é apresentar um método de odometria visual utilizando câmeras comuns e sem a necessidade de GPU, baseado em técnicas de Structure from Motion (SFM) com *features* esparsos, utilizando as informações de uma reconstrução densa. A Odometria visual é o processo de estimar a orientação e posição de um agente (um robô, por exemplo), a partir das imagens. Esta dissertação fornece uma comparação entre a precisão da odometria calculada pelo método proposto e pela reconstrução densa utilizando o Kinect Fusion. O resultado desta pesquisa é diretamente aplicável na área de realidade aumentada, tanto pelas informações da odometria que podem ser usadas para definir a posição de uma câmera, como pela reconstrução densa, que pode tratar aspectos como oclusão dos objetos virtuais com reais.

Palavras–chave

Odometria visual; Reconstrução 3D densa; SFM; Kinect Fusion; Visão Computacional; Realidade Aumentada;

Abstract

Nascimento, Marcelo de Mattos; Raposo, Alberto Barbosa (Advisor). **Using dense 3D reconstruction for visual odometry based on structure from motion techniques**. Rio de Janeiro, 2015. 59p. MSc Thesis — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Aim of intense research in the field computational vision, dense 3D reconstruction achieves an important landmark with first methods running in real time with millimetric precision, using RGBD cameras and GPUs. However these methods are not suitable for low computational resources. Having low computational resources as requirement, the goal of this work is to show a method of visual odometry using regular cameras, without using a GPU. The proposed method is based on technics of sparse Structure From Motion (SFM), using data provided by dense 3D reconstruction. Visual odometry is the process of estimating the position and orientation of an agent (a robot, for instance), based on images. This dissertation compares the proposed method with the odometry calculated by Kinect Fusion. Results of this research are applicable in augmented reality. Odometry provided by this work can be used to model a camera and the data from dense 3D reconstruction, can be used to handle occlusion between virtual and real objects.

Keywords

Visual Odometry; Dense 3D Reconstruction; SFM; Kinect Fusion; Computer Vision; Augment Reality;

Sumário

Lista de figuras

1
Introdução

Aplicações que utilizam câmeras como um dos sensores principais estão cada vez mais presentes, como realidade aumentada em transmissões ao vivo, propagandas e jogos eletrônicos. Outra área em que as câmeras são usadas como sensores é a robótica, usada como um dos sensores para permitir o controle de drones[1] e carros autônomos[2]. O uso comum e crucial nos casos acima é estimar a posição e orientação, por exemplo: modelar uma câmera para aumentar uma cena e na robótica, para estimar a posição e orientação de um agente. Este processo de estimar a orientação e posição com uma câmera é conhecido como odometria visual. O primeiro trabalho sobre este assunto, desenvolvido por Moravec[3], data do início da década de 80 e tinha como objetivo o controle de uma sonda robótica. Blocos do seu *pipeline* continuam relevantes, como o uso de pontos de interesse. Nesse trabalho é apresentado um dos primeiros detectores de cantos.

A odometria visual é um subconjunto de um problema mais geral conhecido como *structure from motion* (SFM)[4], que consiste em recuperar pose de câmeras junto com a estrutura 3D a partir de um conjunto de imagens. O problema é mais geral, porque não se supõe que o conjunto de imagens tenha uma ordem específica. A reconstrução frequentemente é refinada com *bundle adjustment*[4] *offline*. Exemplo de aplicações que usam SFM e são refinadas off-line são o Google Maps e o Microsoft Photo Synth[5]. A odometria visual, por outro lado, se concentra em calcular a posição sequencialmente a medida que processa novos quadros. Quando além da odometria visual é criado um mapa do ambiente, surgem os métodos *visual simoultaneous localization and mapping*(SLAM) como em Davison et al.[6], Klein e Murray[7], Grisetti et al.[8]. Estes métodos apresentam maior precisão devido às restrições que podem ser impostas à trajetória, quando uma área é revisitada ou simplesmente pela não repetição do cálculo de informações já conhecidas com acúmulo de erro.

Ao contrário dos métodos visual SLAM que buscam o tempo real, os métodos de reconstrução densa que recuperam a geometria de uma cena, também baseados em odometria visual, até pouco tempo concentravam-se em métodos *offline* como Furukawa e Ponce[9]. Recentemente, métodos de reconstrução densa como Kinect Fusion[10], alcançaram taxas de tempo real para reconstruir cenas com volume de um cômodo, com o uso de GPUs. O

Kinect Fusion usa um modelo global denso e uma câmera RGBD[1], alcançando uma grande precisão na reconstrução. As aplicações são diversas, pois tanto a odometria em tempo real está disponível, como a geometria densa da cena. Por exemplo, essas informações podem servir para um agente desviar de obstáculos durante sua navegação. A geometria também pode ser usada em aplicações de realidade aumentada, por exemplo: pode ser utilizada para tratar oclusão entre real e virtual e simular com realismo a interação entre objetos reais e virtuais via simulação física.

1.1 Motivação

A motivação deste trabalho é unir os dados de uma reconstrução densa 3D previamente feita pelo Kinect Fusion, com a odometria calculada por um *pipeline* clássico de SFM com *features* esparsos, sem uso de uma GPU nem de uma câmera RGBD. Deste modo, as informações como a geometria densa de uma cena estão disponíveis e a odometria pode ser calculada em dispositivos de menor poder computacional.

1.2 Objetivo

O objetivo deste trabalho é implementar um método para o cálculo da odometria visual usando *pipeline* para *features* esparsos com ajuste local e informações de reconstrução densa, no caso, a posição 3D de pontos da reconstrução junto com as informações 2D dos *features*. Para avaliar a precisão e o desempenho do sistema aqui proposto, seus resultados serão comparados com os obtidos com o Kinect Fusion.

1.3 Organização do Trabalho

O texto desta dissertação está organizado da seguinte forma. O capítulo 2 fornece um resumo dos trabalhos relacionados com a reconstrução densa e a odometria visual não densa. O capítulo seguinte apresenta os conceitos teóricos usados, como câmera *pinhole*, movimento de corpo rígido, um método para o cálculo de mínimos quadrados não linear e uma descrição detalhada de trabalhos utilizados que são: o Kinect Fusion, o detector SURF e o descritor FREAK. O quarto capítulo descreve a arquitetura do método proposto, detalhando a construção de um mapa com informações da reconstrução densa

[1]Câmeras RGBD capturam simultaneamente a cor RGB e a profundidade de um pixel

e o *pipeline* não denso para o cálculo da odometria visual. O quinto capítulo fornece os resultados da comparação do método proposto com o Kinect Fusion em duas cenas. Finalmente, o sexto capítulo apresenta uma conclusão sobre este trabalho e propostas de trabalhos futuros.

2
Trabalhos Relacionados

A odometria visual é um tema extensamente estudado[11]. Possui uma miríade de técnicas e uso de diferentes tipos de câmeras. As câmeras podem ser RGB, RGBD, variando-se o tipo de lente, sua calibração prévia ou auto-calibração e a disposição mono, estéreo ou mais câmeras. Este texto se concentrará em técnicas em tempo real que utilizam uma única câmera comum calibrada ou uma única câmera RGBD.

Apesar da grande variedade de técnicas, as mesmas podem ser classificadas em dois tipos: as que se baseiam em informações esparsas e as que se baseiam em densas. As técnicas esparsas utilizam apenas uma parte da imagem disponível, como pontos ou uma área, que possam ser identificados em uma outra imagem. Por outro lado as técnicas densas utilizam toda a imagem. Até recentemente, a abordagem densa se restringia a métodos *offline*. Com a melhora dos recursos computacionais como o uso de GPUs, essas técnicas puderam ser implementadas em tempo real.

2.1
Odometria Visual Esparsa

Diversas técnicas de odometria visual esparsa foram implementadas com sucesso para o uso em aplicações de realidade aumentada e também para o controle de agentes na robótica. O *pipeline* típico destes métodos, como descrito por Nistér et al.[12], consiste em primeiramente extrair pontos de interesse com uso de detectores como Harris[13] e FAST[14] e depois estabelecer a correspondência entre os pontos de interesse calculados do quadro atual com o anterior. Essa correspondência é feita comparando uma área ao redor dos pontos, usando a métrica de distância do erro fotométrico ou com uso de descritores como SURF[15] e FREAK[16]. O uso de descritores torna a correspondência mais robusta, pois os mesmos são feitos para serem resistentes, por exemplo, à escala, rotação e mudanças na iluminação. Esses benefícios geralmente vêm com maior uso de recursos computacionais. Finalmente, a transformação de corpo rígido entre quadros consecutivos é estimada com métodos 2D-2D decompondo a matriz essencial, ou 2D-3D com triangulação[17] entre pares de pontos anteriores com a pose conhecida e a correspondência 2D-3D entre pontos 3D e suas projeções. Recuperar a pose utilizando estas correspondências é conhecida como *space resection*, como

descrito em Haralick et al.[18]. A vantagem do cálculo 2D-3D sobre 2D-2D é a quantidade de pontos necessários. Na decomposição da matriz essencial são no mínimo 5 pontos e para o *space resection* no mínimo 3 pontos[18]. Uma desvantagem dos métodos 2D-3D é que precisam ser inicializados com métodos 2D-2D afim de triangular o primeiro conjunto de pontos 3D. O estágio de correspondência de pontos de interesse não é livre de erros e a quantidade de pontos é importante para a de filtragem de *outliers* utilizando métodos como RANSAC[19]. No RANSAC a validade dos *inliers* é feita usando conjunto mínimo de valores para testar o modelo, ao contrário do método de mínimos quadrados que utiliza todos os dados disponíveis. Após o cálculo da pose, a precisão pode ser melhorada com *bundle adjustment* entre os N últimos quadros.

Métodos de optimização global com uso de mapa são utilizados para aprimorar a odometria reduzindo o acúmulo de erro. Os métodos SLAM concentram-se em três diferentes categorias:

- o uso de *bundle adjustment* global e um modelo de *keyframes* como no PTAM.
- Os filtros probabilísticos como em Davison et al.[6].
- As optimizações de grafo de poses como em Grisetti et al.[8].

Quando a odometria é feita apenas com a informação de uma câmera calibrada não é possível obter a escala original, entretanto é possível atualizar para uma reconstrução métrica, conhecendo alguma medida do ambiente. Esses dados podem ser obtidos com sensores IMU[1] como são comumente utilizados na robótica. Quando há, além da odometria visual, a odometria de diversos sensores, o método *Extended Kalman Filter* pode ser usado para fundir os dados, como em Weiss et al.[20] .

2.2 Odometria Visual Densa

As técnicas de odometria densa, ao contrário da esparsa, utilizam informações de toda a imagem. Há basicamente duas abordagens principais: as baseadas no erro fotométrico e as baseadas no *iterative closest point* (ICP). As abordagens baseadas no erro fotométrico calculam o alinhamento entre duas imagens, usando uma transformação de corpo rígido que melhor alinhe a projeção dos pontos 3D do quadro anterior no quadro seguinte, minimizando o erro fotométrico da projeção 2D entre ambos os pixels correspondentes. Essa

[1] *Inertial measurement unit*(IMU) são sensores como giroscópios, acelerômetros e magnetômetros que fornecem o valor instantâneo para medidas físicas

abordagem é similar ao fluxo ótico, mas entre pontos 3D e pixels da imagem. Como é necessária a coordenada 3D, este método é geralmente feito com uso de uma câmera RGBD. Deste modo um pixel na imagem 2D, conhecendo-se sua profundidade, pode ser retro-projetado para sua coordenada 3D em relação a câmera. Utilizando apenas uma câmera, o DTAM[21] consegue extrair a profundidade para um quadro, através da análise de um conjunto de imagens preenchendo uma distância d, construindo um volume de custo discreto para atribuir um valor de profundidade a um determinado pixel. Deste modo é possível construir um conjunto de *keyframes* com a profundidade conhecida e criar novos *keyframes* projetando o mapa em uma pose específica. Com a posse da profundidade gerada pelo modelo de *keyframes*, o alinhamento pelo erro fotométrico pode ser feito. Embora o método descrito em DTAM funcione com uma câmera comum, o mesmo necessita de uma GPU para funcionar em tempo real.

O segundo tipo de abordagem para odometria visual densa são os métodos baseados no ICP. Estes métodos buscam alinhar com uma transformação de corpo rígido, um conjunto de pontos 3D a outro conjunto de pontos 3D, minimizando a distância entre pontos correspondentes. Há diversas variações do ICP. Uma muito utilizada é *plane to point* [22], em que, além da coordenada do ponto utiliza-se a normal dos mesmos na métrica para o cálculo do alinhamento. Um gargalo do método ICP é o cálculo dos pontos correspondentes. A correspondência pode ser estabelecida com uma busca de vizinho mais próximo usando, por exemplo, uma estrutura hierárquica como uma *octree*. Quando o dado está disponível em mapa de profundidade vindo de uma projeção como em uma câmera RGBD, a técnica conhecida como *projective data association*[23] pode ser utilizada com desempenho superior a busca de vizinhos mais próximos. O alinhamento com ICP e *projective data association* são usadas no Kinect Fusion e serão detalhadas no Capítulo 3.

Há ainda métodos híbridos com a função de custo utilizando tanto a métrica do erro fotométrico, quanto do alinhamento via ICP, como descrito em Tykkala et al.[24].

3
Base Teórica

Neste capítulo estão os conceitos matemáticos usados nesta dissertação como o modelo de câmera utilizado, parametrização de movimento de corpo rígido e método de mínimo quadrados. Também são discutidos com mais detalhes trabalhos relacionados utilizados que são: Kinect Fusion, detector de pontos de interesse SURF e descritor FREAK.

3.1
Modelo de Câmera Pinhole

O modelo de câmera adotado descreve a projeção de uma coordenada 3D passando pelo centro de projeção de uma câmera escura ideal, onde a abertura é descrita como um ponto e nenhuma lente é usada para focar a luz. Por não modelar o efeito de lentes ou abertura, nenhuma distorção ou desfoque é descrito. Apesar das limitações do modelo utilizado, o mesmo aproxima uma câmera moderna com mundo em foco e baixa distorção.

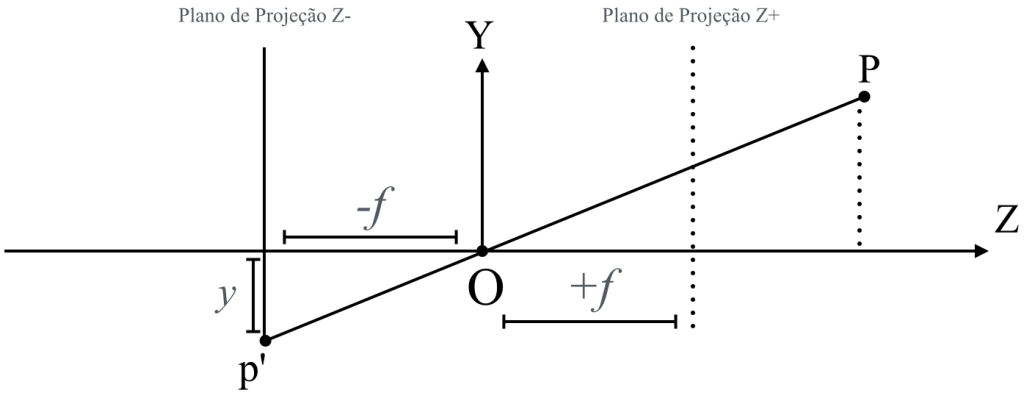


Figura 3.1: Corte lateral de uma câmera escura

3.1.1
Imagem por uma câmera escura

Pelo modelo de câmera escura, todos os raios são forçados a passar pelo centro ótico O, consequentemente, os únicos pontos que contribuem com irradiança no ponto da imagem p', com coordenada 2D $x = [x, y]^T$, são os

que estão em linha com o centro O. P é um ponto com coordenadas 3D $X = [X, Y, Z]^T$ relativas aos eixos de referência com origem no centro óptico, com eixo-z perpendicular ao plano plano de projeção e f a distância até o plano de projeção (figura 3.1) . Portanto por semelhança de triângulos tem-se a chamada equação da projeção perspectiva.

$$x = -f\frac{X}{Z} \quad , \quad y = -f\frac{Y}{Z} \tag{3.1.1}$$

3.2 Modelo Geométrico Para Formação de Imagem

A simplificação do modelo *pinhole* reduz o processo de formação de imagem ao traçado de raios de pontos de objetos em pixels. Para modelar este processo deve ser levado em conta as seguintes características:

1. Transformação entre o sistema de coordenadas da câmera e mundo.
2. Projeção das coordenadas 3-D em coordenadas 2-D de imagens.
3. Transformação da coordenada de imagem em outro sistema 2-D de coordenadas. representado por *pixels*.

3.2.1 Câmera perspectiva ideal

Considerando um ponto P com coordenadas relativas ao mundo $\mathbf{X_0} = [X_0, Y_0, Z_0]^T$, podemos rescrever o mesmo ponto nas coordenadas relativas a origem da câmera por uma transformação de corpo rígido dada por $g = (R, T)$ de $\mathbf{X_0}$:

$$\mathbf{X} = R\mathbf{X_0} + T \quad \in \mathbb{R}^3$$

Adotando a câmera *pinhole* com plano de projeção em Z+ (ver figura 3.1), pode-se re-escrever a equação da geometria perspectiva (eq. 3.1.1) como

$$\mathbf{x} = \begin{pmatrix} x \\ y \end{pmatrix} = \frac{f}{Z}\begin{pmatrix} X \\ Y \end{pmatrix}$$

e usando coordenadas homogêneas:

$$Z\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}\mathbf{X} \quad , \quad \mathbf{X} = \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

acrescentando a transformação de corpo rígido:

$$Z\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \mathbf{X_0}$$

e definindo duas matrizes como

$$K_f = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad , \quad \pi_0 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

pode-se formar a seguinte equação matricial

$$\lambda \mathbf{x} = K_f \pi_0 \mathbf{X} = K_f \pi_0 g \mathbf{X_0} \tag{3.2.1}$$

3.2.2 Parâmetros Intrínsecos

A câmera do modelo ideal tem as coordenadas de imagem centradas no centro ótico de projeção. Entretanto, na prática as imagens estão em pixels (i, j) com tamanho dos pixels diferentes dos pontos projetados no plano retinal e com a origem das coordenadas de imagem em outro ponto. Portanto é necessário modificar a equação 3.2.1 para refletir estas características.

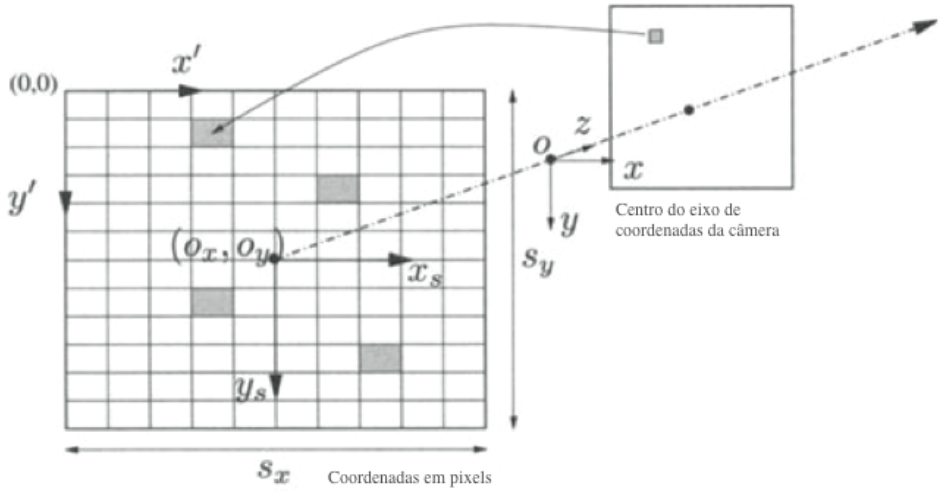


Figura 3.2: Transformando coordenadas do plano retinal para pixels

Primeiramente para tratar o tamanho diferente entre pontos (x, y) no plano retinal e pixels (i, j) é utilizada uma transformação de escala dada por:

$$\begin{bmatrix} xs \\ ys \end{bmatrix} = \begin{bmatrix} sx & 0 \\ 0 & sy \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

onde sx é o tamanho do pixel na direção x e sy o tamanho na direção y. Como a origem dos pixels (i, j) adotada neste trabalho e na literatura de visão computacional está no canto superior esquerdo (figura 3.2), é necessário adicionar uma translação na equação acima:

$$\mathbf{x}' = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & o_x \\ 0 & s_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad , \quad K_s = \begin{bmatrix} s_x & 0 & o_x \\ 0 & s_y & o_y \\ 0 & 0 & 1 \end{bmatrix}$$

onde x' e y' são as coordenadas em pixels e (o_x, o_y) é a coordenada da origem dos pixels em relação a origem do plano retinal. A matriz que transforma os pontos. $\lceil x, y, 1 \rceil^T$ para $\lceil x', y', 1 \rceil^T$ é conhecida como K_s. Deste modo a equação 3.2.1 é modificada para:

$$\lambda \mathbf{x}' = K_s K_f \pi_0 g \mathbf{X_o} \tag{3.2.2}$$

A concatenação das transformações K_s e K_f é conhecida como K e contém todos os parâmetros intrínsecos de uma câmera particular. A matriz K também é conhecida como matriz de calibração e pode ser obtida por um método de calibração como OpenCV[25].

Matriz K:

$$K = \begin{bmatrix} f s_x & 0 & o_x \\ 0 & f s_y & o_y \\ 0 & 0 & 1 \end{bmatrix}$$

substituindo a matriz K na equação tem-se

$$\lambda x' = K \pi_0 g X_o \tag{3.2.3}$$

Normalmente é utilizado apenas uma transformação projetiva π dada por

$$\pi = K \pi_0 g \quad \pi \in \mathbb{R}^{3x4}$$

e finalmente

$$\lambda x' = \pi X_o \tag{3.2.4}$$

3.3 Mapa Exponencial

Grande parte do trabalho proposto envolve resolver problemas de optimização com funções de custo na qual parte da parametrização utiliza transformações de corpo rígido. Portanto o modo de descrever essas transformações é crucial para este trabalho e deve levar em conta a facilidade

de diferenciação e que a parametrização não envolva funções que limitem o sub-espaço utilizado, tal qual uma matriz de rotação em que

$$R \in R^{3x3} \quad | \quad RR^T = I \quad \det(R) = 1$$

Este sub-espaço, mais a multiplicação de matrizes pertencentes ao mesmo é chamado de grupo das rotações SO(3)

3.3.1 Mapa Exponencial para Rotações

Dada uma trajetória $R(t) : \mathbb{R} \rightarrow SO(3)$ que descreve um movimento contínuo de rotação, a trajetória deve satisfazer a seguinte restrição:

$$R(t)\, R^T(t) = I$$

Derivando o produto acima como em[26], chega-se a equação:

$$\dot{R}(t)R^T(t) + R(t)\dot{R}^T(t) = 0$$

Manipulando a equação chega-se à: $\dot{R}(t)R^T(t) = -\dot{R}(t)R^T(t)^T$, lembrando que $(AB)^T = B^T A^T$ e que matrizes da forma $A = -A^T$ são conhecidas como antissimétricas e dão origem ao conjunto $so(3)$.

$$so(3) = \left\{ \omega \in R^3 \quad \hat{\omega} = \begin{bmatrix} 0 & \omega_3 & \omega_2 \\ -\omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix} \right\}$$

Portanto tem-se:

$$\dot{R}(t)R^T(t) = \hat{\omega}(t) \quad \omega \in R^3$$

Multiplicando a equação acima pela direita por $R(t)$, obtém-se:

$$\dot{R}(t) = \hat{\omega}R(t)$$

que tem forma de uma equação matricial diferencial ordinária de primeira ordem

$$\dot{x}(t) = Ax(t)$$

com solução para condição de contorno $R(0) = I$ dado por

$$R(t) = e^{\hat{\omega}t}$$

e com a expansão da série de Taylor

$$e^{\hat{\omega}t} = I + \hat{\omega}t + \frac{(\hat{\omega}t)^2}{2!} + \cdots + \frac{(\hat{\omega}t)^n}{n!} + \cdots$$

Fazendo $t = \|\omega\|$ e normalizando ω chega-se a seguinte identidade com a potencia de $\hat{\omega}$:

$$\hat{\omega}^2 = \omega\omega^T - I \quad \hat{\omega}^3 = -\hat{\omega} \quad \hat{\omega}^4 = -\hat{\omega}^2 \quad \hat{\omega}^5 = \hat{\omega} \quad \cdots$$

Simplificando a série do mapa exponencial para:

$$e^{\hat{\omega}t} = I + \left(t - \frac{t^3}{3!} + \frac{t^5}{5!} + \cdots\right)\hat{\omega} + \left(\frac{t^2}{2!} - \frac{t^4}{4!} + \frac{t^6}{6!} - \cdots\right)\hat{\omega}^2$$

onde a primeira série entre parênteses é de $\sin(t)$ e a segunda $(1 - \cos(t))$. Deste modo chega-se a fórmula fechada do mapa exponencial, conhecida como *Rodrigues*[4]:

$$e^{\hat{\omega}t} = I + \text{sen}(t)\hat{\omega} + (1 - \cos(t))\hat{\omega}^2, \quad \exp : so(3) \to SO(3)$$

O operador $log(R) = \omega$ é definido como[4]:

$$\|\omega\| = \text{acos}(\frac{\text{traço } (R) - 1}{2}), \quad \frac{\omega}{\|\omega\|} = \frac{1}{2\text{sen}(\|\omega\|)} \begin{bmatrix} r_{32} - r_{23} \\ r_{13} - r_{31} \\ r_{21} - r_{12} \end{bmatrix}$$

3.3.2 Mapa exponencial para Transformações de Corpo Rígido

Nesta seção são apresentadas as fórmulas que permitem o mapa de uma transformação de corpo rígido $g(R,T)$ para coordenadas exponenciais e no sentido inverso com a definição dos operadores $log(g)$ e $exp(\xi)$[4].

ξ é conhecido como vetor tangente de $g(t)$ e $\hat{\xi}$ a matriz conhecida como *twist* dado por:

$$se(3) = \left\{ \hat{\xi} = \begin{bmatrix} \hat{\omega} & v \\ 0 & 0 \end{bmatrix} \quad \hat{\omega} \in \quad so(3), \quad v \in R^3 \right\}$$

e o mapa de exp:$se(3) \to SE(3)$ é dado por:

$$e^{\hat{\xi}} = \begin{bmatrix} e^{\hat{\omega}} & \frac{(I - e^{\hat{\omega}})\hat{\omega}v + \omega\omega^T v}{\|\omega\|} \\ 0 & 1 \end{bmatrix}$$

O operador $log(g)$ pode ser calculado usando o $log(R)$ para parte da rotação

seguido sistema abaixo para o calculo de v:

$$\frac{(I - e^{\hat{\omega}})\hat{\omega}v + \omega\omega^T v}{\|\omega\|} = T$$

3.4 Métodos de Mínimos Quadrados

A técnica de mínimos quadrados estima os melhores parâmetros de um modelo utilizando medidas reais, minimizando o quadrado do erro entre o valor observado e o previsto.

3.4.1 Definição Matemática

O método busca estimar o melhor valor para um modelo $f(x, \theta) \quad \mathbb{R}^n \to \mathbb{R}$ sendo θ os valores dos parâmetros e y o conjunto dos valores observados. Minimizando com a seguinte equação:

$$argmin \sum_i^n (y_i - f(x_i, \theta))^2$$

a diferença $r_i(\theta, x_i) = y_i - f(x_i, \theta)$ é conhecida como residual. Os possíveis mínimos ou mínimo, caso f seja linear, são obtidos quando a derivada parcial da soma dos residuais é zero.

$$\sum_i^n 2r_i(\theta, x_i)\frac{\partial r_i(\theta, x_i)}{\partial \theta} = 0 \tag{3.4.1}$$

Um sistema linear pode ser representado pela seguinte forma:

$$f = X\theta - Y \quad x_i = \begin{bmatrix} x_1 & \dots & x_k \end{bmatrix} \quad \theta = \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_k \end{bmatrix} Y = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$$

usando a equação 3.4.1 com o sistema linear tem se:

$$\sum_i^n (x_i\theta - y_1){x_i}^T = 0$$

rescrevendo a equação acima na forma matricial chega-se na equação normal

$$X^T X\theta = X^T Y \tag{3.4.2}$$

O sistema linear resultante pode ser resolvido usando um método de solução para sistemas lineares como a decomposição QR.

3.4.2 Mínimos Quadrados não Linear

Caso $f(x, \theta)$ seja não linear, o problema deve ser linearizado usando um ponto próximo à solução, pois a solução da equação 3.4.1 pode não convergir, ou convergir para um mínimo não global e até mesmo para um ponto de máximo ou sela.

Sendo θ^k o ponto onde o residual r será linearizado, usando a expansão da série de Taylor de primeira ordem:

$$r_i(\theta) \approx r_i(\theta^{(k)}) + \nabla r_i(\theta^{(k)})^T(\theta - \theta^{(k)})$$

o novo valor de θ^{k+1} é o mínimo da soma dos quadrados dos resíduos linearizados:

$$\sum_i^n \left(r_i(\theta^{(k)}) + \nabla r_i(\theta^{(k)})^T(\theta - \theta^{(k)})\right)^2$$

Rescrevendo com $\theta - \theta^{(k)} = \Delta$ e $\theta^{(k+1)} = \Delta + \theta^{(k)}$

$$\sum_i^n \left(r_i(\theta^{(k)}) + \nabla r_i(\theta^{(k)})^T\Delta\right)^2$$

o sistema fica com a forma $f = X\theta + Y$ com $X = J(\theta)$, $Y = r(\theta)$ e $\theta = \Delta$, sendo $J = \partial r/\partial\theta$. Deste modo a minimização pode ser resolvida pela equação normal (3.4.2) com a fórmula:

$$J(\theta^{(k)})J(\theta^{(k)})^T\Delta = -J(\theta^{(k)})^T r(\theta^{(k)})$$

O processo iterativo é repetido até que se ache a solução com a precisão necessária ou um número máximo de iterações tenha sido excedido.

3.4.3 Mínimos Quadrados não Linear Ponderado

A presença de dados fora do modelo afeta bastante a minimização pois a mesma é baseada no quadrado do erros do residuais. Geralmente esses dados são devidos a *outliers* que não são explicados pelo modelo. Uma forma de mitigar esses residuais é criar uma função robusta que os atenue, caso passem de determinado valor. Os residuais podem ser minimizados por uma nova

função com a equação:

$$\text{argmin} \sum_i \rho\left(r_i^2\right)$$

Fazendo a equação igual a zero como em 3.4.1, a equação normal fica com a forma:

$$J(\theta^{(k)})^T W J(\theta^{(k)}) = -J(\theta^{(k)})^T W r(\theta^{(k)})$$

onde W é uma matriz diagonal em que cada elemento da diagonal é dado por $W_{ii} = \omega(r_i) \quad \omega = \rho'$. O cálculo da matriz W deve ser feito a cada nova estimativa de $\theta^{(k)}$.

A função ω é conhecida na literatura de análise de regressão como *M-Estimator*, termo cunhado por Huber et al.[27].

3.5 FREAK (Fast Retinal Keypoint)

FREAK é um descritor binário usado para medir a similaridade entre *keypoints*(pontos de interesse), composto de 3 partes comuns aos descritores deste tipo[16]:

- um padrão para amostra de pares de pontos em um *patch* centrado em um *keypoint*.
- uma métrica para medir a orientação do descritores e rotacioná-los para compensar mudanças de orientação.
- cálculo do descritor comparando os pares do padrão com alguma heurística.

3.5.1 Padrão de pares de Pontos

O descritor FREAK possui um padrão da distribuição dos pares de pontos similar a distribuição de células da retina humana (figura 3.3). O conjunto de 512 pares entre os pontos são previamente treinados; os detalhes desta técnica podem ser vistos em [16]. Os pares são escolhidos para permitir uma comparação em cascata (figura 3.4), somente se a distância de Hamming entre os primeiros n bits for menor que um limiar, que os outros próximos bits serão avaliados.

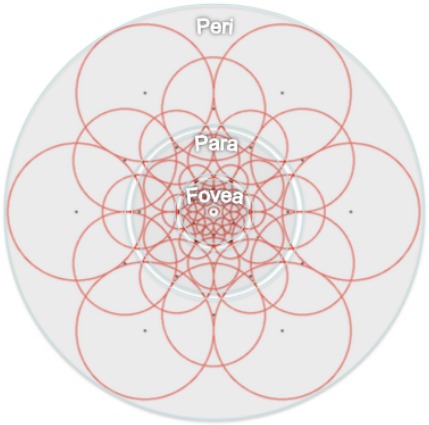


Figura 3.3: *Patch* FREAK com distribuição de pontos similar a distribuição de células na retina, os círculos representam diferentes tamanhos de *kernel* para suavização gaussiana

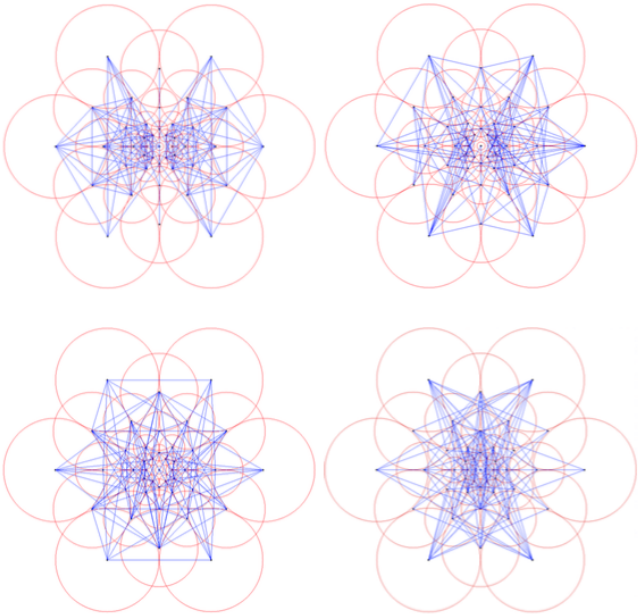


Figura 3.4: Todos os 512 pares e grupos para comparação em cascata

3.5.2 Cálculo da Orientação

O cálculo da orientação utiliza o gradiente entre um subconjunto dos pares da amostra definido por:

$$g\left(p_i, p_j\right) = \left(p_j - p_i\right) \cdot \frac{I\left(p_j, \sigma_j\right) - I\left(p_i, \sigma_i\right)}{\left\|p_j - p_i\right\|}$$

$$O = \frac{1}{M} \sum_{(p_i, p_j) \in G} g(p_i, p_j)$$

sendo G o conjunto de pares de pontos (p_i, p_j) usados no cálculo do gradiente e a função I é o ponto p suavizado pelo *kernel* σ de acordo com a posição do mesmo (figura3.3). O ângulo do descritor é dado pelo arco tangente das componentes O_x/O_y. Com o ângulo do descritor calculado, os pontos da amostra são rotacionados pelo mesmo.

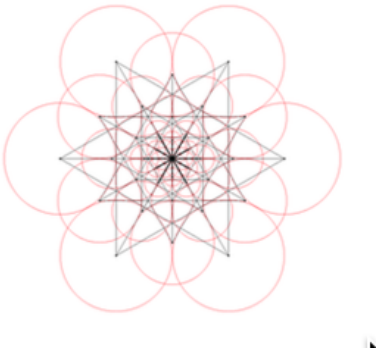


Figura 3.5: conjuntos de pares de pontos usados para o cálculo da orientação

3.5.3 Cálculo do Descritor

O cálculo do descritor é o resultado da comparação entre os pares do *patch* suavizados pelo seu respectivo kernel, dado por:

$$b = \begin{cases} 1 & I(p_j, \sigma_j) > I(p_i, \sigma_i) \\ 0 & \text{senão} \end{cases}$$

A concatenação do resultado dessas comparações resulta no descritor. O descritor pode ser comparado usando a técnica de comparações em cascata ou comparando a distância de Hamming diretamente entre os dois descritores.

3.6 SURF Detector(Fast Hessian Detector)

O detector SURF é projetado para achar *keypoints* invariantes a operações de escala em uma imagem. Isto é geralmente feito analisando a resposta a uma função na imagem em diferentes escalas.

O detector SURF implementa este espaço sem reduzir a imagem sub-amostrando sucessivos kernels gaussianos. Os cálculos são todos feitos na imagem original aumentando-se o tamanho do kernel; esses cálculos são

feitos de forma eficiente utilizando imagens integrais[1] e aproximando o filtro gaussiano por filtros de caixa. O detector SURF utiliza a resposta ao Hessiano para achar os pontos de interresse e o mesmo é dado por:

$$\mathcal{H}(x,\sigma) = \begin{bmatrix} L_{xx}(x,\sigma) & L_{xy}(x,\sigma) \\ L_{xy}(x,\sigma) & L_{yy}(x,\sigma) \end{bmatrix}$$

onde

$$L_{xx}(x,\sigma) = I(x) * \frac{\partial^2}{\partial x^2} g(\sigma)$$

L_{xx} é a convolução da imagem pela derivada de segunda ordem do *kernel* gaussiano. O detector SURF funciona analisando a resposta do determinante do Hessiano aproximado em uma vizinhança espacial constituída de 3x3x3 pontos, verificando se o ponto é o máximo emtre os vizinhos(*non-maximum suppression*). Caso seja, este ponto é marcado como um *keypoint*. Esta vizinhança constitui-se de pontos vizinhos na mesma escala e pontos nas escalas vizinhas (figura 3.6) . O detector SURF constrói uma escala fazendo um pirâmide de respostas ao Hessiano variando o valor de σ. Este espaço está subdividido em oitavas que consistem em dobrar o valor σ, e as oitavas estão subdivididas uniformemente (figura 3.6).

O cálculo das derivadas é feito aproximando a derivada do *kernel* gaussiano por filtros de caixa. A figura 3.7 mostra a configuração dos filtros de caixa com $\sigma \simeq 1.2$ com tamanho 9x9. Valores maiores de σ correspondem ao aumento do tamanho do *kernel* com a seguinte formula, $s = 1.2\frac{K}{9}$ onde K é o tamanho do kernel e $s \simeq \sigma$. O determinante aproximado é dado pela equação:

$$Det(H_{aprox}) = D_{xx}D_{yy} - (\omega D_{xy})^2$$

onde D_{xx} são aproximações para L_{xx}. O valor de ω deve ser ajustado de acordo com o tamanho do kernel a fim de manter a conservação da energia do *kernel* gaussiano[15]; em implementações práticas esse valor é mantido constante com $\omega = 0.9$.

Após achar os *keypoints*, suas posições são refinadas usando o método proposto por [28] . O método usa a resposta da função $H(x, y, s)$, ou seja, o Hessiano com suas escalas, expandindo a mesma em série de Taylor e fazendo

[1]Imagem integral é uma estrutura de dados para cálculo eficiente da soma de pixels em uma área retangular. Cada pixel da imagem integral contém a soma dos pixels a esquerda e acima,incluindo o próprio pixel da imagem original

o resultado igual a zero(ponto extremo):

$$H(x) = H + \frac{\partial H}{\partial x}^T x + \frac{1}{2} x^T \frac{\partial^2 H}{\partial x^2} x = 0$$

com solução

$$\hat{x} = -\frac{\partial^2 H}{\partial^2 x}^{-1} \frac{\partial H}{\partial x}$$

As derivadas são calculadas por diferenças finitas no espaço da função $H(x, y, s)$ e $\hat{x}$ é o novo valor interpolado para o *keypoint*.

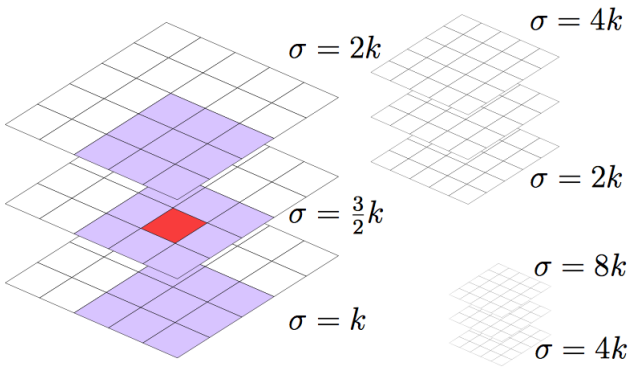


Figura 3.6: Vizinhos para *non-maximum suppression*, 3 oitavas com suas escalas

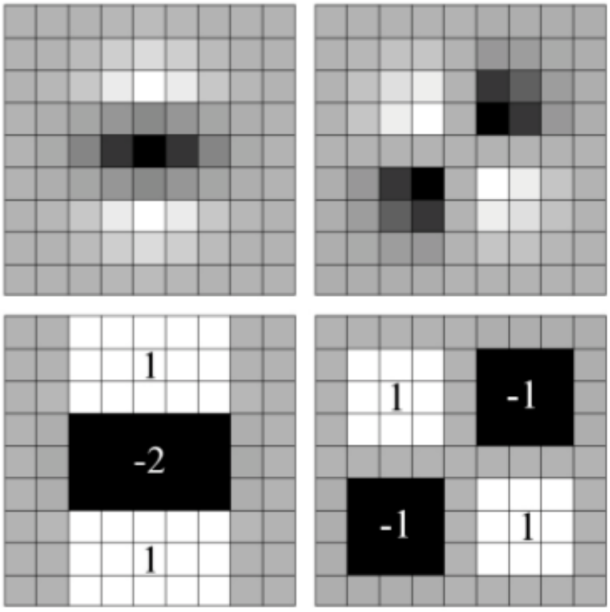


Figura 3.7: Aproximações L_{yy} e L_{xy} conhecidas como D_{yy} e D_{xy}, usando conjuntos de filtros de caixa com $s = 1.2$

3.7 Kinect Fusion

O Kinect Fusion permite a reconstrução 3D densa de uma cena interna em tempo real com uso de uma câmera RGBD e uma GPU. O método é resistente à pequenas variações de movimento de objetos da cena, podendo atualizar a reconstrução com a nova configuração da mesma. O método fornece uma odometria de alta precisão, um modelo 3D baseado em *voxels* e mapas de profundidade[2] de alta qualidade. O *pipeline* do Kinect Fusion pode ser descrito nas seguintes etapas[10]

- Obter o mapa de profundidade, suavizar o mapa preservando as arestas e calcular as normais para cada ponto.
- Estimar a pose do novo mapa de profundidade.
- Atualizar o volume global com o novo mapa de profundidade.
- Gerar um mapa de profundidade por *raycast* utilizando uma predição para próxima pose e usar esse mapa para calcular a próxima pose.

3.7.1 Processando o mapa de profundidade

O objetivo desta etapa é criar um conjunto de pontos 3D definidos como V_k e as normais N_k na coordenada da câmera, dispostos num mapa 2D U como no mapa de profundidade original.

O mapa de profundidade vindo do Kinect é bastante ruidoso sendo necessário aplicar o filtro bilateral[29] que suaviza o mesmo mas ao mesmo tempo mantém as arestas. Pequenos furos no mapa original são preenchidos com este filtro e o mesmo é fundamental para melhorar a qualidade do cálculo das normais. Em seguida os pontos podem ser calculados pela equação:

$$v_i(u) = D(u)K^{-1}u$$

onde K é a matriz de parâmetros intrínsecos da câmera infravermelha do Kinect e D o valor da profundidade deste pixel. A equação é o resultado da multiplicação da equação(3.2.3) por K^{-1} com g igual a identidade, já que o ponto está na coordenada da câmera. Já as normais n_i são calculados com o produto vetorial entre as diferenças finitas dos pixels vizinhos nas direções x e y.

[2]Mapas de profundidade são imagens que contém a distância das superfícies de uma cena para determinado ponto de vista

3.7.2 Estimando a Pose

Nesta etapa é estimada a pose utilizando o mapa de profundidade gerado pelo *raycast* entre a pose anterior e a atual. O cálculo dessa transformação é feito com uma variação do ICP conhecida como *Point to Plane Error*, utilizando a função de custo:

$$\sum_{u} (Tv_i(u) - v_{i-1}^g(u) \cdot n_{i-1}^g)^2$$

sendo T a transformação a ser achada e u os pontos 2D correspondentes entre o mapa de profundidade atual e anterior; $v_{i-1}^g(u)$ é o ponto 2D do mapa anterior transformado para coordenada global com a pose anterior e n_{i-1}^g a normal do mapa anterior. Supõe-se uma variação pequena entre um *frame* e outro e deste modo a transformação de corpo rígido $T_i = T_{i-1}\exp(\xi)|_{\xi=0}$ pode ser escrita com a aproximação do mapa exponencial ao redor de zero com a matriz:

$$\begin{bmatrix} 1 & \omega_3 & -\omega_2 & v_1 \\ -\omega_3 & 1 & \omega_1 & v_2 \\ \omega_2 & -\omega_1 & 1 & v_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

A equação normal do problema linearizado é reduzida em GPU. O sistema linear resultante é calculado em CPU com a fatoração Cholesky. Finalmente a transformação é calculada com $T_{i-1}\exp(\omega, v) = T_i$.

A associação dos pixels entre os mapas de profundidade é feita transformando o ponto 3D v_{i-1}^g com a transformação para coordenadas da câmera atual dada por T_i^{-1}. O ponto é projetado e verifica-se se o mesmo pertence ao *frustum* de visão. Caso seja, é calculada a distância entre a coordenada global do ponto v_i^g, que possui a coordenada 2D da projeção do ponto anterior, com v_{i-1}^g. O ângulo entre as normais de ambos os pontos também é calculado. De posse dos dois valores verifica-se se estão abaixo de um limiar e a correspondência entre os pontos com mesma coordenada 2D pode ser estabelecida. A interação entre a correspondência de pontos e o cálculo da pose via ICP é repetida até a precisão desejada.

3.7.3 Integração Volumétrica

Após calcular a pose global, os pontos do mapa de profundidade (não processados com filtro bilateral) são convertidos para uma coordenada global.

O Kinect Fusion usa um grid 3D denso para integrar os mapas de profundidade medidos, conhecido como TSDF(*Truncated Surface Distance Function*). O TDSF é subdivido uniformemente em um conjunto de *voxels* com uma certa quantidade por eixo. A quantidade e o volume físico que um *voxel* possui define a resolução do volume reconstruído.

Os valores dos *voxels* são calculados percorrendo cada *voxel* do grid e projetando-os na pose atual. Caso esteja no *frustum* de visão o valor é atualizado com a seguinte formula:

$$
\begin{aligned}
sdf_i &= \|t_i - v^g\| - D_i(p) \\
tsdf_i &= \begin{cases} min(1, sdf_i/t_{max}) & \text{se} \quad sdf_i > 0 \\ max(-1, sdf_i/t_{min}) & \text{senão} \end{cases} \\
w_i &= min(w_{max}, w_{i-1} + 1) \\
tsdf^{med} &= \frac{tsd_{i-1}w_{i-1} + tsd_i w_i}{w_{i-1} + w_i}
\end{aligned}
$$

A primeira equação mede a distância do *voxel* com a profundidade medida(superfície física), consultando o mapa de profundidade com a projeção do voxel. $D_i(p)$ é o valor medido da projeção, $\|t_i - v^g\|$é a distância do *voxel* v^g para o centro de projeção atual dado por t_i (translação da pose atual), $tsdf_i$ é o valor truncado por uma constante e $tsdf^{med}$ a média deste valor ao longo do tempo. Na figura 3.8 há uma ilustração do preenchimento dos *voxels*, a superfície implícita está indicada em vermelho.

3.7.4 Raycast

Nesta etapa é gerada uma nuvem de pontos de alta qualidade, utilizando o volume reconstruído. Este conjunto de pontos é gerado pelo *raycast* por pixel do volume TSDF. O *raycast* procede da seguinte maneira: para cada pixel u do *frustum* de visão atual, é calculada a direção do raio pela diferença da retro-projeção do pixel u com profundidade 0 e 1 (Algoritmo 1).

Algoritmo 1 Cálculo da direção do raio

1: $\text{raio}^{inicio_g} \leftarrow T_i K^{-1}(u, 0)$ ▷ início do raio coordenada global
2: $\text{raio}^{proximo_g} \leftarrow T_i K^{-1}(u, 1)$ ▷ ponta do vetor coordenada global
3: $\text{raio}^{inicio_{grid}} \leftarrow \text{converte}(\text{raio}^{inicio_g})$ ▷ converte para coordenadas do grid
4: $\text{raio}^{proximo_{grid}} \leftarrow \text{converte}(\text{raio}^{inicio_g})$
5: $\text{raio}^{direcao} \leftarrow \text{normaliza}(\text{raio}^{proximo_{grid}} - \text{raio}^{inicio_{grid}})$

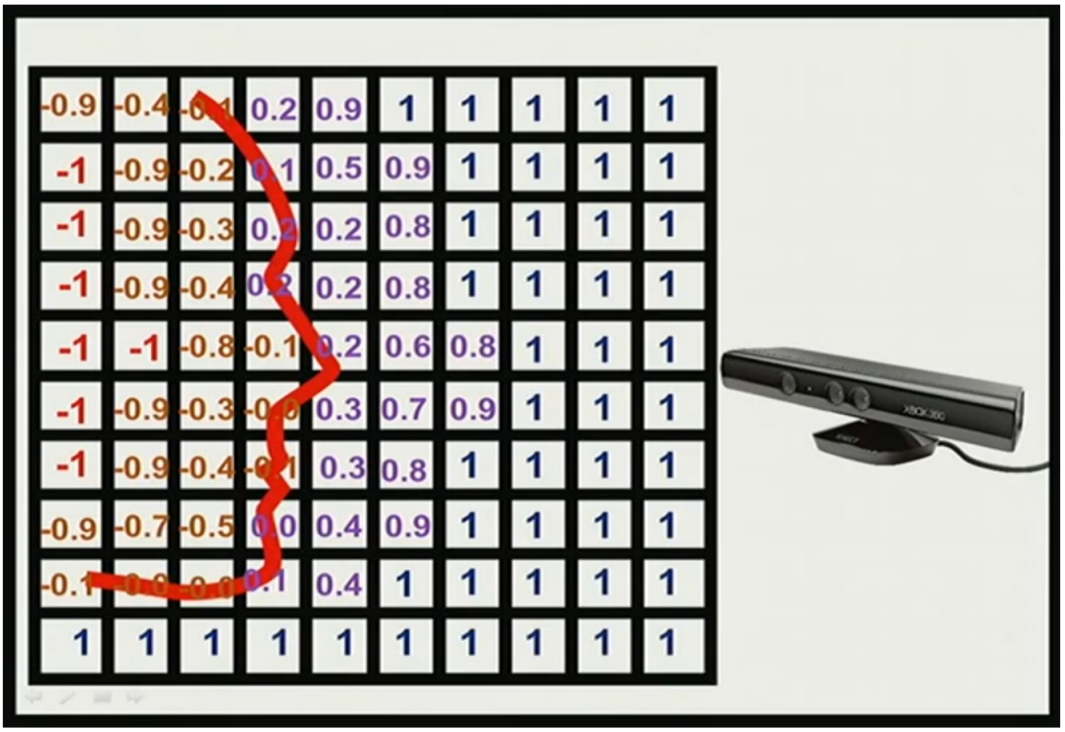


Figura 3.8: Representação do volume TSDF. Cada elemento do grid representa um *voxel*; a superfície está implicitamente representada pelo *zero-crossing* onde há a troca do sinal do valor TSDF. Os valores TSDF aumentam em módulo com a distância da superfície; valores negativos estão atrás e positivos na frente.

O raio deve marchar com um passo suficiente para atingir o próximo *voxel* e o passo pode ser acelerado com o tamanho da constante de truncamento. Achando um *voxel* com valor menor que zero o passo é ajustado para atingir o próximo, garantindo que seja encontrado a troca de sinal do valor TSDF, achando a superfície implícita. Ao cruzar a superfície implícita, a posição do zero é interpolada linearmente entre o *voxel* atual e o anterior. O cálculo da normal é feito com a derivada numérica entre as diferenças finitas dos valores TSDF.

4
Método Proposto

O método consiste em dois blocos: o primeiro, a construção de um mapa *off-line* usando a reconstrução densa e o segundo a odometria em tempo real utilizando este mapa com *keypoints*/descritores esparsos.

4.1
Mapa off-line

No Kinect há duas câmeras, uma infravermelha que é usada para gerar o mapa de profundidade e a RGB. As câmeras estão rigidamente acopladas, ou seja há uma transformação de corpo rígido que leva uma pose da câmera RGB para câmera IR. Deste modo é possível usar a pose dada pela odometria do Kinect Fusion, por exemplo para projetar um ponto 3D X_w dado na coordenada do mundo, na câmera RGB. Usando a pose atual fornecida pelo Kinect Fusion T_w^{ir}, que leva um ponto na coordenada do mundo para coordenada da câmera IR, concatenada com a pose relativa entre as câmeras T_{ir}^{rgb} é possível projetar um ponto X_w na câmera RGB.

$$x = K_{rgb}\pi_o T_{ir}^{rgb} T_w^{ir} X_w$$

4.1.1
Criando Mapa

Utilizando o Kinect Fusion é feita uma reconstrução conforme os passos descritos na seção 3.7. O Kinect Fusion funciona a 30FPS, mesma taxa de quadros das câmeras do Kinect, portanto é possível calcular uma pose para cada quadro capturado. Com essas informações, mais o *stream* RGB (figura 4.1) é construído um mapa de *keyframes* constituído de um conjunto de *keypoints*, descritores e uma pose. Esse conjunto será usado para o cálculo da odometria na próxima etapa. O início do processo começa lendo o *stream* das câmeras IR e RGB sequencialmente, (algoritmo 2, linha 2 e 3). Após é executado o *pipeline* do Fusion (linha 4). Verifica-se se o *tracking* é válido (convergência dado pela API do Kinect Fusion) , se o intervalo de tempo entre a captura das duas imagens é inferior a um limiar e finalmente se houve um movimento com distância maior que 10cm ou ângulo maior que 10 graus entre a última imagem armazenada no *buffer* e a atual. Após isso, é feito outro *raycast* do volume usando a pose da câmera RGB e seus parâmetros intrínsecos, com

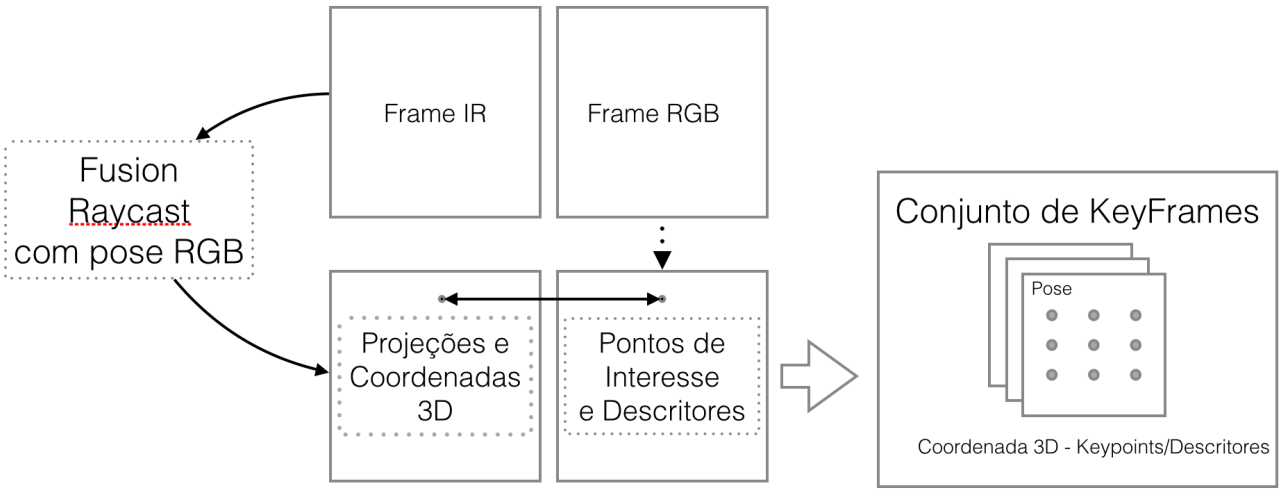


Figura 4.1: Fluxo da construção do mapa *off-line*

esses dados é gerada uma imagem onde os pixels possuem a coordenada 3D do mundo, (linha 7). As imagens RGB e 3D são guardadas num *buffer*. Após um determinado número de quadros o processo é finalizado e o *buffer* é gravado em disco. Posteriormente, para cada conjunto de imagens 3D e RGB são gerados seus respectivos pares de *keypoints* SURF e descritores FREAK.

Os *keypoints* SURF e descritores FREAK, linhas 12 e 13, são calculados utilizando a biblioteca OpenCV[25]. O detector SURF do OpenCV segue a descrição da seção 3.6. O cálculo do *non max suppresion* é feito comparando todos vizinhos pixel por pixel de forma simples, embora haja algoritmos de maior desempenho. Isso permite que a implementação seja trivialmente paralelizável. Os parâmetros utilizados no cálculo dos *keypoints* são: valor acima 300 para determinante do Hessiano, 4 oitavas e duas escalas por oitava. A implementação do descritor FREAK utiliza o tamanho dos *keypoints*, (no caso do *keypoint* SURF, o tamanho do *kernel* utilizado na escala onde o *keypoint* foi encontrado), para ajustar a posição e área de amostra (seção 3.5.1). A escala na área da amostra permite que o descritor FREAK seja resistente a operações de escala. Outra optimização é a substituição dos *kernel* gaussianos, por uma média dos pontos na área da amostra usando imagem integral.

Algoritmo 2 Criando Mapa de Keyframes

loop
  $\text{Im}_{ir} \leftarrow$ lerImagemIR
  $\text{Im}_{RGB} \leftarrow$ lerImagemRGB
  fusion(Im_{ir}) ▷ executa pipeline do fusion
  $T_w^{ir} \leftarrow$ fusion.lerPose(Im_{ir})
  se movimentoAngulo(T_{ir}) **e** timeStampOk **e** fusion.trackOk **então**
    $\text{Im}_{3d} \leftarrow$ fusion.raycast($T_{ir}^{rgb} T_w^{ir}, K_{rgb}$) ▷ novo raycast
    buffer $\leftarrow$ ($\text{Im}_{rgb}, \text{Im}_{3d}, T_{ir}^{rgb} T_w^{ir}$) ▷ salva imagem 3D, imagem 2D e pose RGB
  fim se
fim loop
para todo ($\text{Im}_{rgb}, \text{Im}_{3d}$, pose) $\in$ buffer **faça**
  keypoints $\leftarrow$ surf(Im_{rgb})
  descritores $\leftarrow$ freak(keypoints,Im_{rgb})
  mapa $\leftarrow$ (pose,keypoints,descritores,Im_{3d}) ▷ adiciona tupla ao mapa
fim para

4.2 Odometria em Tempo Real

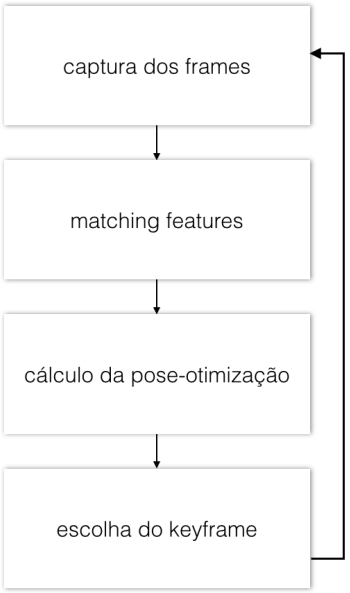


Figura 4.2: Fluxo do cálculo da odometria

A odometria visual fornece uma pose $g(R,T)$ utilizando o *stream* da câmera RGB e o mapa de *keyframes*. A pose é calculada nas seguintes etapas (figura 4.2)

- Ler imagem do *stream* RGB (algoritmo 3, linha 2)
- Calcular os *keypoints* SURF e descritores FREAK; buscar os descritores similares do *keyframe* atual e imagem e criar o conjunto dos pontos 2D e 3D (linhas 3-5)
- Calcular a pose minimizando o erro de reprojeção dos pontos 3D na imagem atual (linha 6)
- Localizar *keyframe* mais próximo seguindo uma métrica entre ângulo e distância da pose atual com a pose calculada (linha 7)

Algoritmo 3 Odometria

```
1: loop
2:     Im_RGB ← lerImagemRGB
3:     kps ← surf(Im_rgb)
4:     desc ← freak(keypoints,Im_rgb )
5:     (2Dkey,2Datual,3D) ← matching(k.keypoints,k.descritores,kps,desc)
6:     pose ← calcPose(2Dkey,2Datual,3D,k.pose,poseAtual)
7:     k ← buscaProximo(mapa,pose)
8: fim loop
```

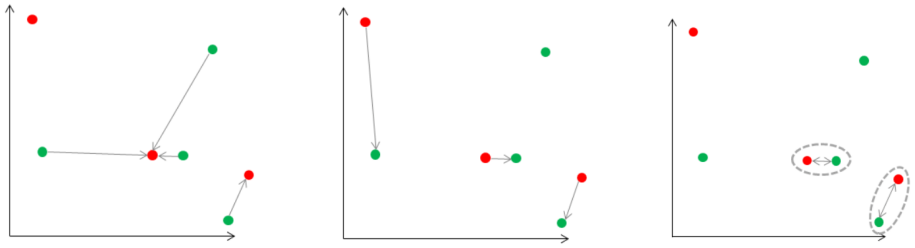

4.2.1 Matching

Nesta etapa é feita a associação entre os descritores do quadro atual e os descritores do *keyframe*. Primeiro são calculados os *keypoints* SURF (algoritmo 3 linha 3) e após os descritores FREAK, linha 4. Em seguida é feita uma busca dos vizinhos mais próximos(NN) utilizando como métrica a distância de Hamming, que é dada por $\mathbf{soma}(\mathbf{xor}(s_1, s_2))$ sendo s_1 e s_2 duas *strings* binárias, **xor** o **ou** exclusivo e **soma** conta os números 1 na *string* resultante.

A busca é feita por força bruta comparando cada descritor do conjunto do quadro atual com o conjunto dos descritores *keyframes*, selecionando o descritor com menor distância para cada par. Apesar de ser feita por força bruta foi utilizada a classe BFMatcher do OpenCV que possui algumas optimizações. O cálculo das distâncias entre as combinações é paralelizado, embora a busca pelo menor par continue sequencial. O cálculo das distâncias de Hamming utiliza as instruções SIMD quando disponível, utilizando **xor** com várias palavras de

32 ou 64 bits ao mesmo tempo, dependendo da arquitetura. A **soma** também é feita com instruções SIMD contando o número de 1 em várias palavras ao mesmo tempo. Uma estrutura hierárquica não cabe para este problema devido a dimensão do descritor ser 512. São possíveis apenas buscas aproximadas como FLANN[30], sem ganho significativo de desempenho para quantidade dos *keypoints* normalmente encontrada pelo detector SURF.

Após a busca pelo vizinho mais próximo é feita uma filtragem selecionando as correspondências mais fortes, verificando se a menor distância para o descritor i do conjunto A é o descritor j do conjunto B, e se essa relação é válida no sentido oposto (figura 4.3). Esse cálculo é feito de forma aproximada percorrendo as tuplas calculadas da busca nn(A,B), para cada tupla (a_i, b_j, d) atualizar a tupla $(b_j, a_k, dmin)$ com (b_j, a_j, d) caso $d < dmin$. As tuplas resultantes formadas contém os pares filtrados. A figura 4.3 exemplifica o processo com distância euclidiana, mas o mesmo exemplo é válido para a distância de Hamming que satisfaz os axiomas de distância, notadamente no exemplo a simetria $d(a, b) = d(b, a)$.



(a) nn(verde,vermelho) (b) nn(vermelho,verde) (c) cross

Figura 4.3: Procedimento de filtragem das correspondências mais fortes

4.2.2 Cálculo da Pose

O cálculo da pose é feito minimizando o erro de reprojeção dos pontos 3D encontrados no *matching* e projetados na imagem atual. O residual é dado por:

$$r_i = \pi(X_i) - x_i$$

onde π é a projeção (seção 3.2.4), X_i a coordenada 3D e x_i o pixel encontrado no quadro atual (este residual fornece duas equações). Esse é um problema de mínimos quadrados não linear contendo *outliers*, resolvido de forma semelhante como na seção 3.4.3. A projeção pode ser parametrizada usando o espaço tangente da transformação de corpo rígido, e supondo uma pequena

transformação $\xi \approx 0$ em relação a uma pose anterior $G(R,T)$. Esta suposição permite calcular a jacobiana da projeção π de modo simples com a fórmula como descrito em [31].

$$\frac{\partial \pi(e^{\hat{\xi}} \oplus G \oplus X)}{\partial \xi} = \left.\frac{\partial \pi(X')}{\partial X'}\right|_{X'=G\oplus X=g} \left.\frac{\partial e^{\hat{\xi}} \oplus G \oplus X}{\partial \xi}\right|_{\xi=0}$$

$$\begin{pmatrix} f_x/g_z & 0 & -f_x g_x/g_z^2 \\ 0 & f_y/g_z & -f_y g_y/g_z^2 \end{pmatrix} (I_3 \quad -\hat{g})$$

A simplificação $\xi = 0$ não é sempre o caso do problema atual. O uso de um descritor robusto permite que o *matching* funcione com diferença considerável entre as poses e a parametrização deve levar em conta esta caraterística. Deste modo a projeção π é parametrizada com a equação:

$$K(e^{\hat{\omega}}X + T) = X_\pi \quad x = {}^{X^1_\pi}/_{X^3_\pi} \quad y = {}^{X^2_\pi}/_{X^3_\pi}$$

com a variável $\mathbf{x} = (\omega_1, \omega_2, \omega_3, t_1, t_2, t_3)$

Este problema é resolvido com auxilio da biblioteca Ceres Solver[32], com uso de derivadas automáticas[33] e com o método Levenberg-Marquardt(LM) variação do Gauss-Newton, que é mais robusto para soluções iniciais mais distantes. O método LM aumenta a matriz Hessiana com uma matriz diagonal D, fazendo os valores da diagonal variarem entre o método de Gauss-Newton $D = 0$ e o gradiente descendente $D = \infty$, de acordo com a solução encontrada no momento. A biblioteca Ceres-Solver implementa o LM como descrito em [34] com algumas variações, sendo as principais o cálculo da matriz diagonal com escala e o suporte às funções de perda, algoritmo 4. A qualidade da solução atual, calculada na linha 11, é o custo da solução atual menos a anterior, sobre a diferença estimada pelo modelo linearizado. O valor linearizado (denominador) sempre é positivo, portanto, caso $\varrho < 0$ (linha 7) o custo aumentou e o raio μ caminha na direção gradiente descendente (linha 19). Caso $\varrho > 0$ o raio é ajustado de acordo com a qualidade encontrada (linha 17). O custo da solução $F(x)$ é dado pela soma do quadrado dos residuais $F(x) = \frac{1}{2} r(x)^T r(x)$. A matriz diagonal D (linha 6) é calculada como a raiz do produto do raio μ com o valor da norma da coluna da jacobiana, sendo o valor da coluna i dado por J_i, $D_i = \mu \left\| J_i \right\|$. A escala no residual e jacobiana são calculadas usando a função de perda $\rho(F_i(x) = r_i(x)^2)$. A função que é usada na escala é a derivada $\rho'(F_i(x))$ e o residual e a jacobiana são multiplicados por $r_k = \sqrt{\rho'(F_k)} r_k$ e $J_k = \sqrt{\rho'(F_k)} J_k$ (linha 2 e 14), sendo r_k valor k do residual e J_k linha k da

Jacobiana. A função de perda utilizada é *Tukey's biweight*[35], com seguinte comportamento:

$$\rho'(x) = \begin{cases} x\left(1 - \frac{x^2}{c^2}\right) & \text{se} \quad x < c \\ 0 \quad \text{se} \quad x > c \end{cases}$$

Esta função suprime os *outliers*. Como não é convexa, um novo mínimo local difere do valor ótimo. O valor ótimo de c é 4,68 para residuais com variância $\sigma = 1$; deste modo deve ser feita uma escala para o residual r_i/σ. O valor de σ é aproximado por $\mathrm{MAD(r)}/0.6745$ sendo MAD *median of absolute deviation* calculado por:

$$\text{MAD} = \frac{1}{n}\sum_{i}^{n} |x_i - \text{mediana}(x)|$$

O valor de σ é calculado apenas uma vez após a convergência. O sistema linear que calcula o passo Δ (linha 6) é chamado sistema linear regularizado aumentado, na forma:

$$\begin{pmatrix} J \\ D \end{pmatrix} \Delta = \begin{pmatrix} -r \\ 0 \end{pmatrix}$$

O sistema resultante é resolvido por mínimos quadrados com decomposição QR. Fazendo $A = (J, D)$, $\Delta = x$ e $b = (-r, 0)$ e decompondo $A = QR$, chega-se ao sistema $Rx = Q^T b$; x é calculado via retro-substituição. São usados como solução termos do tamanho original de J. Após a solução $x = (\omega_1, \omega_2, \omega_3, t_1, t_2, t_3)$ convergir, a parte devido a rotação é convertida para matriz por $e^{\hat{\omega}}$. A solução atual é usada como solução inicial para a próxima interação. Caso não haja convergência $x_0 = \mathbf{0}$.

Algoritmo 4 Levenberg-Marquardt

1: $k \leftarrow 0 \quad v \leftarrow 2 \quad x \leftarrow x_0$
2: J(x)=scaleloss(J(x)) r(x)=scaleloss(r(x))
3: $g \leftarrow J(x)^T r(x)$
4: achou $\leftarrow \|g\|_\infty < \varepsilon_1 \quad \mu \leftarrow \mu_0$
5: **enquanto não** achou **e** $(k < k_{max})$ **faça**
6: $k \leftarrow k + 1$ Resolva $\min(\|Jx - r\| + \|D\|)$
7: **se** $\|\Delta\| \leq \varepsilon_2 (\|x\| + \varepsilon_2)$ **então**
8: achou $\leftarrow$ **true**
9: **senão**
10: $x_{novo} \leftarrow x + \Delta$
11: $\varrho \leftarrow (F(x) - F(x_{novo}))/(\frac{1}{2}\Delta^T(\Delta - g))$
12: **se** $\varrho > 0$ **então**
13: $x \leftarrow x_{novo}$
14: J(x)=scaleloss(J(x)) r(x)=scaleloss(r(x))
15: $g \leftarrow J(x)^T r(x)$
16: achou $\leftarrow \|g\|_\infty < \varepsilon_1$
17: $\mu \leftarrow \mu * max(1/3, \quad 1 - (2 - \varrho)^3) \quad v \leftarrow 2$
18: **senão**
19: $\mu \leftarrow \mu v \quad v \leftarrow 2v$
20: **fim se**
21: **fim se**
22: **fim enquanto**

4.2.3 Escolha do Keyframe

O *keyframe* que será usado no *matching* é o que possui o valor mínimo da fórmula $(2 - \cos(a))^2 d$, onde d é a distância entre a origem da pose atual P e *keyframe* k_i e a o ângulo entre as componentes z (figura 4.4). As poses P e P_k (pose do k-ésimo *keyframe*) levam as coordenadas do mundo para a câmera e podem ser vistas como matrizes de mudança de base ortonormal com mudança de origem:

$$\begin{bmatrix}[X] & [Y] & [Z] & [T]\end{bmatrix}$$

A componente Z aponta para o centro de projeção (ver figura 3.2) e T é a distância para origem da coordenada do mundo. O coseno do ângulo a e a

distância d são dados por:

$$cos(a) = \frac{Z_k \cdot Z_p}{\|Z_k\| \, \|Z_p\|} \quad d = \|T_k - T_p\|$$

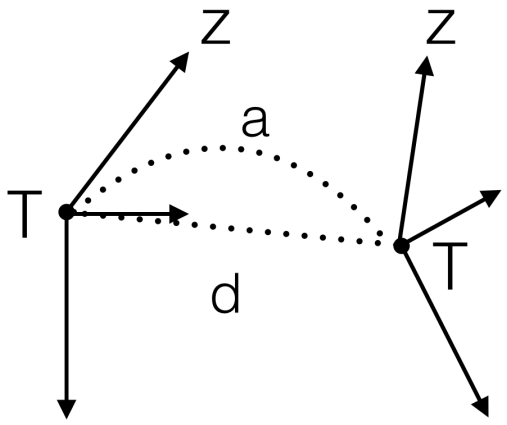


Figura 4.4: Ângulo e distância entre duas poses

5
Resultados

Neste capítulo é apresentada uma comparação do método proposto com a odometria calculada pelo Kinect Fusion e uma análise do desempenho.

5.1
Experimento

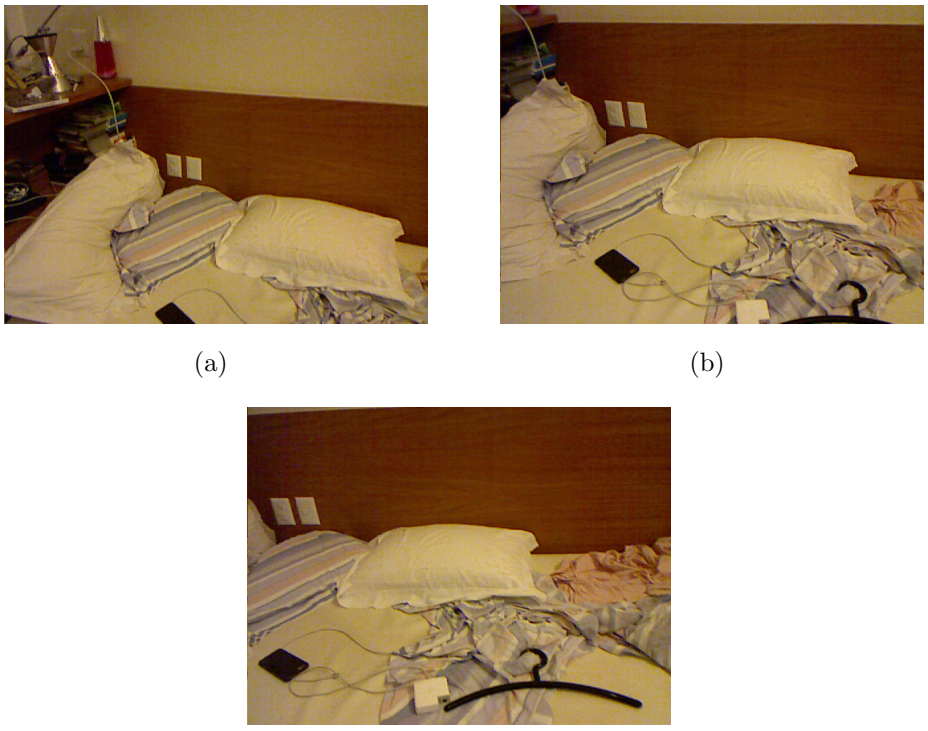
O experimento realizado possui duas partes: a montagem do mapa e a captura de quadros e dados da odometria do Kinect Fusion para comparação com o método proposto.

O mapa foi feito utilizando os seguintes parâmetros para reconstrução do Kinect Fusion: resolução de 128 *voxels* por metro, com um volume de 512x384x512 (X, Y, Z) *voxels*, resultando num volume de 4x3x4 metros e resolução por *voxel* de 8x10x8 milímetros. A captura dos quadros utilizados na montagem do mapa foi feita a cada movimento de 15cm entre os quadros ou ângulo de 10 graus, ambos medidos com base na odometria do Fusion. Como há apenas o critério da distância e ângulo para captura dos quadros, a captura manual procurou evitar que a câmera passasse pela mesma região várias vezes. O cálculo dos parâmetros intrínsecos das câmeras RGB e IR e da pose relativa entre ambas foi feito com MRPT[36].

Os quadros capturados para comparar a odometria diferem dos utilizados no mapa, e foram capturados de forma contínua a 20FPS com resolução 640x480 pixels. É gravada uma imagem Im_i e uma pose P_i para cada interação do Fusion formando um conjunto de imagens e poses. Posteriormente utilizando este conjunto de imagens, é calculada uma pose P'_i de forma contínua para cada imagem Im_i com o método proposto. A precisão da odometria é medida calculando-se a distância entre as poses P_i e P'_i e o ângulo entre as mesmas. É utilizado o mesmo volume de reconstrução em ambas as etapas, portanto, estão no mesmo sistema de coordenadas. A taxa de 20FPS usada para testar o método proposto é inferior a taxa de 30FPS alcançada pelo Kinect Fusion durante a reconstrução. Este valor foi escolhido para testar um maior movimento entre dois quadros e também para representar um valor mais realista do desempenho atual do método com média de 19FPS para a cena 2.

O manuseio do Kinect foi feito com apenas uma mão, evitando movimentos bruscos.

5.2 Cena 1



(a) (b)

(c)

Figura 5.1: Cena 1

O primeiro teste, foi feito em quarto com luz artificial rico em texturas (funcionamento do *matching*) e variação de geometria (funcionamento do Kinect Fusion), ver figura 5.1. Durante o teste foram feitos poucos movimentos bruscos.

5.2.1 Análise dos resultados

Durante a fase de *matching* e criação do mapa o detector SURF achou uma alta quantidade de *keypoints*, média de 652 por quadro. Na figura 5.2 há uma visualização do *matching* mostrando, à direita, uma imagem de um *keyframe* e, à esquerda, um quadro onde é calculada a odometria. Os pontos brancos são os *keypoints*/descritores e as linhas as correspondências encontradas entre ambos. Na região do quadro quadriculado pode-se notar a presença de vários *outliers*.

Na figura 5.3 é mostrada a mediana do erro de reprojeção usado no cálculo da pose. A área sombreada é o intervalo entre mediana+MAD e mediana-MAD. Valores altos desses parâmetros indicam a presença de *outliers*. Embora haja vários picos desses valores, a distância e orientação para pose calculadas em

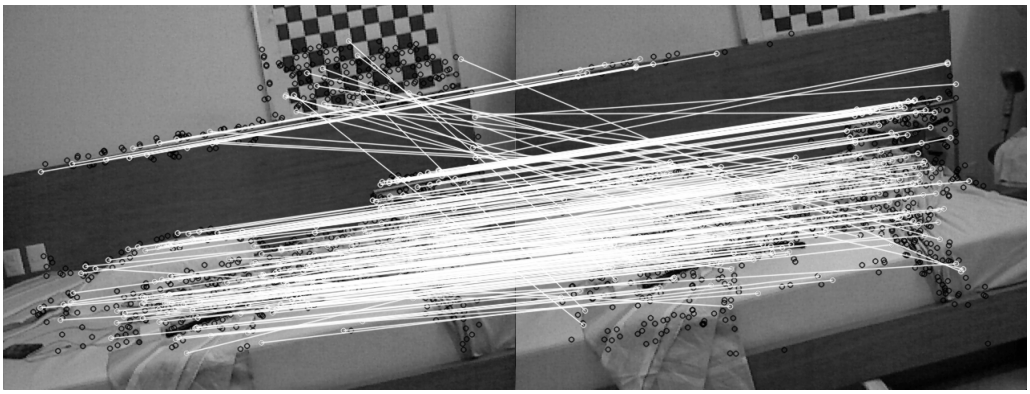


Figura 5.2: *Outliers* presentes no *matching*

relação ao Fusion se mantiveram, respectivamente, menor que 4cm (figura 5.4) e menor que 2 graus (figura 5.5) por todo o experimento. A função de perda conseguiu limitar a influência dos *outliers*. O parâmetro da função de perda justamente por essa grande variação na mediana foi mantido fixo. O uso de descritores robustos permite o cálculo da pose com ângulo de até 25 graus entre as componentes Z do *keyframe* e a pose calculada.

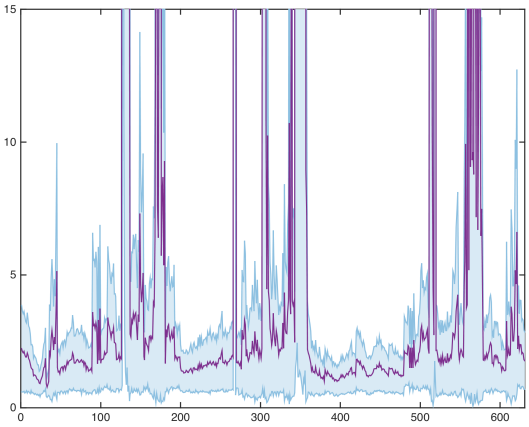
Tabela 5.1: Estatística dos ângulos em relação Fusion, cena 1

	média	max	min	std
pitch	0,5	1,6	0	0,4
yaw	0,4	1,9	0	0,3
roll	0,2	0,9	0	0,2

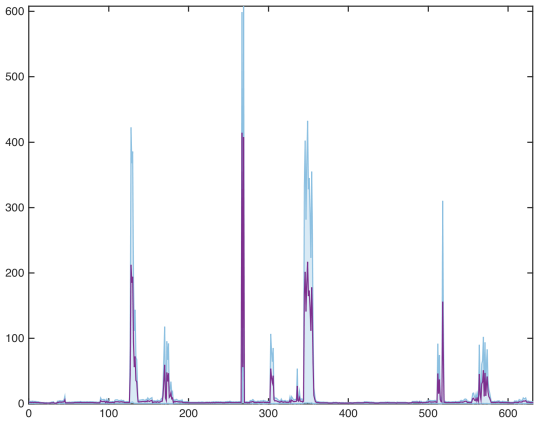
O valor máximo para a distância entre a pose calculada pelo Fusion e pelo método proposto foi 3,75cm com mínimo de 0,14cm. O valor médio foi de 1,83cm com mediana de 1,83cm e desvio padrão de 0,67cm (figura 5.4). A estatística para os dados dos ângulos (figura 5.5) está na tabela 5.1.

Na execução (tabela 5.2) o tempo dominante ficou com o detector SURF. Apesar de ser constante, o cálculo do Hessiano possui 41 operações aritméticas para o cálculo de um determinante Det(H) (figura 3.7) e é repetido em várias escalas para cada pixel (4 oitavas com 2 escalas por oitava).

Outra parte custosa é o *non maximal supression* com 26 comparações, no caso do *keypoint* ser um ponto de máximo em sua oitava, junto com a interpolação para ajuste da posição (resolução de um sistema linear 3x3). A quantidade de *keypoints* impacta diretamente o *matching* $O(n^2)$ e o cálculo da pose $O(mn^2)$ (parte mais custosa do cálculo da pose é a resolução via mínimos quadrados com decomposição QR), onde m é a quantidade de



(a) detalhes



(b) visão geral

Figura 5.3: Eixo Y, mediana do erro de reprojeção em roxo. Área em ciano, valor da mediana limitada por MAD. Eixo X, imagens

Tabela 5.2: Tabela com tempo de execução dos componentes principais do método proposto

LM	3,94s	18,6%
SURF	11,9s	56%
FREAK	1,03s	4,6%
NN	3,97s	19%
Total	21,2s	
FPS*	29,8	

*Quadros por segundo

parâmetros da jacobiana e n a quantidade resultante de *keypoints* filtrados pela correspondência mais forte. O descritor FREAK embora tenha um custo

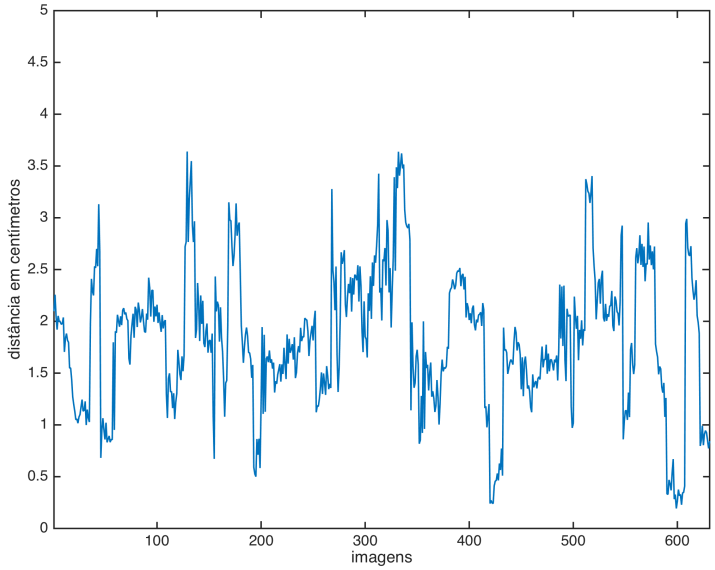


Figura 5.4: Distância em centímetros em relação ao Fusion

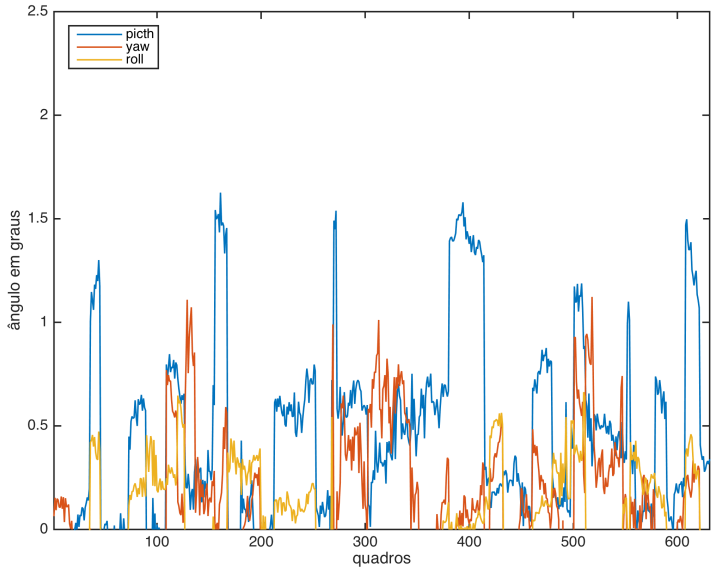
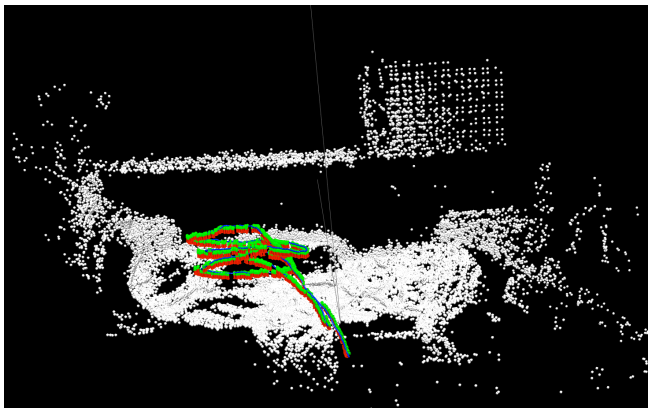


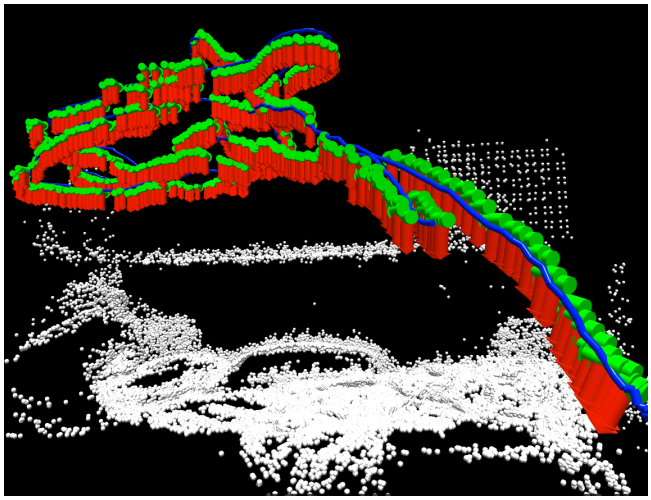
Figura 5.5: Ângulos de Euler sobrepostos da pose calculada em relação ao Kinect Fusion. Convenção: roll(X)-pitch(Y)-yaw(Z)

maior por pixel, são 37 somas de área (figura 3.3) usando imagem integral mais as 512 diferenças das combinações (figura 3.4), o mesmo só é calculado para cada *keypoint*, menos de 700 por imagem.

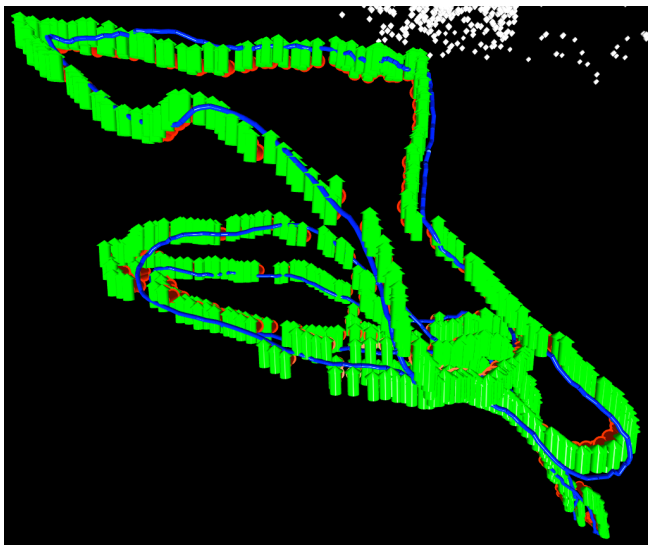
Na figura 5.6 há uma visualização do experimento. Pontos brancos representam *keypoints*/descritores do mapa e a linha azul é trajetória da câmera calculada pelo Kinect Fusion. As setas verdes (eixo Z) e vermelha (eixo Y), dispostas na posição da pose (figura 3.2), representam a odometria calculada.



(a) visão geral



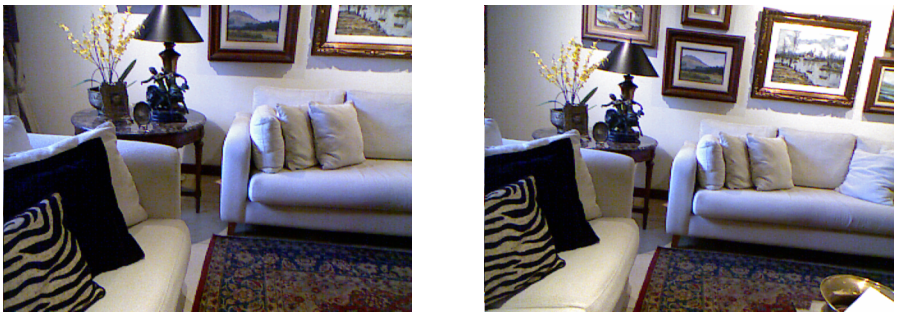
(b) vista lateral



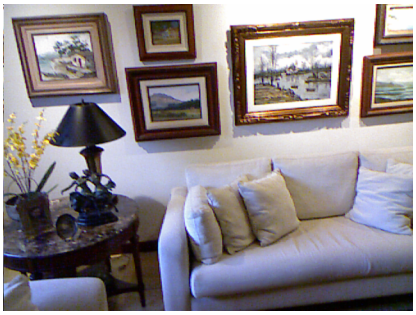
(c) vista de topo

Figura 5.6: Visualização 3D da odometria calculada (cena 1)

5.3 Cena 2



(a) (b)



(c)

Figura 5.7: Cena 2

Na cena 2 mantivemos os requisitos para o funcionamento do método. A cena 2 representa um ambiente rico em textura e geometria (figura 5.7), que foi iluminado parcialmente com luz natural (horário próximo do por do sol) e focos de luz artificial.

5.3.1 Análise dos resultados

Tabela 5.3: Tabela com tempo de execução dos componentes principais do método proposto, cena 2

LM	4,12s	17,8%
SURF	11,6s	50%
FREAK	1,18s	5%
NN	5,9s	25%
Total	23,2s	
FPS*	21,2	

*Quadros por segundo

Neste experimento foi detectado um número maior de *keypoints*, 1400 em média, o que explica o tempo maior utilizado pelo detector SURF. São mais vezes chegando até ao final das comparações achando pontos de máximo e principalmente calculando o ajuste de posição do *keypoint*. Os outros fatores como dependem também da quantidade de *keypoints* tiveram um desempenho pior, resultando num FPS mais baixo (tabela 5.3). O experimento na cena 2 teve uma qualidade visivelmente pior do que na cena 1. A piora na qualidade fica latente com os seguintes dados: o valor máximo para a distância em relação ao Fusion foi de 22,6cm com mínimo de 0,18. A média teve valor de 4,63 com mediana de 3,68cm e desvio padrão de 3,80cm (figura 5.8). A estática para os dados dos ângulos (figura 5.9), está na tabela 5.4. A piora na qualidade

Tabela 5.4: Estatística dos ângulos em relação Fusion, cena 2

	média	max	min	std
pitch	0,6	4,6	0	0,8
yaw	0,7	3,37	0	0,7
roll	0,5	2,2	0	0,3

deve-se à performance do *matching* (figura 5.11), resultando erros maiores de reprojeção (figura 5.10). O parâmetro μ_0 do LM($\mu_0 = 10^4$) é sempre mantido alto, ou seja o método sempre começa na região de gradiente descendente, o que ajuda o método a voltar a convergir. Por exemplo, há um pico nas figuras 5.8 e 5.9 após o quadro 150 que dura aproximadamente até o quadro 180.

Na figura 5.12, há uma visualização da odometria calculada para a cena 2. É possível observar a diferença entre o Fusion e o método proposto. São várias setas distantes da linha azul e uma faixa sem setas próximas, faixa esta relativa ás poses no intervalo dos quadros 150 à 200.

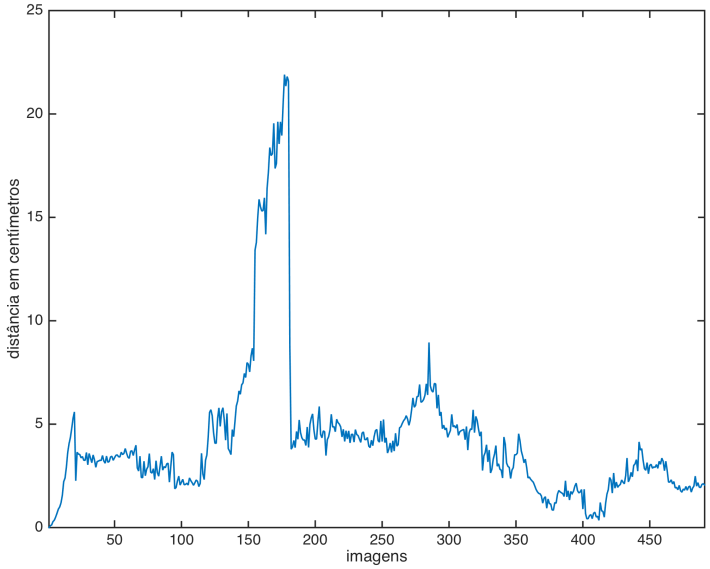


Figura 5.8: Distância em centímetros em relação ao Fusion (cena 2)

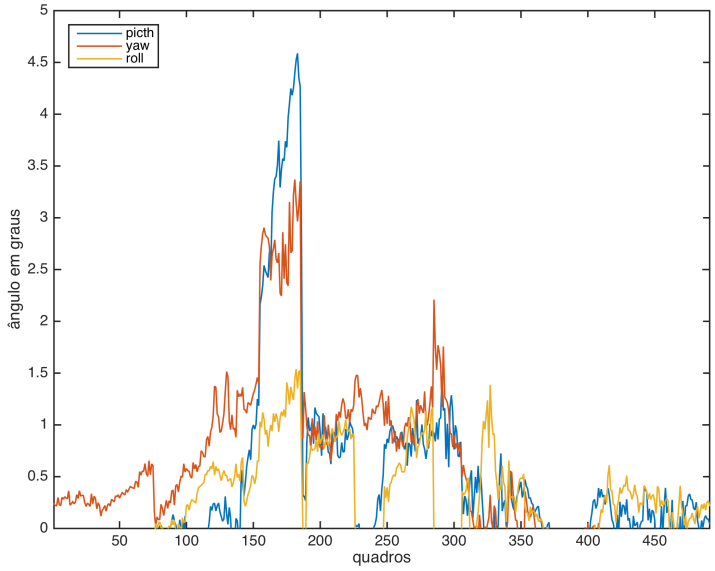
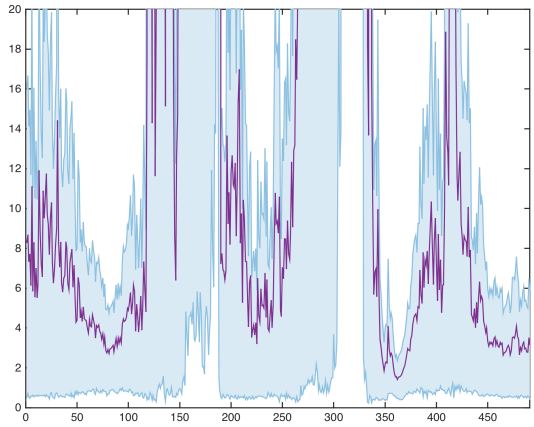
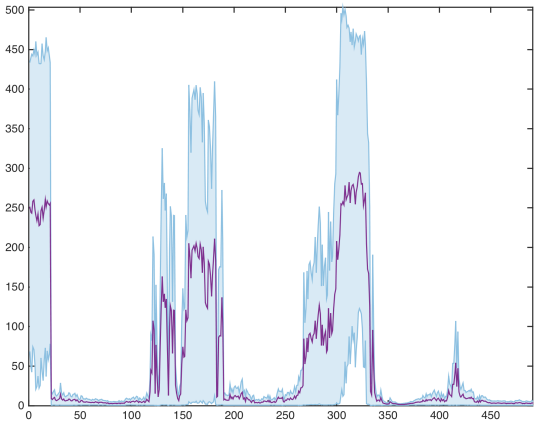


Figura 5.9: Ângulos de Euler sobrepostos da pose calculada em relação ao Kinect Fusion. Convenção: roll(X)-pitch(Y)-yaw(Z) (cena 2)



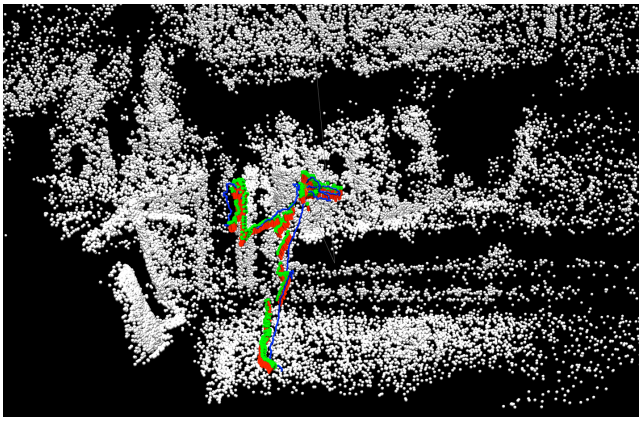
(a) detalhes



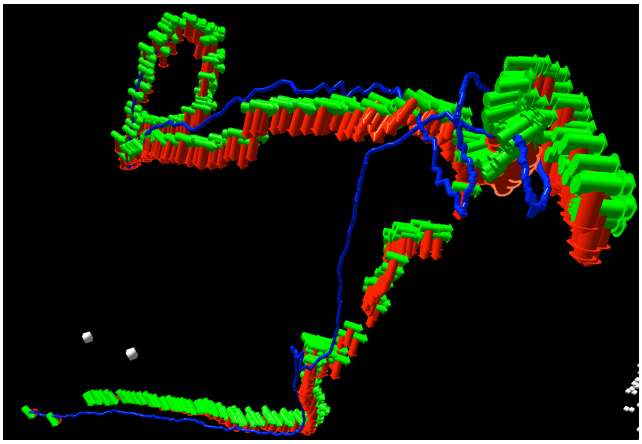
(b) visão geral

Figura 5.10: Eixo Y, mediana do erro de reprojeção em roxo. Área em ciano, valor da mediana limitada por MAD. Eixo X, imagens

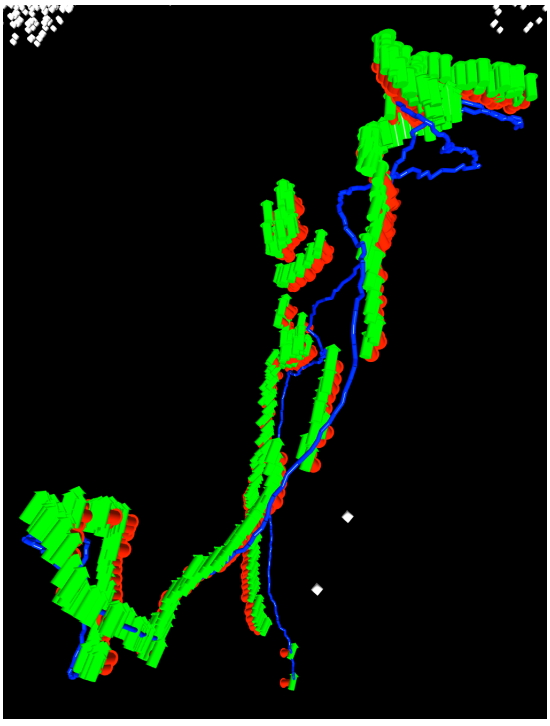
Figura 5.11: *outliers* no *matching* resulta na piora da qualidade do cálculo da pose (cena 2)



(a) visão geral



(b) vista lateral



(c) vista de topo

Figura 5.12: Visualização 3D da odometria calculada (cena 2)

6
Conclusão e Trabalhos Futuros

Este trabalho propôs um método para utilizar dados de uma reconstrução densa feita pelo Kinect Fusion, com um *pipeline* para cálculo da odometria utilizando *features* esparsos. Nos experimentos realizados o método conseguiu calcular em tempo real a pose, observando os requisitos para o funcionamento do mesmo, tendo no pior caso uma distância de 23cm em relação a odometria do Fusion. Os requisitos restrigem o método para áreas internas com iluminação e a utilização de uma agente que não mova a câmera com grande velocidade (o ângulo máximo registrado entre pose e *keyframe* foi de 20 graus). Na primeira cena foram obtidos resultados mais próximos aos da odometria calculada pelo Kinect Fusion. Neste experimento a iluminação estava mais uniforme sem áreas com sombra como na cena 2. Embora observada a diferença na iluminação, não foi possível precisar sua influência no experimento. O método proposto não apresenta nenhum estado para recuperação caso não haja convergência no cálculo da pose ou haja um movimento grande entre duas poses.

A cena 2 possui o dobro da quantidade de *keypoints* por quadro. Entretando os resultados da odometria foram piores que na cena 1. O maior número de *keypoints* impactou a performance com a cena 2 tendo 22,2 FPS contra 29,8 FPS da cena 1. No estágio de *matching* encontra-se o gargalo do método proposto com o cálculo dos *keypoints* SURF. A quantidade dos *keypoints* impacta todos os componentes e um maior número não significou melhora na qualidade do cálculo da pose. O descarte de *keypoints* é feito apenas com um limiar no valor do determinante da Hessiana do detector SURF. Um método que leve em conta a distribuição espacial dos *keypoints* como em [37] pode escolher menos *keypoints* com mais qualidade.

O uso de um valor fixo para a região de *outlier* na função de perda não é o ideal e uma análise estatística da distribuição do erro de reprojeção se faz necessária, algo similar ao feito em [38]. Outra sugestão de trabalho futuro é a utilização de um modelo de velocidade como usado no PTAM; a posição dos *keypoints* poderia ser prevista e a busca de vizinhos mais próximos poderia ser feita apenas em uma área ao redor desta posição. Este estudo pode melhorar a performance e qualidade do *matching*.

Uma aplicação prática para o método apresentado seria utilizar a odometria calculada em aplicações de realidade aumentada, junto à malha 3D fornecida pelo Kinect Fusion. Esta malha pode ser usada para a oclusão

de objetos virtuais e reais.

Outra utilização do método seria em aplicações multiusuário de realidade aumentada. Quando há mais de um Kinect observando uma mesma área, há uma interferência na leitura do mapa de profundidade. Uma forma de contornar esta limitação é que apenas um usuário use o Kinect Fusion. Os demais usariam a odometria visual deste trabalho, com uso de câmeras comuns.

Referências Bibliográficas

[1] Rafael PD Vivacqua, Felipe N Martins, and Raquel F Vassallo. Visual lane tracking and prediction for autonomous vehicles. In *Simpósio Brasileiro de Automação Inteligente (SBAI)*, 2013. 1

[2] Alexandre S Brandão, Felipe N Martins, and Higor B Soneguetti. A vision-based line following strategy for an autonomous uav. In *International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, 2015. 1

[3] Hans P Moravec. Obstacle avoidance and navigation in the real world by a seeing robot rover. Technical report, DTIC Document, 1980. 1

[4] Yi Ma. *An invitation to 3-d vision: from images to geometric models*, volume 26. Springer Science & Business Media, 2004. 1, 3.3.1, 3.3.2

[5] Microsoft. photosynth is based on photo tourism, a research project by university of washington graduate student noah snavely. `https://www.photosynth.net`, 2015. 1

[6] Andrew J Davison, Ian D Reid, Nicholas D Molton, and Olivier Stasse. Monoslam: Real-time single camera slam. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(6):1052–1067, 2007. 1, 2.1

[7] Georg Klein and David Murray. Parallel tracking and mapping for small ar workspaces. In *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*, pages 225–234. IEEE, 2007. 1

[8] Giorgio Grisetti, Rainer Kummerle, Cyrill Stachniss, and Wolfram Burgard. A tutorial on graph-based slam. *Intelligent Transportation Systems Magazine, IEEE*, 2(4):31–43, 2010. 1, 2.1

[9] Yasutaka Furukawa and Jean Ponce. Accurate, dense, and robust multi-view stereopsis. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 32(8):1362–1376, 2010. 1

[10] Richard A Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J Davison, Pushmeet Kohi, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. Kinectfusion: Real-time

dense surface mapping and tracking. In *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on*, pages 127–136. IEEE, 2011. 1, 3.7

[11] Davide Scaramuzza and Friedrich Fraundorfer. Visual odometry [tutorial]. *Robotics & Automation Magazine, IEEE*, 18(4):80–92, 2011. 2

[12] David Nistér, Oleg Naroditsky, and James Bergen. Visual odometry. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 1, pages I–652. IEEE, 2004. 2.1

[13] C. G. Harris and J. M. Pike. 3d positional integration from image sequences. *Image Vision Comput.*, 6(2):87–90, May 1988. ISSN 0262-8856. doi: 10.1016/0262-8856(88)90003-0. URL `http://dx.doi.org/10.1016/0262-8856(88)90003-0`. 2.1

[14] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *Computer Vision–ECCV 2006*, pages 430–443. Springer, 2006. 2.1

[15] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *Computer vision–ECCV 2006*, pages 404–417. Springer, 2006. 2.1, 3.6

[16] Alexandre Alahi, Raphael Ortiz, and Pierre Vandergheynst. Freak: Fast retina keypoint. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 510–517. Ieee, 2012. 2.1, 3.5, 3.5.1

[17] Richard I Hartley and Peter Sturm. Triangulation. *Computer vision and image understanding*, 68(2):146–157, 1997. 2.1

[18] Bert M Haralick, Chung-Nan Lee, Karsten Ottenberg, and Michael Nölle. Review and analysis of solutions of the three point perspective pose estimation problem. *International journal of computer vision*, 13 (3):331–356, 1994. 2.1

[19] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981. 2.1

[20] Stephan Weiss, Markus W Achtelik, Simon Lynen, Margarita Chli, and Roland Siegwart. Real-time onboard visual-inertial state estimation and

self-calibration of mavs in unknown environments. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 957–964. IEEE, 2012. 2.1

[21] Richard A Newcombe, Steven J Lovegrove, and Andrew J Davison. Dtam: Dense tracking and mapping in real-time. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2320–2327. IEEE, 2011. 2.2

[22] Yang Chen and Gérard Medioni. Object modeling by registration of multiple range images. In *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, pages 2724–2729. IEEE, 1991. 2.2

[23] Szymon Rusinkiewicz and Marc Levoy. Efficient variants of the icp algorithm. In *3-D Digital Imaging and Modeling, 2001. Proceedings. Third International Conference on*, pages 145–152. IEEE, 2001. 2.2

[24] Tommi Tykkala, Cédric Audras, and Andrew I Comport. Direct iterative closest point for real-time visual odometry. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 2050–2056. IEEE, 2011. 2.2

[25] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000. 3.2.2, 4.1.1

[26] Kaare Brandt Petersen, Michael Syskind Pedersen, et al. The matrix cookbook. *Technical University of Denmark*, 7:15, 2008. 3.3.1

[27] Peter J Huber et al. Robust estimation of a location parameter. *The Annals of Mathematical Statistics*, 35(1):73–101, 1964. 3.4.3

[28] Matthew Brown and David G Lowe. Invariant features from interest point groups. In *BMVC*, number s 1, 2002. 3.6

[29] Buyue Zhang and Jan P Allebach. Adaptive bilateral filter for sharpness enhancement and noise removal. *Image Processing, IEEE Transactions on*, 17(5):664–678, 2008. 3.7.1

[30] Marius Muja and David G. Lowe. Scalable nearest neighbor algorithms for high dimensional data. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 36, 2014. 4.2.1

[31] Jose luis Blanco. A tutorial on se(3) transformation parameterizations and on-manifold optimization, 2014. 4.2.2

[32] Sameer Agarwal, Keir Mierle, and Others. Ceres solver. `http://ceres-solver.org`, 2015. 4.2.2

[33] Wikipedia. Automatic differentiation, 2015. `http://en.wikipedia.org/wiki/Automatic_differentiation`. 4.2.2

[34] K. Madsen, H. B. Nielsen, O. Tingleff, and Mathematical Modelling. Imm methods for non-linear least squares problems, 2004. 4.2.2

[35] John Sachs. Robust dual scaling with tukey's biweight. *Applied psychological measurement*, 18(4):301–309, 1994. 4.2.2

[36] Jose Luis Blanco et al. Mobile robot programming toolkit (mrpt). `http://www.mrpt.org`, 2015. 5.1

[37] Steffen Gauglitz, Luca Foschini, Matthew Turk, and Tobias Höllerer. Efficiently selecting spatially distributed keypoints for visual tracking. In *Image Processing (ICIP), 2011 18th IEEE International Conference on*, pages 1869–1872. IEEE, 2011. 6

[38] Christian Kerl, Jurgen Sturm, and Daniel Cremers. Robust odometry estimation for rgb-d cameras. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 3748–3754. IEEE, 2013. 6