UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

Eduardo Arrial Speroni

UM SISTEMA DE BAIXO CUSTO PARA LOCALIZAÇÃO UTILIZANDO SENSORES POSICIONAIS E ESTEREOSCOPIA VISUAL

Santa Maria, RS
2016

Eduardo Arrial Speroni

UM SISTEMA DE BAIXO CUSTO PARA LOCALIZAÇÃO UTILIZANDO SENSORES POSICIONAIS E ESTEREOSCOPIA VISUAL

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Informática, Área de Concentração em Ciências Exatas e da Terra, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de
Mestre em Informática.

ORIENTADORA: Prof.ª Dr.ª Simone Regina Ceolin

Santa Maria, RS
2016

Ficha catalográfica elaborada através do Programa de Geração Automática
da Biblioteca Central da UFSM, com os dados fornecidos pelo(a) autor(a).

Speroni, Eduardo Arrial
Um sistema de baixo custo para localização utilizando sensores posicionais e estereoscopia visual / Eduardo Arrial Speroni.- 2016.
87 p.; 30 cm

Orientadora: SImone Regina Ceolin
Dissertação (mestrado) - Universidade Federal de Santa Maria, Centro de Tecnologia, Programa de Pós-Graduação em Informática, RS, 2016

1. Robótica 2. Visão Computacional 3. Estereoscopia 4. SLAM 5. Sensores I. Ceolin, SImone Regina II. Título.

End. Eletr.: edusperoni@gmail.com

Eduardo Arrial Speroni

UM SISTEMA DE BAIXO CUSTO PARA LOCALIZAÇÃO UTILIZANDO SENSORES POSICIONAIS E ESTEREOSCOPIA VISUAL

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Informática, Área de Concentração em Ciências Exatas e da Terra, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de
Mestre em Informática.

Aprovado em 29 de agosto de 2016:

Simone Regina Ceolin, Dr.ª (UFSM)
(Presidenta/Orientadora)

Cesar Tadeu Pozzer, Dr. (UFSM)

Bruno Zatt, Dr. (UFPel)

Santa Maria, RS
2016

RESUMO

UM SISTEMA DE BAIXO CUSTO PARA LOCALIZAÇÃO UTILIZANDO SENSORES POSICIONAIS E ESTEREOSCOPIA VISUAL

AUTOR: Eduardo Arrial Speroni
ORIENTADORA: Simone Regina Ceolin

Um dos problemas na área da robótica é chamado de *Simultaneous Location and Mapping* (SLAM), e consiste da necessidade de um robô localizar-se no ambiente e mapeá-lo. O uso de sistemas estereoscópicos é uma abordagem utilizada para a resolução deste problema. Estes sistemas são compostos de câmeras sincronizadas via *hardware* de alto custo, enquanto câmeras de baixo custo tem seu uso mais restrito para aplicações onde há pouca movimentação. Desta maneira, este trabalho visa propor sistema de baixo custo ao utilizar estereoscopia com baixa *baseline* e câmeras com baixo ângulo de visão horizontal, e sua sincronização realizada via *software*, em conjunto com um filtro baseado na densidade do mapa de disparidades das imagens capturadas, que tem por objetivo descartar *frames* não retificados corretamente, o que implica a sua dessincronização. Adicionalmente, foi desenvolvido um aplicativo Android capaz de obter e transmitir dados sensoriais de um *smartphone*, como GPS e orientação, reduzindo o custo e aumentando a acessibilidade do sistema. A partir destes dados foi possível gerar *datasets* de calibração e processamento para que pudessem ser analisados posteriormente. Ao combinar a odometria visual com os dados de sensores do *smartphone* contidos nos *datasets*, obteve-se um sistema capaz de obter sua localização sem conhecimento prévio do ambiente com um erro similar aos obtidos por técnicas já consolidadas de alto custo. Contudo, os dados de GPS mostraram-se imprecisos em cenários de baixa velocidade, enquanto a interferência eletromagnética e a baixa quantidade de pontos de referência laterais prejudicaram a leitura da orientação do dispositivo e o cálculo da odometria visual no cenário de alta velocidade. Constata-se que o sistema não é capaz de realizar processamento em tempo real, dada a necessidade de avaliação todos os *frames* para que sejam filtrados, descartando-se em torno de 60% dos mesmos. Desta maneira, foi demonstrado que o sistema proposto de baixo custo foi capaz de manter um erro baixo em troca de um alto tempo de processamento, potencialmente reduzindo o custo e aumentando a acessibilidade de aplicações VSLAM. Devido à modularidade do sistema, é possível substituir seus componentes sem grandes alterações em sua implementação, viabilizando o uso de dispositivos com maior precisão em trabalhos futuros.

Palavras-chave: Robótica. Visão Computacional. Estereoscopia. SLAM. Sensores. Android.

ABSTRACT

A LOW COST SYSTEM FOR LOCALIZATION USING POSITIONAL SENSORS AND VISUAL STEREOSCOPY

AUTHOR: Eduardo Arrial Speroni
ADVISOR: Simone Regina Ceolin

One of the problems in robotics is called Simultaneous Location and Mapping (SLAM), and lies in the necessity of a robot to localize itself on the environment while simultaneously mapping it. The use of stereoscopic systems is one approach to solve this problem. Theses systems are composed by high cost cameras synchronized via hardware, while low cost cameras are more restrict to applications with low or no movement. This research proposes a low cost system by using stereoscopy with a low baseline and low horizontal field of view cameras, synchronizing them via software, along with a filter based on the density of the disparity map of the captured images, with the intent to discard badly rectified frames, which implies desynchronization. Additionally, an Android app capable of obtaining and transmitting sensory data from a smartphone, like GPS and orientation, was developed, reducing the cost and increasing the system's accessibility. From these data, calibration and processing datasets were generated, so they could be analyzed afterward. The combination of visual odometry and the smartphone's sensory data contained in the datasets resulted in a system capable of obtaining its localization without previous knowledge of the environment with a similar error to the ones obtained by well established high cost techniques. However, the GPS data was imprecise in low speed scenarios, while the high electromagnetic interference and the low amount of lateral points of reference harmed the device's orientation data and the visual odometry calculation in the high speed scenario. The system isn't capable of real time processing, given the need to analyze every frame so they can be filtered, discarding about 60% of them. It was demonstrated that the proposed low cost system was capable of keeping a low error in return of a high processing time, potentially reducing the cost and increasing the accessibility of VSLAM applications. Due to the system's modularity, it's possible to replace its components without many implementation changes, allowing the use of better precision devices in future work.

Keywords: Robotics. Computer Vision. Stereocopy. SLAM. Sensors. Android.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

$SLAM$	*Simultaneous Location and Mapping*
$VSLAM$	*Visual Simultaneous Location and Mapping*
GPS	*Global Positioning System*
$UFSM$	*Universidade Federal de Santa Maria*
BMP	*Bad Matched Pixel*
EKF	*Extended Kalman Filter*
MCL	*Monte Carlo Localization*
LCE	*Loop Closure Error*
$ISPKF$	*Iterated Sigma Point Kalman Filter*
SBM	*Stereo Block Matching*
$SGBM$	*Semi-global Block Matching*
GLM	*OpenGL Mathematics*
MP	*Matched Pixel*
FPS	*Frames per second*
$NRMSE$	*Normalized Root Mean Square Error*

LISTA DE ALGORITMOS

SUMÁRIO

1 INTRODUÇÃO

A necessidade de criação de robôs autônomos para a realização de tarefas é uma das bases da área da robótica, e problemas devem ser resolvidos para que este objetivo seja atingido. Um robô autônomo móvel, por exemplo, necessita localizar-se no ambiente para que possa mover-se evitando obstáculos.

Simultaneous Localization And Mapping (SLAM) é uma área da robótica que consiste do problema em que um robô móvel, sem conhecimento prévio do ambiente no qual se encontra, necessita construir um mapa do local, e simultaneamente localizar-se nele. Uma solução de SLAM deve lidar com problemas como erros de sensor, problema de correspondência, fechamento de *loop*, complexidade de tempo e memória (THRUN, 2002; FRESE; HIRZINGER, 2001; BEGUM; MANN; GOSINE, 2008). A maior parte das soluções propostas na área é direcionada à criação, armazenamento, processamento e obtenção do mapa.

Podem-se dividir as formas de representação do ambiente para SLAM como: métricas, que utilizam informações geométricas; topológicas, que utilizam grafos cujos nodos são lugares bastante distintos; e híbridas, como propostas por Blanco et al. (BLANCO; GONZÁLEZ; FERNÁNDEZ-MADRIGAL, 2009), compostas por ambas as representações. Nestas técnicas, utiliza-se o conceito de *landmarks*, que são formas distintas e identificáveis no mundo nas quais o robô pode avaliar constantemente suas posições e estimar seu progresso (THRUN et al., 2001).

Uma das técnicas utilizadas para solucionar o problema de SLAM é um modelo baseado em características utilizando o filtro de Kalman, proposto por Kalman (KALMAN, 1960) e aplicado à Ciência da Computação por Welch e Bishop (WELCH; BISHOP, 1995). Esta abordagem é muito compacta e tem sido bastante otimizada com o crescimento da área de pesquisa, porém são limitadas ao número de modelos disponíveis. Outra abordagem envolve o uso de filtros de partículas, porém esta tem um custo computacional maior devido ao grande número de partículas e seus valores estatísticos individuais, o que também gera um grande custo de memória (THRUN et al., 2001). Estas técnicas podem utilizar dados odométricos da movimentação do robô e sensoriais, como sensores *laser* e sistemas estereoscópicos, para auxiliar na criação do mapa e localização do robô.

A estereoscopia é uma técnica de visão computacional, que torna possível por meio da aquisição de duas imagens de diferentes pontos de uma cena, aferir a profundidade de um determinado objeto. Em outras palavras, determinar a distância do observador ao objeto observado, obtendo-se a profundidade da cena (GONZALEZ; WOODS, 2000). Também é possível mensurar distâncias entre objetos que compõem uma cena e dessa forma obter um modelo ou mapa tridimensional de um ambiente utilizando apenas duas câmeras.

A partir do uso de estereoscopia, é possível gerar uma cópia 3D do mundo real, dado um bom emparelhamento entre as imagens capturadas, e é a base da subárea de SLAM chamada Visual SLAM (VSLAM) (KARLSSON et al., 2005), que busca solucionar o problema de SLAM ao calcular a odometria do sistema de acordo com as imagens reconhecidas.

Por serem utilizados em robôs móveis, sistemas estereoscópicos devem ter sua captura de imagens sincronizadas para que não sejam capturadas em posições diferentes, dificultando a estimativa da posição do sistema. Em diversos trabalhos, utilizam-se soluções prontas de sistemas estereoscópicos sincronizados via *hardware*, como as câmeras Bumblebee, o que eleva muito o custo de uma implementação VSLAM (KIM et al., 2011; AGRAWAL; KONOLIGE, 2006; KITT; GEIGER; LATEGAHN, 2010).

Além de estereoscopia, podem-se utilizar sensores de posicionamento global (GPS) e inerciais para a complementação do sistema (AGRAWAL; KONOLIGE, 2006). Ao longo do tempo, estes tipos de sensores tornaram-se mais acessíveis, e atualmente podem ser encontrados em diversos *Smartphones* de baixo custo. Desta maneira, o uso destes dispositivos vem crescendo em trabalhos que buscam solucionar problemas de SLAM, predominantemente utilizando o sistema operacional Android (FARAGHER; SARNO; NEWMAN, 2012; SHIN; CHON; CHA, 2012; KAO; HUY, 2013; VENTURA et al., 2014).

Desta maneira, este trabalho visa propor um sistema SLAM com câmeras de baixo custo, sem sincronização via *hardware*, utilizando um *Smartphone* Android para auxiliar a reduzir o erro obtido, reduzindo-se o custo total do sistema e mantendo um erro de $1{,}6\%$ a $2{,}1\%$, comparável com sistemas de valor elevado, para que possa ser utilizado em ambientes externos, como em ruas, calçadas, e campos abertos.

1.1 SISTEMA PROPOSTO E CONTRIBUIÇÃO

O sistema proposto visa estimar sua localização com precisão a cada passo do trajeto percorrido, sem conhecimento prévio do ambiente, a partir de dados visuais de duas câmeras estereoscópicas e dados sensoriais presentes em um dispositivo Android, utilizando *hardware* de baixo custo. Estes dados podem ser utilizados para uma reconstrução 3D do ambiente.

O diagrama *top-down*, na Figura 1.1, apresenta o sistema proposto, dividido em 3 partes: geração de *datasets*, processamento da localização, e reconstrução 3D. Os *datasets*[1] são gerados realizando uma sincronização via *software* em duas câmeras e obtendo dados sensoriais a partir de um aplicativo Android utilizando comunicação via USB. Devido

[1]Foram gerados 3 *datasets* em ambientes externos na Universidade Federal de Santa Maria (UFSM), denominados *dataset* COPERVES, *dataset* Planetário e *dataset* Carro. Mais detalhes sobre os mesmos são descritos no Capítulo 5.

ao alto tempo de processamento, os *datasets* são calibrados e processados separadamente, onde são filtrados os *frames* considerados sincronizados para o cálculo da odometria visual, realizado pela biblioteca LIBVISO2, e seu resultado é integrado com os dados sensoriais obtidos. A partir do resultado do processamento, é possível calcular o erro e desempenho do sistema e realizar a reconstrução 3D, realizada predominantemente pelas bibliotecas OpenCV e OpenGL.

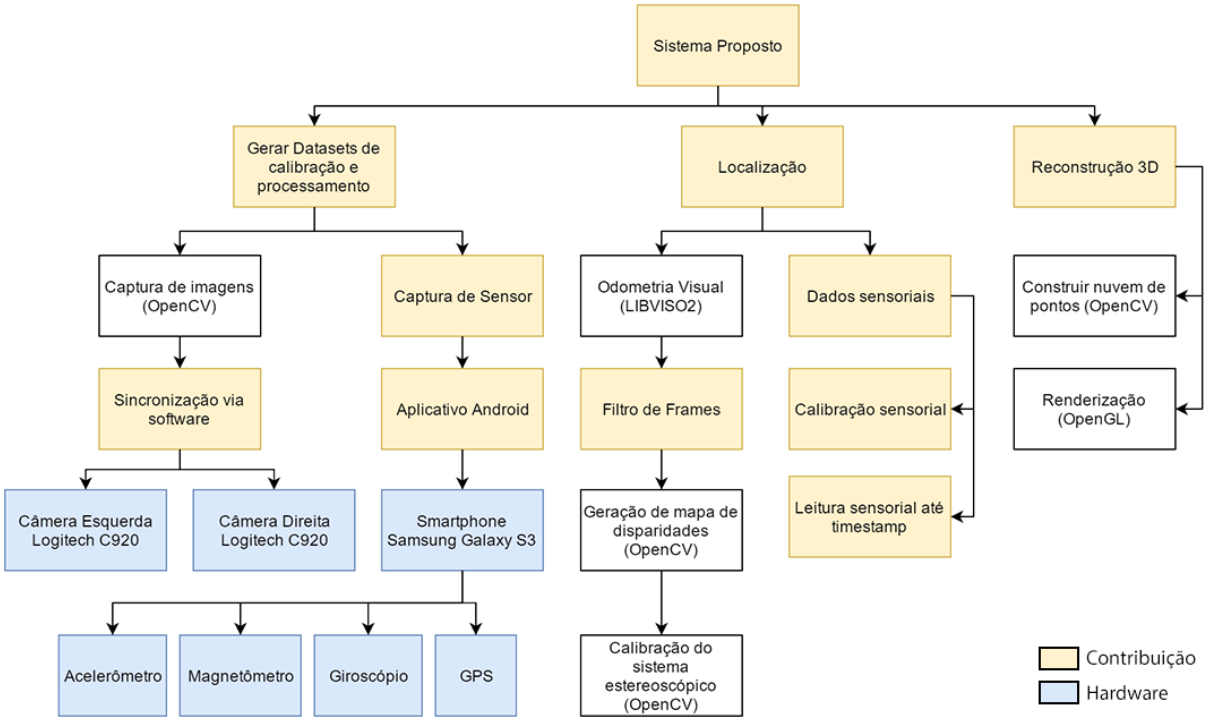


Figura 1.1 – Diagrama *top-down* do sistema proposto.

A “Sincronização via *software*” é realizada utilizando *threads* com semáforos para capturar os *frames* em tempos similares de ambas as câmeras. O “Aplicativo Android” capta os dados de sensores e envia-os via TCP/IP, que são recebidos e armazenados pelo programa na “Captura de Sensor”. Estes dados compõe os *datasets*.

O “Filtro de *Frames*” descarta os *frames* dessincronizados a partir do mapa de disparidades, encaminhando os restantes para o processamento da “Odometria Visual (LIBVISO2)”. Os sensores são calibrados (Calibração sensorial) e lidos até o momento de obtenção do *frame* (Leitura sensorial até *timestamp*), compondo os “Dados sensoriais” que são então integrados aos dados de odometria visual, resultando na “Localização”.

A “Reconstrução 3D” é a integração da nuvem de pontos gerada pelo OpenCV e a “Localização”, renderizadas em OpenGL.

Os blocos em tom de azul representam o *hardware* utilizado para a geração de *datasets*, comunicando-se via USB, sendo a biblioteca OpenCV responsável pela interface de comunicação com as câmeras no bloco “Captura de imagens (OpenCV)”, e o “Aplicativo Android” pela obtenção de dados dos sensores pertencentes ao *smartphone*.

Desta maneira, os passos realizados são: geração de *dataset* de calibração; geração de *dataset* de processamento; calibração do sistema; processamento do *dataset*, obtendo-se a localização do sistema a cada passo; geração de resultados, incluindo cálculo de desempenho e erro; e reconstrução 3D.

1.2 OBJETIVOS

O objetivo deste trabalho é a implementação de um sistema Visual SLAM de baixo custo para uso em ambientes externos, utilizando, além de um sistema de câmeras estereoscópico, sensores que podem ser comumente encontrados em *smartphones* Android. Este sistema deve ser capaz de identificar sua posição com precisão e seus dados podem ser utilizados para a reconstrução da nuvem de pontos.

1.3 OBJETIVOS ESPECÍFICOS

De acordo com a linha definida pelo foco do trabalho, os objetivos podem ser especificados da seguinte maneira:

- Implementação de um aplicativo Android que permita obter dados precisos dos sensores do dispositivo;
- Implementação e calibração do sistema estereoscópico;
- Geração de *datasets* de calibração e processamento, tendo em vista isolar a captura de dados e o seu processamento, que demanda mais tempo, e a possibilidade de trabalhos futuros;
- Realizar uma sincronização via *software* na captura de imagens;
- Utilizar a imagens obtidas para computar a odometria visual do sistema;
- Integrar os dados de odometria visual com sensores contidos no *smartphone* Android;
- Comparar a solução obtida com outras soluções já consolidadas.

1.4 JUSTIFICATIVA/MOTIVAÇÃO

Em grande parte dos trabalhos na área, o uso de câmeras sincronizadas via *hardware* e soluções fechadas de placas sensoriais acaba por elevar o custo do sistema SLAM (KIM et al., 2011; AGRAWAL; KONOLIGE, 2006; KITT; GEIGER; LATEGAHN, 2010). O alto custo destas aplicações acaba por limitar sua utilização onde o orçamento disponível para sua implantação é reduzido. Desta maneira, vê-se a necessidade de alternativas de baixo custo e acessíveis de implementação de sistemas SLAM, utilizando um sistema estereoscópico composto por câmeras comuns e sensores encontrados em dispositivos de alta disponibilidade, como *Smartphones* Android, para uma implementação SLAM de baixo custo e comparável com técnicas utilizadas atualmente.

1.5 ESTRUTURA DO TRABALHO

No Capítulo 2 é apresentado o referencial teórico sobre (Visual) SLAM, abrangendo diversas técnicas, assim como fundamentos de estereoscopia, e o uso de sensores posicionais e de movimento, focado na plataforma Android. O Capítulo 3 apresenta os trabalhos relacionados que são utilizados como embasamento para decisões de utilização de técnicas e comparação de resultados. Com base nestes, o Capítulo 4 descreve o sistema proposto, implementação do trabalho e problemas enfrentados, e seus resultados são apresentados no Capítulo 5. Por fim, o Capítulo 6 contém as considerações finais deste trabalho e trabalhos futuros a serem desenvolvidos.

2 REFERENCIAL TEÓRICO

Neste capítulo são explanados os fundamentos básicos de estereoscopia, SLAM e as técnicas utilizadas para sua solução.

2.1 FUNDAMENTOS DE ESTEREOSCOPIA

Estereoscopia é um processo pelo qual por meio de duas imagens de um objeto, assumindo que essas sejam obtidas por duas câmeras idênticas e separadas por uma determinada distância, é possível adquirir a informação de tridimensionalidade da cena (GONZALEZ; WOODS, 2000). Para buscar o entendimento de visão estéreo e a geometria envolvida para obter as informações da cena, como a distância dos objetos observados, é interessante destacar a geometria envolvida em apenas uma câmera. O modelo geométrico de uma câmera é ilustrado na Figura 2.1, onde é chamado basicamente de *pinhole* (buraco de agulha).

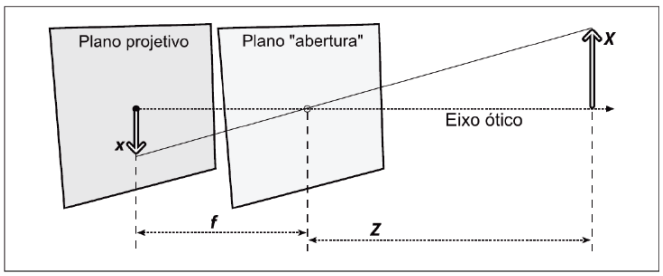


Figura 2.1 – Modelo geométrico de câmera pinhole.
Fonte: (BRADSKI, 2008)

O principal parâmetro desse modelo é a distancia focal (MENDES, 2012), representada por f, esta é distância entre a abertura da câmera e o plano projetivo. O plano projetivo é o local onde a luz é projetada após ter passado pela abertura da câmera. A distância entre a abertura e o objeto é denotada por Z, X é o comprimento do objeto no mundo real e x na projeção. Sendo conhecida a distância focal é possível obter o comprimento de qualquer objeto no plano projetivo. É importante destacar que no plano projetivo a imagem encontra-se invertida. Também é importante ressaltar que, a partir da geometria apresentada, não é possível determinar a distância entre um ponto do mundo e a câmera (BRADSKI, 2008). Sendo assim, por meio de um sistema canônico se torna possível aferir a distância entre um determinado ponto 3D P na cena e o sistema de câmeras. A Figura 2.2 ilustra o sistema.

Conforme a Figura 2.2, é possível triangular a posição do ponto P, obtendo-se assim sua coordenada 3D (x,y,z) em relação ao sistema de câmeras. Sendo assim, a dispari-

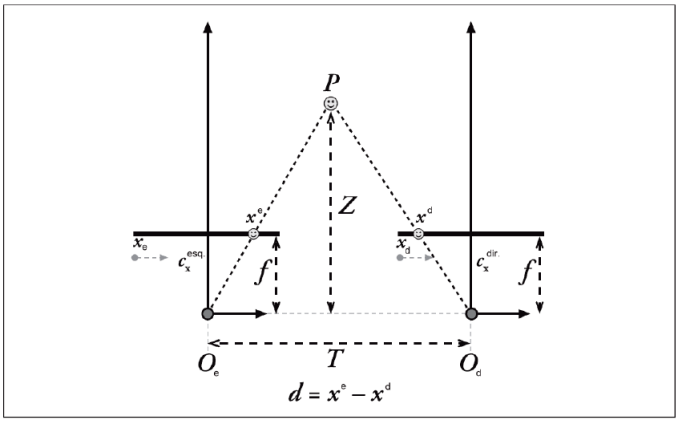


Figura 2.2 – Modelo de Sistema Canônico com distância focal f e câmeras com deslocamento T.
Fonte: (BRADSKI, 2008)

dade ou d, é obtida pela equação:

$$d = x^e - x^d \tag{2.1}$$

onde

$x^e =$ Distância do ponto na imagem esquerda no eixo x
$x^d =$ Distância do ponto na imagem direita no eixo x

Neste contexto, a equação presente em (2.2) mostra a similaridade entre os triângulos O_ePO_d e x^ePx^d à esquerda e a parte da direita o calcula para distância Z (BRADSKI, 2008).

$$\frac{T-(x^e-x^d)}{Z-T} = \frac{T}{Z} \Rightarrow Z = \frac{fT}{x^e-x^d} = \frac{fT}{d} \tag{2.2}$$

Entretanto, câmeras reais estão sujeitas a distorções, e podem não estar perfeitamente alinhadas horizontalmente no sistema canônico. Diante disso, através da geometria epipolar, é possível virtualmente alinhar as câmeras e definir suas inter-relações, conforme ilustrado na Figura 2.3.

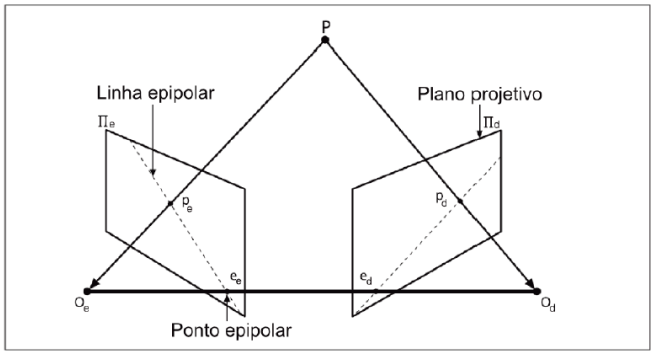


Figura 2.3 – Modelo de geometria epipolar.
Fonte: (BRADSKI, 2008)

Conforme a Figura 2.3, o centro da projeção é representado por O_e e O_d, da imagem da direita e esquerda respectivamente. As intersecções que ligam os centros das projeções e os planos de projeção são chamadas de pontos epipolares, são eles e_e e e_d, levando em consideração o ponto P. Dessa forma, se tem a intersecção do plano epipolar em cada plano projetivo. Assim como mencionado por Scaramuzza et al. (MENDES, 2012), quando se torna necessário encontrar o ponto P na imagem da direita, a partir do conhecimento da projeção deste ponto na esquerda, ou seja, conhecendo P_e e E_d, é possível calcular a linha epipolar e restringir a busca ao ponto P_d.

A correspondência entre pontos de ambas as imagens é uma das tarefas que mais dispensa processamento (CARDOSO, 2012), e as câmeras ainda possuem distorções radial, formada pelo formato imperfeito da lente, e tangencial, provocada pelo desalinhamento da lente em relação ao sensor (BRADSKI, 2008). Neste contexto o processo de calibração destas câmeras é necessário.

2.2 ESTEREOSCOPIA

A estereoscopia é composta por três etapas, sendo elas: Calibração, Emparelhamento, e a geração do Mapa de Disparidade (CARDOSO, 2012).

- Calibração: Estima parâmetros intrínsecos e extrínsecos de cada câmera. Os parâmetros intrínsecos englobam a matriz de coeficientes de distorção e a matriz fundamental, que inclui a distância focal e o centro aproximado da imagem. Por outro lado, parâmetros extrínsecos são as matrizes de projeção, que normalmente é a matriz fundamental homogênea transladada no eixo x, as matrizes de rotação e translação para cada câmera, e as transformações de rotação e translação entre as duas câmeras. Uma câmera com parâmetros intrínsecos é chamada de calibrada, e é chamada de calibrada e retificada quando seus parâmetros extrínsecos são conhecidos;

- Emparelhamento: Processo de determinar os pontos 2D de um objeto 3D em ambas as imagens, tornando possível a triangulação do objeto;

- Mapa de Disparidades: é uma terceira imagem que armazena as informações de profundidade da cena, que é gerada após a correspondência. O fluxo das etapas é ilustrado na Figura 2.4 conforme segue.

Existem, porém, problemas com os métodos descritos. O processo de calibração em ambientes dinâmicos é mais complexo, o que torna interessante o uso de técnicas mais autônomas com descrito por Orghidan et al.(ORGHIDAN et al., 2014).

Como descrito por Bradski (BRADSKI, 2008), a calibração tem como objetivo retificar o sistema estereoscópico, como demonstrado na Figura 2.5. Portanto, na etapa de

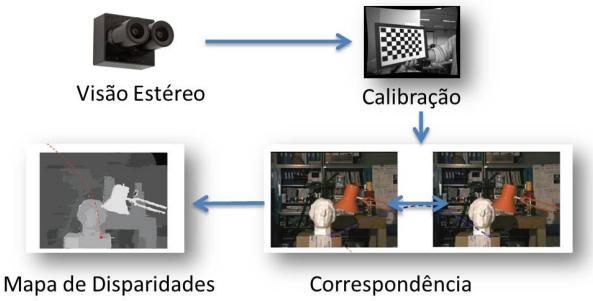


Figura 2.4 – Etapas do sistema estéreo.

calibração, define-se a matriz fundamental da câmera A

$$A = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

onde:

f_x = Distância focal em x
f_y = Distância foal em y
(c_x,c_y) = Centro ótico expresso em *pixels*

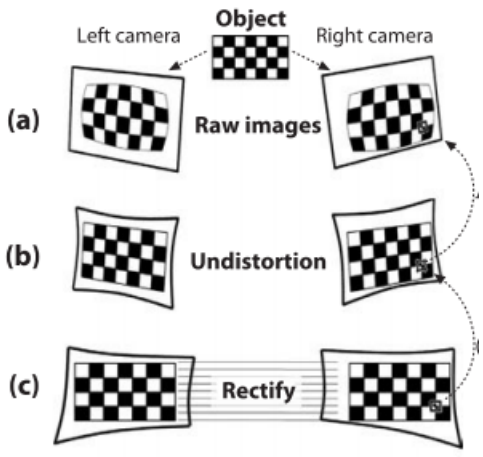


Figura 2.5 – Etapas da retificação do sistema estéreo.
Fonte: Adaptado de (BRADSKI, 2008)

E a matriz $[R|t]$, que é composta dos parâmetros de rotação e translação extrínsecos da câmera usados para definir seu posicionamento em torno de uma cena estática, e é responsável por traduzir as coordenadas 3D (X,Y,Z) do mundo para um sistema de coordenadas fixo em relação à câmera (BRADSKI, 2008). Em estereoscopia, estas matrizes são utilizadas para converter o sistemas de coordenadas de uma câmera para a outra, definindo-se a transformação de perspectiva:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = A[R|t] \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{2.4}$$

onde:

(X,Y,Z) = Coordenadas do ponto 3D no mundo
(u,v) = Coordenadas do ponto de projeção
A = Matriz fundamental da câmera

A transformação para correção de distorção, demonstrada na na imagem (b) da Figura 2.5, é dada por:

$$\begin{bmatrix} u' \\ v' \end{bmatrix} = \left(\frac{1 + k_1 r^2 + k_2 r^4 + k_3 r^6}{1 + k_4 r^2 + k_5 r^4 + k_6 r^6} \right) \begin{bmatrix} u \\ v \end{bmatrix} \begin{bmatrix} 2p_1 uv + p_2(r^2 + 2u^2) \\ p_1(r^2 + 2v^2) + 2p_2 uv \end{bmatrix} \tag{2.5}$$

onde:

(u',v') = Coordenada da imagem distorcida
(u,v) = Coordenada da imagem sem distorção
k_{1-6} = Coeficientes de distorção radial
p_1, p_2 = Coeficientes de distorção tangencial

A partir das matrizes A (2.4) de ambas as câmeras, e as transformações R e t entre ambas, é possível realiza a retificação de ambas, para que seja eliminada a disparidade vertical, demonstrada na imagem (c) da Figura 2.5. Define-se, então, as matrizes de projeção R_e, para a câmera esquerda, e R_d para a câmera direita, que são utilizadas para definir a câmera esquerda como o centro de projeção, da mesma maneira que define-se as matrizes de projeção P_e e P_d

$$P_e = \begin{pmatrix} & & & 0 \\ & A & & 0 \\ & & & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.6}$$

$$P_d = \begin{pmatrix} & & & t_x * f \\ & A & & 0 \\ & & & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.7}$$

onde:

A = Matriz fundamental da câmera
f = Distância focal
t_x = Componente x da matriz de translação t
$t_x * f$ = Conhecido como o *baseline* T_x

A matriz de conversão disparidade-profundidade, utilizada para transformar a disparidade disposta no mapa de disparidades pela correspondência em profundidade:

$$Q = \begin{bmatrix} 1 & 0 & 0 & c_x \\ 0 & 1 & 0 & c_y \\ 0 & 0 & 0 & f \\ 0 & 0 & \frac{-1}{T_x} & 0 \end{bmatrix} \tag{2.8}$$

onde:

(c_x,c_y) = Centro ótico da câmera
f = Distância focal
T_x = *baseline* entre as câmeras

A partir da calibração, é possível dar continuidade à etapa de emparelhamento, ou correspondência, onde mais dispensa-se processamento. Baydoun et al. (BAYDOUN; ALALAOUI, 2014) propõem uma nova abordagem que visa obter maior desempenho, nele ainda é destacado o maior problema na correspondência, a oclusão, onde alguns pixels aparecem em uma imagem porém na outra não. Outras abordagens que buscam solucionar desempenho bem como resolver oclusões são descritas por Cyganek et al.(CYGANEK; SIEBERT, 2011) e Ju et al., (JU; KANG, 2009). Como resultado da correspondência é gerado um mapa de disparidades composto por *pixels* com valores referentes a diferença de posição entre o *pixel* na imagem esquerda e direita, ou *pixels* sem valores, que são pontos onde não foram encontradas correspondências entre as imagens, chamados de *Bad Matched Pixels* (BMP), métrica que pode ser utilizada para identificar a eficácia do mapa, ou ainda o seu inverso, *Matched Pixels*, também chamados de densidade do mapa, como utilizados por Lazaros et al. (LAZAROS; SIRAKOULIS; GASTERATOS, 2008) e Geiger et al. (GEIGER; LENZ; URTASUN, 2012).

Os mapas de disparidade podem ser densos ou esparsos. O denso é um mapa mais completo porém requer a avaliação de todos os pontos envolvidos no par de imagens previamente, o que ocasiona em um custo computacional muito elevado. Em contrapartida mapas esparsos são computacionalmente menos exigentes, pois são apenas uma parte do anterior. Algumas abordagens, como a proposta por Yao et al. (YAO et al., 2012), buscam um equilíbrio entre as técnicas utilizando interpolação de mapas intermediários, ou realizando segmentação de imagens, como sugerido por Klaus et al. (KLAUS; SORMANN; KARNER, 2006).

Com o mapa de disparidades gerado, é possível realizar a reconstrução da cena 3D detectada pelas câmeras ao reprojetar a imagem de acordo com a disparidade utilizando a matriz Q e transformando o vetor homogêneo resultante

$$\begin{bmatrix} x' \\ y' \\ z' \\ w \end{bmatrix} = Q \begin{bmatrix} d_x \\ d_y \\ d_d \\ 1 \end{bmatrix} \implies \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{x'}{w} \\ \frac{y'}{w} \\ \frac{z'}{w} \\ \frac{w}{w} \end{bmatrix} \tag{2.9}$$

onde:

(x',y',z',w) = Ponto homogêneo resultado da reprojeção
Q = Matriz de transformação disparidade-profundidade
(d_x,d_y) = Ponto da imagem esquerda no mapa de disparidades
d_d = Disparidade associada ao ponto (d_x,d_y)
(x,y,z) = Posição do ponto 2D reprojetado em 3D

Quando a transformação é aplicada a cada ponto no mapa de disparidades, obtém-se cenas 3D semelhantes aos demonstrado na Figura 2.6.

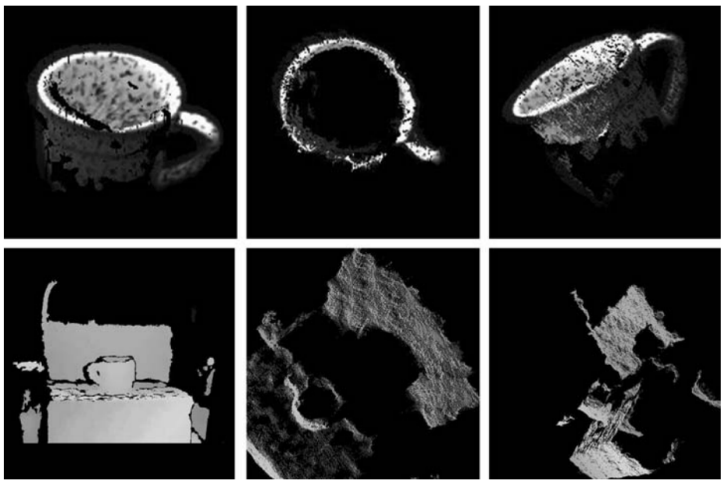


Figura 2.6 – Cenas 3D reconstruídas a partir do mapa de disparidades.
Fonte: (BRADSKI, 2008)

2.3 CONCEITOS BÁSICOS DE SLAM

Simultaneous Location and Mapping (SLAM) é uma área da robótica que surgiu com a necessidade de localizar um robô e, simultaneamente, mapear o ambiente a sua volta, e algumas das técnicas mais utilizadas são explanadas nas Seções 2.4 e 2.5. Primeiramente, alguns conceitos básicos de SLAM devem ser considerados.

Define-se que um robô tem informações *perceptuais*, que são informações observadas, de onde extraem-se *landmarks*, que são pontos de referência identificados no ambiente, e *odométricas*, que contém dados sobre o movimento realizado pelo robô (THRUN et al., 2001). Neste contexto, o *input* que faz as rodas de um robô terrestre andar é considerado uma informação *odométrica*, que é o ponto de partida para a predição da *pose*, enquanto todas as outras que devem ser observadas, como imagem de câmeras, sensores de distância, e sistemas estereoscópicos são consideradas informações *perceptuais*, de onde extraem-se os *landmarks*. Sistemas que utilizam informações perceptuais visuais,

como estereoscopia, definida na Seção 2.2, são considerados parte da subárea de *Visual* SLAM (VSLAM) (KARLSSON et al., 2005).

Uma *pose* é normalmente representada pela coordenada 2D ou 3D do robô e sua orientação (THRUN et al., 2001). Consequentemente, pode ser definido como as informações pertinentes da localização do robô.

Landmarks são objetos que o robô pode utilizar como pontos de referência para sua localização (WELCH; BISHOP, 1995). Estes objetos são identificados durante a movimentação do robô e reobservados em movimentos subsequentes (MOUTARLIER; CHATILA, 1990).

2.4 LOCALIZAÇÃO VIA FILTRO ESTENDIDO DE KALMAN

Inicialmente proposto por Smith and Cheeseman (SMITH; CHEESEMAN, 1986), com o algoritmo de mapeamento estocástico, que é uma abordagem ao SLAM com o filtro estendido de Kalman (EKF) (KALMAN, 1960), e posteriormente aplicado à computação por Welch e Bishop (WELCH; BISHOP, 1995). Nesta técnica, o estado do veículo e todas as características do mapa são coletados em um único vetor de estados, e, quando as características do mapa são reobservadas, tanto a *pose* e a localização destas características são atualizadas.

O filtro de Kalman, por sí, é um conjunto de equações matemáticas que providenciam uma solução eficiente e recursiva para o problema dos mínimos quadrados, e suporta aproximação do passado, presente e até estados futuros, podendo fazê-lo mesmo quando a natureza do sistema modelado é desconhecida. Já o *Extended Kalman Filter* (EKF), é a técnica de linearizar um sistema dinâmico não linear para uso no filtro de Kalman (ZUNINO, 2002).

$$x_{k|k} = X_k + \eta_k \tag{2.10}$$

Nesse contexto, a Equação 2.10 é usada para representar o vetor de estados $x = [x_r^T x_1^T x_2^T ... x_N^T]^T$ onde $x_r = [x_r y_r \theta_r]^T$ é uma estimativa da *pose* do robô, e $x_i = [x_i y_i]^T$ é a estimativa do estado do *landmark*, X é o estado real do vetor e η é a estimativa de erro. k é o indicador de tempo, significando que $x_{k+1|k}$ é usado para denotar o estado de x no tempo $k + 1$ dadas as informações do tempo k.

O erro de covariância estimado é dado pela matriz representada na Equação 2.11, onde as submatrizes P^{rr}, P^{ri} e P^{ii} são covariâncias de, respectivamente, robô para robô, robô para característica (*landmark*), e característica para característica. As submatrizes P^{ij} são as correlações característica-característica. O robô e o mapa são representados por um único vetor de estados x com a covariância estimada P a cada espaço de tempo.

Pode-se dizer, portanto, que esta matriz é uma matriz gaussiana de dimensão $(3 + 2n)$ onde n é o número de *landmarks* e representa o *belief*, a provável *pose* do robô.

$$P_{k|k} = \begin{bmatrix} P^{rr} & P^{r1} & \dots & P^{rN} \\ P^{1r} & P^{11} & \dots & P^{1N} \\ \vdots & \vdots & \ddots & \vdots \\ P^{Nr} & P^{N1} & \dots & P^{NN} \end{bmatrix} \tag{2.11}$$

Inicialmente, a matriz anterior contém apenas a *pose* do robô (P^{rr}), e os *landmarks* (P^{ii}) são adicionados de acordo com a sua observação, assim como suas matrizes Jacobianas de correlação *landmark*-*landmark* (P^{ij}) e de robô-*landmark* (P^{ri}), conforme descrito por Moutarlier et al. (MOUTARLIER; CHATILA, 1990). Posteriormente, os *landmarks* são reobservados e o estado do mapa é atualizado, também atualizando a *pose* do robô. O esboço desse processo pode ser visualizado na Figura 2.7, onde o robô prediz sua posição em (a) de acordo com a odometria, observa um *landmark* em (b) e reobserva o mesmo *landmark* em (c).

Já o fluxograma do algoritmo de EKF pode ser visto na Figura 2.8. Pode-se perceber que a odometria serve de *input* para a predição, que depois é comparada com a observação dos *landmarks* (*features*) já conhecidos e a adição de novos *landmarks* que são posteriormente adicionados ou alterados no mapa. Este processo é repetido constantemente enquanto o robô está ativo.

2.5 LOCALIZAÇÃO DE MONTE CARLO VIA FILTRO DE PARTÍCULAS

O método de localização de Monte Carlo (MCL) é um filtro de Bayes recursivo que estima a distribuição das *poses* do robô, dadas as informações dos sensores, como apontado por Thrun et al. (THRUN et al., 2001). O termo sensores é genericamente usado e pode se referir a sensores de distância e odômetros, assim como imagens de câmeras e dados computados com base em tais imagens. A ideia principal do filtro de Bayes é estimar a probabilidade da densidade das *poses*, o também chamado de *belief*.

A Equação 2.12 demonstra um passo do filtro de Bayes. De acordo com os conceitos de informações odométricas e perceptuais introduzidos na Seção 2.3, define-se que: x_t denota o estado no tempo t, a_t a informação odométrica (ação) em dado tempo, e o_t a informação perceptual (observação) em dado tempo, com t variando de 0-t.

$$Bel(x_1) = p(x_t|o_t,a_{t-1},o_{t-1},a_{t-2},...,o_0) \tag{2.12}$$

Levando em consideração que o método de Localização de Monte Carlo é o filtro de Bayes recursivo, algumas alterações na fórmula são feitas para tal. A fórmula final

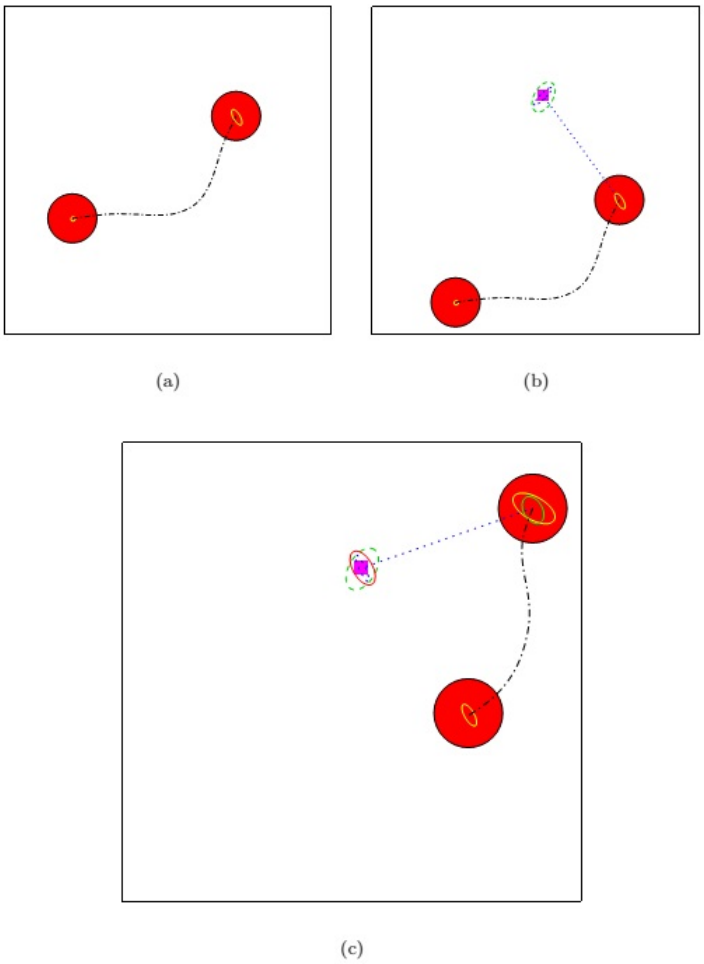


Figura 2.7 – (a) demonstra a predição em mapeamento estocástico, (b) demonstra a adição de *landmarks*, e (c) demonstra a reobservação de *landmarks*.
Fonte: (ZUNINO, 2002)

recursiva é demonstrada na Equação 2.13.

$$Bel(x_t) = \eta p(o_t|x_t) \int p(x_t|x_{t-1},a_{t-1}) Bel(x_{t-1}) dx_{t-1} \tag{2.13}$$

O resultado simplificado deste algoritmo aplicado pode ser visto na Figura 2.9, onde as linhas sólidas representam o movimento do robô, partido de *Starting location*, e as partículas representam o *belief* em pontos diferentes de tempo. Como foram utilizadas apenas informações odométricas (a_t), nota-se que o *belief* é menos preciso com cada movimentação do robô devido ao acúmulo de erros. Enquanto nas primeira iterações as partículas encontram-se bem definidas em torno da posição real, nas últimas o *belief* está mais espalhado, dificultando a predição da *pose* do robô. O uso de informações perceptuais (o_t) visa diminuir a área do *belief*.

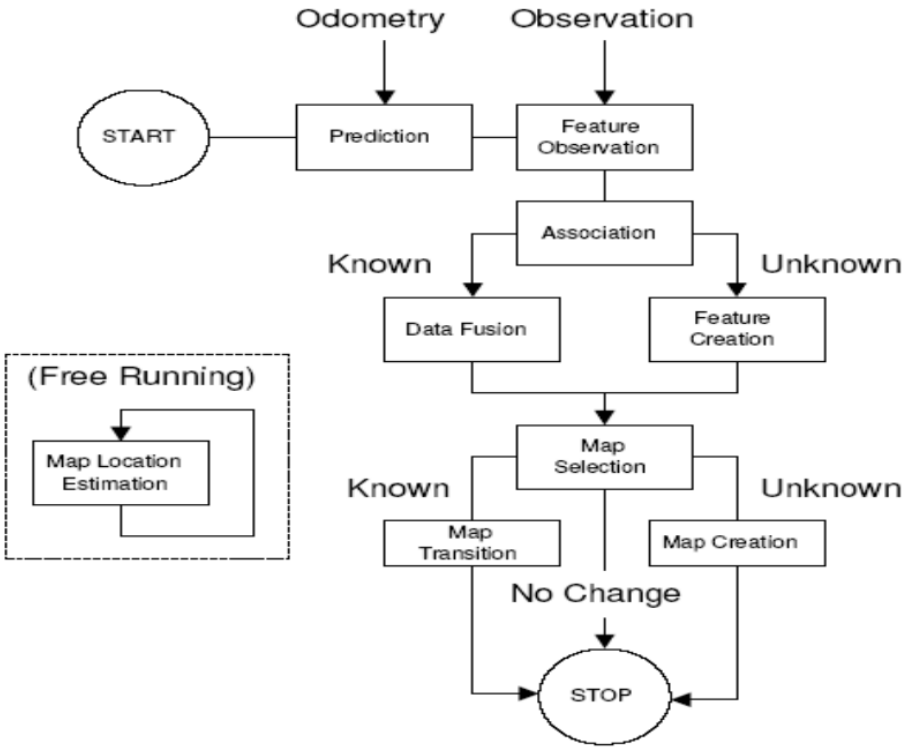


Figura 2.8 – Fluxograma do algoritmo de EKF.
Fonte: (LEONARD; NEWMAN, 2003)

2.6 REPRESENTAÇÃO DE POSIÇÃO

Uma *pose* é normalmente representada pela coordenada 2D ou 3D do robô e sua orientação (THRUN et al., 2001). As coordenadas 3D do sistema (x, y e z) podem ser representadas como a matriz de translação t demonstrada na Equação 2.14:

$$t = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{2.14}$$

A rotação é uma transformação linear, que pode ser expressada por uma matriz R ortogonal $3x3$ para base ortonormal R^3 (BOAS, 2006). Desta maneira, pode-se definir a *pose* como uma matriz homogênea $P = [R|t]$ demonstrada por:

$$Pose = \begin{pmatrix} & & & x \\ & R & & y \\ & & & z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.15}$$

onde:

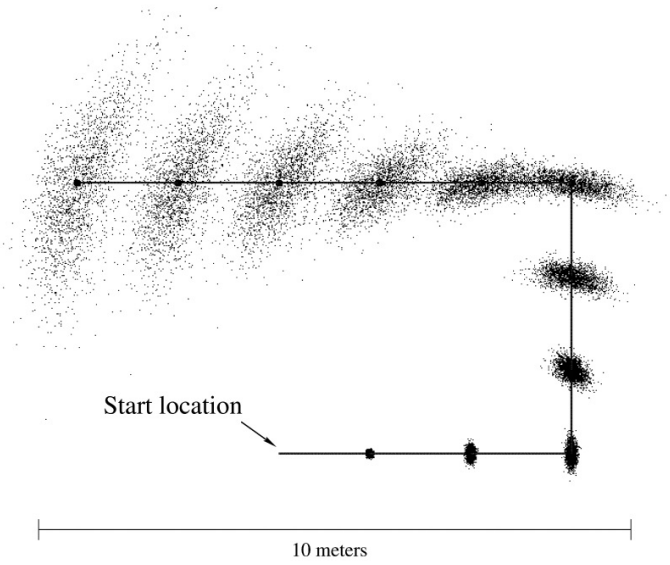


Figura 2.9 – Aproximação do *belief* do robô apenas com informação odométrica.
Fonte: (THRUN et al., 2001)

$Pose =$ Matriz que representa a *pose*
$R \quad =$ Submatriz de rotação 3x3
x,y,$z =$ Coordenadas 3D do sistema

A matriz de rotação R pode ser construída a partir da combinação de ângulos de Euler or por *quaternions*, por exemplo (MURRAY et al., 1994).

2.7 ÂNGULOS DE EULER

Os ângulos de rotação ϕ, θ, ψ associados aos eixos z, y e x, são chamados de ângulos de Euler, e também são conhecidos como *yaw*, *pitch*, *roll*. A partir da aplicação das matrizes de rotações $3x3$ ZYX, é obtida a matriz de rotação correspondente à transformação descrita pelos ângulos (MURRAY et al., 1994), definida como:

$$R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.16}$$

Os passos desta transformação podem ser visualizados na Figura 2.10, onde os vetores, em azul, são rotacionados em torno dos eixos ZYX, sucessivamente, em seus respectivos ângulos ϕ, θ, ψ, resultando nos vetores em vermelho.

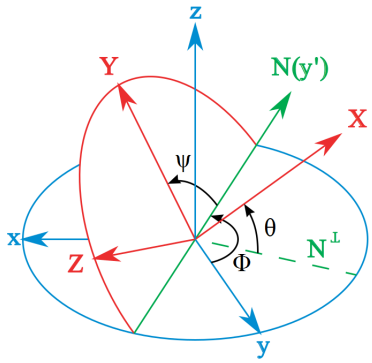


Figura 2.10 – Rotações aplicadas em sucessão.

2.7.1 *Gimbal lock*

A utilização de ângulos de Euler em aplicações 3D pode, em diversos casos, ocasionar em um *Gimbal lock*, que é a perda de um grau de liberdade na rotação. Como demonstrado na Equação 2.16, uma matriz de rotação resultante é dada a partir da aplicação de 3 rotações, uma para cada ângulo. Em casos onde $\theta = \frac{\pi}{2}$, por exemplo, a seguinte matriz de rotação é obtida:

$$R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.17}$$

Aplica-se a multiplicação de matrizes:

$$R = \begin{bmatrix} 0 & 0 & 1 \\ \sin\phi & \cos\phi & 0 \\ -\cos\phi & \sin\phi & 0 \end{bmatrix} \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ \sin\phi\cos\psi + \cos\phi\sin\psi & -\sin\phi\sin\psi + \cos\phi\cos\psi & 0 \\ -\cos\phi\cos\psi + \sin\phi\sin\psi & \cos\phi\sin\psi + \sin\phi\cos\psi & 0 \end{bmatrix} \tag{2.18}$$

Finalmente, aplicando as fórmulas trigonométricas:

$$R = \begin{bmatrix} 0 & 0 & 1 \\ \sin(\phi+\psi) & \cos(\phi+\psi) & 0 \\ -\cos(\phi+\psi) & \sin(\phi+\psi) & 0 \end{bmatrix} \tag{2.19}$$

A alteração dos ângulos ϕ e ψ acaba por ter o mesmo efeito de alterar o ângulo $\phi+\psi$, porém o eixo de rotação permanece o mesmo (z), visto que a última coluna e primeira linha da matriz não mudam. Desta maneira, um método alternativo de representação e obtenção de rotação deve ser utilizado para garantir os 3 graus de liberdade do sistema, como *Quaternions*.

2.8 *QUATERNIONS*

Quaternions são um sistema numérico que expande números complexos, e foram descritos por William Rowan Hamilton em 1843 e definidos como o quociente de dois vetores em um espaço tridimensional (SHENITZER; ROSENFELD; GRANT, 1988).

Quaternions podem ser utilizados para representar rotações (MURRAY et al., 1994), e são vetores quadridimensionais representados por:

$$Q = q_0 + q_1 i + q_2 j + q_3 k \tag{2.20}$$

onde:

Q = *Quaternion*
q_0 = Componente escalar
$\vec{q} = (q_1,q_2,q_3)$ = Componente vetorial

ou ainda:

$$Q = (w,x,y,z) \tag{2.21}$$

Considerando *quaternions* unitários, onde

$$\|Q\| = 1 \tag{2.22}$$

uma rotação ao redor de um eixo unitário arbitrário ω, demonstrado pela Figura 2.11, pode ser expressa da seguinte maneira:

$$Q = (\cos\frac{\theta}{2},\omega\sin\frac{\theta}{2}) \tag{2.23}$$

onde:

Q = *Quaternion*
θ = Ângulo de rotação
ω = Eixo de rotação de Euler

Resultando, desta maneira, na construção da seguinte matriz de rotação:

$$R = \begin{bmatrix} 1-2y^2-2z^2 & 2xy+2wz & 2xz-2wy \\ 2xy-2wz & 1-2x^2-2z^2 & 2yz+2wx \\ 2xz+2wy & 2yz-2wx & 1-2x^2-2y^2 \end{bmatrix} \tag{2.24}$$

Nota-se que, enquanto a matriz de rotação de ângulos de Euler é gerada por três rotações, uma em cada eixo principal, o *quaternion* é definido como uma rotação em torno de um eixo único, o que mantém seus 3 graus de liberdade e impede o problema de *Gimbal Lock*.

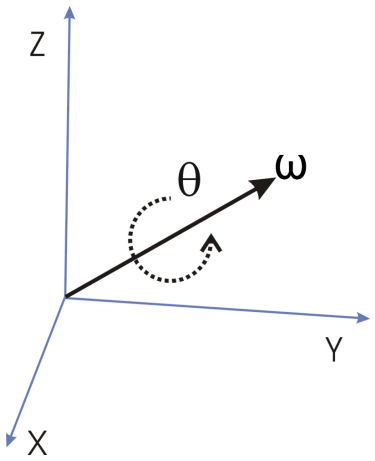


Figura 2.11 – Rotação θ em torno do eixo ω.

2.9 *SMARTPHONES* E SENSORES

Smartphones são dispositivos que estão crescendo no mercado desde o lançamento do *iPhone*, em 2007, e contêm diversos sensores que auxiliam nas diversas tarefas que podem ser realizadas nos mesmos. Devido à disponibilidade destes dispositivos e a gama de sensores contidos nos mesmos, há um aumento em trabalhos sobre SLAM e VSLAM que utilizam-os, predominantemente na plataforma Android, devido ao baixo custo associado (FARAGHER; SARNO; NEWMAN, 2012; SHIN; CHON; CHA, 2012; KAO; HUY, 2013; VENTURA et al., 2014).

Dentre os sensores utilizados para complementar aplicações SLAM contidos em *smartphones* estão: *Global Positioning System* (GPS), acelerômetro, giroscópio e magnetômetro. Algumas abordagens de SLAM para ambientes internos também utilizam redes WiFi, mensurando a intensidade de seus sinais visando triangular a posição, como proposto por Kao e Huy (KAO; HUY, 2013).

Leituras do sensor GPS na plataforma Android são compostas de latitude, longitude e o *timestamp* da leitura. Adicionalmente, podem conter informações de *bearing*, direção, em relação ao norte, do movimento (em °), altitude, velocidade de deslocamento (em m/s), e precisão da leitura (em m). Desta maneira, são possíveis extrair, além da posição do sistema, dados de sua movimentação.

Leituras de acelerômetros são compostas da aceleração detectada em cada um dos eixos 3D do sensor, e normalmente são ruidosas (FRADEN, 2003). No contexto de aparelhos Android, estes dados são obtidos em m/s^2 nos eixos dispostos de acordo com a Figura 2.12. Além das informações de aceleração, o *timestamp* da leitura é obtido, assim como sua precisão, calculada internamente pela plataforma, e dada como alta, média, ou baixa.

Utilizando a plataforma Android, tem-se a possibilidade de obter dados diretos do sensor em questão, assim como dados específicos obtidos a partir da decomposição das

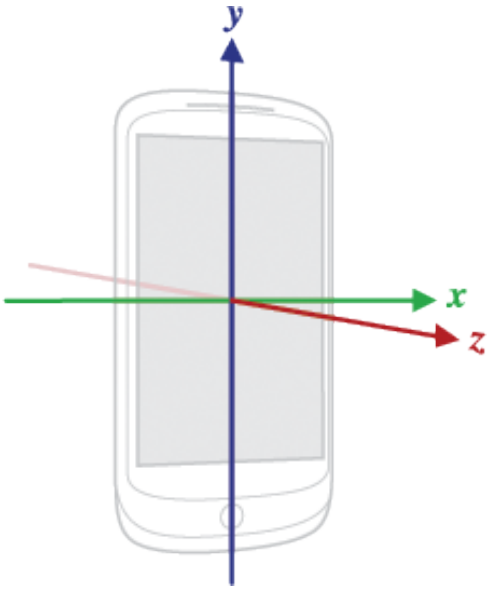


Figura 2.12 – Eixos do acelerômetro em um dispositivo Android.
Fonte: (GOOGLE, 2016)

informações originais, processo chamado de *fusão de sensores*. Desta maneira, a aceleração obtida pelo dispositivo é decomposta de acordo com a seguinte equação:

$$a = g + La \tag{2.25}$$

onde:

a = Aceleração
g = Gravidade
La = Aceleração Linear

Resultando assim nos sensores virtuais de gravidade e aceleração linear a partir de um filtro passa-baixa (GOOGLE, 2016). Desta maneira, o sensor de aceleração linear pode ser utilizado para detectar as forças sendo exercidas sobre o dispositivos, desconsiderando a força da gravidade, enquanto o sensor de gravidade pode ser utilizado para identificar a orientação do dispositivo em relação ao centro do planeta, obtendo assim seu *roll* e *pitch*.

O sensor giroscópio é responsável por medir a velocidade de rotação angular em cada eixo do sistema, em $radianos/segundo$, tendo valores positivos em rotações anti-horárias. A velocidade angular pode ser definida como a derivada da posição angular no tempo t:

$$\dot{\theta} = \frac{d\theta}{dt} \tag{2.26}$$

e, portanto, a posição angular pode ser obtida de acordo com a integração da velocidade angular

$$\theta(t) = \int_0^t \dot{\theta}(t)dt \approx \sum_t^0 \dot{\theta}(t)T_s \tag{2.27}$$

Embora o giroscópio seja um sensor com pouco ruído, a integral da velocidade angular aproximada sobre pontos discretos no tempo de amostragem T_s, o que resulta no acúmulo de erros, também chamado de *drift*.

A partir da medição do campo magnético em cada um dos eixos do dispositivos, realizada pelo magnetômetro, em micro-Tesla (μT), pode-se obter a orientação do dispositivo de acordo com o norte magnético, agindo como uma bússola, obtendo-se o *yaw* do dispositivo. Esta medição é suscetível a erros devido à variações no campo magnético provocadas por outros equipamentos eletrônicos ou metais ferromagnéticos contidos no ambiente.

Nota-se que a orientação completa do dispositivo (*row*, *pitch* e *yaw*) não pode ser obtida pela utilização de apenas um sensor. O acelerômetro é eficaz em obter o *row* e o *pitch*, porém contém muito ruído e o *yaw* não pode ser obtido. A partir do magnetômetro obtém-se o *yaw* em relação ao norte magnético da terra, mas é prejudicado por interferências externas. Em contraste, o giroscópio detecta a rotação em cada ângulo, porém não contextualiza-a em coordenadas reais e acumula erros com o tempo. Desta maneira, a plataforma Android realiza uma fusão entre os três sensores para determinar a orientação do dispositivo. O acelerômetro e o magnetômetro são utilizados para a identificação da orientação do dispositivo a longo termo, corrigindo o *drift* ocasionado pelo giroscópio, responsável por identificar a orientação do dispositivo a curto termo. A combinação destas informações resultam na obtenção da orientação do dispositivo em relação com coordenadas globais, dadas tanto na forma de ângulos de Euler (sensor de orientação), ou de *quaternion* (sensor de vetor de rotação).

O sensor de orientação da plataforma é dado em ângulos de Euler, com o *yaw* no intervalo $[0,360)$, representando o ângulo entre o dispositivo e o norte magnético, *pitch* no intervalo $[-180, 180]$, e o *roll* no intervalo $[-90,90]$.

Em contraste, o sensor vetor de rotação é dado por um *quaternion*, com suas componentes demonstradas na Figura 2.13, onde o componente x é o produto do vetor $y.z$ tangente ao solo, alinhado ao leste, y também é tangente ao solo, porém alinhado ao norte magnético, e z é perpendicular ao solo, apontando para o céu.

Como estabelecido na Seção 2.7.1, a utilização de ângulos de Euler pode gerar a perda de um grau de liberdade na rotação, o que torna-os inferiores a *quaternions* para a representação de rotação em aplicações práticas. Desta maneira, a utilização do sensor vetor de rotação da plataforma Android é mais precisa, tornando-se a melhor escolha para o sistema.

Devido ao fato de o magnetômetro ser utilizado para orientar o dispositivo de acordo com o norte magnético e o fenômeno de declinação magnética, demonstrado na Figura

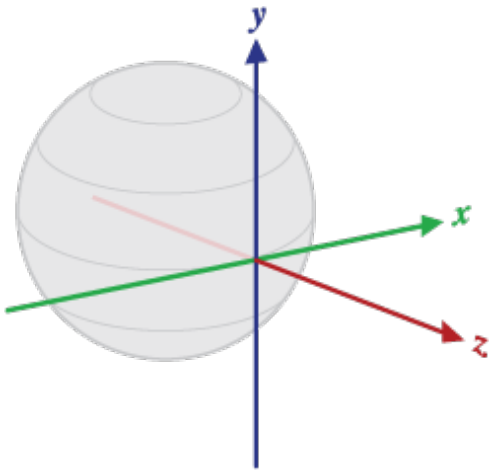


Figura 2.13 – Eixos do *quaternion* em relação ao mundo.
Fonte: (GOOGLE, 2016)

2.14, onde o campo magnético varia de acordo com a latitude da medição, é necessária uma correção para que o dispositivo seja alinhado com o norte real (FINLAY et al., 2010). Utiliza-se, então, as coordenadas obtidas pelo GPS para a estimação da declinação do campo magnético no local da medição, que então é subtraída da componente y do *quaternion*, que equivale ao *azimuth* (*yaw*) dos ângulos de Euler.

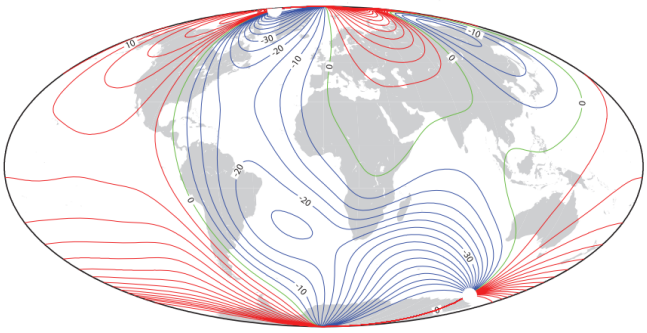


Figura 2.14 – Declinação magnética medida em 2010, com seus valores em graus.
Fonte: Adaptado de (FINLAY et al., 2010)

2.10 BIBLIOTECAS UTILIZADAS

Para facilitar a implementação do sistema, e levando em consideração o objetivo de baixo custo, foram utilizadas as seguintes bibliotecas livres: OpenCV, LIBVISO2, OpenGL e GLM.

2.10.1 OpenCV

A biblioteca OpenCV, por Brasdski (BRADSKI, 2000), é uma ferramenta de visão computacional que, no sistema proposto, é responsável pela gerência das câmeras (extração de frames e definição de parâmetros de configuração), algoritmo de calibração de sistema estereoscópico, implementação dos algoritmos de geração de mapa de disparidades de *Block Matching* (StereoBM), por Konolige (KONOLIGE, 1998), e *Semiglobal Block Matching* (StereoSGBM), por Hirschmuller (HIRSCHMULLER, 2008), e algoritmo de construção da nuvem de pontos a partir do mapa de disparidade.

Os exemplos de uso da biblioteca são demonstrados no Algoritmo 2.1, onde as linhas de 1 a 4 realizam uma captura de imagem, a linha 6 realiza uma conversão de cores, e as linhas de 6 a 8 geram o mapa de disparidades e constroem a nuvem de pontos.

Algoritmo 2.1 Exemplos de uso da biblioteca OpenCV

```
1  Mat image; //Classe Matriz da biblioteca
2  VideoCapture vCapture;
3  vCapture.open(deviceID); // Abre a camera para captura de video
4  vCapture.retrieve(imagem); //obtem a imagem
5
6  cvtColor(leftFrameColor, leftFrameGray, COLOR_BGR2GRAY); // conversao de cor
7
8  Mat bmdisp;
9  bm->compute(leftFrame, rightFrame, bmdisp); //computa a disparidade usando SBM
10 reprojectImageTo3D(bmdisp, pointCloud, Q); //construi a nuvem de pontos pointCloud
```

2.10.2 LIBVISO2

A biblioteca LIBVISO2, por Geiger et al. (GEIGER; ZIEGLER; STILLER, 2011), é responsável pela odometria visual computada a partir de ambas as imagens do sistema estereoscópico, utilizando a extração de características para identificar pontos que podem ser rastreados, que então são computados com uma variação de um *Kalman Filter*, descrito na Seção 2.4, resultando em uma estimativa de movimento. O Algoritmo 2.2 apresenta os exemplos de uso da biblioteca, em conjunto com o OpenCV, onde os parâmetros de distância focal, centro da imagem e *baseline* são extraídos nas linhas de 1 à 4, instancia-se o objeto da biblioteca na linha 5, e a linha 6 realiza a odometria visual.

Algoritmo 2.2 Exemplo de uso da biblioteca LIBVISO2

```
1 param.calib.f = leftCam->projectionMatrix.at<double>(0, 0); //distancia focal
2 param.calib.cu = leftCam->projectionMatrix.at<double>(0, 2); //centro cx
3 param.calib.cv = leftCam->projectionMatrix.at<double>(1, 2); //centro cy
4 param.base = -rightCam->projectionMatrix.at<double>(0, 3) /
      rightCam->projectionMatrix.at<double>(0, 0); //baseline a partir de -TX/f
5 VisualOdometryStereo viso(param); //instancia objeto a partir de param
6 viso.process(leftFrameGray.ptr(), rightFrameGray.ptr(), imageSize);
```

```
7 //processa frames equerdo e direito
```

2.10.3 OpenGL e GLM

A biblioteca gráfica OpenGL é utilizada para a renderização dos resultados e nuvem de pontos, enquanto a biblioteca *OpenGL Mathematics* (GLM) é utilizada pra auxiliar no cálculo de transformações lineares, como as de rotação. O Algoritmo 2.3 apresenta exemplos de uso, onde a renderização da nuvem de pontos dá-se nas linhas de 1 a 14, uma rotação dada por um *quaterion* é realizada na linha 16, e uma translação na linha 18.

Algoritmo 2.3 Exemplos de uso da biblioteca OpenGL e GLM

```
1  for (int i = 0; i < m_currentPointCloud.size().height; i++) //para cada linha
2  {
3    for (int j = 0; j < m_currentPointCloud.size().width; j++) //para cada coluna
4    {
5      Vec3f pt3f = m_currentPointCloud.at<Vec3f>(i, j); // obtem ponto da nuvem
6      if (pt3f[2] == 10000) //Nao renderiza bad matched pixel
7        continue;
8      Vec3b& pt3fc = m_currentPointCloudImage.at<Vec3b>(i, j); // obtem cor da imagem original
9      glColor3f(((float)pt3fc[2]) / 255.0, ((float)pt3fc[1]) / 255.0, ((float)pt3fc[0]) /
           255.0); // aplica cor
10     glBegin(GL_POINTS); //inicia desenho de pontos
11     glVertex3f(pt3f[0], pt3f[1], pt3f[2]); //desenha ponto
12     glEnd();
13   }
14 }
15
16 glMultMatrixd(glm::toMat4(quaternion)); //transforma quaterion em matriz e adiciona na pilha
       de matrizes
17
18 glTranslated(t[0],t[1],t[2]); //translada para o ponto t
```

3 TRABALHOS RELACIONADOS

Uma análise sobre trabalhos relacionados foi realizada para que fosse possível a comparação dos resultados obtidos no sistema proposto, assim como contextualizar o uso de determinados sensores e dispositivos.

3.1 *HIGH-ACCURACY DIFFERENTIAL TRACKING OF LOW-COST GPS RECEIVERS* (HEDGECOCK et al., 2014)

O trabalho de Hedgecock et al. (HEDGECOCK et al., 2014) demonstra o uso de GPS de frequência única em localização em tempo real. Neste trabalho é interessante destacar o baixo tempo de calibração do sistema, chegando a $1{,}79ms$ utilizando um computador, e $17{,}93ms$ utilizando um *smartphone* HTC Desire.

Adicionalmente, em outro trabalho do mesmo autor (HEDGECOCK et al., 2013) foram utilizados múltiplos GPS de baixo custo em dispositivos Android, e foi possível obter um erro médio de $9{,}1cm$ em um objeto estacionário, $25{,}3cm$ dirigindo em um ambiente obstruído, e $37{,}6cm$ à uma velocidade de $90km/h$.

Nestes dois trabalhos por Hedgecock et al. (HEDGECOCK et al., 2013, 2014) destaca-se a utilização de GPS de baixo custo, software relativamente rápido e erro geralmente abaixo de $1m$. Reforçando a eficácia do sistema, o trabalho de Agarwal e Habani (AGARWAL; HABLANI, 2011) demonstra a utilização de GPS em aeronaves com *smoothed pseudorange measurements*, onde foi utilizado um GPS de frequência dupla que não alcança o erro máximo horizontal e vertical permitidos de, respectivamente, $4{,}1m$ e $0{,}5m$. Contudo, utilizando o método proposto de melhor aproximação foi obtido um erro médio de $0{,}07m$ horizontal e $0{,}25m$ vertical. O autor ainda cita o possível uso de sistemas de navegação inerciais, que utilizam giroscópios, acelerômetros e magnetômetros, para a diminuir ainda mais o erro obtido.

Estes trabalhos servem de embasamento para o uso do sensor de GPS para uma estimação precisa da posição do sistema, assim como o uso de dispositivos Android para a implementação do sistema.

3.2 *A LOW-COST STEREO SYSTEM FOR 3D OBJECT RECOGNITION* (OLEARI; RIZZINI; CASELLI, 2013)

O trabalho de Oleari et al. (OLEARI; RIZZINI; CASELLI, 2013) propõe um sistema estereoscópico de baixo custo utilizando câmeras Logitech C720 de qualidade HD fixas em um pedestal, como demonstrado na Figura 3.1. Entretanto, as câmeras utilizadas eram capazes de capturar em resolução máxima apenas 10 *frames* por segundo, sendo necessário utilizar uma resolução menor para uma maior frequência de captura e sincronização via *software*. Apesar das limitações do sistema, os resultados do reconhecimento e reconstrução 3D dos objetos mostraram-se satisfatórios, obtendo-se uma taxa de reconhecimento acima de 80%.

Figura 3.1 – Sistema estereoscópico proposto por Oleari et al. (OLEARI; RIZZINI; CASELLI, 2013).

Fonte: Adaptado de (OLEARI; RIZZINI; CASELLI, 2013).

Devido à proposta de um mapeamento estático do trabalho, foram utilizados modelos 3D dos objetos mapeados como *ground truth*, não sendo possível relacionar o desempenho pelo trabalho e o obtido pelo sistema proposto. Destaca-se, contudo, o uso de estereoscopia em câmeras de baixo custo e não sincronizadas via *hardware*, da similar ao sistema proposto neste trabalho.

3.3 *VISION SYSTEM FOR MOBILE ROBOTS FOR TRACKING MOVING TARGETS, BASED ON ROBOT MOTION AND STEREO VISION INFORMATION* (KIM et al., 2011)

Com o intuito de rastrear objetos em movimento utilizando um robô móvel, no trabalho de Kim et al. (KIM et al., 2011) é proposto o sistema estereoscópico apresentado na Figura 3.2, que é composto de câmeras *Bumblebee2* sincronizadas via *hardware* utilizando a captura de imagens à 5 *frames* por segundo com uma resolução de $320x240$ *pixels*. Dois motores são responsáveis por rotacionar o sistema de câmeras, direcionando-as ao objeto em movimento, e um giroscópio para compensar a movimentação do robô por curvas e inclinações, como demonstrado na Figura 3.3.

A partir da combinação do giroscópio e o sistema estereoscópico foi possível detectar o objeto e direcionar as câmeras em ambos os cenários de teste, o que não foi

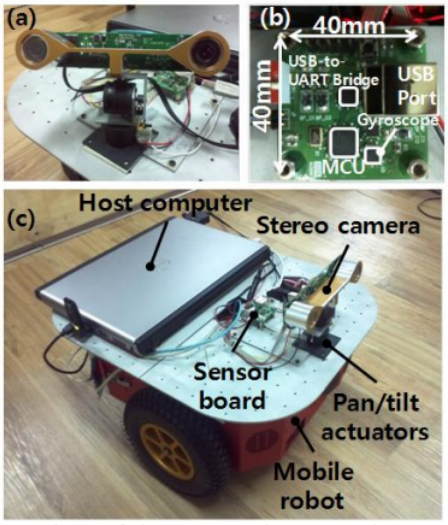


Figura 3.2 – Sistema proposto por Kim et al. (KIM et al., 2011). (a) Sistema estereoscópio atrelado aos motores de compensação de movimento. (b) Placa de sensores. (c) Sistema completo.

Fonte: Adaptado de (KIM et al., 2011).

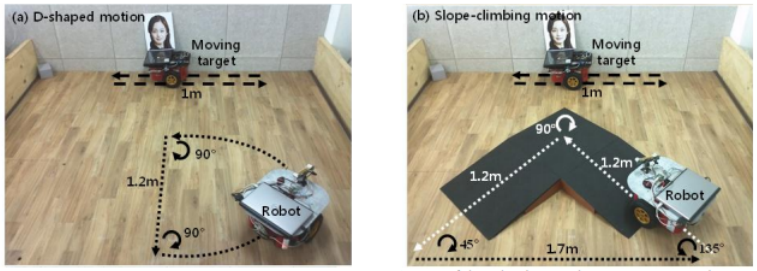


Figura 3.3 – Trajetórias (a) e (b) do robô proposto por Kim et al. (KIM et al., 2011).

Fonte: Adaptado de (KIM et al., 2011).

possível utilizando apenas dados visuais. O erro em cenários onde o robô movimenta-se linearmente ou rotaciona em seu próprio eixo também foram reduzidos.

Os resultados apresentados no trabalho são relacionados ao acerto de identificação do objeto em cada cenário, assim como o erro visual (em *pixels*) observado nas imagens. Desta maneira, os resultados não podem ser diretamente relacionados ao sistema proposto neste trabalho, porém destaca-se a utilização do giroscópio para compensar a rotação do sistema, de uma maneira similar à que é apresentada neste trabalho.

3.4 *REAL-TIME LOCALIZATION IN OUTDOOR ENVIRONMENTS USING STEREO VISION AND INEXPENSIVE GPS* (AGRAWAL; KONOLIGE, 2006)

O trabalho de Agrawal e Konolige (AGRAWAL; KONOLIGE, 2006) propõe um robô terrestre, demonstrado na Figura 3.4, que utiliza informações odométrica de suas rodas, sensores inerciais proprietários, um GPS Garmin e um par de sistemas estetoscópicos que utilizam câmeras *Bumblebee* sincronizadas via *hardware* com uma *baseline* de $12cm$. O robô anda por um terreno e retorna ao mesmo ponto, sendo possível calcular o erro de sua trajetória a partir do erro de fechamento de *loop*, ou *loop closure error* (LCE). Os autores ainda destacam que a baixa *baseline* das câmeras ocasiona em um maior erro no cálculo de distâncias.

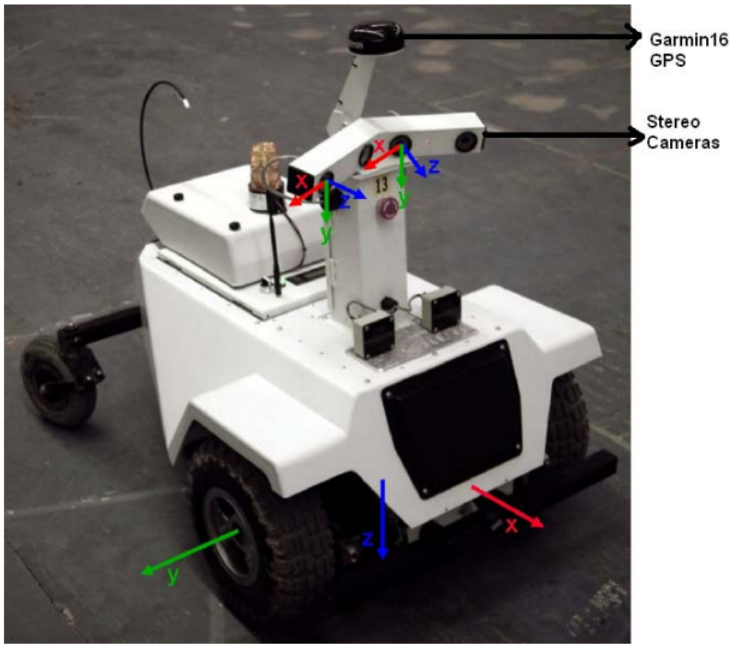


Figura 3.4 – Robô proposto por Agrawal e Konolige (AGRAWAL; KONOLIGE, 2006).

Fonte: Adaptado de (AGRAWAL; KONOLIGE, 2006).

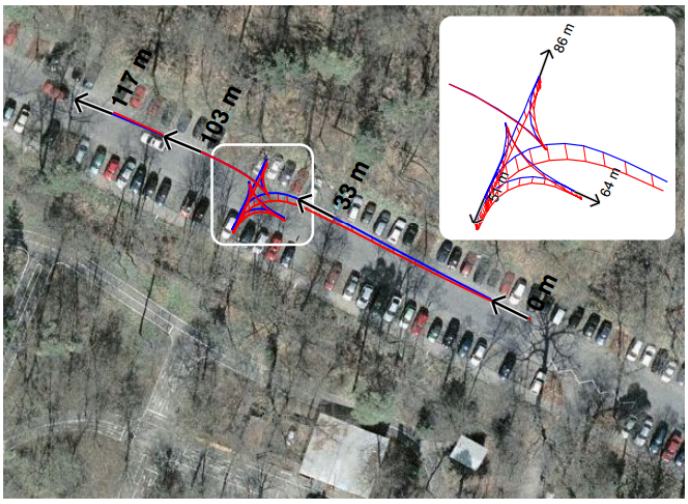
Para minimizar o erro do GPS, os *loops* realizados pelo robô são acima de $50m$, e a leitura do sensor só é considerada após o sistema viajar uma determinada distância, evitando que a posição do robô mude quando ele está estacionário. Dados de velocidade também são apenas utilizados quando o robô locomove-se à mais de $0{,}5m/s$. Desta maneira, foi possível reduzir consideravelmente o LCE do sistema.

Nos cenários propostos, foi possível obter um LCE de $2{,}2\%$ à 5% utilizando apenas odometria visual auxiliada por sensores inerciais, e $1{,}3\%$ à 31% utilizando odometria veicular. Ao realizar a combinação entre o GPS e ambas as odometrias, o erro obtido foi de $0{,}3\%$ à 2%. Desta maneira, podem-se relacionar diretamente os resultados obtidos entre

o trabalho de Agrawal e Konolige e o sistema proposto, visto que os *datasets* gerados por este também formam um *loop* que pode ser utilizado para estimar o erro.

3.5 *VISUAL ODOMETRY BASED ON STEREO IMAGE SEQUENCES WITH RANSAC-BASED OUTLIER REJECTION SCHEME* (KITT; GEIGER; LATEGAHN, 2010)

O trabalho de Kitt et al. (KITT; GEIGER; LATEGAHN, 2010) propõe uma técnica de odometria visual utilizando um método de *Kalman Filter* (KF) chamado *Iterated Sigma Point Kalman Filter* (ISPKF), que foram aplicadas em diversos *datasets* por Geiger et al. (GEIGER; ROSER; URTASUN, 2010). Desta maneira, os autores não dedicam tempo à geração das imagens, apenas ao seu processamento e comparação de seus resultados ao *ground truth* do *dataset*, dado por sensores posicionais de alta qualidade. A Figura 3.5 apresenta os trajetos processados pelo algoritmo de odometria visual.



(a) Driven path for a sequence with 1200 frames.

(b) Driven path for a sequence with 1200 frames.

Figura 3.5 – Trajetos (a) e (b) processados no trabalho de Kitt et al. (KITT; GEIGER; LATEGAHN, 2010).

Fonte: Adaptado de (KITT; GEIGER; LATEGAHN, 2010).

O algoritmo resultante do trabalho de Kitt et al. é utilizado na biblioteca LIBVISO1, versão anterior à LIBVISO2. Também é demonstrado um comparativo entre os algoritmos de *Iterated Extended Kalman Filter*, *Extended Kalman Filter*, *Unscented Kalman Filter* e *Iterated Sigma Point Kalman Filter* para odometria visual em simulações, onde as técnicas apresentaram erros similares, entre $33{,}3m$ e $34{,}3m$, com exceção do EKF, que apresentou um erro de $105{,}9m$. Desta maneira justifica-se o uso do ISPKF devido ao seu menor tempo de processamento e erro similar às alternativas.

A partir da aplicação do algoritmo sobre os *datasets*, foi possível obter um erro de

$24{,}29m$, correspondente a $1{,}3\%$ da distância percorrida. Este erro pode ser diretamente relacionado ao erro de fechamento de *loop* obtido no sistema proposto neste trabalho.

3.6 CONSIDERAÇÕES

Destaca-se o uso de câmeras sincronizadas via *hardware* em robôs móveis (AGRAWAL; KONOLIGE, 2006; KIM et al., 2011), enquanto não há esta preocupação em sistemas estáticos (OLEARI; RIZZINI; CASELLI, 2013), visto que não há movimentação que cause dessincronização dos *frames* capturados. Desta maneira, há um desafio em utilizar câmeras de baixo custo e sincronizá-las via *software* para que sejam viáveis em robôs móveis.

O uso de GPS e sensores é utilizado em diversos trabalhos (AGRAWAL; KONOLIGE, 2006; KIM et al., 2011; HEDGECOCK et al., 2014), reduzindo o erro obtido em comparação com a utilização de apenas um tipo de odometria. Com base nestas informações, busca-se corrigir erros de odometria visual ocasionados pela dessincronização dos *frames* das câmeras.

É possível fazer um comparativo direto entre o erro baseado na distância total percorrida do sistema com os trabalhos de Kitt et al. (KITT; GEIGER; LATEGAHN, 2010) e Agrawal e Konolige (AGRAWAL; KONOLIGE, 2006), assim como validar os métodos de sincronização ao aplicá-lo em um *dataset* de Geiger et al. (GEIGER; ROSER; URTASUN, 2010).

Embora o trabalho de Kitt et al. (KITT; GEIGER; LATEGAHN, 2010) também faça o processamento sobre *datasets*, não há a preocupação com a geração dos mesmos, que é realizada com câmeras de alto custo sincronizadas via *hardware*, com uma alta *baseline* de em torno de $60cm$, e um ângulo de visão horizontal de 100, o que difere deste trabalho, onde a geração de *datasets* é um dos objetivos apresentados, e as câmeras comuns utilizadas não dispõe desta sincronização, tem uma *baseline* de $12cm$, um menor ângulo de visão horizontal, de 70, e utiliza-se dados sensoriais para reduzir o erro obtido.

Destaca-se o alto custo associado ao sistema de Agrawal e Konolige (AGRAWAL; KONOLIGE, 2006), devido ao uso de dois pares de câmeras sincronizadas via *hardware* com uma *baseline* de $12cm$, uma placa de sensores inerciais, GPS, e um robô com dados odométricos. Em contraste, o sistema proposto neste trabalho utiliza alternativas de menor custo, utilizando apenas um par estereoscópico, composto de câmeras comuns sincronizadas via *software* com a mesma *baseline*, dados sensoriais e GPS unificados em um dispositivo Android, e sem informações odométricas. A *baseline* utilizada é de $12cm$.

Portanto, visando um custo menos elevado que o dos sistemas de Agrawal e Konolige (AGRAWAL; KONOLIGE, 2006) e o utilizado por Geiger et al. (GEIGER; ROSER; URTASUN, 2010) na geração dos *datasets* utilizados por Kitt et al. (KITT; GEIGER; LATEGAHN, 2010), este trabalho propõe um sistema que utilizam câmeras sem sincronia

via *software*, similar às utilizadas por Oleari et al. (OLEARI; RIZZINI; CASELLI, 2013) em um ambiente estático, e sensores posicionais (AGRAWAL; KONOLIGE, 2006; KIM et al., 2011; HEDGECOCK et al., 2014) integrados em um dispositivo Android para detectar a sua localização durante seu movimento. Os resultados da localização composta de uma sincronização via *software*, em conjunto com a odometria visual e dados sensoriais podem ser comparados com os trabalhos de Kitt et al. (KITT; GEIGER; LATEGAHN, 2010) e Agrawal e Konolige (AGRAWAL; KONOLIGE, 2006).

4 SISTEMA PROPOSTO

Neste capítulo é proposto um ambiente para aplicação de SLAM e analisar-se-ão os possíveis problemas das propostas existentes.

4.1 AMBIENTE PROPOSTO

O ambiente proposto para a aplicação é um sistema estereoscópico em um ambiente externo, de baixa velocidade, e dinâmico, não limitado a determinados tipos de terreno ou características. Este sistema é responsável pela obtenção de imagens para a extração de *landmarks*, além de reconstruir a nuvem de pontos da cena em 3D em relação ao sistema. Em adição às câmeras, o sistema conta com os seguintes sensores perceptuais: *Global Positional System* (GPS), acelerômetro, giroscópio, e magnetômetro, através de um *smartphone* Android. Semelhante ao trabalho de Agrawal e Konolige (AGRAWAL; KONOLIGE, 2006) é utilizado uma *baseline* de $12cm$ entre as câmeras.

Percebe-se que não existem informações odométricas disponíveis para os métodos MCL e EKF, descritos no Capítulo 2, Seções 2.5 e 2.4, exceto a odometria visual. Em um robô terrestre comum, as informações odométricas da movimentação é dada pelas rodas, as quais têm o movimento facilmente medido a partir do *input* e a derrapagem pode ser corrigida, como proposto por Boucher et al. (BOUCHER; ABABSA; MALLEM, 2013).

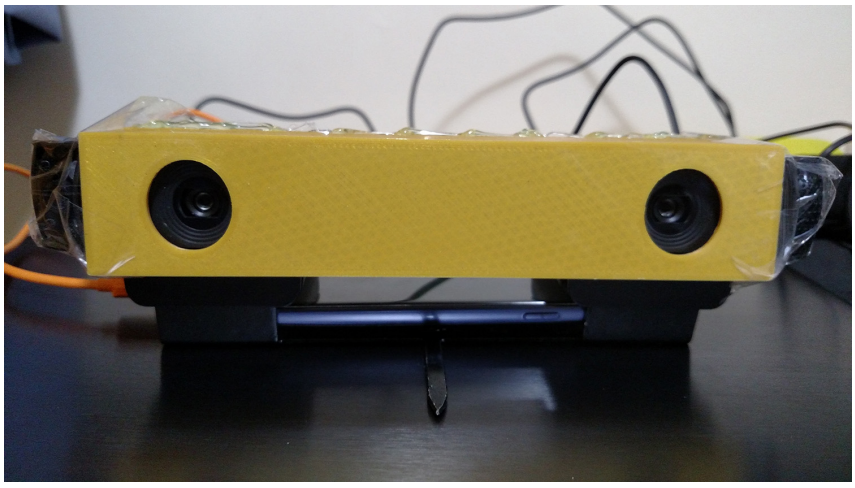


Figura 4.1 – Sistema estereoscópico utilizado.

A Figura 4.1 mostra o sistema estereoscópico proposto, composto por duas câmeras Logitech C920 de qualidade Full HD com um ângulo de visão horizonal de em torno de 70, uma geração superior ao modelo utilizado por Oleari et al. (OLEARI; RIZZINI; CASELLI, 2013) em seu sistema, um suporte auxiliar de alinhamento impresso em impressora 3D, e

um Samsung Galaxy S3 acoplado em sua parte inferior. Destaca-se a natureza modular do sistema, com cada elemento comunicando-se via USB ou *Bluetooth*, sendo possível a substituição de cada módulo por um dispositivo diferente. Para a geração de *dataset* o sistema foi ligado a um *ultrabook* Dell com processador Intel i7-4500U, 8GB de memória e um SSD Samsung 840 EVO de 500GB, e os processamentos foram realizados em um computador com processador Intel i7-4790k e 24GB de memória, ambos com sistema operacional Windows.

4.1.1 Correção de foco da câmera

A câmera utilizada apresenta foco automático de 20 estágios, porém, devido às suas especificações, o último estágio de foco tem distância máxima de $3m$, o que torna ruins as imagens capturadas em ambientes maiores que a distância do foco. Para corrigir este problema, as câmeras necessitaram ser modificadas para que fosse possível alterar o estágio de foco máximo, rotacionando sua lente interna no sentido horário como apresentado na Figura 4.2 pela seta azul. Desta maneira, foi possível atingir o nível de foco infinito, possibilitando sua aplicação em ambientes maiores que $3m$.

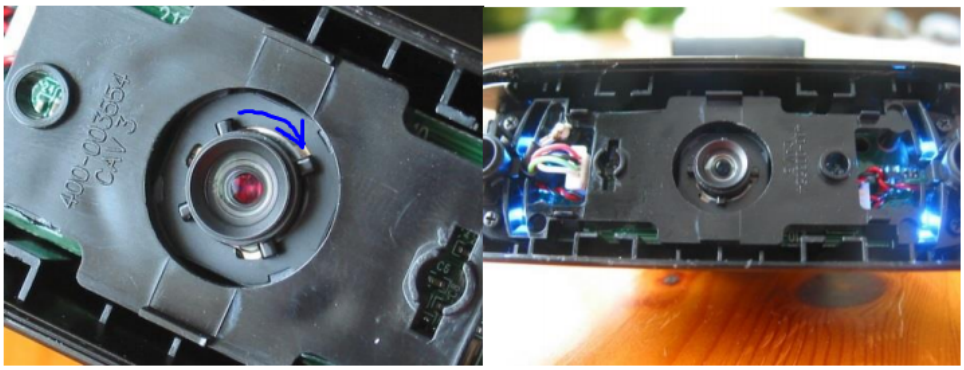


Figura 4.2 – Ajuste de estágio máximo de foco.

4.2 ARQUITETURA DO SISTEMA

A arquitetura do sistema pode ser descrita de acordo com o diagrama *top-down* apresentado na Figura 4.3. As câmeras comunicam-se com o *notebook* via USB, tendo sua interface de comunicação gerida pela biblioteca OpenCV. Os dados dos sensores do *smartphone* são obtidos pelo aplicativo Android desenvolvido, e são transmitidos utilizando o protocolo TCP/IP, também via USB.

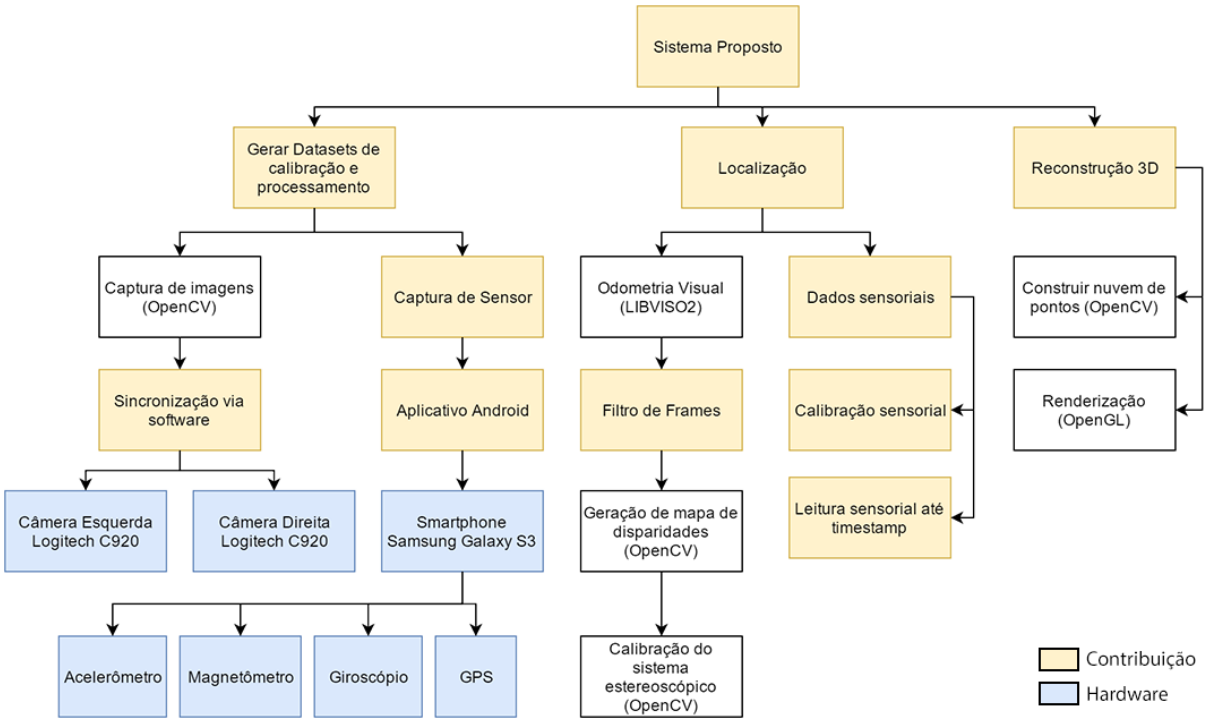


Figura 4.3 – Diagrama *top-down* do sistema proposto.

Devido ao alto tempo de processamento, optou-se pela geração de *datasets* para que possam ser processados posteriormente. A partir destes, é possível calibrar os componentes, retificando o sistema estereoscópico e os sensores, e extrair a localização do sistema, filtrando os *frames* sincronizados para que possam ser processados pela odometria visual e integrando-a aos dados sensoriais obtidos até o momento de captura do *frame*.

Os dados de localização são então utilizados para calcular o erro de fechamento de *loop* (LCE) do sistema e realizar a reconstrução 3D, onde a nuvem de pontos é projetada em 3D utilizando OpenGL. A partir dos resultados obtidos é possível traçar um comparativo entre o sistema proposto e os trabalhos relacionados.

4.3 INFORMAÇÕES PERCEPTUAIS DO SISTEMA

Na Seção 4.1 foram definidos alguns sensores perceptuais do sistema que são utilizados para obter os seguintes dados:

- GPS: Informação posicional horizontal e vertical;
- Acelerômetros: Informações de movimento fino que podem auxiliar os outros sensores;
- Magnetômetro e Giroscópio: Informação de orientação do sistema;

- Sistema estereoscópico: Identificação de *landmarks*, objetos a serem mapeados e suas respectivas localizações em relação ao sistema.

Para estimar a precisão do SLAM para o sistema deve-se levar em consideração o erro dos sensores de localização. De acordo com o trabalho de Hedgecock, pode-se concluir que os erros horizontais e verticais devem ser inferiores a $0{,}5m$, considerando os erros de GPS apresentados, a baixa velocidade de movimento do ambiente proposto, e a baixa interferência em um ambiente externo.

O sistema estereoscópico pode ser utilizado com grande precisão para localizar objetos relativos ao sistema, de acordo com Legg et al. (LEGG et al., 2014). Neste contexto, o sistema estereoscópico serve como o sensor de distância, simultaneamente identificando possíveis *landmarks* e adicionando objetos ao mundo virtual 3D. Este processo é definido na Seção 4.5.2.

Como destacado na Seção 4.1, não rodas para cálculo de odometria no sistema proposto. Isto é, não há maneiras, a partir do *input*, de informar a movimentação do robô ao sistema. Assim, de acordo com a Figura 2.7, o passo de predição da posição é inexistente. Sendo assim, o sistema deve orientar-se apenas pelas informações perceptuais.

4.4 MÓDULO DE GERAÇÃO DE *DATASET*

Para uma melhor consistência de resultados, foi implementado um módulo de geração de *dataset* que permite um maior controle sobre a captura de informações, permitindo com que diferentes tipos de processamento possam ser realizados posteriormente, para que não causem interferência com a captura dos dados. Um *dataset* deste sistema é composto por:

- Imagens de ambas as câmeras, em forma de vídeo;
- Arquivo com o tempo de obtenção da imagem;
- Arquivo com dados sensoriais, identificados com o seu tempo de obtenção.

4.4.1 Captura de imagens

Para a realização da captura das imagens do sistema estereoscópico, a biblioteca OpenCV (Capítulo 2, Seção 2.10.1) foi utilizada como interface entre hardware e software devido à sua ampla documentação e uso (YEO; LEE; LIM, 2015; RUSU et al., 2010; SZELISKI, 2010).

Devido ao objetivo de criar um sistema estereoscópico de baixo custo, as câmeras não são sincronizadas via *hardware*, visto que estes tem um custo elevado, como as câmeras Bumblebee, utilizadas nos trabalhos de Kim et al. (KIM et al., 2011), e Argwal e Konolige (AGRAWAL; KONOLIGE, 2006). Desta maneira, visando minimizar a diferença de tempo entre a captura dos *frames* de cada câmera, foi implementado um algoritmo de sincronização via *software* utilizando *threads* e um método de sincronização estilo semáforo, demonstrado pelo Algoritmo 4.1. As linhas de 1 a 8 demonstram a função executada em paralelo, enquanto as linhas 11 e 12 demonstram a criação das *threads* responsáveis por sua execução. A sincronização entre as *threads* é dada pelas linha 5 e de 16 a 19. Finalmente, a escrita dos *frames* no disco é dada pela linha 24.

Algoritmo 4.1 Captura de imagens em diferentes *threads*.

```
1  void retrieve_thread(camera c) //thread de captura
2  {
3    while(shouldRun) //enquanto o programa estiver executando
4    {
5      lock.wait(); //aguarda ate que ambas as threads estejam sincronizadas e ate receber o
           sinal de inicio de captura
6      c.retrieve(imagem) //decodifica o ultimo frame.
7    }
8  }
9
10 //MAIN THREAD
11 leftThread = new thread(retrieve_thread,leftCam); //cria thread para camera esquerda
12 rightThread = new thread(retrieve_thread,rightCam); //cria thread para camera direita
13 lastFrameTimestamp = 0; //inicializa o timestamp inicial
14 while(true)
15 {
16   lock.wait_threads(); //aguarda ate as 2 threas estarem sincronizadas
17   lock.notify(); //ordena a captura de frame
18   leftThread.wait_until_frame(); //aguarda o frame ser obtido
19   rightThread.wait_until_frame();
20   currentTimestamp = lastTimestamp; //o tempo do frame atual e o tempo da captura do ultimo
         frame
21   lastTimestamp = now();
22   if(currentTimestamp != 0) //descarta o frame inicial, que nao estava sincronizado
23   {
24     recordFrames(leftFrame,rightFrame, currentTimestamp); //armazena o frame em disco e o
           timestamp em um arquivo
25   }
26 }
```

Para otimizar a escrita dos dados no disco foi utilizado *buffers* de escrita, armazenando os *frames* em memória até o recurso ser liberado. O processamento destes *buffers* é executado em *threads* individuais, otimizando a escrita em disco ao realizá-la simultaneamente às *threads* de captura, também evitando seu bloqueio quando a escrita leva mais tempo que o esperado. Desta maneira foi possível manter uma taxa de captura de 30 *frames* por segundo com uma diferença de sincronia inferior a um *frame*, o que resulta em uma diferença máxima de tempo de $33ms$ entre a obtenção dos *frames* de cada câmera.

4.4.2 Informação sensorial utilizando Android

Para obter as informações sensoriais foi desenvolvido um aplicativo Android que pode ser executado em qualquer *smartphone* que utiliza o sistema operacional na versão 4.3 ou superior. A escolha desta plataforma também traz a vantagem da possibilidade de uso de sua API, que faz um pré-tratamento dos sensores. Assim, pode-se definir que os sensores relevantes para a aplicação, de acordo com a API, são:

- GPS: Informação posicional horizontal e vertical;
- Vetor de rotação: Um *quaternion* que representa a rotação do dispositivo;
- Aceleração linear: Um vetor que representa a força de aceleração exercida no sistema, já excluindo a gravidade.

A velocidade de obtenção dos dados foi configurada para a máxima possível, que, nos testes realizados, mostrou-se abaixo de $10ms$ para o vetor de rotação e aceleração linear, e abaixo de $1s$ para o GPS. Esta abordagem foi escolhida, pois, considerando o pior caso de uma câmera capturar um quadro a cada $33ms$ (totalizando aproximadamente 30 quadros por segundo), o maior *delay* entre a captura e o sensor será menor que $10ms$ atingindo uma melhor sincronização.

Sensores de dispositivo *android* devem ser calibrados para que os resultados de seus sensores sejam precisos (GOOGLE, 2016). Sendo assim, a calibração do magnetômetro é realizada rotacionando o dispositivo duas vezes em cada um de seus eixos. Devido á diferenças eletromagnéticas em pontos diferentes do planeta, o GPS é utilizado para estimar o campo eletromagnético no local e definir um *offset* entre o norte magnético (detectado pelo magnetômetro), e o norte verdadeiro. Quanto ao acelerômetro, sua calibragem é feita no módulo de processamento de *dataset*, descrito na Seção 4.5.1, e consiste em manter o dispositivo estacionário para a definição de um *offset*, dado pela média do acelerômetro no período.

Considerando que o acelerômetro é dado em relação aos eixos do dispositivo, e não às coordenadas globais, este deve ser corrigido de acordo com a orientação do dispositivo no momento. Esta correção é feita pela multiplicação da matriz dada pelo vetor de rotação e o vetor de aceleração.

De acordo com o Capítulo 2, Seção 2.9, outro fator importante é o fato de o vetor de rotação utilizar simultaneamente o giroscópio e magnetômetro. Como sensores podem ter acúmulo de erros ao longo de seu uso (*drift*), a rotação dada pelo giroscópio é corrigida pelo magnetômetro (utilizando o campo magnético da terra) para garantir a mesma orientação dado o norte magnético. Desta maneira, momentos que o sistema para de rotacionar são suficientes para que o vetor seja realinhado, aumentando sua precisão.

Visando uma menor latência na transferência dos dados, a comunicação entre o *smartphone* e o programa de geração de *dataset* é realizada utilizando protocolo TCP/IP,

o que mantém os dados na mesma ordem em que foram enviados, via USB, mantendo uma conexão cabeada com menor latência que as alternativas disponíveis nativamente no dispositivo, como *Bluetooth* e *Wi-Fi*. A Figura 4.4 mostra o aplicativo Android resultante (a), assim como a saída de dados (b) e a orientação do dispositivo representada em 3D (c). A reta de cor preta, ao lado do *Teapot* representa a direção para onde as câmeras apontam.

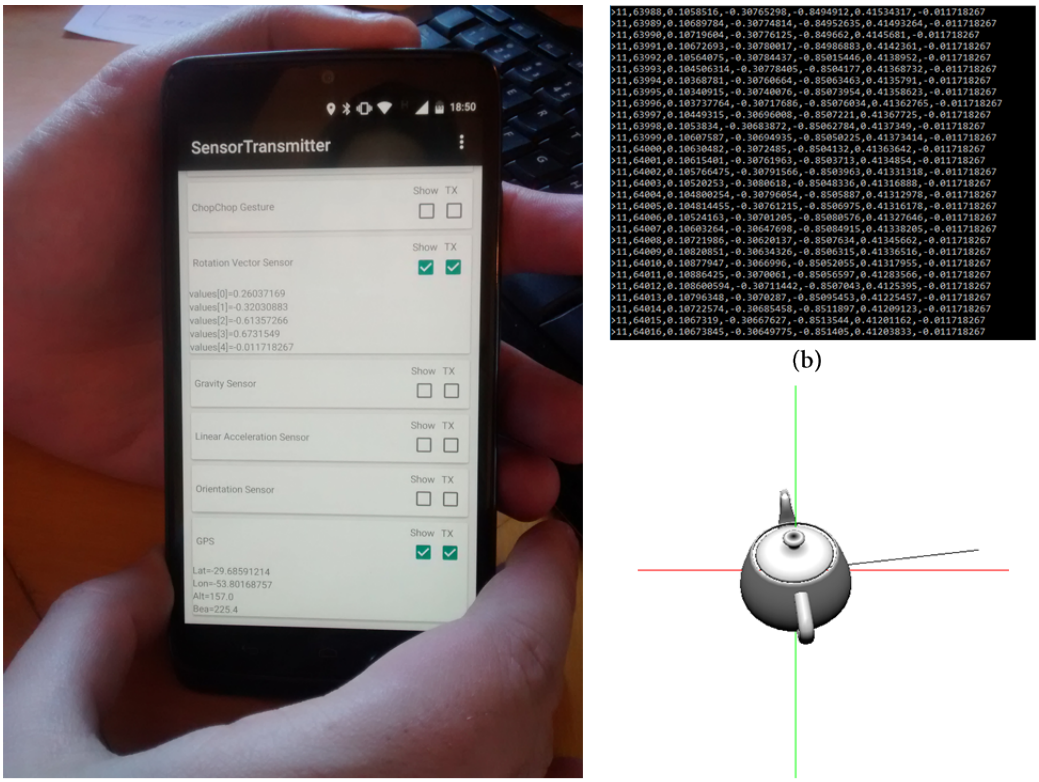


Figura 4.4 – Aplicativo Android sendo executado com o dispositivo inclinado (a), a saída dos dados coletados (b), e o dispositivo representado pelo *Teapot*, visto do eixo Z (c).

4.4.3 *Dataset* de Calibração

Além da geração de um *dataset*, é necessária a geração de um *dataset* de calibração composto de imagens contendo um *chessboard* de calibração 9x6, como apresentado na Figura 4.5 de sistema estereoscópico do OpenCV, assim como dados sensoriais do sistema enquanto ele permanece estacionário, para que seja computado qualquer *bias* presente.

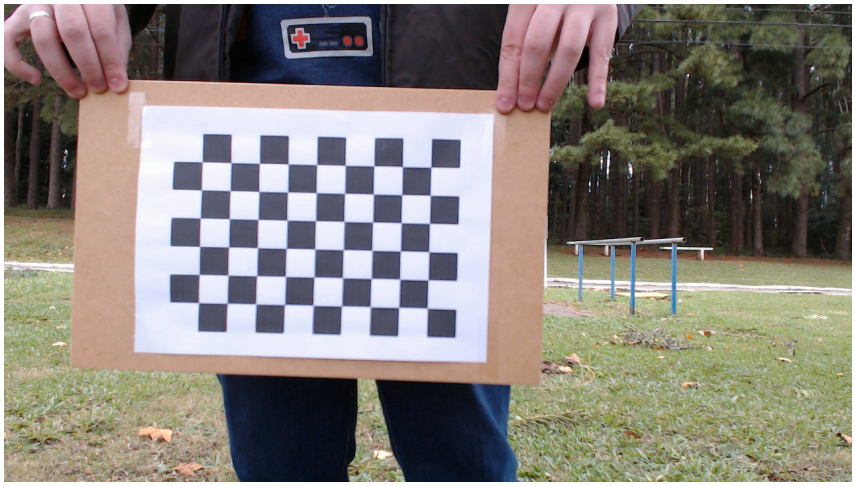


Figura 4.5 – Exemplo de imagem de calibração.

4.5 MÓDULO DE PROCESSAMENTO DE IMAGENS DE *DATASET*

Após a geração do *dataset*, ele é processado pelo módulo de processamento de imagens, responsável por gerar arquivos auxiliares, como informações de odometria visual e mapa de disparidades.

4.5.1 Calibração do sistema

Para que seja possível a análise correta das imagens e sensores, o sistema deve ser calibrado utilizando o *dataset* de calibração gerado na Seção 4.4.3, e de acordo com o Capítulo 2, Seção 2.2. Desta maneira, utilizando a biblioteca OpenCV (BRADSKI, 2000), são identificadas imagens contendo o *chessboard*, que são utilizadas em seu método de calibração. Também de acordo com o Capítulo 2, Seção 2.2, como resultado, os parâmetros intrínsecos e extrínsecos do sistema são obtidos.

Devido ao *dataset* de calibração ser gerado em formato de vídeo, apenas são utilizadas para calibração as imagens onde a movimentação do *chessboard* é inferior a 1 *pixel*, garantindo que, no momento de captura, não havia movimento que pudesse gerar *frames* dessincronizados.

Para a calibração do acelerômetro, é feita uma média dos dados coletados no período de calibração, estimando, assim, o seu *bias*. No processamento das informações no restante do trabalho, a aceleração é corrigida a partir da subtração do *bias* computado.

4.5.2 Estereoscopia em (*Visual*) SLAM

Por meio dos parâmetros intrínsecos e extrínsecos obtidos na Seção 4.5.1 os pontos da imagem podem ser reprojetados em 3D e utilizados para triangular a posição do objeto. Pode-se definir o pseudo-algoritmo de VSLAM utilizando visão estéreo, demonstrado pelo Algoritmo 4.2.

Algoritmo 4.2 Demonstração de algoritmo de VSLAM

```
1  leftImage = leftCam.capture(); //captura frame da camera esquerda
2  rightImage = rightCam.capture(); //captura frame da camera direita
3  leftCam.undistort_rectify(leftImage,rightCam); //remove as distorcoes radiais e tangenciais e
       retifica as imagens de acordo com o par estereo
4  rightCam.undistort_rectify(rightImage,leftCam); //o mesmo que o anterior, mas para a outra
       camera
5  landmarks_2D_extraidos = extrair_landmarks(leftImage,rightImage); //extrai os landmarks das
       duas imagens, colocando seus pontos 2D em leftPoints e rightPoints do objeto Landmark
       correspondente
6  for(landmark2D : landmarks_2D_extraidos) {
7    Ponto3D coordenada3D = triangulatePoints(average(landmark2D.leftPoints),
         average(landmark2D.rightPoints)); //triangula a coordenada 3D a partir do ponto medio
         do landmark
8    float distancia = coordenada3D.modulo(); //considerando que o sistema de cameras esta na
         origem do sistema, a distancia e o modulo do vetor dado pela coordenada3D.
         Consequentemente, a coordenada x e a distancia horizontal, a coordenada y e a distancia
         vertical e a coordenada z e a distancia de profundidade.
9    landmarks_globais.update(landmark2D.id,coordenada3D); //atualiza a informacao global do
         landmark e atualiza a pose do robo
10 }
```

A Figura 4.6 mostra as imagens de entrada (capturadas nas linhas 1 e 2 do Algoritmo 4.2) já sem distorção e retificadas (linhas 3 e 4). A partir destas imagens, a linha 5 extrai ambos os *chessboards* na forma de pontos 2D em ambas as imagens, e agruparia-os em um objeto *landmark* que contem as coordenadas 2D de ambas as câmeras do *landmark* em questão. Em seguida, os *landmarks* (*chessboards* neste caso) teriam suas coordenadas 3D extraídas pela linha 7. E, por fim, a linha 9 atualiza o mapa do robô e sua *pose*. A Figura 4.7 mostra como os *chessboards* são representados em um ambiente 3D após a extração de suas coordenadas pela linha 7.

Figura 4.6 – Imagens de estrada devidamente sem distorção e retificadas.

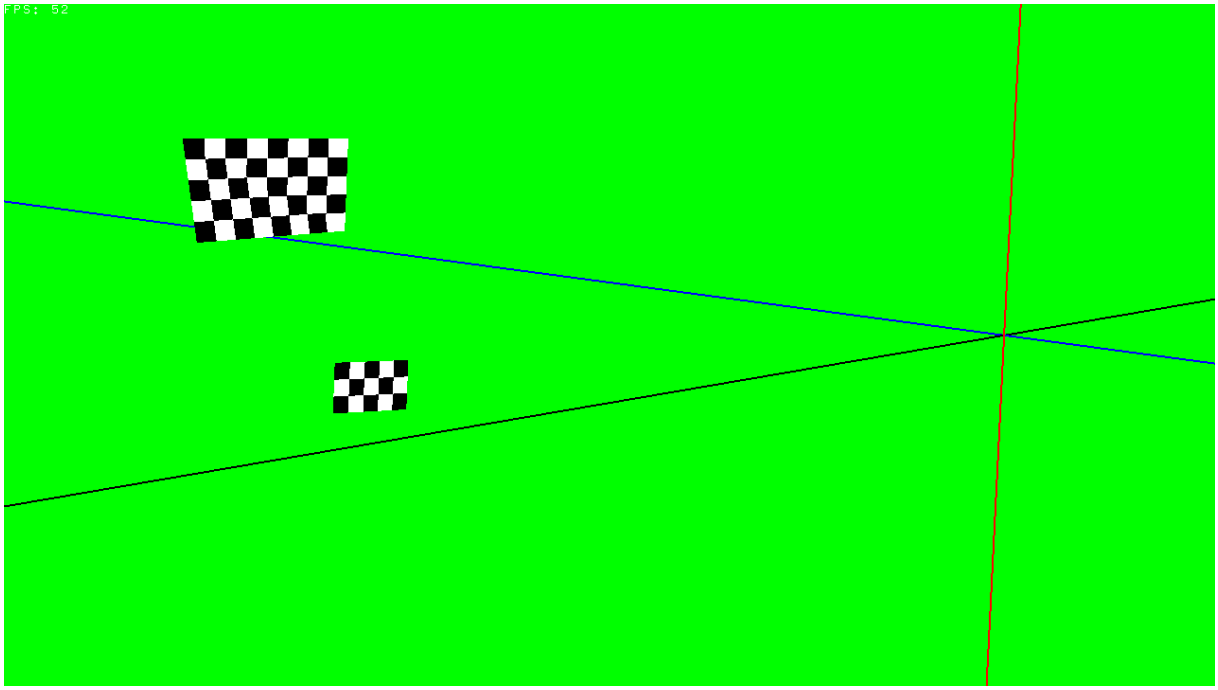


Figura 4.7 – *Chessboards* das imagens de entrada (Figura 4.6) representadas em 3D a partir de estereoscopia.

A cada movimentação do robô, o processo é repetido e a nova *pose* do robô pode ser estimada a partir de novas capturas das câmeras e reavaliação dos *landmarks*. Nota-se que novos *landmarks* são necessários caso os atuais não sejam mais visíveis, e quando a *pose* é atualizada, os *landmarks* não observados também devem ter suas localizações reavaliadas para que, quando reobservados, o erro do sistema possa ser calculado. Estes passos de estimativa do movimento (*motion*) do sistema são realizados pela biblioteca LIBVISO2, descrita no Capítulo 2, Seção 2.10.2. Desta maneira, as imagens sem distorção e retificadas são enviadas à biblioteca, que é responsável por realizar o trecho ilustrado nas linhas de 5 a 10 do Algoritmo 4.2. Estes dados são então salvos em um arquivo de *motion* a cada *frame* processado, para que possam ser interpretados separadamente.

4.5.3 Geração do mapa de disparidades

Os mapas de disparidades são gerados a partir dos algoritmos de *Block Matching*, por Konolige (KONOLIGE, 1998), e *Semiglobal Block Matching*, por Hirschmuller (HIRSCHMULLER, 2008), que fazem parte da biblioteca OpenCV, descrita no Capítulo 2, Seção 2.10.1. Os resultados destes algoritmos são salvos em arquivos de vídeo independente, utilizando compressão sem perdas, para que não haja necessidade de processamento extra a cada geração de resultado. O algoritmo StereoSGBM demanda mais processamento, e gera mapas de disparidades mais densos devido à sua abordagem semiglobal (HIRSCH-

MULLER, 2008), enquanto o StereoBM obtém mapas menos densos, como demonstrado por Geiger et al. (GEIGER; LENZ; URTASUN, 2012). A Figura 4.8 apresenta os mapas de disparidades gerados por ambas as técnicas no mesmo *frame*.

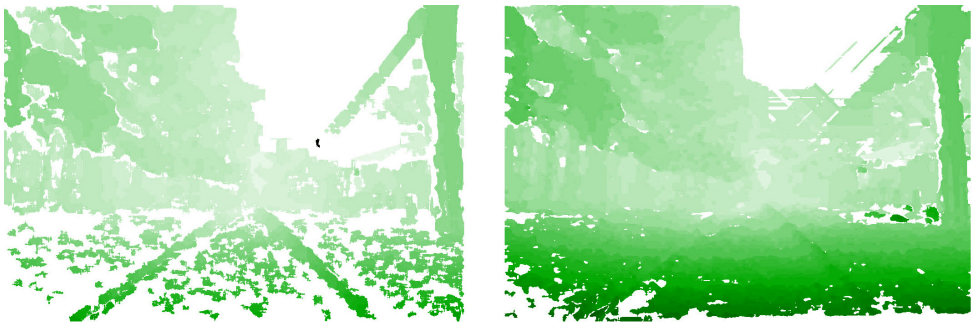


Figura 4.8 – Comparação entre o mapa de disparidades gerado por *block matching* (KONOLIGE, 1998) (a), e gerado por *semiglobal block matching* (HIRSCHMULLER, 2008) (b).

4.6 MÓDULO DE GERAÇÃO DE RESULTADOS

O módulo de geração de resultados é responsável por processar o *dataset* completo e unificar as informações sensoriais e de odometria visual em uma variedade de testes.

4.6.1 Avaliação de sincronização e mapa de disparidades

Como apresentado na Seção 2.2 do Capitulo 2, um mapa de disparidades é composto por *pixels* correspondidos em ambas as imagens, com sua intensidade definida como a distância em relação ao sistema, e áreas onde uma correspondência não foi encontrada têm seus *pixels* sem valor, chamados de *Bad Matched Pixels*. A quantidade de *Bad Matched Pixels* é utilizada em comparações entre algoritmos de geração de mapas de disparidade, como por Lazaros et al. (LAZAROS; SIRAKOULIS; GASTERATOS, 2008), e pode ser utilizada para identificar *frames* com boa correspondência (ZHANG et al., 2003).

Como descrito na Seção 4.5.3, o método StereoSGBM gera mapas mais densos devido à sua abordagem semiglobal, o que acaba por erroneamente encontrar correspondência em imagens não retificadas. A Tabela 4.1 apresenta a densidade do mapa de disparidades (porcentagem de *Matched Pixels*) em ambos os métodos em um *frame* sincronizado, que contém uma boa retificação, e outro dessincronizado, resultando numa má retificação, assim como a variação entre os *frames*. A maior variação apresentada no método de StereoBM reforça que o algoritmo é mais sensível à *frames* não retificados, tornando uma análise sobre o seu mapa gerado ideal para identificação destes.

Tabela 4.1 – Comparativo de *matched pixels* entre StereoBM e StereoSGBM para um *frame* dessincronizado e outro sincronizado em sucessão.

Frame	SBM	SGBM
Dessincronizado	14,97%	40,4%
Sincronizado	60,54%	82,41%
Variação	404%	203%

Desta maneira, uma análise sobre os *Matched Pixels* de cada *frame* pode determinar se o *frame* está ou não sincronizado, visto que *frames* dessincronizados perdem sua retificação. Utilizando esta informação, pode-se verificar se as informações relativas a este *frame* devem ser utilizadas ou descartadas. Partindo deste pressuposto, Zhang et al. (ZHANG et al., 2003) recria o mapa de disparidades ao transladar uma das imagens, buscando obter um mapa de disparidades mais denso, o que implica uma melhor retificação. Devido às câmeras já estarem retificadas pelo passo de calibração, foi desenvolvido um filtro adaptativo linear simples que limita a variação máxima entre o último *frame* sincronizado e o *frame* avaliado, descartando *frames* com mapas de disparidades menos densos, e é dado pela asserção:

$$mp_i + mp_i(vb + (i-k)vpf) > mp_k \tag{4.1}$$

onde:

i = *Frame* atual
k = Último *frame* utilizado
mp_i,mp_k = Densidade no *frame* i e k
vb = Variação base
vpf = Variação a ser incrementada a cada *frame*

O filtro utilizado parte do pressuposto que, nos cenários propostos, com imagens capturadas a 30 *frames* por segundo, a variação de conteúdo a cada $33ms$ seja baixa. Sendo assim, definiu-se uma variação base de 7% com $1{,}5\%$ extra a cada *frame*. Este método acaba por rejeitar *frames* que tenham mapas de disparidades muito incompletos, em relação com *frames* anteriores.

Nota-se que não há um *ground truth* associado aos *frames*, o que impossibilita a avaliação da confiança do mapa de disparidades, como proposto por Hu e Mordohai (HU; MORDOHAI, 2010), e limita a quantidade de informações disponíveis para a identificação dos *frames* dessincronizados.

4.6.2 Fusão de informações

A partir dos dados sensoriais e de odometria visual coletados, é possível realizar uma fusão de informações. Desta maneira, definiram-se os seguintes ambientes de testes:

- Apenas informações de odometria visual;
- Apenas informações de GPS;
- Odometria visual interpolada com informações de rotação;
- Fusão completa: Informações de localização GPS, com movimento fino corrigido utilizando odometria visual e dados sensoriais do dispositivo Android.

No cenário de odometria visual, o processamento de odometria visual é puramente executado pela biblioteca LIBVISO2 (GEIGER; ZIEGLER; STILLER, 2011), tanto em *frames* considerados sincronizados quanto dessincronizados, verificando, assim, a eficácia do filtro. Em ambos os casos, o resultado é uma *pose* para cada *frame* processado.

Devido à sua precisão, como descrito no Capítulo 2, Seção 2.9, e no Capítulo 3, as informações de GPS são utilizadas como base para comparação entre os cenários, inclusive para determinar o ponto de *loop* nos *datasets* gerados.

Visando aprimorar a odometria visual, suas informações são unidas com as de rotação, efetivamente aplicando o deslocamento computado pela odometria na direção que o sistema está apontando, e é realizado ao substituir a parte referente à matriz de rotação na matriz de *pose* pelos dados do vetor de rotação, que finalmente é multiplicada pelo inverso da movimentação detectada pela odometria. Estas operações podem ser visualizadas na Equação 4.2, sendo $Pose$ a *pose* do sistema, R o vetor de rotação em formato matricial, x, y, e z as coordenadas do sistema, e $Motion$ a matriz de movimento computada pela odometria visual. Destaca-se que a rotação da *pose* anterior é sempre substituída para o cálculo da *pose* futura.

$$Pose_i = \begin{pmatrix} & & & x_{i-1} \\ & R_{i-1} & & y_{i-1} \\ & & & z_{i-1} \\ 0 & 0 & 0 & 1 \end{pmatrix} * Motion_{i-1}^{-1} \tag{4.2}$$

onde:

$Pose$ = Matriz que representa a *pose*
R = Matriz de rotação 3x3
x,y,z = Coordenadas 3D do sistema
$Motion$ = Matriz que representa a movimentação do sistema

No último cenário, todas as informações são combinadas para gerar um traçado mais detalhado do caminho percorrido. Este cenário difere do anterior por atualizar os

valores de x, y, e z, representados na Equação 4.2, ao receber uma nova coordenada de GPS.

4.6.3 Reconstrução 3D

Para a realização da reconstrução 3D foi obtida a *pose* do sistema em cada ambiente de teste, e as imagens das câmeras são extraídas de acordo com a Seção 4.5.2. A partir destes dados, uma nuvem de pontos é reconstruída utilizando a biblioteca OpenCV, e a combinação desta nuvem com a *pose* resultam no ambiente reconstruído em 3D.

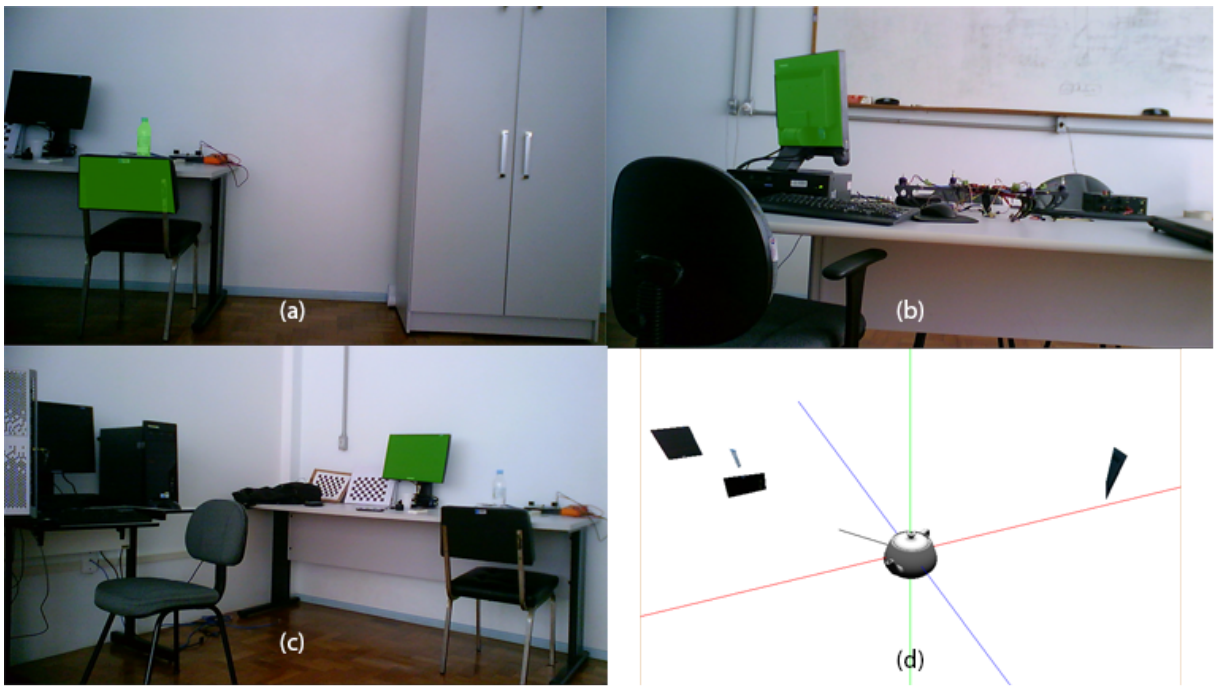


Figura 4.9 – (a), (b) e (c) Imagens de entrada com os objetos detectados marcados em verde, (d) resultado do mapeamento.

A Figura 4.9 demonstra alguns objetos mapeados utilizando o método descrito. Para fins de demonstração, os objetos são identificados manualmente e suas coordenadas 2D são fornecidas como entrada para o algoritmo. Assim que os objetos são posicionados, uma textura é extraída da área 2D do objeto na imagem e utilizada para sua visualização.

4.7 DESAFIOS ENFRENTADOS

Segundo a documentação oficial do OpenCV, o problema de sincronização entre as câmeras pode ser suavizado utilizando o método $grab$ descrito no Algoritmo 4.3, facilitando o processo de captura de imagens descrito na Seção 4.4.1.

Algoritmo 4.3 Captura de imagens de acordo com a documentação.

```
1 leftCam.grab(); //requisita que a camera esquerda capture o proximo frame
2 rightCam.grab(); //requisita que a camera direita capture o proximo frame
3 currentTimestamp = now(); //armazena a hora de geracao do frame, em milissegundos
4 leftCam.retrieve(leftImage); //decodifica o frame capturado e armazena em leftImage
5 rightCam.retrieve(rightImage); //decodifica o frame capturado e armazena em rightImage
```

O Algoritmo 4.3 baseia-se em que a captura de imagens é rápida (linhas 1 e 2), e sua decodificação e transferência é demorada (linhas 4 e 5). Este método, porém, não obteve o resultado esperado no sistema estereoscópico proposto. A função $grab$ não desempenha seu papel, o que resulta em a função $retrieve$ decodificar o último *frame* capturado e requisitar um novo *frame*. Adicionalmente, mesmo que o funcionamento estivesse correto, não haveria garantias que os *frames* estivessem sincronizados, devido à execução sequencial do código.

Este problema adicionou complexidade método de sincronização utilizado, sendo necessário contornar a situação ao armazenar o *timestamp* após a captura, que é relativo ao próximo *frame*, e não ao recém capturado. Também há uma menor confiabilidade na sincronia e *timestamp*, visto que o método $retrieve$ tem um maior tempo de processamento, e não é possível precisar o tempo exato que a captura do próximo *frame* tem início.

Independente da sincronização via *software* realizada, os mapas de disparidades resultantes e os dados de odometria visual indicavam que muitos *frames* não apresentavam a sincronia necessária para uma aplicação em movimento. Devido à este problema, diversas iterações foram realizadas sobre o algoritmo de captura, porém nenhuma foi capaz de eliminar os *frames* dessincronizados. Desta maneira, a solução desenvolvida é apresentada na Seção 4.6.1, onde foi utilizado um filtro baseado na densidade do mapa de disparidades para descartar *frames* com má sincronia, o que aumenta o tempo de processamento ao gerar o mapa de disparidades para cada *frame* processado. Contudo, esta solução foi eficaz em reduzir a quantidade de *frames* dessincronizados.

Primeiramente a comunicação entre *smartphone* Android e o programa, descrita na Seção 4.4.2, foi realizada utilizando comunicação serial via *Bluetooth*. Posteriormente, os testes realizados mostraram que a comunicação *Bluetooth* apresentava atrasos crescentes quando combinado com a alta taxa de atualização dos sensores. A combinação da transmissão do acelerômetro e do vetor de rotação era suficiente para que os atrasos fossem observados, impossibilitando o uso deste tipo de comunicação na aplicação. Devido à este problema, foi utilizada uma conexão USB para implementar uma comunicação via protocolo TCP/IP, eliminando problema.

Os dados de GPS também mostraram-se imprecisos durante a geração de *datasets* em baixa velocidade, o que não era o resultado esperado de acordo com os trabalhos relacionados, Capítulo 3. *Datasets* gerados em lugares abertos também apresentavam problemas, o que indicava a causa raiz não era relacionada ao sinal de GPS recebido.

Devido ao modelo de *smartphone* utilizado ter sido lançado em 2012, havia a possibilidade de que seu sensor GPS não fosse capaz de obter uma maior precisão, o que confirmava-se com os dados obtidos da plataforma Android, indicando um erro de $6m$ a $12m$.

Visando obter dados de GPS mais precisos, foi utilizado um segundo *smartphone* Android, o Motorola Moto Maxx, de 2014, executando o mesmo aplicativo, porém utilizando comunicação *Bluetooth* para que pudesse ficar em local mais elevado em relação ao sistema, e visto que o GPS realiza uma leitura a cada segundo, não ocorrendo os atrasos constatados ao utilizar os outros sensores. Contudo, os resultados foram similares aos obtidos com apenas um *smartphone*, mesmo que o erro apontado pela plataforma fosse menor, de $3m$ a $6m$. Desta maneira, não foi possível integrar os dados de GPS na fusão de informações, Seção 4.6.2.

Originalmente, dados de aceleração também seriam utilizados na tentativa de obter a posição do sistema partindo apenas de informações sensoriais de rotação e aceleração, no passo de fusão de informações, Seção 4.6.2. Porém não foi possível obter dados de velocidade e deslocamento precisos nos testes preliminares. Como destacado por Woodman et al. (WOODMAN; HARLE, 2008), o *drift* da posição gerada pela integral dupla sobre aceleração cresce rapidamente com o tempo, tornando o acelerômetro inutilizável para estas aplicações. Adicionalmente, o erro é agravado ao converter o vetor de aceleração linear em coordenadas globais utilizando o vetor de rotação.

4.8 DIAGRAMAS DO SISTEMA

O diagrama de classes do aplicativo Android está disposto na Figura 4.10. A classe *MainActivity* instancia os sensores e GPS (classes *SensorView* e *LocationView*), que realizam a comunicação tanto por *Socket* TCP/IP quanto por *Bluetooth* (classes *TCPServerComm* e *BluetoothSerialCom*).

Quanto a aplicação de geração e processamento de *dataset*, seu diagrama de classes é dado pela Figura 4.11, utilizando o padrão de projeto fábrica para geração dos objetos câmera projetado para comportar múltiplos sistemas estereoscópicos, Figura 4.12, responsável pela obtenção de dados sensoriais via *Bluetooth* e TCP/IP, e Figura 4.13 responsável pela integração entre os módulos e controle sobre processamento e geração de *dataset*, assim como a classe *EduardoUtils*, que contém funções auxiliares de transformações. Destaca-se o uso da biblioteca OpenCV para as matrizes na estrutura $cv{::}Mat$, da biblioteca OpenGL com suas *callbacks* na classe *AppController* nas funções com nomes iniciados por $openGL$, e da biblioteca GLM nas estruturas do *namespace* glm, responsáveis por armazenar vetores e *quaternions*, assim como auxiliar em sua manipulação.

O diagrama de atividades da geração de *dataset* é representado pela Figura 4.14, onde há uma *thread* para obtenção de *frames* de cada câmera, armazenando-os a um *buf-*

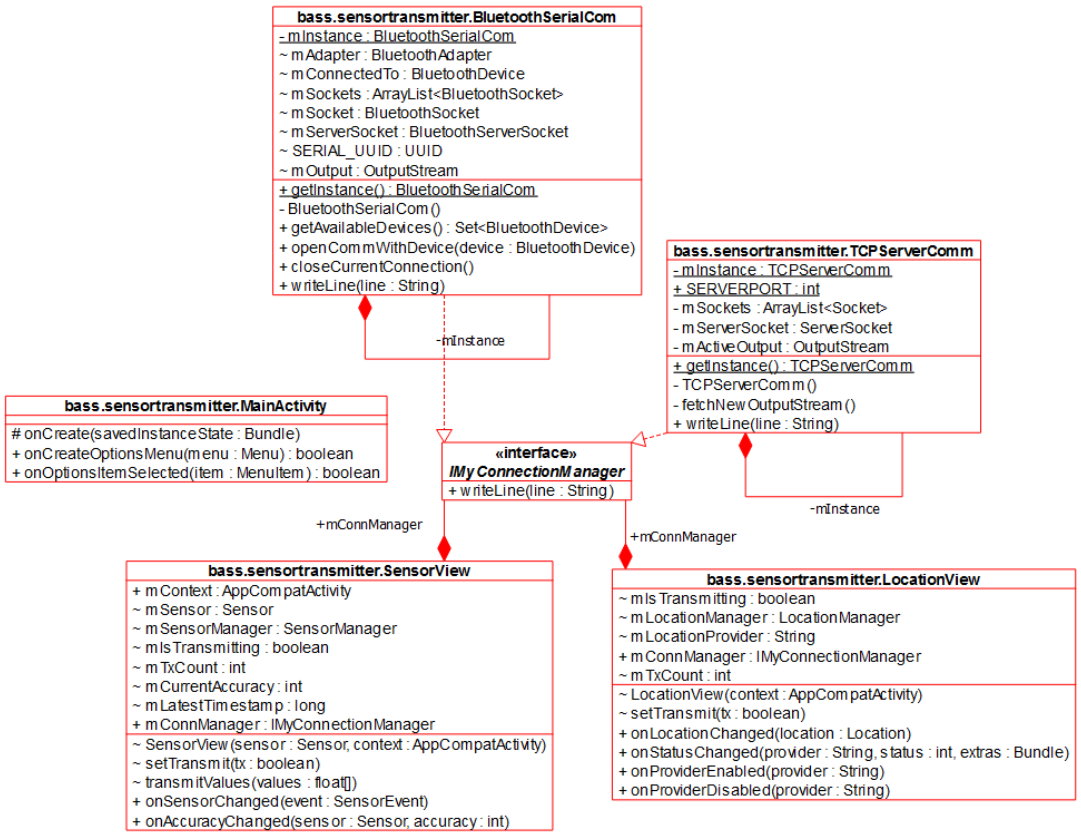


Figura 4.10 – Diagrama de classes da aplicação Android.

fer de escrita com seu *timestamp*. *Threads* secundárias escrevem o *frames* armazenados no *buffer* para o disco. A *thread* responsável por receber os dados sensoriais e armazená-los no disco é representada de forma única à direita da Figura, porém são criadas de acordo com o número de dispositivos utilizados. Destaca-se as bibliotecas em parênteses a cada passo, que, no caso da geração de *dataset* foi apenas a biblioteca OpenCV que realiza a interface de comunicação com as câmeras, recebendo o *frame* gerado no passo "Captura imagem esquerda/direita".

A Figura 4.15 apresenta o diagrama de atividades de processamento dos dados (a) e visualização dos mesmos (b). Em (a), os vídeos gerados no *datasets* são abertos e os *frames* de ambas as câmeras são obtidos em *loop*. Para cada par de *frames*, seu mapa de disparidades é gerado pelo algoritmo StereoBM (KONOLIGE, 1998), implementado na biblioteca OpenCV. A partir do mapa de disparidades, o filtro verifica a retificação das imagens na condição "*Frame* sincronizado?", prosseguindo ao par seguinte caso rejeitados. Se os *frames* são considerados sincronizados, sua odometria visual é computada pela biblioteca LIBVISO2, gerando uma nova *pose*. Os dados sensoriais, armazenados em arquivo, são então lidos e processados até o *timestamp* de obtenção das imagens e a *pose* é atualizada de acordo. Estes passos são então armazenados em disco para que possam ser analisados e exibidos posteriormente.

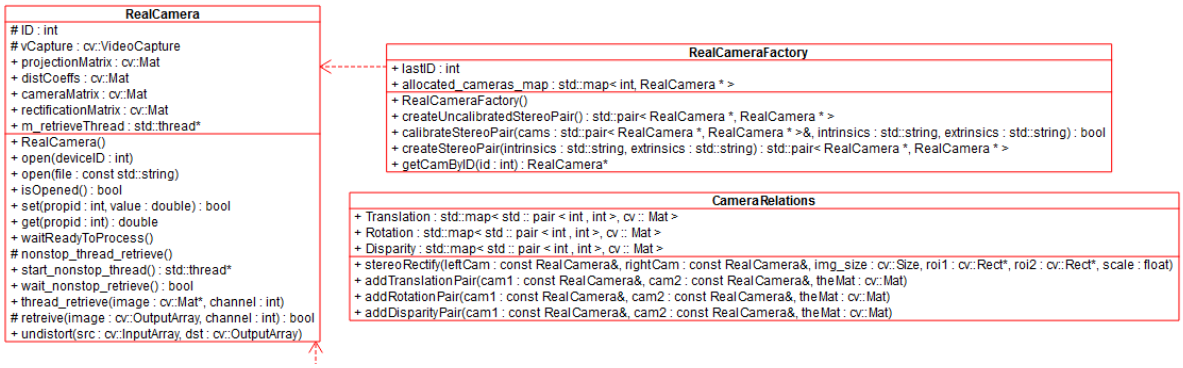


Figura 4.11 – Diagrama de classes referente à criação e utilização de câmeras.

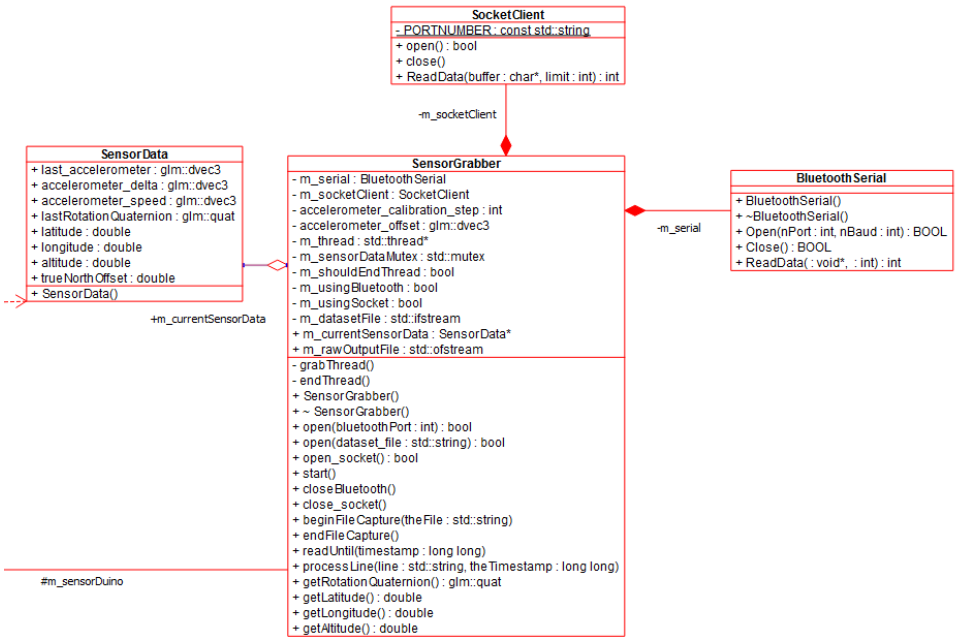


Figura 4.12 – Diagrama de classes referente à obtenção de sensores.

No passo de visualização de resultados (b), o mapa de disparidades processado em (a) é reprojetado em 3D pela biblioteca OpenCV, gerando uma nuvem de pontos, assim como os dados de *pose* são plotados em um gráfico. A biblioteca OpenGL é responsável por esta plotagem de gráfico e renderização da nuvem de pontos.

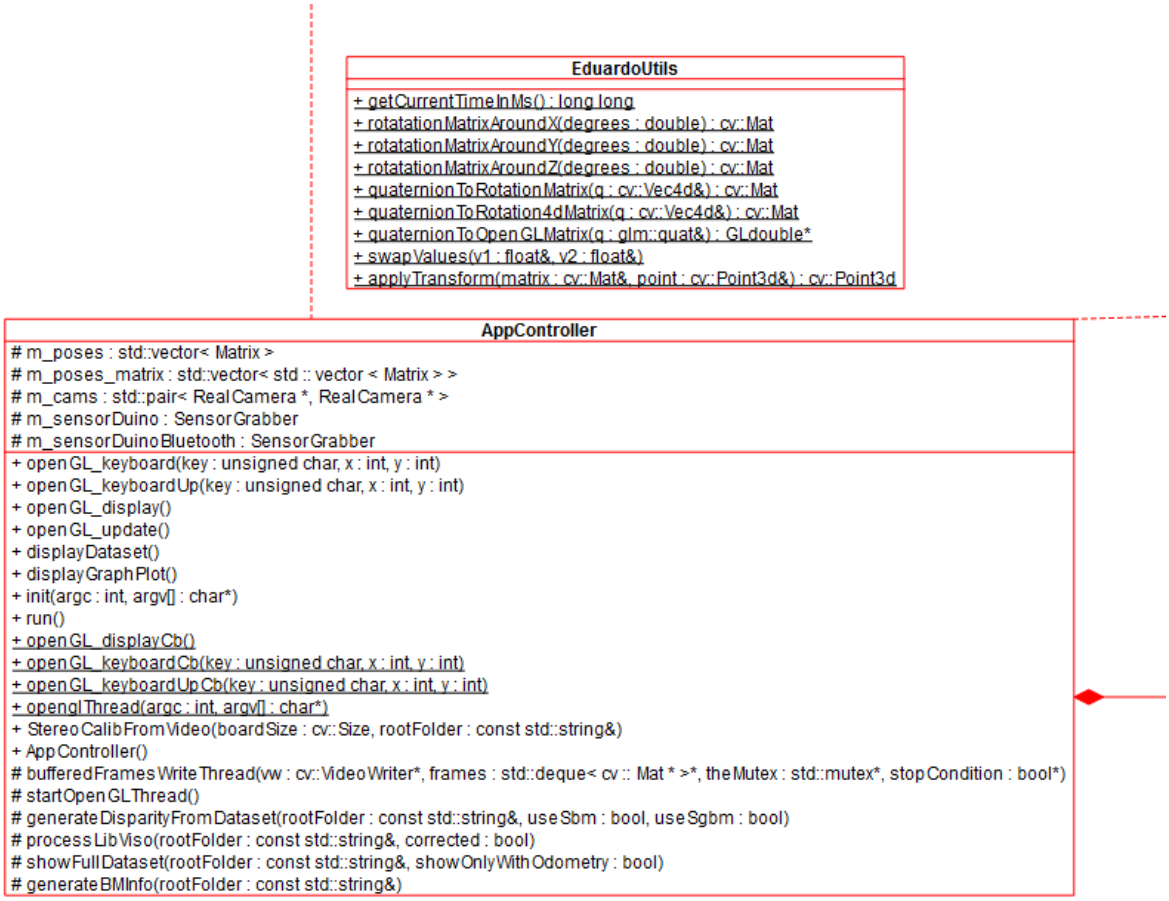


Figura 4.13 – Diagrama de classes principal.

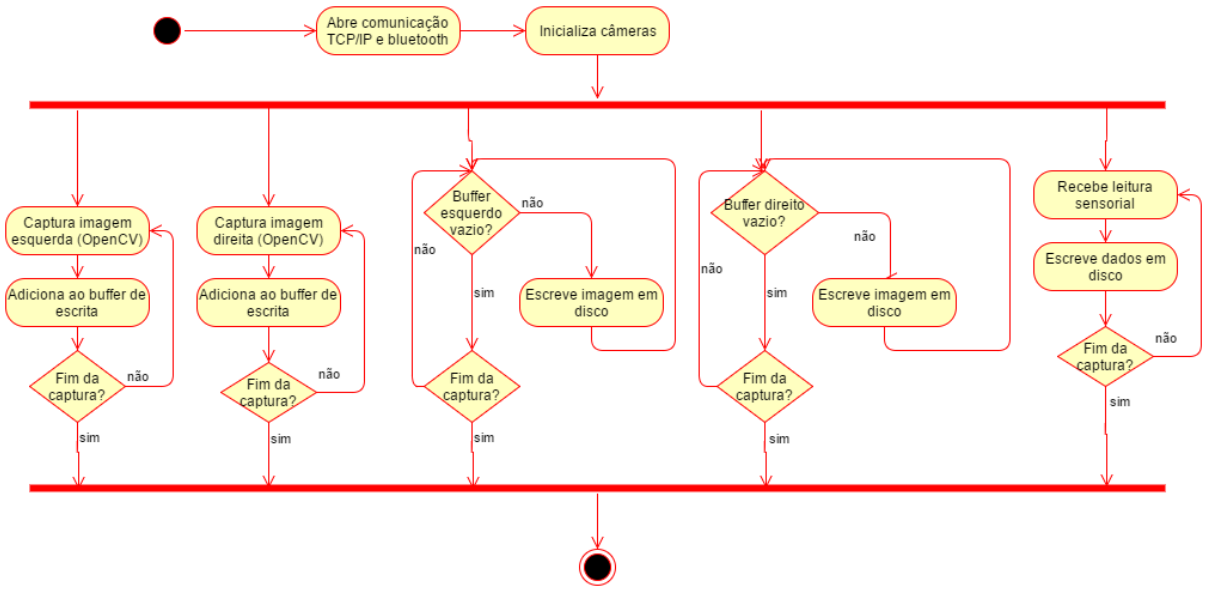


Figura 4.14 – Diagrama de atividades de geração de *dataset*.

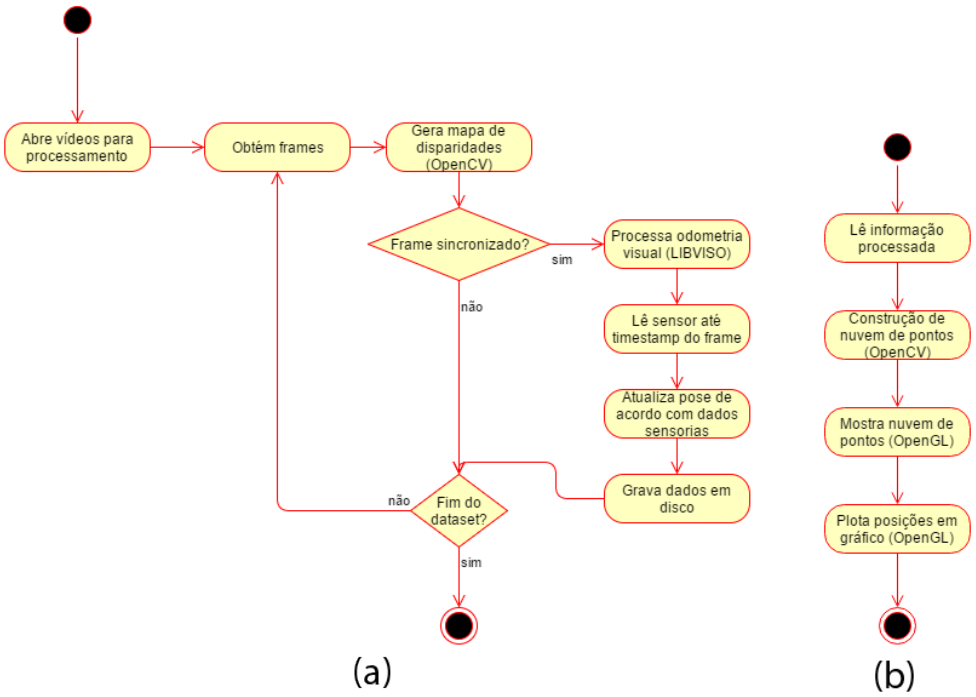


Figura 4.15 – Diagrama de atividades de processamento de *dataset* (a) e visualização de resultados (b).

5 RESULTADOS

O ambiente proposto no Capítulo 4 foi utilizado para gerar *datasets* com um vídeo para cada câmera do sistema estereoscópico, assim como dados sensoriais de rotação, aceleração e GPS. A partir das imagens capturadas, foram filtrados os *frames* considerados sincronizados para que fosse realizada a odometria visual. Finalmente, uniram-se os dados sensoriais e deslocamento odométrico para obter um mapeamento e localização mais precisos.

5.1 MÉTRICAS UTILIZADAS

A densidade do mapa de disparidades é utilizada como métrica para verificar sua qualidade, e representa a quantidade de *pixels* com correspondência nas imagens, dada a partir do inverso dos *Bad Matched Pixels*, e utilizada para filtrar *frames* sincronizados e dessincronizados, como definido no Capítulo 4, Seção 4.6.1. Desta maneira, esta métrica também é utilizada para traçar um comparativo entre os mapas obtidos com e sem filtragem de *frames*, avaliando sua eficácia.

Devido ao sistema não contar com *ground truth*, como o trabalho de Kitt et al. (KITT; GEIGER; LATEGAHN, 2010), os *datasets* foram gerados formando um *loop*, sendo possível verificar o erro de fechamento do *loop*, ou *Loop Closure Error* (LCE), como utilizado por Agrawal e Konolige (AGRAWAL; KONOLIGE, 2006), e é dado em m e porcentagem da distância percorrida. Desta maneira, é possível fazer um comparativo direto entre o erro obtido pelo sistema e os obtidos nos trabalhos relacionados.

5.2 *DATASETS*

Utilizando o módulo de geração de *dataset*, foram gerados 3 *datasets* distintos, identificados como:

1. *Dataset* COPERVES: gerado em estacionamento e calçada;
2. *Dataset* Planetário: gerado em lugar aberto;
3. *Dataset* Carro: gerado em carro em movimento.

Visando comparar os resultados com *datasets* disponíveis publicamente, foi utilizado o *dataset* 2009_09_08_drive_0021 por Greiger et al. utilizado como exemplo na biblioteca LIBVISO2, também de sua autoria (GEIGER; ZIEGLER; STILLER, 2011; GEIGER;

ROSER; URTASUN, 2010). Este *dataset* foi gerado a partir de um sistema estereoscópico que utiliza câmeras sincronizadas via *hardware* e uma *baseline* de em torno de $60cm$, e seus dados técnicos são exibidos no Quadro 5.1.

Resolução	1344x391
Frames	2579
Taxa de atualização	15FPS
Duração	2m 51s

Quadro 5.1 – Dados do *dataset* 2009_09_08_drive_0021.

Destaca-se o uso de câmeras com um alto campo de visão horizontal, ressaltado pela resolução com alta proporção $comprimento/altura$.

5.2.1 *Dataset* COPERVES

A Figura 5.1 representa o traçado do *dataset* COPERVES, gerado no estacionamento e calçada do prédio da COPERVES da Universidade Federal de Santa Maria. Arbustos, carros e construções encontram-se nas laterais do trajeto. O Quadro 5.2 apresenta as especificações da geração do *dataset*.

Figura 5.1 – Traçado do trajeto do *dataset* COPERVES. A linha em verde representa sobreposição de traçado.

Resolução	1280x720 (HD)
Frames	4978
Taxa de atualização	30FPS
Duração	2m 45s

Quadro 5.2 – Dados do *dataset* COPERVES.

5.2.2 *Dataset* Planetário

O *dataset* Planetário foi gerado em campo aberto na Universidade Federal de Santa Maria, próximo ao Planetário, e seu traçado é apresentado na Figura 5.2. Este *dataset* tem por objetivo verificar o desempenho do algoritmo com poucos pontos de referência laterais e em ambientes bastante iluminados. O Quadro 5.3 apresenta as especificações da geração do *dataset*.

Figura 5.2 – Traçado do trajeto do *dataset* Planetário.

Resolução	1280x720 (HD)
Frames	6112
Taxa de atualização	30FPS
Duração	3m 23s

Quadro 5.3 – Dados do *dataset* Planetário.

5.2.3 *Dataset* Carro

Este *dataset* foi gerado próximo ao prédio 74 da UFSM. Este *dataset* contém uma volta em torno de um pequeno trecho, em vermelho, e uma última volta pelo trecho em azul, demonstrados na Figura 5.3. Tendo como objetivo traçar um comparativo com o *dataset* público utilizado, este trajeto contém pontos de referência mais distantes que os anteriores. Nota-se que durante o trajeto em azul, os pontos de referência laterais tornam-se escaços. Também diferentemente dos anteriores, a base deste é fixa acima do carro.

Figura 5.3 – Traçado do trajeto do *dataset* Carro.

Quadro 5.4 – Dados do *dataset* Carro.

Resolução	1280x720 (HD)
Frames	6157
Taxa de atualização	30FPS
Duração	3m 25s

5.3 TAXA DE APROVEITAMENTO DE *FRAMES* E AVALIAÇÃO DE MAPA DE DISPARIDADES

Utilizando a densidade do mapa de disparidades, é possível verificar a porcentagem de *pixels* com correspondência (*Matched Pixels*) nas imagens. Desta maneira, a Tabela 5.1 mostra os resultados obtidos com a filtragem dos *frames* dessincronizados dos *datasets*.

Tabela 5.1 – Tabela de análise de aproveitamento de *frames*.

Dataset	Densidade	Densidade (Filtrado)	Frames Desconsiderados
COPERVES	49,09%	65,22%	65,50%
Planetário	28,41%	38,34%	67,60%
Carro	12,16%	13,55%	27,26%
2009_09_08_drive_0021	34,73%	34,73%	0%

Nota-se que todos os *datasets* foram beneficiados pelo filtro utilizado devido ao aumento na quantidade de *pixels* com correspondência (MP). Os dados também mostram que, embora a sincronização via *software* utilizada aproxime a captura dos *frames*, este

filtro mostra-se necessário para reduzir quaisquer erros na análise dos dados resultantes. Também se percebe que há um menor erro de sincronização quando as câmeras são montadas em um ponto fixo, como no carro. Portanto, operações realizadas no *dataset* sem filtro tem uma maior taxa de erro, devido á maior quantidade de *frames* dessincronizados, chegando a até $65{,}98\%$ no *Dataset* Planetário, o que é comprovado na Seção 5.4. Em contraste, não houve nenhum *frame* filtrado pelo algoritmo no *dataset* 2009_09_08_drive_0021, pois as câmeras são sincronizadas via *hardware*.

As Figuras 5.4 e 5.5 demonstram, respectivamente, *frame* aceito e um rejeitado pelo algoritmo com base na variação na densidade do mapa de disparidades. A imagem (a) das figuras apresenta os *frames* obtidos pelas câmeras, já retificados, enquanto a imagem (b) apresenta o mapa de disparidades obtido, e a imagem (c) apresenta um destaque da imagem (a) ampliado. Nota-se que na imagem (c) da Figura 5.5 há uma grande disparidade vertical onde o gramado encontra-se com o poste de energia, e, em contraste, a imagem (c) da Figura 5.4 mostra um alinhamento correto, visualizada na base da árvore à direita do poste. Como resultado, o algoritmo de *Block Matching*, por Konolige (KONOLIGE, 1998), não encontrou correspondências para $39{,}46\%$ da imagem (*Bad Matched Pixels*) no *frame* aceito, enquanto no *frame* rejeitado, $85{,}03\%$ de sua região não foi correspondida em ambas as imagens. Esta diferença é demonstrada na imagem (b) de ambas as Figuras, onde é possível verificar um mapa de disparidades denso na Figura 5.4, o que não ocorre na Figura 5.5.

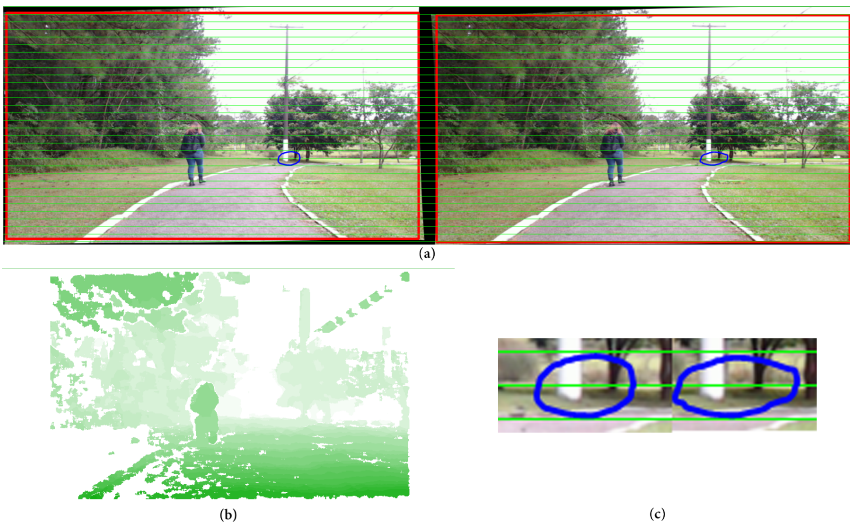


Figura 5.4 – *Frame* considerado. (a) imagem retificada, (b) mapa de disparidades resultante, (c) ponto de referência ampliado.

A baixa densidade do *dataset* Carro dá-se por seus pontos de referência estarem distantes e a rua não ser corretamente emparelhada devido à uniformidade de sua textura. Desta maneira, o mapa é predominantemente composto por objetos laterais, como demonstrado pelo seu mapa de disparidades na Figura 5.6. Esta baixa densidade acabou por prejudicar o desempenho do filtro, visto que havia pouca diferença de densidade

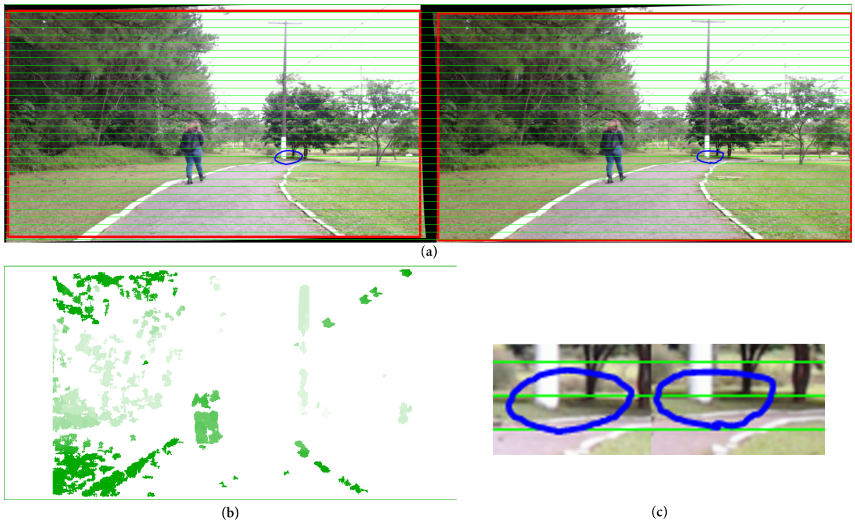


Figura 5.5 – *Frame* desconsiderado. (a) imagem retificada, (b) mapa de disparidades resultante, (c) ponto de referência ampliado.

entre *frames* sincronizados e dessincronizados, resultando na baixa quantidade de *frames* desconsiderados em relação aos outros *datasets*.

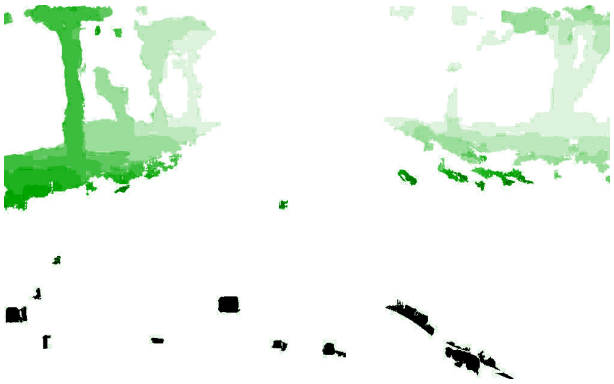


Figura 5.6 – Mapa de disparidades com baixa densidade no *dataset* Carro.

A Figura 5.7 apresenta um comparativo entre os mapas de disparidades de uma sequencia de 20 *frames* capturados, onde (a) mostra *frames* mantidos pelo filtro, e (b) *frames* descartados. Nota-se a grande variação na densidade dos *frames* descartados, o que reforça a necessidade do filtro para a aplicação, uma vez que os *frames* descartados não apresentam uma boa sincronia, e geram mais erros ao serem utilizados para computar a odometria visual. Em contraste, os mapas em (a) apresentam uma alta densidade e contém objetos mais facilmente identificáveis, o que implica uma melhor sincronização e melhores resultados ao serem processados pela odometria visual e reconstruídos em 3D.

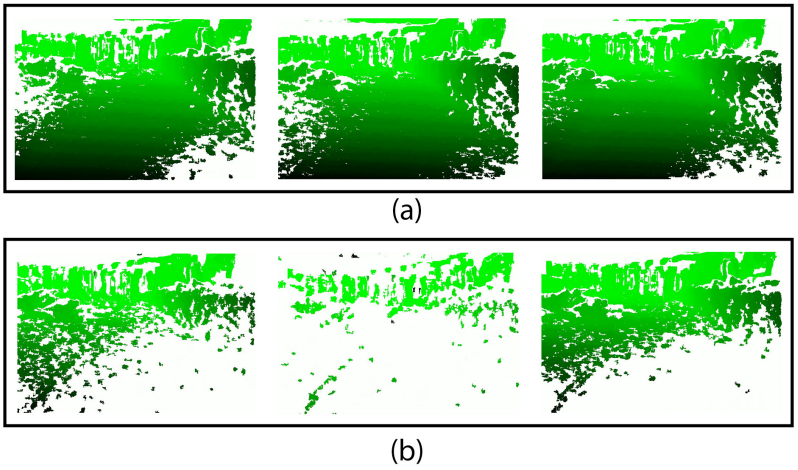


Figura 5.7 – (a) *frames* considerados sincronizados pelo filtro. (b) *frames* descartados pelo filtro.

5.4 ESTIMATIVA DE *POSE*

Nesta seção são tratados o resultado das *poses* e o caminho traçado por cada cenário de teste descrito no Capítulo 4, Seção 4.6.2. Os trajetos obtidos, demonstrados nas Figuras 5.8, 5.9 e 5.10, são classificados pelas cores:

- Preta: odometria visual com *frames* dessincronizados;
- Vermelha: odometria visual com *frames* sincronizados;
- Amarela: combinação de odometria visual com sensor de rotação do dispositivo;
- Azul: dados de GPS.

5.4.1 *Dataset* COPERVES

O resultado do processamento do *dataset* é apresentado na Figura 5.8, de acordo com as cores definidas neste Capítulo. Os dados de GPS não foram satisfatórios pois tiveram grande oscilação durante o trajeto, também parcialmente responsável pela alta discrepância na distância total percorrida. Desta maneira, o sensor não foi utilizado na união dos sensores com a odometria visual, sendo apenas utilizado o seu ponto inicial para georreferenciar o sistema.

A odometria visual foi capaz de identificar o trajeto percorrido tanto com *frames* dessincronizados quanto sincronizados. Contudo, é possível verificar uma discrepância entre ambas a partir da segunda curva do trajeto, onde a odometria com *frames* dessincronizados não realizou a rotação correta, tornando a curva mais aberta, e na terceira curva teve

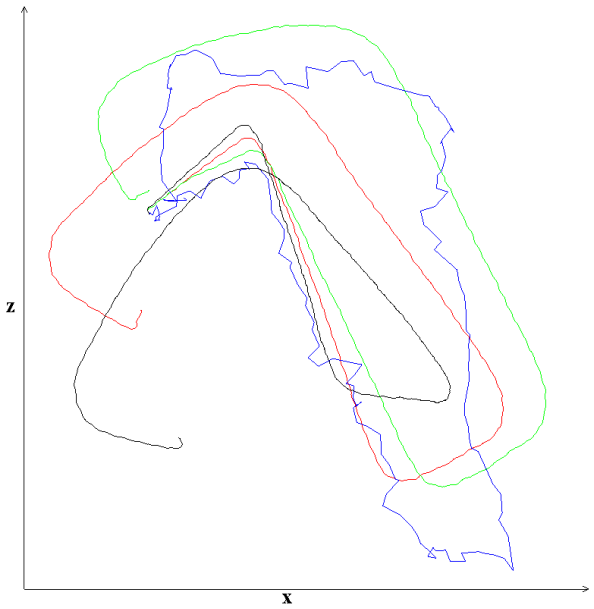


Figura 5.8 – Trajeto obtido no *Dataset* COPERVES.

uma menor angulação, o que também acontece na próxima curva. Em contraste, a odometria visual com *frames* sincronizados obteve resultados similares ao trajeto em verde, composto pela fusão entre a mesma e os sensores, com uma maior diferença inciando na terceira curva. A partir dos trajetos obtidos, é possível calcular o erro de fechamento de *loop* de cada cenário, demonstrados na Tabela 5.2.

Tabela 5.2 – *Loop Closure Error* no *dataset* COPERVES.

Método	Distância Total (m)	LCE (m)	LCE(%)
GPS	$103m$	$0{,}51m$	$0{,}49\%$
Odometria Visual (sem filtro)	$72m$	$11m$	15%
Odometria Visual (com filtro)	$79m$	$5{,}02m$	6%
Odometria Visual + Sensores	$79m$	$1{,}30m$	$1{,}6\%$

Destaca-se uma redução no erro de 15% para 6% ao filtrar os *frames* a serem utilizados na odometria visual. O erro obtido a partir da odometria visual em *frames* filtrados foi de 6%, levemente superior ao erro obtido pela odometria visual e sensores inerciais por Agrawal e Konolige (AGRAWAL; KONOLIGE, 2006), de $2{,}2\%$ a 5%, que utilizam câmeras sincronizas via *hardware* e placa de sensores inerciais proprietária.

Utilizando a correção por sensores, foi possível chegar a um erro de $1{,}6\%$, que é inferior ao erro de $2{,}1\%$ obtido pela odometria visual por Kitt et al. (KITT; GEIGER; LATEGAHN, 2010), e similar ao erro de $0{,}3\%$ a 2% obtido por Agrawal e Konolige (AGRAWAL; KONOLIGE, 2006). Destaca-se que ambos os trabalhos utilizam *hardware* proprietário e câmeras sincronizadas via *hardware*.

5.4.2 *Dataset* Planetário

Da mesma maneira apresentada na Seção 5.4.1, os dados da Figura 5.9 representam as poses do sistema, com cada cor associada a seu método de obtenção. Nota-se que novamente, os dados de GPS não foram satisfatórios devido à sua grande variação em um percurso em linha reta, e, portanto, não foi possível realizar a integração dos dados de GPS com os dados obtidos pela odometria visual. Também houve um problema com a escala da odometria visual, onde foi detectado um percurso de $548m$. Devido a este problema, os dados tiveram sua escala reduzida para que fossem melhor visualizados na Figura.

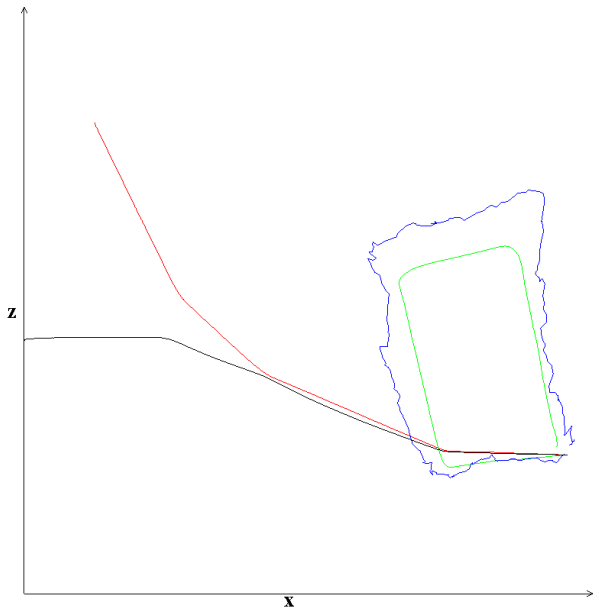


Figura 5.9 – Trajeto obtido no *Dataset* Planetário.

Devido à baixa quantidade de pontos de referência laterais somado à alta iluminação, o que prejudicou a captura clara do solo, o algoritmo de odometria visual cometeu diversos erros sobre a rotação do sistema, tanto com e sem filtro. É possível observar os pontos onde as curvas foram detectadas pela odometria visual na Figura 5.9, porém não foi possível detectar a rotação completa, não obtendo sucesso em fechar o *loop*. Contudo, os dados de rotação dos sensores foram suficientes para a correção destes erros, sendo possível concluir o *loop* do sistema, como apresentado na Tabela 5.3.

Tabela 5.3 – *Loop Closure Error* no *dataset* Planetário.

Método	Distância Total (m)	LCE (m)	LCE(%)
GPS	$173{,}91m$	$1{,}71m$	$0{,}9\%$
Odometria Visual (sem filtro)	$397{,}92m$	-	-
Odometria Visual (com filtro)	$548{,}2m$	-	-
Odometria Visual + Sensores	$548{,}2m$	$11{,}51m$	$2{,}1\%$

Devido à odometria visual não ser capaz de fechar o *loop*, seu erro não foi computado. Por outro lado, o erro da odometria corrigida por sensores foi de $2{,}1\%$, que é consistente com o erro de $1{,}6\%$ obtido no *Dataset* COPERVES, e similar ao erro de $0{,}3\%$ a 2% obtido por Agrawal e Konolige (AGRAWAL; KONOLIGE, 2006), e de $2{,}1\%$ obtido por Kitt et al. (KITT; GEIGER; LATEGAHN, 2010). Em ambos os *Datasets*, o sistema de baixo desenvolvido teve desempenho similar aos sistemas de alto custo.

5.4.3 *Dataset* Carro

Devido à alta interferência eletromagnética do carro, o vetor de rotação obtido pelo *smartphone* não foi satisfatório. Como demonstrado no Capítulo 2, Seção 2.9 e Capítulo 4, Seção 4.4.2, a correção automática do *drift* do vetor de rotação utiliza o magnetômetro, que também posiciona o dispositivo de acordo com o norte magnético, e um forte campo eletromagnético fixa o norte magnético em um ponto incorreto. Este comportamento pode ser observado na Figura 5.10, onde a rotação fina nas curvas é dada pelo giroscópio, mas gradualmente é corrigida para o falso norte magnético (lataria do carro), mantendo uma orientação similar à anterior à curva. Desta maneira, não foi possível utilizar os sensores do dispositivo.

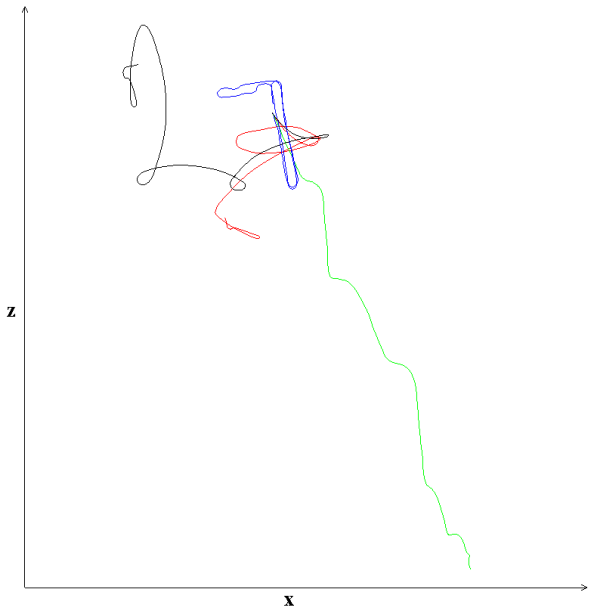


Figura 5.10 – Trajeto obtido no *Dataset* Carro.

Novamente, a odometria visual aplicada sem filtragem de *frames* não foi satisfatória, saindo do traçado proposto e com má detecção em suas curvas, o que pode ser visualizado pelo erro de fechamento de *loop* (*Loop Closure Error*), na Tabela 5.4. Em contraste, houve uma melhora no trajeto ao utilizar apenas *frames* filtrados, mas não foi suficiente para

uma boa estimativa de posição. Portanto, o sistema não funciona corretamente nestas condições.

Tabela 5.4 – *Loop Closure Error* no *dataset* Carro.

	Método	Distância Total (m)	LCE (m)	LCE(%)
Volta 1	Odometria Visual (sem filtro)	$336{,}44m$	$113{,}02m$	$33{,}59\%$
	Odometria Visual (com filtro)	$263{,}30m$	$57{,}75m$	$21{,}93\%$
	GPS	$320{,}55m$	$5{,}62m$	$1{,}75\%$
Volta 2	Odometria Visual (sem filtro)	$573{,}184$	$180{,}78m$	$31{,}54\%$
	Odometria Visual (com filtro)	$413{,}98m$	$103{,}65m$	$25{,}03\%$
	GPS	$467{,}65m$	$10{,}68m$	$2{,}28\%$

Os erros obtidos pelo sistema são muito maiores que os dos trabalhos relacionados. Contudo, destaca-se que no trabalho de Kitt et al. (KITT; GEIGER; LATEGAHN, 2010), onde utiliza-se odometria visual em carros, o sistema estereoscópico utiliza uma alta *baseline* e câmeras sincronizadas via *hardware* com um maior campo de visão horizontal, sendo possível uma melhor identificação de objetos laterais e distantes.

5.4.4 Comparativo entre *Dataset* Carro e *Dataset* 2009_09_08_drive_0021

Pode-se comparar a Figura 5.11, que apresenta a rota tomada pelo sistema utilizando no *Dataset* 2009_09_08_drive_0021 odometria visual, em vermelho, e a rota dada pelo GPS, em azul, com a Figura 5.10. Enquanto a odometria visual mostra-se suficiente no *dataset* gerado com câmeras sincronizadas via hardware e alta *baseline* (GEIGER; ZIEGLER; STILLER, 2011), gerando um erro de $2{,}1\%$, o resultado não foi satisfatório no sistema proposto com sincronização via software, onde o erro é acima de 25%.

5.5 ANÁLISE DA PRECISÃO DO GPS

Para demonstrar o alto erro obtido nos dados de GPS, foram traçadas duas retas onde o trajeto esperado era reto no *dataset* COPERVES, uma entre dois pontos de GPS, e a outra entre dois pontos no traçado obtido pela fusão de informações, demonstrado na Figura 5.12, onde a linha azul representa os dados de GPS, a linha verde representa o traçado obtido pela fusão de informações, e as linhas vermelha e preta representam suas respectivas retas de comparação.

O erro *Normalized Root Mean Square* (NRMSE) foi calculado a partir da distância entre cada ponto e a reta, e normalizado a partir do comprimento total da reta, resultando em $0{,}036604\%$ para os dados de GPS e $0{,}00311\%$ para a fusão de informações. Nota-se

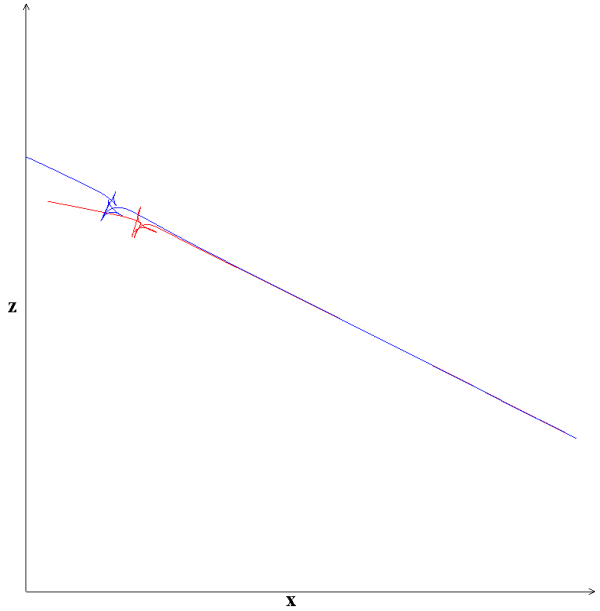


Figura 5.11 – Trajeto obtido no *dataset* 2009_09_08_drive_0021.

que o NRMSE do GPS é em torno de 11 vezes maior que o da fusão de informações. Desta maneira, é possível confirmar que o alto erro do GPS torna-o inutilizável nos cenários de baixa velocidade, mesmo que seu LCE seja inferior ao obtido pelos outros métodos utilizados.

5.6 ANÁLISE DE DESEMPENHO

Durante o processamento dos *datasets*, obteve-se uma média de $708ms$ para a remoção de distorção e retificação das imagens, $726{,}112ms$ para a geração do mapa de disparidades utilizando SBM, e $2428{,}6ms$ para o processamento de odometria visual. No total, estes passos somam $3862{,}712ms$ por *frame*, ou $0{,}25$ *frames* por segundo (FPS).

O *dataset* COPERVES contém 4978 *frames*, o que resulta em aproximadamente $5{,}34h$ dedicadas ao processamento, considerando que nenhum *frame* é descartado. Como a odometria visual não é processada quando o *frame* é descartado, e considerando um descarte de $65{,}50\%$ realizado pelo filtro, o tempo de processamento total torna-se em torno de $3{,}13h$, dedicando-se $1{,}98h$ para retificação e geração de mapa de disparidades, e $1{,}15h$ para o cálculo da odometria visual. Como resultado, o tempo de processamento médio é de $2271{,}979ms$, ou $0{,}44FPS$, enquanto o *dataset* contém em torno de 1717 *frames* aproveitáveis, equivalente à uma taxa de $10{,}35FPS$.

Destaca-se que o tempo base de processamento é de $1434{,}112ms$, que consiste da retificação das imagens e geração do mapa de disparidades, necessários para realizar a filtragem dos *frames*. Desta maneira, o tempo de processamento é muito elevado para

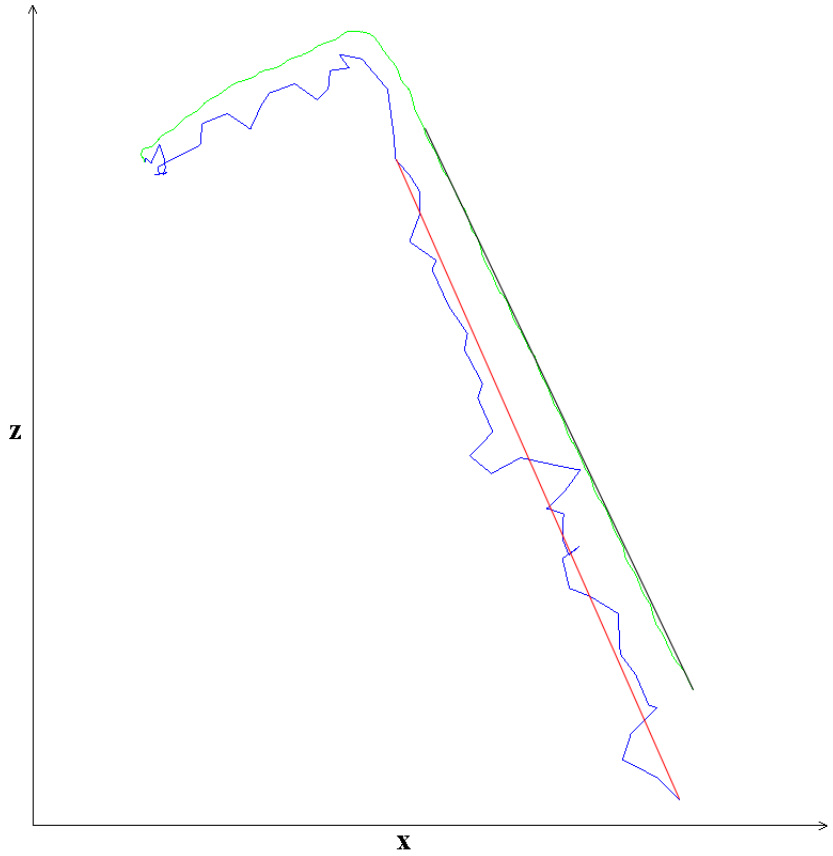


Figura 5.12 – Comparação entre trajetos obtidos e esperados em um trecho predominantemente reto.

uma aplicação em tempo real.

Visando reduzir o tempo de processamento, também foram realizados testes reduzindo a resolução dos *frames* pela metade, $640x360$. O resultado desta redução foi um tempo de processamento de $160ms$ para a remoção de distorção e retificação das imagens, $235{,}456ms$ para a geração do mapa de disparidades para, e $582{,}989ms$ para o processamento da odometria visual.

Considerando um descarte de $65{,}50\%$ de *frames*, o tempo de processamento médio é em torno de $596{,}58ms$, ou $1{,}67FPS$. Embora o tempo de processamento tenha diminuído, todos os *frames* devem ser processados para que o filtro seja efetivo. Sendo assim, o processamento base de $395{,}456ms$, ou $2{,}52FPS$ não é suficiente para processar uma entrada de $30FPS$ em tempo real.

Para que uma aplicação em tempo real seja possível e necessária uma maneira de garantir que o *frame* está sincronizado, utilizando sincronização via *hardware* ou um método de filtragem com maior desempenho, preferencialmente não necessitando avaliar todos os *frames* capturados. Ao presumir que todos os *frames* estão sincronizados obtém-se um tempo de processamento de $756{,}58ms$, ou $1{,}32FPS$, onde é possível realizar o processamento apenas no *frame* atual, tornando-o utilizável em tempo real.

Ao manter a resolução original do *dataset*, o tempo de processamento sem o filtro é de $3136{,}6ms$, ou $0{,}31FPS$, o que também prejudica a execução em tempo real. Desta maneira, o sistema não pode ser utilizado em tempo real em seu estado atual.

5.7 RECONSTRUÇÃO 3D

A partir do mapa de disparidades, foi possível construir uma nuvem de pontos, demonstrado a profundidade obtida na cena, e pode ser visualizada na Figura 5.13, com o bule de chá representando o sistema estereoscópico em relação à imagem, e as linhas coloridas representando os eixos X, Y e Z.

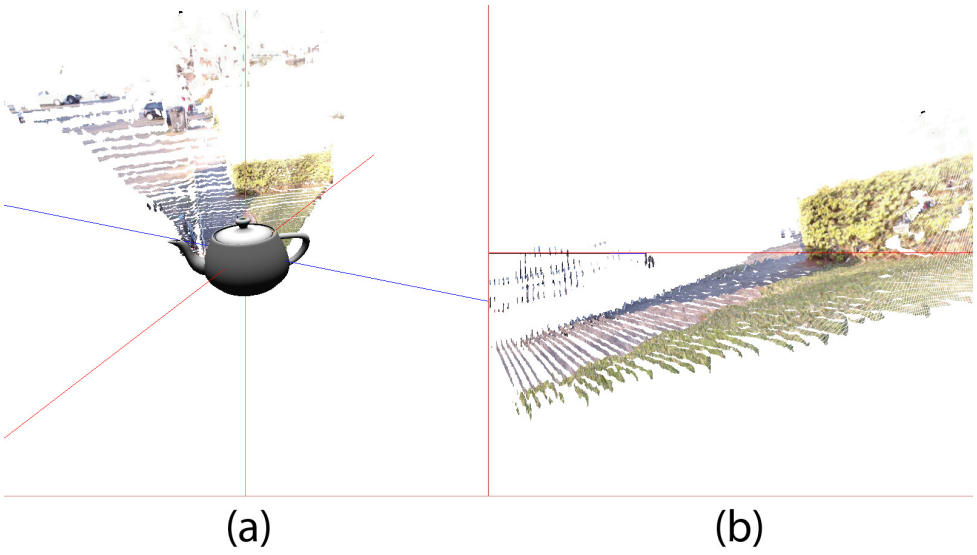


Figura 5.13 – Reconstrução 3D do *dataset* COPERVES.

A Figura 5.14 mostra a reconstrução do outro lado dos arbustos mostrados na Figura 5.13, com (a) logo após a curva e (b) aproximando-se do prédio a frente, melhor visualizado pela posição das plantas à esquerda do sistema.

Devido à complexidade de renderização, além de problemas como uso de memória e filtragem de pontos muito próximos ou distantes, não houve a preocupação com a reconstrução 3D completa do trajeto, porém é demonstrado que é possível realizá-la a partir das *poses* obtidas em conjunto com os mapas de disparidades com uma maior densidade devido à filtragem realizada.

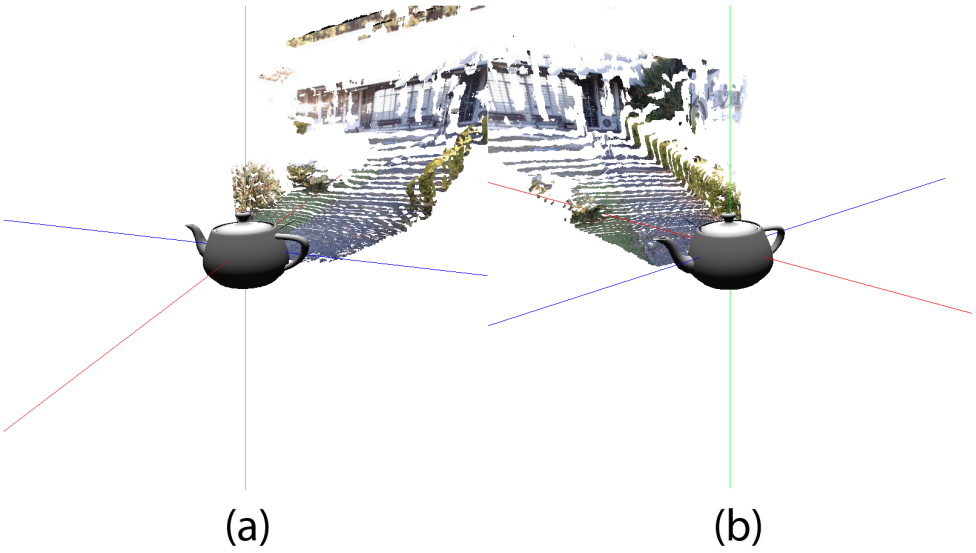


Figura 5.14 – Outras posições na reconstrução 3D do *dataset* COPERVES.

6 CONCLUSÃO

Ao utilizar o sistema proposto, foi possível estimar sua posição com um erro similar aos erros encontrados nos trabalhos relacionados. Entretanto, os dados de GPS não foram precisos nos *datasets* gerados em baixa velocidade, o que impossibilitou seu uso para obtenção de resultados mais precisos.

As câmeras sem sincronização via *hardware* mostraram-se eficazes após terem seus *frames* dessincronizados filtrados, diminuindo seu erro. Devido à baixa *baseline* do sistema combinada com o baixo ângulo de visão horizontal, objetos distantes não são corretamente identificados pela odometria visual, resultando em erros grandes, como identificado no *Dataset* Carro.

Desta maneira, o sistema obteve melhor desempenho em ambientes com objetos mais próximos, como no *dataset* COPERVES, onde a odometria visual corretamente identificou a rota percorrida, e em ambientes abertos de baixa velocidade, como no *dataset* Planetário, onde a odometria foi incapaz de identificar as curvas realizadas, porém obteve um bom fechamento de *loop* ao corrigi-las com dados sensoriais.

De acordo com os resultados obtidos nos *datasets* Carro e Planetário, recomenda-se que sejam utilizados sistemas com uma maior *baseline* e câmeras com um maior ângulo de visão horizontal em ambientes abertos, para que os objetos sejam mais facilmente detectados e sua disparidade melhor identificada, sendo possível obter uma odometria visual mais precisa.

Devido ao alto tempo associado ao processamento dos *datasets*, o sistema não pode ser utilizado em tempo real no momento. Contudo, testes preliminares não mostraram grandes alterações de resultados ao diminuir a resolução das imagens, visto que os pontos mais afetados são onde a disparidade é muito baixa, o implicando uma maior distância em relação ao sistema e não são bons pontos de referência devido às situações onde a dessincronização dos *frames* pode gerar uma pequena diferença na disparidade geral, o que se traduz numa grande diferença onde as disparidades são baixas.

De acordo com a análise de desempenho, muito processamento é realizado em *frames* que são descartados. Desta maneira, melhores métodos de sincronização via *software*, ou utilizar câmeras sincronizadas via *hardware*, podem aumentar o desempenho do sistema, tornando-o utilizável em aplicações em tempo real.

O código foi desenvolvido visando à abstração do sistema estereoscópico utilizado, ou seja, não limitado ao modelo específico das câmeras utilizadas, tamanho de *baseline*, e pode ser utilizado em câmeras com sincronização via *hardware*. O dispositivo de leitura sensorial também pode ser alterado, desde que utilize o aplicativo desenvolvido, ou tenha os dados enviados em forma serial.

O mapa de disparidades também se mostrou denso em ambientes menores, sendo

possível realizar uma reconstrução 3D mais completa dos ambientes mapeados.

Embora tenha um funcionamento mais restrito que o descrito nos objetivos do trabalho, destaca-se que o uso de câmeras de baixo custo para SLAM auxiliadas por sensores mostrou-se possível e viável, desde que seus dados sejam devidamente filtrados para obter uma sincronia mais precisa, sendo possível a construção de um robô de baixo custo e sem necessidade de odometria, como em *drones* aéreos.

6.1 TRABALHOS FUTUROS

Devido à flexibilidade do código e o uso de bibliotecas multiplataforma, o sistema pode ser executado em dispositivos embarcados, sendo possível construir um robô controlado remotamente para a geração de *dataset*.

Dada a impossibilidade de uso do GPS nos cenários de baixa velocidade, e o baixo erro encontrado apesar desta limitação, é possível analisar o desempenho do sistema para ambientes fechados. Deve-se levar em consideração que as câmeras utilizadas apresentaram uma menor taxa de *frames* quando seu tempo de exposição foi aumentado.

Pode-se também utilizar câmeras de baixo nível que possam ser sincronizadas via *hardware*, propondo uma alternativa por um preço mais elevado, porém mais barato que soluções proprietárias.

A partir do lançamento de novos sensores e versões mais precisas dos atuais em dispositivos Android, o aplicativo desenvolvido poderá ser utilizado para realizar suas leituras e a comunicação entre os dispositivos, visto que ele faz uma varredura em todos os sensores disponibilizados pela plataforma. Destaca-se o trabalho publicado recentemente por Chen et al. (CHEN; ZHAO; FARRELL, 2016), onde é proposto um método de cálculo em tempo real, utilizando GPS comum, obtendo-se precisão centimétrica na estimativa de posição. Este método não requer sensores extras, e pode ser implementado em futuras iterações do Android.

O aplicativo também será disponibilizado na loja de aplicativos do sistema para que possa ser utilizado em trabalhos similares no futuro. Da mesma maneira, os *datasets* gerados poderão ser utilizados em pesquisas futuras para obter uma melhor sincronização e calibração do sistema, assim como diferentes métodos de odometria visual, técnicas de visão computacional, e combinação sensorial.

REFERÊNCIAS BIBLIOGRÁFICAS

AGARWAL, S.; HABLANI, H. B. Precise positioning using gps for category-iii aircraft operations using smoothed pseudorange measurements. In: **Proceedings of the International Conference & Workshop on Emerging Trends in Technology**. New York, NY, USA: ACM, 2011. (ICWET '11), p. 846–850. ISBN 978-1-4503-0449-8. Disponível em: <http://doi.acm.org/10.1145/1980022.1980205>.

AGRAWAL, M.; KONOLIGE, K. Real-time localization in outdoor environments using stereo vision and inexpensive gps. In: IEEE. **18th International Conference on Pattern Recognition (ICPR'06)**. [S.l.], 2006. v. 3, p. 1063–1068.

BAYDOUN, M.; AL-ALAOUI, M. A. Enhancing stereo matching with classification. **Access, IEEE**, IEEE, v. 2, p. 485–499, 2014.

BEGUM, M.; MANN, G. K.; GOSINE, R. G. Integrated fuzzy logic and genetic algorithmic approach for simultaneous localization and mapping of mobile robots. **Applied Soft Computing**, Elsevier, v. 8, n. 1, p. 150–165, 2008.

BLANCO, J.-L.; GONZÁLEZ, J.; FERNÁNDEZ-MADRIGAL, J.-A. Subjective local maps for hybrid metric-topological slam. **Robotics and Autonomous Systems**, Elsevier, v. 57, n. 1, p. 64–74, 2009.

BOAS, M. L. **Mathematical methods in the physical sciences; 3rd ed.** Hoboken, NJ: Wiley, 2006. Disponível em: <https://cds.cern.ch/record/913305>.

BOUCHER, M.; ABABSA, F.-E.; MALLEM, M. Reducing the slam drift error propagation using sparse but accurate 3d models for augmented reality applications. In: **Proceedings of the Virtual Reality International Conference: Laval Virtual**. New York, NY, USA: ACM, 2013. (VRIC '13), p. 11:1–11:6. ISBN 978-1-4503-1875-4. Disponível em: <http://doi.acm.org/10.1145/2466816.2466828>.

BRADSKI, A. **Learning OpenCV, [Computer Vision with OpenCV Library ; software that sees]**. 1. ed.. ed. [S.l.]: O'Reilly Media, 2008. Gary Bradski and Adrian Kaehler.

BRADSKI, G. Opencv library. **Dr. Dobb's Journal of Software Tools**, 2000.

CARDOSO, D. M. S. Uso de estereoscopia na movimentação autônoma de robôs. In: MENOTTI, A. C. N. J. D. (Ed.). **Workshops of SIBGRAPI)**. Ouro Preto, MG, Brazil: [s.n.], 2012. Disponível em: <http://www.decom.ufop.br/sibgrapi2012/eproceedings>.

CHEN, Y.; ZHAO, S.; FARRELL, J. A. Computationally efficient carrier integer ambiguity resolution in multiepoch gps/ins: A common-position-shift approach. **IEEE Transactions on Control Systems Technology**, v. 24, n. 5, p. 1541–1556, Sept 2016. ISSN 1063-6536.

CYGANEK, B.; SIEBERT, J. P. **An introduction to 3D computer vision techniques and algorithms**. [S.l.]: John Wiley & Sons, 2011.

FARAGHER, R. M.; SARNO, C.; NEWMAN, M. Opportunistic radio slam for indoor navigation using smartphone sensors. In: **Position Location and Navigation Symposium (PLANS), 2012 IEEE/ION**. [S.l.: s.n.], 2012. p. 120–128. ISSN 2153-358X.

FINLAY, C. C. et al. International geomagnetic reference field: the eleventh generation. **Geophysical Journal International**, v. 183, n. 3, p. 1216–1230, 2010. Disponível em: <http://gji.oxfordjournals.org/content/183/3/1216.abstract>.

FRADEN, J. **Handbook of Modern Sensors: Physics, Designs, and Applications (Handbook of Modern Sensors)**. [S.l.]: SpringerVerlag, 2003. ISBN 0387007504.

FRESE, U.; HIRZINGER, G. **Simultaneous Localization and Mapping - A Discussion**. 2001.

GEIGER, A.; LENZ, P.; URTASUN, R. Are we ready for autonomous driving? the kitti vision benchmark suite. In: **Conference on Computer Vision and Pattern Recognition (CVPR)**. [S.l.: s.n.], 2012.

GEIGER, A.; ROSER, M.; URTASUN, R. Efficient large-scale stereo matching. In: **Asian Conference on Computer Vision (ACCV)**. [S.l.: s.n.], 2010.

GEIGER, A.; ZIEGLER, J.; STILLER, C. Stereoscan: Dense 3d reconstruction in real-time. In: **Intelligent Vehicles Symposium (IV)**. [S.l.: s.n.], 2011.

GONZALEZ, R.; WOODS, R. **Processamento de imagens digitais**. 1nd. ed. [S.l.]: Edgard Blucher, 2000. ISBN 9788521202646.

GOOGLE. **Android Developers**. 2016. Disponível em: <http://developer.android.com/reference/>.

HEDGECOCK, W. et al. Accurate real-time relative localization using single-frequency gps. In: **Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems**. New York, NY, USA: ACM, 2014. (SenSys '14), p. 206–220. ISBN 978-1-4503-3143-2. Disponível em: <http://doi.acm.org/10.1145/2668332.2668379>.

____. High-accuracy differential tracking of low-cost gps receivers. In: **Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services**. New York, NY, USA: ACM, 2013. (MobiSys '13), p. 221–234. ISBN 978-1-4503-1672-9. Disponível em: <http://doi.acm.org/10.1145/2462456.2464456>.

HIRSCHMULLER, H. Stereo processing by semiglobal matching and mutual information. **Pattern Analysis and Machine Intelligence, IEEE Transactions on**, IEEE, v. 30, n. 2, p. 328–341, 2008.

HU, X.; MORDOHAI, P. Evaluation of stereo confidence indoors and outdoors. In: **Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on**. [S.l.: s.n.], 2010. p. 1466–1473. ISSN 1063-6919.

JU, M.-H.; KANG, H.-B. A new method for stereo matching using pixel cooperative optimization. In: IEEE. **Image Processing (ICIP), 16th IEEE International Conference on**. [S.l.], 2009. p. 2105–2108.

KALMAN, R. E. A new approach to linear filtering and prediction problems. **Transactions of the ASME–Journal of Basic Engineering**, v. 82, n. Series D, p. 35–45, 1960.

KAO, W. W.; HUY, B. Q. Indoor navigation with smartphone-based visual slam and bluetooth-connected wheel-robot. In: **Automatic Control Conference (CACS), 2013 CACS International**. [S.l.: s.n.], 2013. p. 395–400.

KARLSSON, N. et al. The vslam algorithm for robust localization and mapping. In: **Proceedings of the 2005 IEEE International Conference on Robotics and Automation**. [S.l.: s.n.], 2005. p. 24–29. ISSN 1050-4729.

KIM, T. I. et al. Vision system for mobile robots for tracking moving targets, based on robot motion and stereo vision information. In: **System Integration (SII), 2011 IEEE/SICE International Symposium on**. [S.l.: s.n.], 2011. p. 634–639.

KITT, B.; GEIGER, A.; LATEGAHN, H. Visual odometry based on stereo image sequences with ransac-based outlier rejection scheme. In: **Intelligent Vehicles Symposium (IV)**. [S.l.: s.n.], 2010.

KLAUS, A.; SORMANN, M.; KARNER, K. Segment-based stereo matching using belief propagation and a self-adapting dissimilarity measure. In: IEEE. **Pattern Recognition. ICPR. 18th International Conference on**. [S.l.], 2006. v. 3, p. 15–18.

KONOLIGE, K. Small vision systems: Hardware and implementation. In: **Robotics Research**. [S.l.]: Springer London, 1998. p. 203–212.

LAZAROS, N.; SIRAKOULIS, G. C.; GASTERATOS, A. Review of stereo vision algorithms: from software to hardware. **International Journal of Optomechatronics**, Taylor & Francis, v. 2, n. 4, p. 435–462, 2008.

LEGG, A. et al. Estereoscopia para mapeamento 3d. In: **SIBGRAPI 2014, Proceedings of the XXVII Brazilian Symposium on Computer Graphics and Image Processing, Rio de Janeiro, Brazil, 27-30 August 2014**. [S.l.]: IEEE Computer Society, 2014. p. 20–23.

LEONARD, J.; NEWMAN, P. Consistent, convergent, and constant-time slam. In: **IJCAI**. [S.l.: s.n.], 2003. p. 1143–1150.

MENDES, C. C. T. **Navegação de robôs móveis utilizando visão estéreo**. 2012. Dissertação (Dissertação de Mestrado) — Universidade de São Paulo, 2012. Disponível em: <http://www.teses.usp.br/teses/disponiveis/55/55134/tde-18062012-162436/publico/dissertacao_rev_a4.pdf>.

MOUTARLIER, P.; CHATILA, R. An experimental system for incremental environment modelling by an autonomous mobile robot. In: **The First International Symposium on Experimental Robotics I**. London, UK, UK: Springer-Verlag, 1990. p. 327–346. ISBN 3-540-52182-8. Disponível em: <http://dl.acm.org/citation.cfm?id=645621.661279>.

MURRAY, R. M. et al. **A mathematical introduction to robotic manipulation**. [S.l.]: CRC press, 1994.

OLEARI, F.; RIZZINI, D. L.; CASELLI, S. A low-cost stereo system for 3d object recognition. In: **Intelligent Computer Communication and Processing (ICCP), 2013 IEEE International Conference on**. [S.l.: s.n.], 2013. p. 127–132.

ORGHIDAN, R. et al. Structured light self-calibration with vanishing points. **Machine vision and applications**, Springer, v. 25, n. 2, p. 489–500, 2014.

RUSU, R. B. et al. Fast 3d recognition and pose using the viewpoint feature histogram. In: IEEE. **Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on**. [S.l.], 2010. p. 2155–2162.

SHENITZER, A.; ROSENFELD, B.; GRANT, H. **A History of Non-Euclidean Geometry: Evolution of the Concept of a Geometric Space**. Springer New York, 1988. (Studies in the History of Mathematics and Physical Sciences). ISBN 9780387964584. Disponível em: <https://books.google.com.br/books?id=DRLpAFZM7uwC>.

SHIN, H.; CHON, Y.; CHA, H. Unsupervised construction of an indoor floor plan using a smartphone. **IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)**, v. 42, n. 6, p. 889–898, Nov 2012. ISSN 1094-6977.

SMITH, R. C.; CHEESEMAN, P. On the representation and estimation of spatial uncertainly. **Int. J. Rob. Res.**, Sage Publications, Inc., Thousand Oaks, CA, USA, v. 5,

n. 4, p. 56–68, dez. 1986. ISSN 0278-3649. Disponível em: <http://dx.doi.org/10.1177/027836498600500404>.

SZELISKI, R. **Computer vision: algorithms and applications**. [S.l.]: Springer Science & Business Media, 2010.

THRUN, S. Robotic mapping: A survey. In: LAKEMEYER, G.; NEBEL, B. (Ed.). **Exploring Artificial Intelligence in the New Millenium**. [S.l.]: Morgan Kaufmann, 2002. To appear.

THRUN, S. et al. Robust monte carlo localization for mobile robots. **Artificial Intelligence**, v. 128, n. 1–2, p. 99 – 141, 2001. ISSN 0004-3702. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0004370201000698>.

VENTURA, J. et al. Global localization from monocular slam on a mobile phone. **IEEE Transactions on Visualization and Computer Graphics**, v. 20, n. 4, p. 531–539, April 2014. ISSN 1077-2626.

WELCH, G.; BISHOP, G. **An introduction to the Kalman filter**. 1995.

WOODMAN, O.; HARLE, R. Pedestrian localisation for indoor environments. In: ACM. **Proceedings of the 10th international conference on Ubiquitous computing**. [S.l.], 2008. p. 114–123.

YAO, G. et al. A rapid stereo matching algorithm based on disparity interpolation. In: IEEE. **World Automation Congress (WAC)**. [S.l.], 2012. p. 5–10.

YEO, H.-S.; LEE, B.-G.; LIM, H. Hand tracking and gesture recognition system for human-computer interaction using low-cost hardware. **Multimedia Tools and Applications**, Springer, v. 74, n. 8, p. 2687–2715, 2015.

ZHANG, H. et al. A linear trinocular rectification method for accurate stereoscopic matching. In: **In Proceedings of British Machine Vision Conference(BMVC'03**. [S.l.: s.n.], 2003. p. 281–290.

ZUNINO, G. Simultaneous localization and mapping for navigation in realistic environments. In: **In Proc. IEEE Conference**. [S.l.: s.n.], 2002.