Universidade do Minho
Escola de Engenharia

José Pedro Araújo Soares

Localização e Mapeamento Simultâneo utilizando uma câmara RGB-D

Dissertação de Mestrado

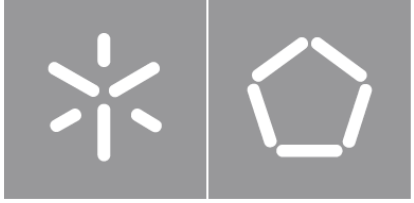
Mestrado Integrado em Engenharia

Eletrónica Industrial e Computadores

Trabalho efetuado sob a orientação do

Professor Doutor Fernando Ribeiro

janeiro 2021



Universidade do Minho
Escola de Engenharia

José Pedro Araújo Soares

Localização e Mapeamento Simultâneo utilizando uma câmara RGB-D

Dissertação de Mestrado

Mestrado Integrado em Engenharia

Eletrónica Industrial e Computadores

Trabalho efetuado sob a orientação do

Professor Doutor Fernando Ribeiro

janeiro 2021

DIREITOS DE AUTOR E CONDIÇÕES DE UTILIZAÇÃO DO TRABALHO POR TERCEIROS

Este é um trabalho académico que pode ser utilizado por terceiros desde que respeitadas as regras e boas práticas internacionalmente aceites, no que concerne aos direitos de autor e direitos conexos.

Assim, o presente trabalho pode ser utilizado nos termos previstos na licença abaixo indicada.

Caso o utilizador necessite de permissão para poder fazer um uso do trabalho em condições não previstas no licenciamento indicado, deverá contactar o autor, através do RepositóriUM da Universidade do Minho.

Atribuição
CC BY

https://creativecommons.org/licenses/by/4.0

Agradecimentos

Chegando ao fim deste trabalho, é importante agradecer a todos aqueles que contribuíram de forma direta e indireta na realização deste trabalho e que levaram a concretização dos objetivos propostos. Posto isto, gostaria de agradecer:

- Ao meu orientador Dr. Fernando Ribeiro, que sempre me apoiou durante o projeto. Sempre disponível para dúvidas que iam surgindo bem como ideias para resolver problemas.
- A todos os meus colegas de laboratório que chateei sempre que me foi oportuno.
- A toda a minha família e amigos que sempre me deram apoio ao longo do percurso académico.
- À Mafalda, que tolerou e compreendeu várias épocas de exames e testes e ainda me deu força para terminar esta etapa.
- À Eva que me deu o apoio final para a conclusão deste trabalho.

DECLARAÇÃO DE INTEGRIDADE

Declaro ter atuado com integridade na elaboração do presente trabalho académico e confirmo que não recorri à prática de plágio nem a qualquer forma de utilização indevida ou falsificação de informações ou resultados em nenhuma das etapas conducente à sua elaboração.

Mais declaro que conheço e que respeitei o Código de Conduta Ética da Universidade do Minho.

Localização e Mapeamento Simultâneo utilizando uma câmara RGB-D

Resumo

Esta dissertação tem como objetivo desenvolver um sistema capaz de fazer um mapeamento e auto-localização simultaneamente, conhecido na robótica como SLAM, recorrendo a visão por computador.

A localização e a capacidade de desenvolver um mapa de um determinado ambiente sempre foram áreas de estudo importantes para a robótica. Esta importância deve-se a uma tendência em procurar construir robôs que consigam fazer tarefas de forma independente.

O sistema proposto estima a odometria visual com um sistema mono câmara. Pontos-chave são encontrados utilizando o algoritmo FAST. Entre frames, são verificadas as transformações que ocorrem nestes pontos. O algoritmo de 5-pontos de Nister é utilizado para prever a matriz essencial utilizando as transformações identificadas. Esta matriz permite estimar o movimento realizado pela câmara. Uma Kinect é utilizada para recuperar a escala e fornecer nuvens de pontos para a construção do mapa.

O sistema foi desenvolvido em Python e utiliza ROS e os seus recursos para atingir os objetivos propostos.

Palavras-Chave: SLAM, Mapeamento, Auto-localização, Visão por Computador, Kinect, nuvens de pontos, Python, ROS.

Simultaneous Localization and Mapping using an RGB-D camera

Abstract

This dissertation aims to develop a system capable of self-localization and mapping, known in robotics as SLAM. The goal is to use computer vision.

Self-localization and the ability of developing a map of a given environment are important areas of study in robotics. The intention and tendency of build robots that can do basic tasks independently is growing each year.

The proposed system estimates visual odometry with a mono camera system. Keypoints are found using the FAST algorithm. Between frames, thoose points are tracked and Nister's 5-point algorithm is used to predict the essential matrix. This matrix allows to calculate the movement performed by the camera. A Kinect is used to recover the scale and provide point clouds for the construction of the map.

The system was developed in Python and use ROS and its resources to achieve the proposed objectives.

Keywords: SLAM, Mapeamento, Auto-localização, Visão por Computador, Kinect, nuvens de pontos, Python, ROS.

Índice

Índice de Figuras

Índice de Tabelas

Acrónimos e abreviaturas

SLAM - **S**imultaneos **L**ocalization **a**nd **M**apping

LIDAR – **Li**ght **D**etection **A**nd **R**anging

RADAR - **Ra**dio **D**etection **a**nd **R**anging

SONAR - **So**und Navigation **a**nd **R**anging

GPS – **G**lobal **P**ositioning **S**ystem

IMU – **I**nertial **M**esurement **U**nit

RGB-D – (**R**ed, **G**reen, **B**lue - **D**epth)

RGB – (**R**ed, **G**reen, **B**lue)

INS – **I**nertial **N**avigation **S**ystem

LMedS – **L**east **Med**ian of **S**quares

RANSAC - Random **Sa**mple **C**onsensus

fps – **f**rames **p**or **s**egundo

Capítulo 1 – Introdução

1.1 Enquadramento e motivação

Cada vez mais são usados robôs totalmente autónomos no nosso dia-a-dia. Saber a posição em que se encontram para determinarem a posição para onde pretendem ir, torna-se assim essencial para estes robôs. *Simultaneos Localization and Mapping* (SLAM) é uma técnica que permite uma auto-localização e um mapeamento simultâneo de um ambiente desconhecido.

Existem diversos sensores que podem ser utilizados em SLAM, como por exemplo, LIDAR, SONAR, RADAR, Bússolas, *encoders,* uma ou mais câmaras, câmaras RGB-D, entre outros. Em muitos exemplos da literatura são utilizados mais que um sensor, de modo a tornar o sistema mais robusto. Como consequência da quantidade de sensores disponíveis, existem diversos algoritmos adaptados a cada sensor. Esta dissertação pretende estudar um dos ramos mais complexos, o SLAM visual.

A visão por computador é amplamente utilizada em robótica. As câmaras existem em diversas resoluções, tamanhos e são economicamente muito competitivas. Existem em *smartphones*, *drones*, robôs, entre outros, devido à sua versatilidade. *Encoders* de rodas apresentam erros quando a tração do terreno é baixa, o que não acontece com uma câmara. Sistemas com GPS não são viáveis em aplicações no interior de edifícios. Sonares e radares podem apresentar problemas quando existe mais do que um robô com esses sensores no mesmo local devido a interferências de sinal. LIDAR's são muito utilizados para estes sistemas pois têm uma excelente precisão, mas apresentam apenas uma linha horizontal de informação. Tendo em conta os argumentos apresentados, o grande interesse em robótica, o gosto por visão por computador e a possibilidade de aplicar este sistema a um robô que está em desenvolvimento, foram os motivos que alimentaram o interesse pelo projeto e levaram à sua realização.

1.2 Objetivos

O principal objetivo desta dissertação é realizar um sistema SLAM visual. Será utilizado o módulo Kinect Xbox 360 que contém embutida uma câmara RGB e um sensor de profundidade infravermelho.

Assim, os objetivos deste projeto passam por:

- Calibração da Kinect.
- Estudar a viabilidade de aplicar um denso fluxo ótico 2D (*machine learning*).
- Estudar diferentes plataformas para a criação do ambiente 3D e escolher a que apresentar mais vantagens.
- Estimar a odometria visual.
- Fundir a odometria visual estimada com a informação sobre a distância para construir um mapa virtual.

1.3 Estrutura da dissertação

Esta dissertação é composta por 5 capítulos:

- O primeiro capítulo serve como introdução ao tema. É feito um enquadramento sobre o trabalho, explicada a motivação que levou à sua execução e os objetivos estipulados.
- No segundo capítulo são expostas as definições gerais e os conceitos sobre o projeto que serviram de base para o seu desenvolvimento. São ainda referidos diferentes métodos para a realização do objetivo pretendido.
- No terceiro capítulo é apresentado todo o trabalho desenvolvido que permitiu chegar ao protótipo final do projeto. Será mostrado o *hardware* utilizado, todos os procedimentos realizados e o algoritmo final. Por fim, apresentar-se-á um estudo sobre a viabilidade da implementação de um método no sistema.

- No quarto capítulo serão apresentados os testes práticos realizados sobre o sistema final e os seus resultados.
- No quinto e último capítulo, descrever-se-ão as conclusões do projeto e feita uma análise geral do sistema. Serão descritos trabalhos futuros que tornariam o sistema mais preciso e robusto.

Capítulo 2 – Estado da arte

Este capítulo serve como contextualização do tema. Para isso, serão apresentados os problemas associados ao SLAM, bem como, algumas formas de os resolver. SLAM, como referido anteriormente, tem dois grandes objetivos a cumprir, sendo estes a localização e o mapeamento. Embora a realização de um tenha implicação direta no outro, em algumas partes do capítulo serão abordados de forma independente.

2.1 Definição de odometria visual

Odometria visual é um processo que visa calcular a posição e orientação de um agente (robô, humano, veículo, entre outros) com base num conjunto de imagens captadas por uma ou mais câmaras [1]. A partir deste ponto do documento, o significado de odometria visual será referente à definição apresentada, pelo que, pode estar a referenciar sistemas com uma, duas ou mais câmaras.

2.2 Tipos de sistemas de localização

Este subcapítulo focar-se-á na localização, contudo, serão dados alguns exemplos sobre sistemas SLAM. Como explicado anteriormente, neste documento existem diversos sensores que podem ser usados nestes sistemas, no entanto, o objetivo desta dissertação passa por utilizar visão por computador.

Como consequência, o estudo da literatura foi mais focado nos sistemas acima referidos. Assim, os projetos estudados foram divididos em: projetos que não utilizam câmara, projetos que utilizam uma fusão entre câmaras e outro tipo de sensores e projetos que utilizam apenas visão por computador.

2.2.1 Sensores-não-câmaras

Existem diversos exemplos na literatura sobre sistemas que não utilizam câmaras. O odómetro é um equipamento que mede a distância percorrida por um veículo. Para realizar esta tarefa é necessário fazer odometria de roda. Por outras palavras, é necessário contar o número de voltas que a roda dá para calcular a distância que o veículo percorreu. Estes sistemas utilizam *encoders* que permitem uma alta precisão, no entanto, perante terrenos onde a aderência é baixa, apresentam erros de orientação e translação, que aumentam proporcionalmente com a distância percorrida [2][3]. Outro sistema usado para localização é o GPS. Para calcular a posição do recetor, são necessários no mínimo 3 satélites para fazer uma triangulação do sinal. Contudo, de forma a minimizar erros, são utilizados quatro, pois assim um deles serve para mitigar o erro que ocorre devido ao tempo necessário na comunicação entre os satélites e o recetor. Como o GPS permite calcular onde o utilizador se encontra, cruzando os dados com mapas anteriormente cartografados, é possível calcular uma rota que permite ir do ponto A ao ponto B. Uma grande vantagem deste sistema é o facto de o satélite e o recetor estarem sempre em comunicação, logo, o erro não acumula ao longo do tempo [4][5][6]. Em ambientes onde existem materiais ou objetos físicos (edifícios, árvores, entre outros), o sinal pode

tornar-se fraco levando a erros de cálculo. Em alguns casos o sinal pode ser completamente perdido, sendo assim impossível calcular a posição. No interior de edifícios, cavernas, túneis, entre outros, torna-se assim inviável utilizar este sistema.

Também foram desenvolvidos sistemas de localização que utilizam ultrassons e sonares. Para conseguirem calcular distâncias, estes equipamentos têm um emissor e um recetor. Medindo o tempo entre as ondas emitidas e a sua receção, é possível calcular a distância do objeto ao obstáculo. David Ribas et al. propuseram um sistema SLAM utilizando apenas sonares em ambientes subaquáticos. Com o método utilizado conseguem cartografar esses ambientes [7]. Sanchez et al. desenvolveram um sistema de auto-localização para interior de edifícios conseguindo uma precisão média de 3.3 cm [8]. Embora os sensores apresentem boa precisão e sejam economicamente viáveis, demonstram algumas limitações, entre as quais, a orientação e a textura do objeto/superfície onde as ondas do emissor são refletidas, podendo, em alguns casos, fazer com que estas nunca cheguem ao recetor. Além disso, como são sensores sensíveis ao ruído, se existirem na mesma área ondas sonoras em frequências similares, os sinais entrarão em conflito. Sensores laser também são muito utilizados para calcular a posição. Existem duas técnicas de medir a distância: uma delas assimila-se à técnica utilizada com sonares, onde se calcula o tempo entre a emissão e a receção de um pulso laser (LIDAR); a outra técnica baseia-se em sistemas de mudança de fase onde, um sinal contínuo é transmitido e a fase do sinal que retorna é comparada com a sua fonte e a distância ao objeto/obstáculo é medida utilizando a mudança de Doppler [9] [10]. O LIDAR é muito utilizado na deteção e desvio de obstáculos, mapeamento e localização, pois apresenta uma grande precisão. Lingemann et al. desenvolveram um sistema que utiliza dois sensores laser com filtros lineares integrados em simultâneo demonstrando uma grande precisão. Slowak e Kaniewski desenvolveram um sistema SLAM que utiliza um filtro de Kalman em conjunto com dados recebidos de um LIDAR. De seguida, aplicam técnicas de forma e agrupamento para desenvolver o mapa [12]. Métodos de *machine learning* também foram aplicados em conjunto com LIDAR, como foi o caso de Ramezani et al. que apresentaram um sistema SLAM utilizando um LIDAR e um sistema *deep-learning* para calcular o *loop-closure*. Este último, baseia-se em características do ambiente e permite melhorar a precisão do sistema e reduzir o erro que se vai acumulando, conhecido na literatura como *Drift* [13]. Embora utilizando LIDAR seja possível obter grandes precisões, uma grande desvantagem é ser uma solução bastante dispendiosa quando comparada com outros sensores. Além disso, o custo computacional da análise dos dados é alto, o que pode ser um entrave para alguns sistemas.

2.2.2 Fusão de sensores

Este método permite compensar as fraquezas de determinado sensor com outro, tornando assim o sistema mais robusto. Por exemplo, Inertial Navigation System (INS) é uma técnica de posicionamento relativo que calcula a posição e a orientação de um objeto. Esta técnica utiliza acelerómetros e giroscópios para calcular, de forma contínua, a sua posição. Oliver J. Woodman realizou um estudo nesta área e concluiu que quanto maior for o tempo de operação, maior se torna o erro de posicionamento, devendo-se ao facto de o sistema não utilizar nenhum sensor que calcule a distância diretamente, mas apenas sensores que a deduzem. Por outro lado, nesse estudo concluiu-se que a

INS apresenta resultados muito melhores quando combinada com GPS ou magnetómetros [14]. A integração do GPS e INS também foi proposta por outros autores [5][6], e concluiu-se que em ambientes onde o sinal dos satélites não é estável ou não existem satélites suficientes para que seja realizada a triangulação da posição, a INS serve como complemento, pois não é computacionalmente muito dispendioso. Como a posição absoluta vai sendo atualizada constantemente, o erro que num sistema normal INS se acumularia ao longo do tempo, mantém-se baixo. Fang, Xudong Ma e Xianzhong propuseram um sistema SLAM que fundia dados de um sonar e de uma câmara que permitiu melhorar a precisão e a fiabilidade do sistema devido à precisão do sonar [15]. Scheicher et al. combinaram visão estéreo com GPS. A grande vantagem desta combinação, é que o sistema quando combinado, e devido ao posicionamento absoluto resultante do GPS, permite reduzir o erro proveniente do sistema de visão. Por outro lado, quando o sistema GPS perde o sinal, o sistema é capaz de calcular o posicionamento com precisão até o sinal voltar a ser estabelecido [16]. Outro tipo de fusão foi desenvolvido por Shin, Park e Kim recorrendo a uma câmara e um LIDAR para formar um sistema SLAM. Além de conseguirem uma precisão muito alta mesmo em ambientes de grande escala, conseguem construir mapas em medidas reais sem fazer qualquer tipo de verificação de pontos. Os testes foram avaliados utilizando a base de dados KITTI onde o erro obtido foi muito baixo para a distância percorrida [17]. Recentemente Qin et all. propuseram um sistema que serve de *framework* para a fusão de diferentes sensores. É compatível com sistemas de câmaras estéreo, sistemas monos câmara com um IMU, sistemas câmaras estéreo com um IMU e pode ainda ser alterada para acomodar outros sensores, como por exemplo, o GPS [18].

2.2.3 Odometria visual

Para uma melhor compreensão do leitor, os métodos de odometria visual foram divididos em dois grandes grupos. Os métodos geométricos, que são os métodos que não utilizam nenhum tipo de aprendizagem máquina, e os métodos não-geométricos que são os que utilizam [19].

2.2.3.1 Métodos geométricos

2.2.3.1.1 Métodos baseados em aparência

Métodos baseados em aparência também são chamados métodos diretos e utilizam informação da imagem completa. Em ambientes com baixa textura ou poucos objetos apresentam melhores resultados do que métodos baseados em características. Neste método, ou o sistema se baseia em encontrar regiões na imagem para comparação, ou em calcular o fluxo ótico (*optical flow).* Os métodos baseados numa região da imagem dependem de uma correspondência da região entre imagens utilizando *templates* ou com o alinhamento de imagens [20]. Gonzalez et al. incorporaram a técnica da bússola visual, desenvolvida por Frederic Labrosse [21], que se baseia no desvio dos pixéis nas imagens capturadas para estimar a rotação, e métodos de comparação de *templates* com um sistema de duas câmaras (uma a capturar imagens do chão e outra o ambiente) para calcular a translação [22]. Lovegrov et al. apresentaram uma ideia muito interessante ao basearem-se na textura da estrada para calcular a trajetória do veículo em movimento. A câmara utilizada era a câmara de

estacionamento traseira do veículo. Além disso, fundiram os dados com um sinal GPS, o que melhorou ainda mais a precisão do sistema [23]. A viabilidade destes métodos é altamente dependente dos argumentos utilizados na otimização dos dados, pois facilmente podem surgir erros de divergência nas estimativas. Outro grande problema deste método é a existência de objetos autónomos no ambiente, como carros ou pessoas que apresentam trajetórias independentes [24]. O fluxo ótico apresenta melhores resultados quando deparado com estes problemas. Este método foi inspirado em insetos que o usam para se conseguirem deslocar [25]. Um dos trabalhos de referência na literatura sobre *optical flow* foi a formulação desenvolvida por Horn and Schunck [26], embora tivesse algumas falhas relativas a descontinuidades no movimento ou variação da luminosidade. Campbell et al. utilizaram esta técnica dividindo os pontos consoante a sua distância à câmara. Enquanto que os pontos mais próximos da câmara são utilizados para calcular a translação, os pontos mais distantes são usados para calcular a rotação [27].

2.2.3.1.2 Métodos baseados em características

Este método baseia-se em alguns pontos de interesse (*keypoints*) que a imagem possa ter e que sejam de fácil identificação. A ideia principal passa por identificar esses pontos de interesse ao longo do tempo e calcular as suas transformações. Os pontos identificados numa determinada imagem têm de ser novamente identificados e correspondidos na imagem seguinte. Após essa correspondência, é possível calcular a diferença entre a posição dos pixéis alvo e, com isso, determinar o movimento realizado pela sua fonte. Para calcular esse movimento é necessário existir uma relação entre os pixéis da imagem e a distância física e, a esta relação, dá-se o nome de escala. Dependendo do tipo de sistema, uma ou mais câmaras, o método de obtenção desta escala varia. A obtenção dos *keypoints,* foi abordada de diferentes formas resultando assim em diferentes sistemas. Dois grandes pioneiros foram Harris e Moravec quando criaram detetores de beiras e cantos que serviram de base para muitos outros. O "Harris Corner Detector" conseguia correspondências rápidas e foi, portanto, muito utilizado [28][29]. Mais tarde, juntamente com a evolução das técnicas de escala e transformação de imagem, os métodos de deteção evoluíram. A deteção de beiras e cantos não seria assim suficiente, surgindo métodos que detetavam características "invariáveis", isto é, embora uma imagem possa ser escalada ou rodada, essa característica permaneceria detetável. Estas características invariáveis têm uma probabilidade maior de poderem ser detetadas e correspondidas, levando a sistemas com menos erros e, consequentemente, mais robustos. Alguns destes detetores são: SIFT, que foi utilizado, por exemplo, por Tardif et al. [30]; SURF, que serviu de base para o trabalho de Lee, Fraundorfer e Pollefeys [31]; Persson et al., utilizaram um detetor denominado por BRIEF [32] e Konolige et al. fizeram uso de outro detetor, o Censure [33]. Existem muitos outros detetores na literatura e existem variações de alguns já mencionados. Dada a vasta variedade de detetores, a escolha de um para um determinado projeto depende de vários fatores, como por exemplo, luminosidade, margem de erro tolerada, velocidade do sistema geral, *hardware*, entre outros. A escolha é feita consoante essas caraterísticas do sistema, ou muitas vezes é feita uma comparação entre alguns detetores sendo escolhido o que apresentar melhores resultados. Num sistema real, embora os detetores de *keypoints* sejam muito credíveis e cada vez melhores, existem sempre alguns pontos que podem estragar a amostra de pontos adquiridos e, com isso, estragar o resultado. Para resolver este problema, surgiram

alguns métodos que permitem definir uma margem de erro máxima para os conjuntos de pontos a utilizar. Se o erro ultrapassar a margem definida, esse ponto é rejeitado. Por exemplo, RANSAC, MLESAC, filtro de Kalman, entre outros, são utilizados na literatura para eliminar estes *outliers.* Uma lente de uma câmara ideal não apresenta qualquer tipo de distorção. Embora em lentes de grande qualidade o "efeito barril" seja baixo, o conhecimento dos parâmetros intrínsecos da câmara permite reverter esta distorção por *software* [34]. Embora existam métodos que não precisem de uma calibração prévia da câmara, alguns deles ainda precisam. Sistemas estéreo-câmara, na maioria dos casos precisam de uma calibração o mais precisa possível, caso contrário a estimativa da profundidade não será correta. O algoritmo 5-pontos proposto por Nister [34] é um dos casos em que é essencial uma boa calibração da câmara. Embora o algoritmo seja robusto e apresente bons resultados, uma má calibração da câmara pode tornar este algoritmo pouco fiável. Existem muitos outros métodos testados em vários sistemas, como por exemplo, o algoritmo de 2-pontos [31], o algoritmo de 7-pontos [35], o algoritmo de 8-pontos [36], entre outros. Todos estes métodos, em condições específicas, podem apresentar bons resultados, pelo que o seu uso depende do sistema.

2.2.3.1.3 Métodos híbridos

Este método combina os dois apresentados anteriormente, permitindo assim usar as vantagens de cada um deles para colmatar as desvantagens de cada um isoladamente. Existem diversos exemplos na literatura, entre os quais Oliensis e Werman que desenvolveram um sistema híbrido que utilizava características como pontos e linhas e, ainda, intensidades [37]. Scaramuzza et al. criaram um sistema que utilizava a aparência para calcular o "fecho do loop" (*loop closure*) e o movimento inicial, bem como características nas imagens para corrigir a rotação estimada [38]. Silva et al. utilizaram métodos de aparência densos juntamente com características encontradas, de forma a obter a escala com precisão, e posteriormente um filtro de Kalman para refinar os dados [39].

2.2.3.2 Métodos não-geométricos

Nos últimos anos, as técnicas de *machine learning* têm sido cada vez mais utilizadas nas mais diversas aplicações, inclusive em sistemas em tempo-real e o tema em estudo nesta dissertação não é exceção. Na realidade, essas técnicas podem ser aplicadas em qualquer passo para a formação de um sistema de odometria visual. De facto, podemos formar um sistema que utiliza apenas estas técnicas, conhecidas como modelos "*end-to-end".* Estes modelos são treinados para, perante a entrada dos dados provenientes dos sensores, a odometria ser a saída. Uma das grandes vantagens da utilização destes sistemas é que deixa de ser necessário o conhecimento dos parâmetros de calibração da câmara, o que além de eliminar alguns erros provocados por uma má calibração, também torna o sistema mais fácil de aplicar a diferentes câmaras. Tateno et al. desenvolveram um sistema SLAM completo utilizando apenas uma câmara. Para calcular a escala é utilizada uma rede neuronal convolucional (CNN) e a fusão dos dados é feita de forma direta. Além de obterem resultados com um erro baixo, conseguiram superar a maior limitação de sistemas mono câmara, que é a recuperação da escala [40]. Yan Loo et al. desenvolveram um sistema parecido com o anterior e utilizaram o facto de terem uma estimativa muito aproximada da profundidade para reduzir a incerteza, construindo, assim, o mapa de forma muito mais rápida [41]. Zhan et al. utilizaram uma conjugação de métodos,

tendo por base características da imagem e duas redes neuronais convolucionais, uma para calcular o fluxo ótico e outra para calcular a profundidade, conseguindo assim obter um sistema SLAM com uma margem de erro baixa [42]. Liu et al. propuseram um sistema "*end-to-end* siamese convolutional neural network for loop closure detection" para detetar um *loop-closure* em sistemas SLAM. Embora isto não seja um sistema para calcular a odometria, é um sistema importante para SLAM pois permite identificar zonas previamente identificadas que, por sua vez, reduzem o *drif* [43]. Zhao et al. propuseram um sistema *end-to-end* para prever a posição relativa conjugando uma rede neuronal para prever a profundidade e outra para prever o fluxo ótico. Posteriormente, pegam no conjunto dos dados para reproduzir o ambiente [44].

2.2.4 RGB-D

Como irá ser utilizada uma câmara RGB-D neste projeto, foi colocada num subcapítulo de forma isolada. Na realidade, estas câmaras são uma fusão de sensores, uma câmara RGB e um sensor de profundidade. Existe uma grande variedade de trabalhos apresentados na literatura sobre esta câmara. A sua grande vantagem é permitir obter, ao mesmo tempo, imagens RGB e imagens da profundidade. Na imagem da profundidade, cada pixel contém um valor que define a distância do obstáculo à câmara. Fiala e Ufkes propuseram um sistema de odometria visual utilizando as imagens 3D provenientes de uma dessas câmaras, embora o sistema seja capaz de prever a trajetória, apresenta erros de precisão [45]. Yang et al. testaram a possibilidade de construir um sistema SLAM utilizando mais do que uma câmara RGB-D. Devido a limitações de *hardware*, apenas conseguiram utilizar no sistema duas câmaras. Ainda assim, conseguiram construir um sistema SLAM completo [46]. Whelan et all. propuseram um sistema para melhorar a odometria visual utilizando este tipo de câmaras e construir um mapa a cores e com boa precisão. Este sistema permite criar mapas com um grande nível de detalhe [47]. Cheng Zhao et all, desenvolveram um sistema SLAM que utiliza simultaneamente redes neuronais *Pixel-Voxel* para criar um mapa semântico dos objetos captados no ambiente [48].

2.3 Construção do mapa

Os subcapítulos anteriores abordaram diferentes formas de obter informações de câmaras e utilizá-las para a recriação do caminho seguido por elas. Aqui serão abordadas diferentes formas de criar um mapa utilizando diferentes técnicas e métodos. Um método chamado fotogrametria (*photogrammetry*), de forma geral, pega num conjunto de imagens provenientes de uma câmara e agrupa-as de forma a criar um modelo 3D de um determinado objeto ou ambiente. Este método é muito utilizado por satélites para cartografar determinadas áreas ou utilizado para reconstruir digitalmente todo o tipo de objetos e edifícios [49][50]. Outros métodos de construção de um mapa utilizam um conjunto de pontos denominados de nuvens de pontos, e colocam cada um deles na sua posição (x,y,z) no espaço virtual, criando assim um mapa. Por exemplo, um LIDAR envia uma nuvem de pontos com a informação sobre a distância dos obstáculos (Figura 1). Dentro dos métodos que utilizam nuvens de pontos existem muitas variações que procuram melhorar a qualidade dos dados. Algumas aplicam filtros, outras reduzem a quantidade de pontos para tornar o sistema mais rápido, e

outras aplicam ambas. Por exemplo, Benjamin e Thomas desenvolveram um sistema que utiliza uma nuvem de pontos com informações da escala, conseguindo reproduzir um ambiente 3D com grande precisão e detalhe devido a utilizarem o maior número de pontos que conseguirem. Além disso, utilizam filtros e métodos de otimização para preencher espaços que possam não ter informação [51]. Outro algoritmo é a grade de ocupação, utilizado quando os dados provenientes dos sensores têm algum ruído ou incerteza. Para utilizar este tipo de mapeamento, a posição do robô tem de ser conhecida. Este algoritmo acaba por ser um método de otimização que realiza estimativas para a posição dos pontos, melhorando assim a qualidade do mapa. O *octree* é uma estrutura de dados que utiliza uma hierarquia. Cada ponto principal é chamado de nó, em que cada um ou é composto por oito filhos ou nenhum. O nó principal cria uma caixa que engloba todos os oito pontos. Esta associação hierárquica permite fazer uma grande redução do número de pontos na nuvem de pontos. Computacionalmente tem a grande vantagem de tornar o sistema substancialmente mais rápido que muitos outros métodos, embora parte do detalhe seja perdido.

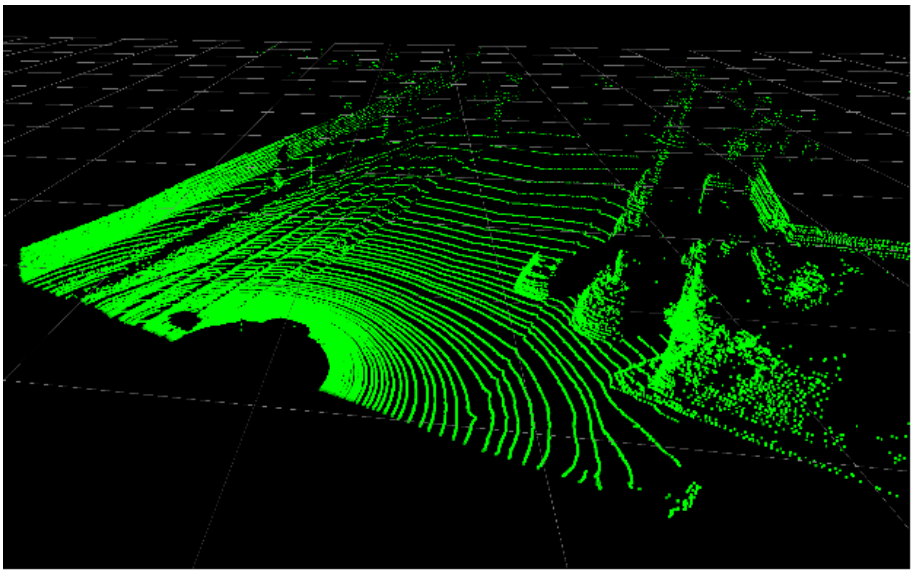


Figura 1-Representação de uma nuvem de pontos proveniente de um LIDAR

2.4 Limitações da visão por computador

Todos os equipamentos têm vantagens e desvantagens. A visão por computador não é exceção, sendo as principais desvantagens o custo computacional, o excesso ou falta de iluminação e a falta de textura que alguns ambientes possam apresentar. Num ambiente ideal a iluminação seria constante e o ambiente apresentaria texturas que permitiriam uma boa identificação dos *keypoints*. Em alguns ambientes, o chão e paredes apresentam pouca textura e têm poucos ou nenhum objeto. Nesses casos, a estimativa da posição torna-se ainda mais difícil (Figura 2). Em ambientes exteriores podem ocorrer mudanças repentinas na iluminação devido ao sol, nuvens, sombras, entre outros, e em ambientes interiores, embora mais fáceis de controlar, podem existir zonas que apresentem diferentes exposições solares. Se essa diferença for grande o suficiente o algoritmo pode encontrar um *keypoint*

errado ou não encontrar um que deveria, levando assim a um aumento do erro na estimativa da posição. De forma a poder exemplificar este problema, desenvolveu-se um pequeno código exemplo para encontrar esquinas e cantos. O algoritmo, devido a uma grande exposição solar, assumiu que encontrou pontos de interesse numa zona onde não existiam, comprovando assim esta limitação (Figura 3). Embora a visão tenha estas limitações, os algoritmos mais recentes estão cada vez melhores e mais preparados para as compensar [2].

Figura 2-Exemplo de um ambiente com superfícies com baixas texturas e sem objetos

Figura 3-Simulação dos efeitos negativos que diferentes intensidades de luz podem provocar

Capítulo 3 – Trabalho realizado

Neste capítulo será descrito todo o trabalho realizado ao longo deste projeto. Inicialmente será apresentado o *hardware* utilizado pois tem uma implicação direta no algoritmo. De seguida, serão enunciadas, de forma resumida, as etapas que levam ao algoritmo do sistema SLAM visual proposto. Posteriormente, cada uma das etapas será descrita com maior detalhe e serão explicadas as diferentes razões que levaram à utilização dos diferentes algoritmos e plataformas no sistema final. Posteriormente será apresentado o algoritmo completo. Para finalizar este capítulo, será descrito o estudo realizado em fluxo ótico e os motivos que justificaram a sua não implementação neste sistema.

3.1 Hardware utilizado

Para alcançar o objetivo desta dissertação foi necessária uma câmara RGB e um dispositivo que consiga medir distâncias de forma a obter a escala. A Kinect 360 (Figura 4) foi um dispositivo criado para ser utilizado em videojogos, mas que acabou por ser utilizado em inúmeros projetos de robótica devido às suas características e potenciais aplicações. Este possui os dois requisitos necessários para o projeto e acabou por ser escolhido para o realizar. Este dispositivo é constituído por uma câmara RGB para aquisição de imagem, um emissor infravermelho (IV) e uma câmara de infravermelhos. Estes dois últimos permitem calcular a distância a um alvo medindo o tempo entre a emissão e a receção do sinal. Este emissor emite um sinal diferenciado por cada pixel, permitindo em vez de calcular a profundidade em apenas um ponto, calculá-la num conjunto de pontos correspondentes à resolução da câmara (640×480 *pixels*). Assim, temos um dispositivo (composto por duas câmaras) que permite obter imagens RGB e imagens da profundidade.

Figura 4-Kinect 360

A Kinect 360 contem as seguintes características:

- Consegue operar, no máximo, a 30 *frames* por segundo(fps);
- O alcance do sensor varia entre 0,5 metros e aproximadamente 4,5 metros;
- A 0,5 metros apresenta um erro aproximado de 1,5 milímetros e a 4,5 metros apresenta um erro aproximado de 40 milímetros.

Existem outras câmaras similares e mais recentes, como a “Kinect V2” ou a “Real Sense” da Intel. Estas últimas são versões mais recentes do que a utilizada neste projeto e têm maior precisão. Embora não fosse possível o acesso aos modelos referidos, a câmara disponível apresenta um erro relativamente baixo e, além disso, migrando para uma mais recente, o erro diminuirá. As câmaras RGB e IV fisicamente não estão no mesmo sítio o que provoca diferentes perspetivas de captura de imagem. Para resolver esta situação é necessário fazer uma calibração das imagens por software, permitindo alinhar as duas imagens e garantir que os pixéis da imagem RGB correspondam aos pixéis da imagem de profundidade. Quando esta correspondência é alcançada, é possível construir uma nuvem de pontos 3D a cores (Figura 5).

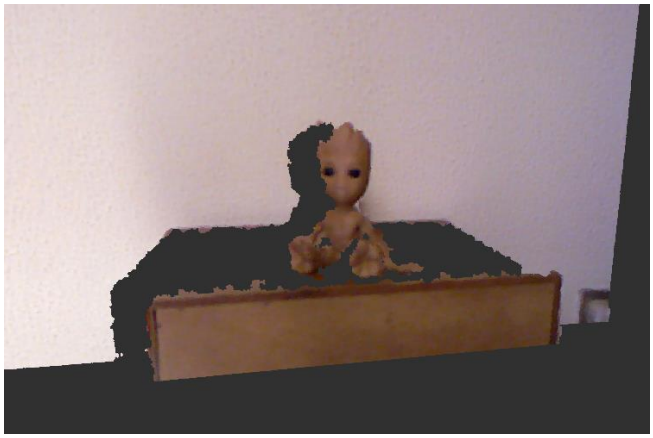


Figura 5-Nuvem de Pontos 3D com correspondência de cores

Além da Kinect 360, o hardware principal utilizado é:

- Processador: Intel core i5 8th Gen (8550)
- Placa gráfica: GeForce MX130
- 8 Gigabytes de memória RAM

3.2 Etapas do algoritmo

Este subcapítulo contém uma introdução mais simplista do algoritmo seguida de uma explicação mais completa de cada etapa. Assim, as etapas principais são:

1) Aquisição e retificação das imagens;
2) Deteção das características chave nas imagens;
3) Estimar o movimento realizado utilizando as características chave (odometria visual);
4) Estimar a escala real com um dispositivo capaz de medir distâncias;
5) Fundir a odometria visual com as informações captadas para construir um mapa 3D.

3.2.1 Aquisição e retificação das imagens

Existem bibliotecas específicas que ajudam na comunicação com este dispositivo tornando assim possível a aquisição das imagens. Assim, do ponto de vista do programador, depois de estar tudo devidamente instalado, a câmara pode ser acedida como uma câmara tradicional com a particularidade de captar, além das imagens RGB, a imagem da profundidade. Os dados recebidos do sensor de profundidade são, posteriormente, convertidos para metros. A figura 6 mostra uma imagem RGB e respetiva profundidade, ambas retiradas da Kinect.

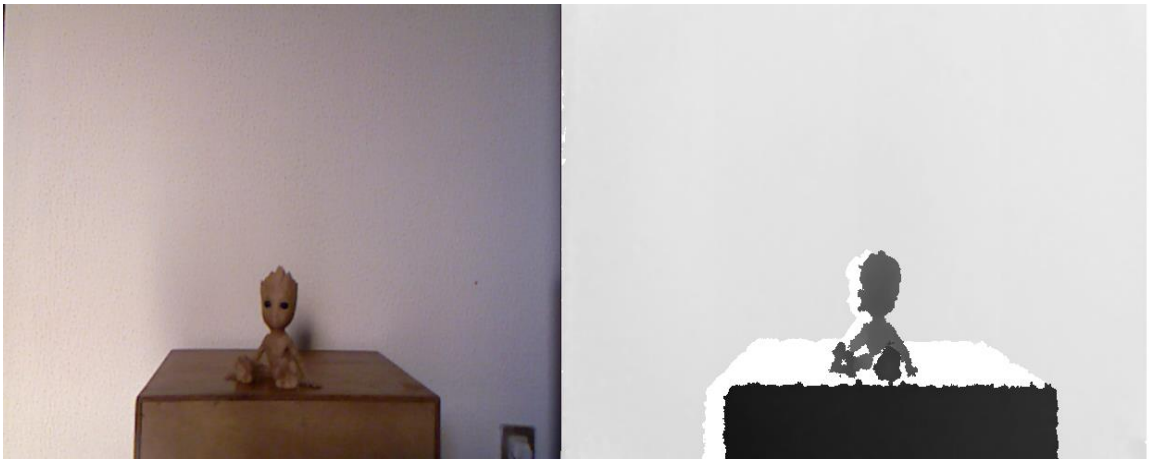


Figura 6-Informação obtida de uma Kinect: À esquerda uma imagem RGB; à direita uma imagem da profundidade representada por tons de cinzento

Tal como foi referido em 3.1, sempre que é necessário conjugar imagens de duas câmaras diferentes e consequentemente separadas fisicamente, é necessária uma calibração das duas imagens para fazer a correspondência dos pixéis das duas imagens. Além disso, as câmaras não têm lentes perfeitas. Estas introduzem uma distorção na imagem capturada e dada a natureza do projeto, minimizar os erros é essencial para obter bons resultados. Esta distorção é possível de minimizar por software com uma calibração. Em suma, a Kinect possui a distorção da lente da câmara RGB e a distorção da lente da câmara IV para compensar por software e a calibração para alinhar os pixéis da imagem a cores com os pixéis da imagem de profundidade.

Existem diferentes métodos de calibração. Neste projeto foram experimentados 3 métodos de calibração diferentes e foi escolhido o que apresenta melhores resultados. Depois de obtidos os parâmetros de calibração, estes são utilizados para retificar as imagens provenientes da Kinect. Todos os parâmetros de calibração obtidos encontram-se nos anexos A, B e C.

3.2.1.1 Calibração de fábrica

Na sua memória interna, a Kinect possui um registo da sua calibração de fábrica e que é diferente para cada câmara. Esta calibração, ao contrário do que seria de esperar, não é utilizada automaticamente quando se usa a câmara, pois a biblioteca não tem conhecimento dos seus parâmetros, sendo necessário um programa externo para aceder a essa calibração. O “anexo A”

contém informações sobre os parâmetros de calibração obtidos e um link para a página que contém informação mais detalhada do processo.

3.2.1.2 Calibração usando um tabuleiro de xadrez num vidro

O método clássico para calibração [52][53] é utilizar um tabuleiro de xadrez, embora para uma melhor calibração seja aconselhado um diferente número de colunas e linhas no tabuleiro. A ideia passa por utilizar um desses tabuleiros onde os quadrados pretos são recortados ficando o vidro transparente no seu lugar e os quadrados brancos permanecem intactos para manter a forma do tabuleiro (o branco é a cor mantida pois reflete melhor a luz). Quando se alinham duas imagens RGB normais, a identificação do tabuleiro é fácil nas duas imagens pois a cor cria um grande contraste. A particularidade de ter uma câmara de infravermelhos permite que a luz emitida atravesse o vidro sendo assim possível alinhar de forma bastante precisa a imagem RGB e a imagem de infravermelhos. Embora a imagem da profundidade possua ruido, este método permite alinhar o tabuleiro com boa precisão. Utilizando o método tradicional, a câmara iria medir o mesmo valor de profundidade para todo o tabuleiro não sendo possível diferenciar o tabuleiro. A figura 7 retrata parte do processo de calibração das duas imagens.

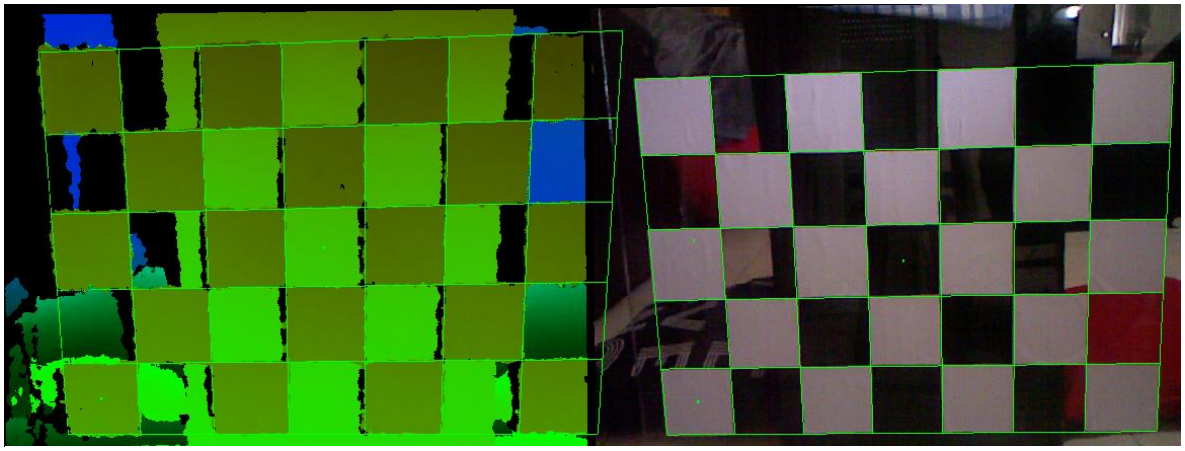


Figura 7-Calibração da Kinect utilizando um tabuleiro semitransparente: À esquerda a imagem de profundidade; à direita a imagem RGB

Com a exceção das diferenças anunciadas, o resto do processo é igual ao método tradicional. São retiradas imagens de diferentes distâncias e de diferentes ângulos. De seguida, são calculadas as transformações para alinhar as duas imagens chegando assim aos parâmetros de calibração. Estes cálculos são feitos de forma automática pelo programa utilizado. O anexo B contém informação sobre os parâmetros de calibração calculados e um link para o sítio do autor original deste método de calibração onde tem toda a informação necessária para a replicação deste processo.

3.2.1.3 Calibração utilizando ROS

Robot Operating System (ROS) é uma plataforma que contém bibliotecas e ferramentas direcionadas para robótica. Esta plataforma possui um pacote denominado "*camera_calibration*" que foi criado justamente para a calibração de câmaras. Para realizar esta calibração é necessária uma folha com um tabuleiro de xadrez impresso pois é um alvo relativamente fácil de identificar facilitando assim a

calibração. A Figura 8 mostra parte do processo de calibração. Para realizar este processo de calibração é necessário capturar imagens de diferentes distâncias e ângulos. Depois desse processo, o cálculo da matriz e coeficientes de transformação é feito de forma automática.

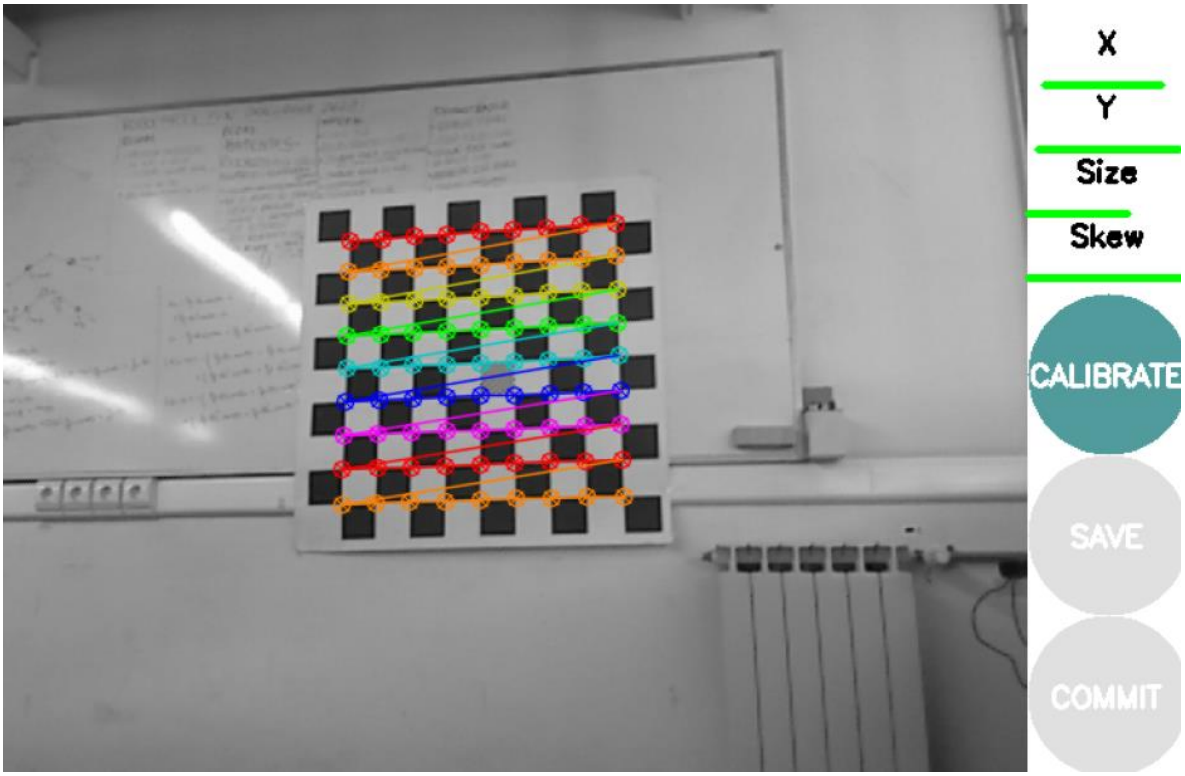


Figura 8-Calibração da Kinect utilizando ROS

É de salientar que esta calibração apenas engloba a câmara RGB, no entanto, ROS utiliza coeficientes por defeito para a câmara de IV. Existem formas de fazer uma calibração da câmara IV, no entanto, a calibração atingida usando os dados calculados da câmara RGB e os dados por defeito da câmara IV superaram as calibrações referidas anteriormente. Os coeficientes estão apresentados no anexo C.

3.2.1.4 Comparação entre as calibrações

Depois de obtidos todos os dados correspondentes às diferentes calibrações, foi utilizado ROS. ROS possui uma ferramenta que utiliza os parâmetros de calibração diretamente nas imagens, e o resultado é uma imagem retificada. Uma ferramenta de visualização chamada "rviz", também pertencente a ROS, foi utilizada para visualizar a conjugação da câmara RGB e da câmara IV, sendo que esta conjugação resulta numa nuvem de pontos 3D com correspondência de cor. Para verificar qual a melhor calibração, a Kinect foi direcionada sempre para o mesmo alvo de duas perspetivas diferentes. Foram capturadas imagens das nuvens de pontos, com a correspondência de cor, resultantes dessas perspetivas. Uma boa calibração faz com que a cor atribuída a cada ponto da nuvem de pontos seja muito próxima da cor real, tornando-se percetíveis os cantos e bordas de objetos. Na prática significa que os pixéis das duas câmaras foram devidamente alinhados. Este alinhamento é importante pois significa que quando encontramos um ponto chave em determinado pixel da imagem RGB da Kinect,

podemos consultar a sua distância no mesmo pixel da câmara IV que sem uma boa calibração não seria possível. A Figura 9 corresponde à calibração de fábrica, a Figura 10 corresponde à calibração realizada com o tabuleiro semitransparente e a Figura 11 corresponde à calibração realizada utilizando a ferramenta anteriormente referida, "camera_calibration", da plataforma ROS. A escolha da calibração foi feita com base na análise direta das imagens. Analisando as imagens, é claro que a calibração que apresentou melhor resultados foi a que utilizou a ferramenta do ROS. A calibração de fábrica apresenta um erro grande e incompatível com os requisitos deste sistema. Isto não significa que a calibração de fábrica seja má, apenas que o fabricante se preocupou em tornar a Kinect apta para videojogos que não precisavam de uma precisão muito grande. A segunda calibração testada apresentou melhores resultados quando comparada com a primeira. Embora ainda apresente alguns erros nas bordas já é possível identificar formas facilmente. O terceiro método, a calibração utilizando ROS, foi a que apresentou melhores resultados. Todas as bordas e arestas estão bem definidas e são facilmente identificadas. O facto de não ser uma correspondência perfeita introduzirá erros no sistema, no entanto é uma calibração boa o suficiente para que as estimativas da distância possam ser utilizadas. Perante estes resultados, a escolha da calibração recaiu nesta última opção. Com uma calibração mais detalhada da câmara IV e depois uma melhor calibração do conjunto das câmaras, o resultado poria ser ainda melhor. No entanto, o incremento na qualidade dos resultados seria pouco para o tempo que seria necessário investir. Quanto melhor for a calibração melhores serão os resultados no sistema final, o que, para extrair o potencial máximo do sistema minimizando erros, torna uma calibração mais precisa parte do trabalho futuro a realizar.

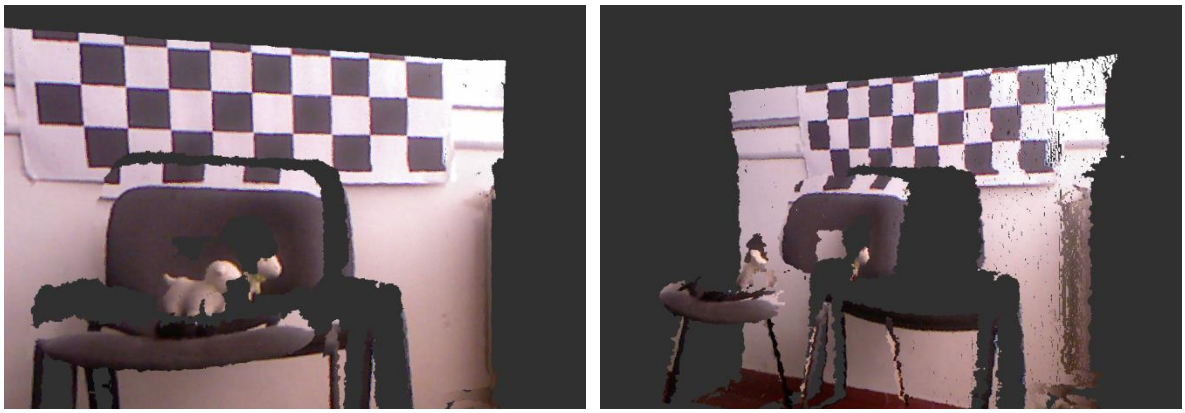


Figura 9-Calibração com os dados de fábrica: À esquerda perspetiva 1; à direita perspetiva 2

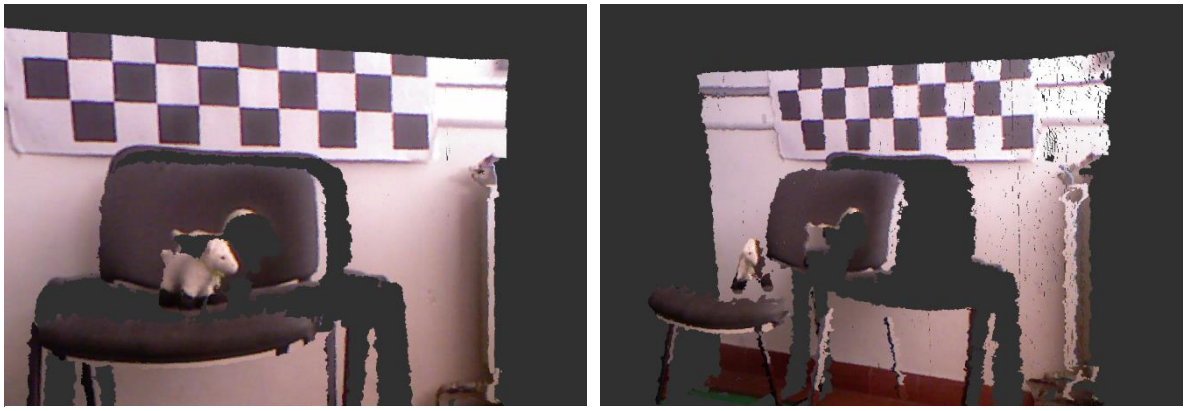


Figura 10-Calibração com os dados do tabuleiro semitransparente: À esquerda perspetiva 1; à direita perspetiva 2

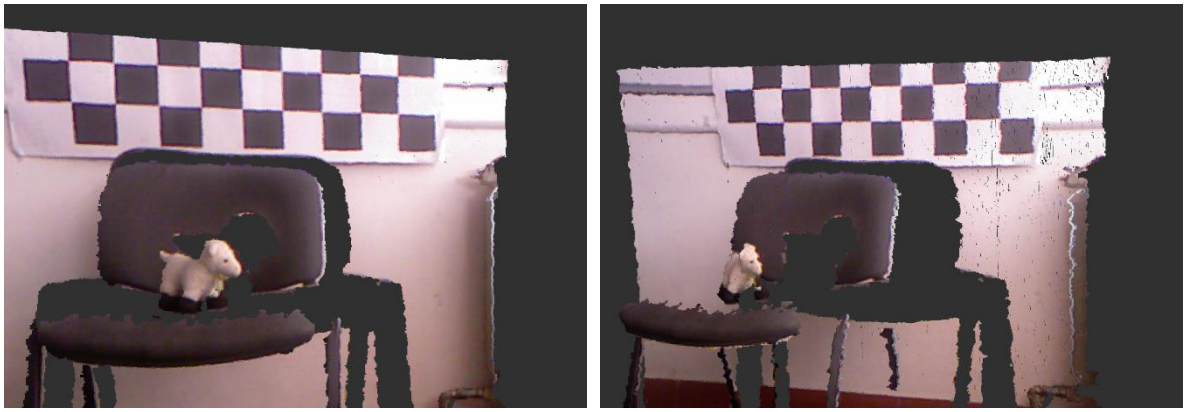


Figura 11-Calibração com os dados da calibração utilizando ROS: À esquerda perspetiva 1; à direita perspetiva 2

3.2.2 Deteção das características chave nas imagens

Para a realização deste projeto é necessário um algoritmo que detete pontos de interesse. Estes pontos são conhecidos na literatura como *keypoints*. Para conseguirmos calcular a trajetória percorrida é necessário quantificar o movimento que ocorreu entre frames. Teoricamente, poderíamos usar toda a informação contida numa imagem, por outras palavras, todos os pixéis. Isto implicaria encontrar o maior número de pontos possível na *frame* seguinte e perceber a transformação ocorrida em cada pixel para finalmente estimar o movimento. No entanto, quando a câmara se move numa direção, esse movimento traduz-se de uma forma semelhante para todos os pixéis. Assim, encontrar alguns pontos de interesse que são de fácil identificação permite poupar recursos computacionais mantendo uma boa estimativa do movimento. É importante utilizar um algoritmo que encontre esses pontos rapidamente e de forma consistente. Na literatura existem diversas opções que cumprem estes requisitos. O algoritmo utilizado foi o FAST pois é computacionalmente muito eficiente quando comparado com muitos métodos populares já referidos no estado da arte deste documento. O algoritmo funciona de uma forma simples, onde, se existir um pixel candidato a ser um canto, é desenhado à sua volta um círculo de raio manipulável. Se for realizada uma comparação entre esse círculo e um relógio, inicialmente são verificadas as intensidades dos pixéis na periferia desse círculo correspondendo à hora 3,6,9,12. Sendo um canto, três desses pixéis terão de ter uma intensidade maior que o quarto canto por um fator μ. Esta é a primeira condição do algoritmo para eliminar a maioria dos pontos candidatos. De seguida é verificada a existência de um conjunto de pontos contínuos com intensidade superior ao fator μ e de outro conjunto de pontos contínuos com intensidade inferior ao mesmo fator. Se estas condições forem verificadas esse ponto é considerado um canto. Para uma explicação mais detalhada é aconselhada a leitura do documento original [54].

3.2.3 Estimativa da trajetória percorrida

O cálculo da estimativa da trajetória percorrida, a odometria visual, é uma tarefa complexa. Esta tarefa torna-se ainda mais difícil quando se tenta usar apenas uma câmara. Embora a Kinect possua uma câmara RGB e uma câmara IV, o cálculo da odometria será feito usando apenas a câmara de cores, sendo a segunda usada apenas para a recuperação da escala. Este método foi utilizado pois torna mais fácil uma futura implementação de novos sensores no projeto. Este tipo de odometria é chamada de odometria monocular. Para ser possível estimar o movimento neste tipo de odometria, é necessário usar duas *frames* consecutivas. Na primeira *frame,* a *frameT,* serão identificados os *keypoints* utilizando o algoritmo anteriormente descrito. Depois disso, na *frameT+1,* esses pontos têm de ser novamente identificados para ser possível calcular as transformações ocorridas e estimar o movimento. Esta sequência de passos é aplicada em cada par de imagens consecutivas, tornando possível a estimativa da trajetória. O maior problema deste tipo de odometria é estimar a escala. Como são imagens isoladas onde apenas existe informação 2D, não é possível fazer nenhuma triangulação para calcular a profundidade nos *keypoints.* Como a Kinect possui um sensor de profundidade, torna-se possível compensar a desvantagem deste sistema. Para estimar a odometria utilizando este método, foram necessários vários passos que serão descritos de seguida pela ordem com que foram realizados.

3.2.3.1 Reidentificação dos *keypoints*

Depois de encontrados os *keypoints* na *frameT* estes têm de ser novamente identificados na *frameT+1*. Tomasi e Kanade propuseram um sistema que identifica estes pontos-chave de uma *frame* para a seguinte. Este método foi concebido com base no identificador de características Lucas e Kanade e é conhecido como "Kanade-Lucas-Tomasi *feature tracker"* [55]. Este foi o método usado para seguir, de *frame* em *frame*, a posição dos pixéis alvo. A biblioteca OpenCV, muito utilizada em visão por computador, possui uma função em Python que aplica este algoritmo, o qual usa uma metodologia de fluxo ótico para estimar o movimento obtido. Os pontos encontrados na *frameT* são os pontos que serão seguidos até que o seu número fique demasiado baixo. Quando isso acontece, uma nova procura de pontos é despoletada. O Código 1 representa a função utilizada bem como os seus parâmetros.

```
pontos_frameT+1 = cv2.calcOpticalFlowPyrLK(frameT, frameT+1, pontos_frameT, None, lk_params)

 lk_params =dict(winSize = (21,21),

                 criteria = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 30, 0.01))
```

Código 1 – Função de OpenCV responsável pela reidentificação dos pontos e os seus parâmetros

3.2.3.2 Verificação dos pontos encontrados

Devido ao movimento, é normal parte dos pontos ficar fora do campo de visão da câmara e, portanto, deixa de ser possível encontrá-los. Os pontos que ficam fora da imagem são retirados pois não podem ser usados. Mais tarde para estimar a escala, os pontos terão de passar por uma seleção mais detalhada que será devidamente explicada.

3.2.3.3 Estimativa da matriz essencial

A matriz essencial relaciona a transformação entre os pontos da primeira e segunda *frame.*. A equação 1 mostra a definição matemática desta matriz. Nessa equação, $\boldsymbol{y^1}$ e $\boldsymbol{y^2}$ correspondem aos pontos comuns entre a primeira e a segunda frame. **E** é a matriz essencial e relaciona a posição dos pontos na primeira e segunda *frame*.

$$(\mathbf{y^1})^{\mathbf{T}}\,\mathbf{E}\,\mathbf{y^2} = \mathbf{0} \qquad (1)$$

Esta matriz é importante, pois quando as imagens estão calibradas permite determinar a translação e rotação relativa entre os pontos calculados. Esta translação é relativa pois as imagens apenas têm informação em duas dimensões. A terceira dimensão será obtida na estimativa da escala.

Existem diversos algoritmos para calcular a matriz essencial. Neste trabalho foi utilizado o algoritmo de 5-pontos de Nister já referido no estado da arte [34]. Este método apresenta cinco graus de liberdade, três graus de rotação e dois vetores unitários de translação. Quando comparado com outros métodos como algoritmo de 7 pontos e o algoritmo de 8 pontos que apresentam mais graus de liberdade, é menos complexo. Este algoritmo exige uma calibração da câmara para ser possível obter bons resultados.

Para melhorar a estimativa da matriz essencial, antes de a calcular, os pontos são verificados. Foram feitos testes de duas formas distintas:

1)RANSAC

RANSAC [56] é um método que permite verificar se o movimento de determinado ponto é muito diferente da maioria dos pontos. Por outras palavras, é um método que permite definir uma margem a partir da qual determinado ponto será retirado da amostra. Torna possível mitigar erros que seriam introduzidos por uma má amostra tornando o sistema mais robusto Este método precisa de alguma afinação de parâmetros para chegar ao seu máximo potencial, sendo esta afinação feita por tentativa e erro. A função de OpenCV utilizada e os seus parâmetros estão representados no Código 2.

```
E = cv2.findEssentialMat(pontos_frameT+1, pontos_frameT, focal_length, principalpoint, cv2.RANSAC, prob, treshold, None)

focal_lenght = 515,6;

principalpoint = (330.27965,259.16179);

prob = 0,99;

treshold = 0,3;
```

Código 2 – Função de OpenCV responsável pelo cálculo da matriz essencial(E) utilizando o método RANSAC e os seus parâmetros

2)LMedS

LMedS [57] é um método que tal como o RANSAC tem como objetivo reduzir o erro provocado por más amostras. Em vez de retirar os pontos que apresentam um valor muito diferente dos restantes, é aplicado um método de minimização não linear. Este método calcula a função que define o melhor ajuste para todo o conjunto de pontos tentando minimizar a soma dos quadrados das diferenças. A vantagem deste método é não precisar de afinação pois o valor calculado já é com base no mínimo erro possível. A função de OpenCV utilizada e os seus parâmetros estão representados no Código 3.

```
E = cv2.findEssentialMat(pontos_frameT+1, pontos_frameT, focal_length, principalpoint, cv2.LMEDS, prob, treshold, None)

focal_lenght = 515,6;

principalpoint = (330.27965,259.16179);

prob = 0,99;

treshold = 1;
```

Código 3 – Função de OpenCV responsável pelo cálculo da matriz essencial(E) utilizando o método LMEDS e os seus parâmetros

LMeds e RANSAC apresentaram resultados semelhantes. Este resultado é muito comum e mencionado diversas vezes na literatura. Dele podemos retirar pelo menos duas conclusões, a primeira é que os dados utilizados apresentam comportamentos semelhantes entre eles, a segunda é que os dados não apresentam muitos *outsiders*. Perante estes resultados, para mitigar erros introduzidos por uma má afinação do método RANSAC, o método LMedS foi utilizado para os restantes testes.

3.2.3.4 Estimar a rotação e translação através da matriz essencial

Em cada iteração uma nova matriz essencial é calculada. Essa matriz traduz as alterações ocorridas entre as duas *frames.* A matriz de rotação (R) e a matriz de translação (t) traduzem a orientação e a posição, respetivamente, de determinado objeto. Para tornar possível a construção de um mapa coeso é necessário ter acesso a estas matrizes. Existe uma relação entre as matrizes R e t e a matriz essencial [58]. A equação (2) traduz a relação principal entre essas matrizes.

$$\boldsymbol{E = R[t]_x} \tag{2}$$

```
R,t = cv2.recoverPose(E, pontos_frameT+1, pontos_frameT, R_total,t_total, focal_length, principalpoint, None)
E = Matriz Essencia;
focal_lenght = 515,6;
principalpoint = (330.27965,259.16179;
```

Código 2 – Função de OpenCV responsável pelo cálculo das matrizes de rotação(R) e translação(t) e os seus parâmetros

3.2.3.5 Construção da trajetória

Entre duas *frames*, conseguimos obter uma estimativa para a rotação e translação ocorridas. No entanto é preciso conjugar todas essas rotações e translações ao longo do tempo para construir toda a trajetória percorrida. As equações abaixo descrevem a forma como os valores da posição e orientação são modificados.

$$\mathbf{R_{total}} = \mathbf{R_{T \rightarrow T+1}} \mathbf{R_{total}} \tag{3}$$

$$\mathbf{t} = \boldsymbol{escala} * [\mathbf{t}]_{\mathbf{x}} \tag{4}$$

$$\mathbf{t_{total}} = \mathbf{t_{total}} + \mathbf{t} \mathbf{R_{total}} \tag{5}$$

Depois de analisar as equações conseguimos perceber que a rotação é alterada em primeiro lugar. Além disso, para calcular a orientação atual, a rotação estimada entre as duas *frames* é aplicada na orientação global. O cálculo da posição até esse instante é resultado da posição anterior somada com o produto da translação estimada, com a rotação multiplicada pela rotação total. A matriz $[\mathbf{t}]_{\mathbf{x}}$ apresenta valores na direção do movimento numa escala unitária. Posto isto, é necessário multiplicar essa matriz pela escala real. Por exemplo, se existir um movimento apenas na direção do eixo do x, o valor de x na matriz será 1 e em y e z será 0. Se a escala entre as frames calculadas for 0,15 metros isso resultará numa translação de 0,15 metros em x e 0 metros em y e z. A obtenção da escala será descrita posteriormente neste documento.

Teoricamente existe mais do que uma possível solução para a matriz **R**. Por esse motivo é aplicada heurística no algoritmo. Se a rotação final calculada resultar numa diferença maior do que 20 graus entre *frames*, essa estimativa é descartada. Isto torna o sistema mais robusto pois um possível erro

de cálculo pode ser ignorado nesta verificação. A maior parte dos erros de estimativa ocorre quando a câmara se encontra parada. Assim, o erro causado pela perda de uma iteração é muito baixo.

3.2.4 Estimativa da escala

Até esta etapa todo o algoritmo usado apenas utilizou a câmara RGB. A necessidade da recuperação da escala métrica do ambiente levou ao uso da Kinect. Como já explicado, a câmara IV permite retirar uma imagem da profundidade do ambiente. Portanto, da mesma forma que são necessárias a *frameT* e a *frameT+1* da câmara RGB também são necessárias as mesmas *frames* da câmara de profundidade. Assim em cada iteração, estamos a trabalhar com 4 *frames* distintas, duas RGB e duas de profundidade. Como explicado anteriormente, em cada iteração existem pontos que são comuns à *frameT* e à *frameT+1.* São esses mesmos pontos que permitem reconstruir a trajetória percorrida. Para calcular a escala o primeiro passo é, para cada um dos pontos da *frameT RGB*, encontrar a sua profundidade na *frameT* da câmara IV (câmara de profundidade). O processo é repetido para a *frameT+1.* No final desse processo terminamos com dois conjuntos de pontos. Eles contêm informações sobre a profundidade de pontos conhecidos em instantes diferentes. Esta verificação da profundidade nos mesmos pontos só é possível porque a câmara foi calibrada. Na prática, e como já foi referido anteriormente, devido à calibração e à separação física da câmara RGB e da câmara IV, parte da informação útil das imagens é perdida quando as imagens são alinhadas. A Figura 12 mostra a "zona morta" da imagem de profundidade criada pelos fatores mencionados. Nesta figura podemos verificar que parte da informação visível na imagem RGB não está presente na imagem de profundidade.

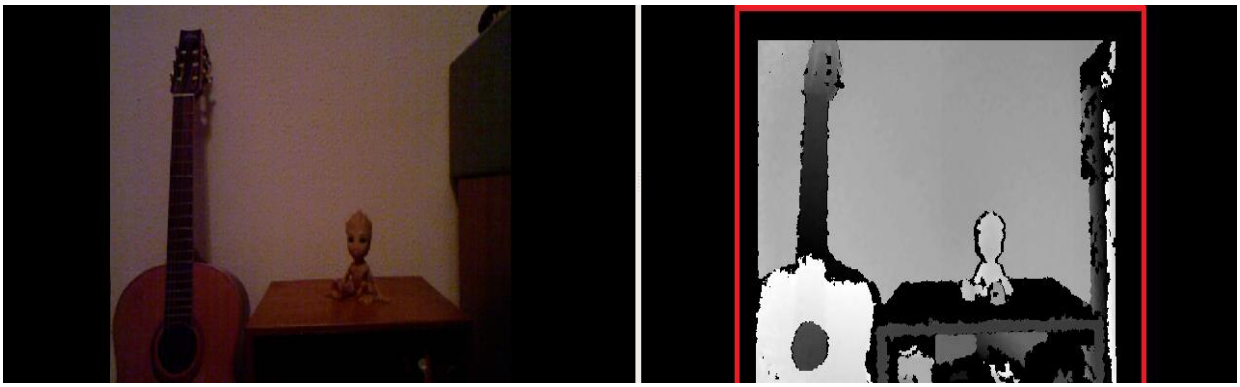


Figura 12-Demonstração da informação perdida na imagem de profundidade: Á esquerda a imagem RGB; à direita a imagem de profundidade com linhas vermelhas a destacar a área perdida relativamente à imagem RGB

Posto isto, alguns pontos de interesse encontrados na imagem RGB podem estar fora do alcance do sensor de profundidade. Estes dois fatores levam à necessidade da implementação de uma etapa de verificação dos pontos com a informação da profundidade. Assim, temos dois fatores que levam à eliminação de pontos, o primeiro é os pontos não terem informação da profundidade, seja por estarem numa zona morta ou por estarem demasiado próximos da câmara. O segundo fator que leva à eliminação de um determinado ponto é a distância. Visto que com o aumento da distância a precisão

do sensor diminui, foi definido que pontos a mais de 3,5 metros de distância seriam eliminados. As Figuras 13 e 14 mostram os *keypoints existentes* antes e depois de terem sido analisados.

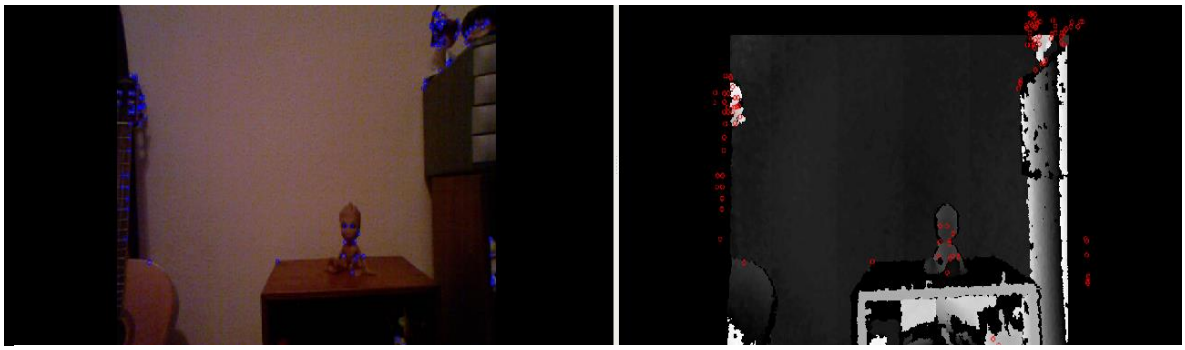


Figura 13 –Pontos-chave antes da sua verificação na imagem de profundidade

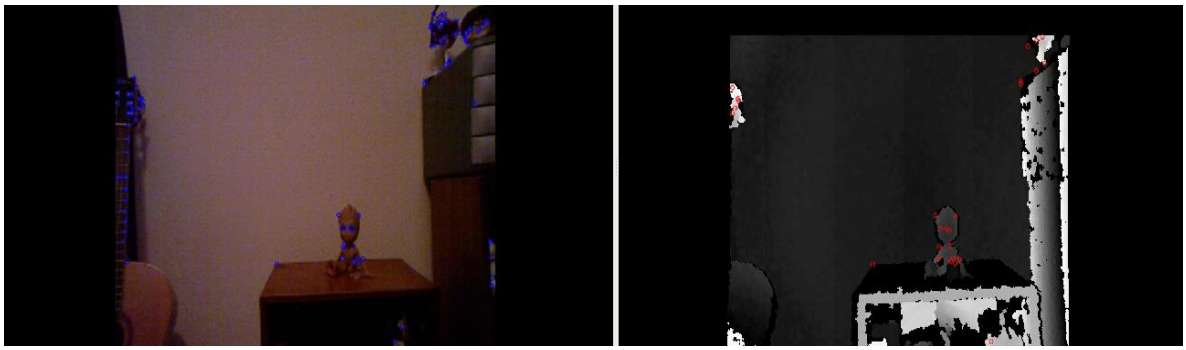


Figura 14 –Pontos-chave depois da sua verificação na imagem de profundidade

A etapa da verificação da profundidade introduz um erro no sistema. Os pontos de interesse de uma imagem podem estar em qualquer sítio dentro dela, por outras palavras, os *keypoints* encontrados podem estar entre os valores inteiros dos pixéis (valores interpolados). Isto só se torna um problema na altura do cálculo da escala. Apenas podemos verificar a profundidade em valores inteiros de pixéis, por exemplo o pixel (320,100) e não em valores intermédios como (320.5,100.3). Para verificar a profundidade, quando o valor do pixel de interesse não é inteiro, é necessário convertê-lo. Na prática não estamos a consultar a profundidade no ponto exato, mas sim ligeiramente ao lado. Esta realidade leva a um erro que com o tempo se vai acumulando. Este erro é diminuído pois a escala final calculada é obtida através da média de todos os pontos.

Os dois conjuntos de pontos têm informação sobre os mesmos pontos em diferentes instantes de tempo. Se existir movimento, a distância dos pontos à câmara irá mudar. Subtraindo a profundidade dos dois conjuntos de pontos, obtemos a distância percorrida pela câmara nesse instante. Esta distância corresponde à escala e será usada para calcular a translação que ocorreu neste instante de tempo, tal como explicado em 3.2.3.5. Caso o movimento seja puramente lateral ou puramente vertical, a distância dos pontos à câmara varia muito pouco, e esses movimentos tornam-se impercetíveis para o sistema. Esta limitação não foi solucionada no presente trabalho.

3.2.5 Construção do mapa 3D

Existem várias formas de construir um mapa para técnicas SLAM. Existem também vários programas que permitem a construção do mapa utilizando nuvens de pontos. Pretende-se que o sistema funcione em tempo real, por isso, o custo computacional destes programas é muito importante. Foram testados três programas diferentes.

3.2.5.1 PPTK

Point Processing ToolKit (PPTK) é uma biblioteca especializada na visualização de nuvens de pontos. A Figura 15 mostra uma renderização de pontos provenientes da Kinect, utilizando esta biblioteca.

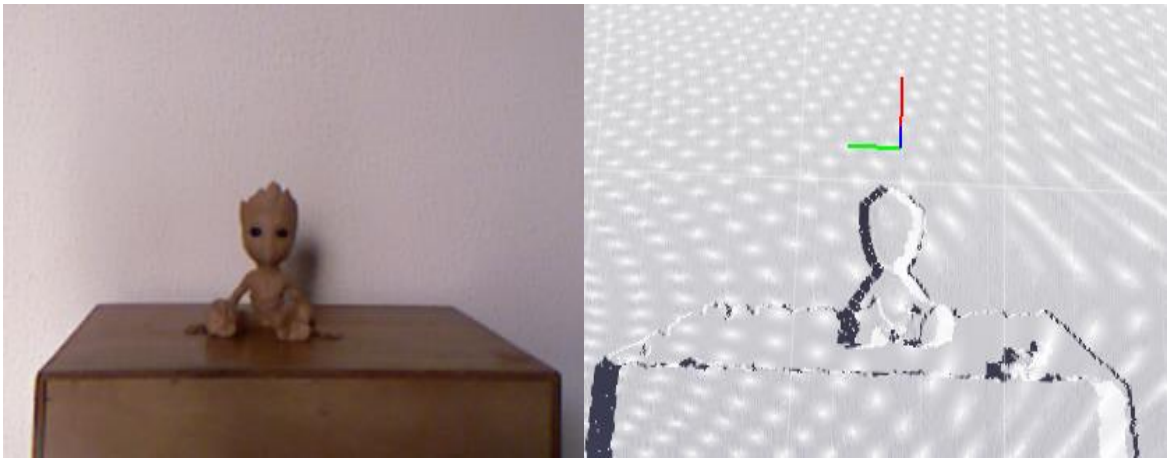


Figura 15-À esquerda: imagem RGB; à direita: nuvem de pontos no ambiente gráfico do PPTK

Esta biblioteca não foi a escolhida por vários motivos. Esta ferramenta não permite alterar a coordenada a partir do qual os pontos são renderizados. Além disso, quando é enviada uma nova nuvem de pontos, os pontos anteriores são apagados, não sendo assim possível a construção de um mapa em tempo real. Estes dois motivos impediram o seu uso no projeto.

3.2.5.2 Open3D

Estas bibliotecas contêm muito mais funcionalidades do que a anterior. A Figura 16 mostra uma renderização de uma nuvem de pontos proveniente da Kinect utilizando esta biblioteca.

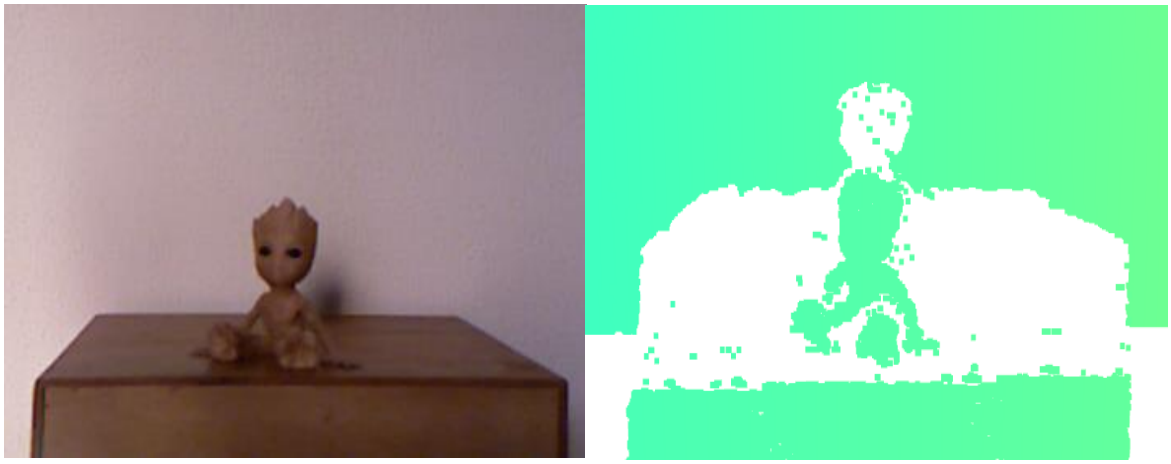


Figura 16-À esquerda: imagem RGB; à direita: nuvem de pontos no ambiente gráfico do Open3D

Esta biblioteca permite que a nuvem de pontos seja atualizada em tempo real sem que a anterior seja apagada. Existem inclusive algumas ferramentas específicas para dispositivos como a Kinect. No entanto, esse suporte só existe para modelos recentes como o *Realsense*. Para trabalhar com esta biblioteca utilizando a Kinect, seria necessário contornar muitas limitações. Isto iria contra o motivo principal do uso de uma ferramenta que é, abster o programador de objetivos secundários para se poder concentrar no essencial. Esta opção foi rejeitada, pois a terceira opção é mais fácil de utilizar e tem ferramentas especificas para a aplicação pretendida. De qualquer forma, para um sensor mais moderno, esta biblioteca tem ferramentas especificas.

3.2.5.3 ROS-Octomap

ROS já foi referido neste trabalho para fins de calibração da câmara. Tal como atrás referido, ROS é focado em robôs e tudo o que os engloba. SLAM é um problema clássico de robótica e já foram elaborados inúmeros sistemas desse tipo e aplicados a robôs. Tal como seria de esperar, existem ferramentas e pacotes específicos em ROS para ajudar a desenvolver este tipo de sistemas. Ros *vizualizer* (Rviz) é uma ferramenta de ROS para visualizar diferentes tipos de dados como nuvens de pontos, mapas de ocupação, imagens, etc. A Figura 17 retrata a visualização de uma nuvem de pontos utilizando o Rviz de ROS.

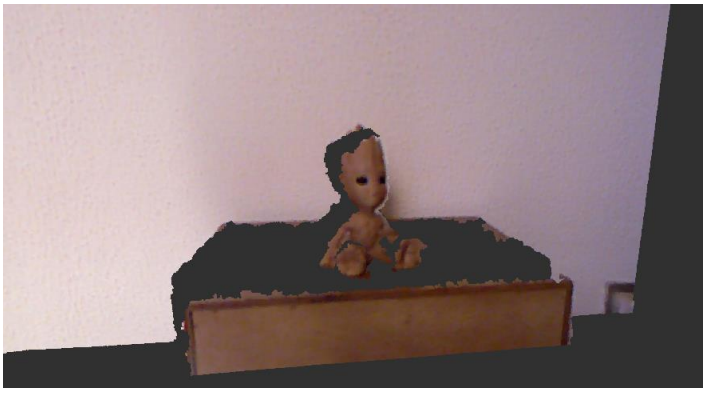


Figura 17-Nuvem de Pontos visualizada no Rviz de ROS.

Como explicado no estado da arte, existem diversas formas de se construir um mapa em 3D. O método escolhido para este trabalho é o *octomap* [59] baseado em *octrees*, o qual agrupa todos os pontos. Como resultado, parte do detalhe é perdido tornando-se computacionalmente muito mais eficiente. Para construir um mapa 3D, o detalhe obtido é o suficiente para este projeto e torna possível a operação em tempo real. ROS possui um pacote chamado "octomap_server" que permite a construção do mapa em tempo real. Este pacote depende da odometria visual pois à medida que esta é atualizada, as nuvens de pontos provenientes da Kinect (já numa escala métrica) são colocadas no ambiente no lugar a que correspondem. As Figuras 18, 19 e 20 mostram a mesma nuvem de pontos convertida para octomap vista de ângulos diferentes.

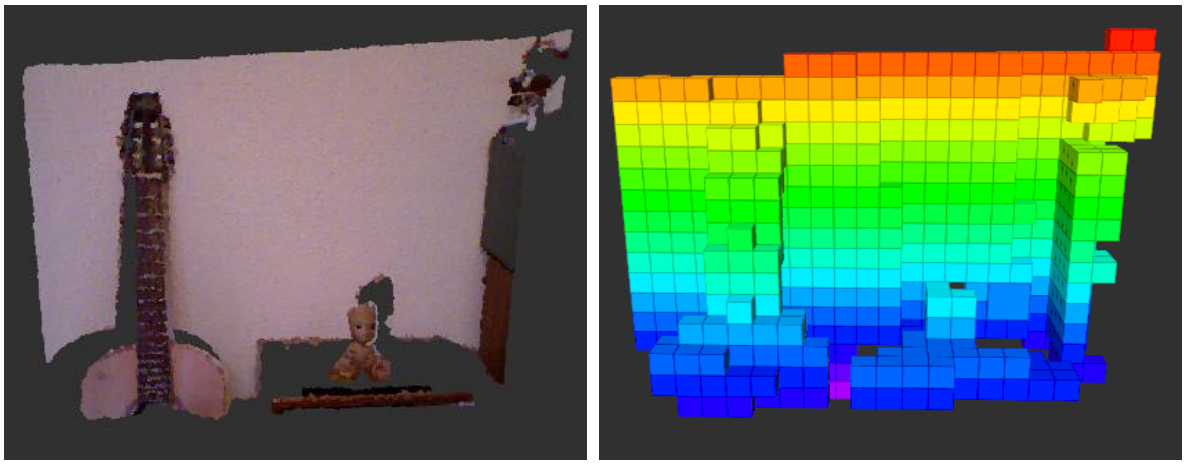


Figura 18-À esquerda uma nuvem de pontos com correspondência de cores; à direita a conversão da nuvem de pontos para octomap (vista de frente)

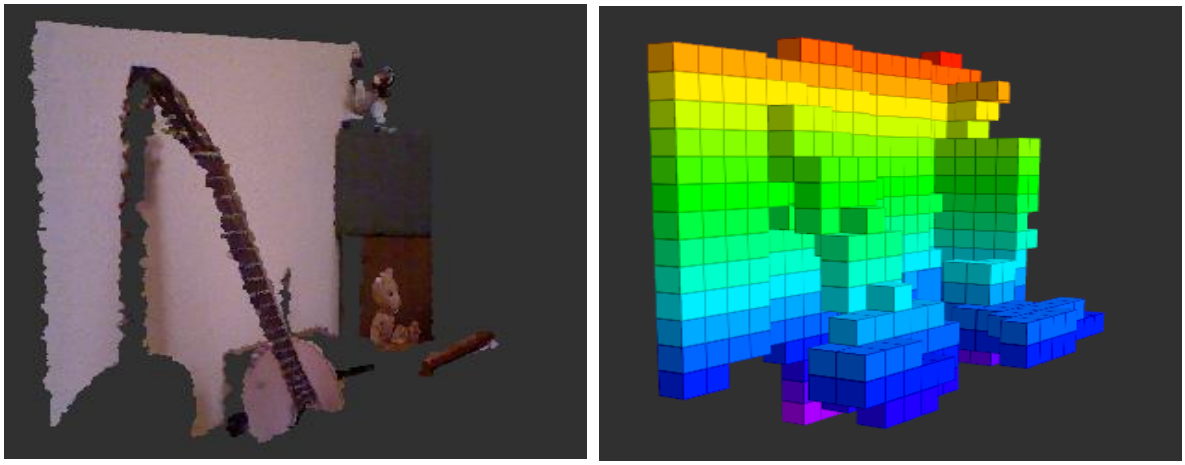


Figura 19-À esquerda uma nuvem de pontos com correspondência de cores; à direita a conversão da nuvem de pontos para octomap (vista na diagonal)

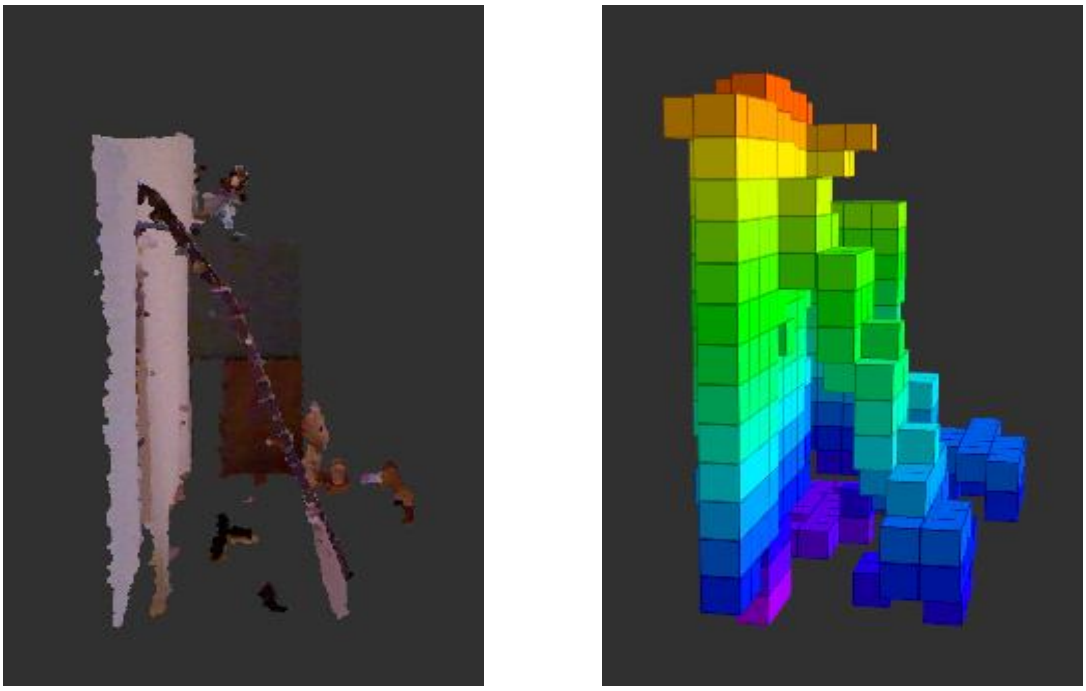


Figura 20- À esquerda uma nuvem de pontos com correspondência de cores; à direita a conversão da nuvem de pontos para octomap (vista de lado)

Observando as imagens é fácil perceber que o octomap faz com que o detalhe da nuvem de pontos seja perdido. Como já foi referido, para reconstruir plantas de habitações o detalhe obtido é suficiente, e mantendo o algoritmo o mais leve possível computacionalmente, permite abrir espaço para novas funções e funcionalidades que não seriam possíveis sem a alteração do hardware.

3.3 Algoritmo completo

Enquanto o ROS e a Kinect estiverem ativos:

- Se o número de keypoints < 150:

-Utilizar FAST para encontrar os keypoints na frameT.

-Retirar imagemT+1 RGB já retificada.

-Retirar imagemT+1 de profundidade já retificada e numa escala métrica.

-Reencontrar os keypoints na imagemT+1.

-Retirar os pontos que foram perdidos entre as imagens.

-Aplicar o algoritmo de 5 pontos de Nister em conjunto com RANSAC/LMedS para

estimar a matriz essencial.

-Recuperar a matriz de rotação (R) e a matriz de translação (t) que definem o movimento

que ocorreu entre as frames T e T+1.

-Verificar a profundidade dos keypoints da imagemT+1 RGB na imagemT+1 de profundidade.

-Verificar a profundidade dos keypoints da imagemT RGB na imagemT de profundidade.

-Percorrer os dois conjuntos de pontos com a informação da profundidade

e retirar aqueles que não têm valores definidos e aqueles que têm profundidade superior a 3.5 metros

-Calcular a escala subtraindo as profundidades dos pontos nos dois instantes

-Se R < 20 graus ()

    -Atualizar os valores da posição total da câmera utilizando (R, t e a escala)

    -Construir o mapa utilizando o Octomap (ROS), as nuvens de pontos da Kinect e a posição estimada.

    -A imagemT RGB= imagemT+1 RGB.

    -A imagemT da profundidade = a imagemT+1 da profundidade.

    -Os keypoints = (keypoints reencontrados)

Se > 20 graus:

    -Ignorar esta iteração

Algoritmo 1 – Algoritmo final

3.4 Estudo sobre fluxo ótico

Como referido no estado da arte, o fluxo ótico procura encontrar movimento entre imagens. Isto torna-o muito interessante pois não só permite uma verificação(2D) da direção do movimento como permite verificar se existem movimentos isolados, por exemplo, se existir uma pessoa, animal, objeto, entre outros, a movimentar-se numa direção diferente do movimento da câmara, é possível isolar este movimento. Esta última característica permite desenvolver sistemas mais competentes perante perturbações externas, e, portanto, mais robustos. Foram desenvolvidos diversos trabalhos relacionados com fluxo ótico sendo alguns deles já referidos neste documento. O fluxo ótico foi estudado utilizando técnicas de *machine learning* e foram alcançados excelentes resultados. FlowNet2 [60] é um trabalho sobre redes neuronais convolucionais *end-to-end* para estimar fluxo ótico. Este trabalho utiliza vários métodos que permitem estimar movimentos a grandes e pequenas velocidades. Este trabalho resultou em várias redes diferentes e o nome do trabalho corresponde à rede que apresenta a melhor estimativa do fluxo ótico, corresponde também à rede mais pesada computacionalmente entre as desenvolvidas. Embora apresente excelentes resultados quando comparada com outros trabalhos da literatura, a sua carga computacional é semelhante. LiteFlowNet [61] é outra rede neuronal convolucional *end-to-end* que estima o fluxo ótico. O objetivo desta rede é atingir uma estimativa tão boa como a de FlowNet2, mas manter o mais baixo possível a carga computacional. Tendo surgido depois do FlowNet2 e tendo usado esta como base, permitiu que a arquitetura interna da rede fosse otimizada.

Foi feito um estudo sobre estes trabalhos para verificar se eram passiveis de serem aplicados ao sistema SLAM em desenvolvimento. Estas duas redes são *opensource* e têm todo o seu trabalho disponível na plataforma *github*. A primeira etapa foi, depois de instalar todas as bibliotecas necessárias, utilizar uma das redes desenvolvidas pelo trabalho FlowNet2 com o código exemplo disponibilizado pelos desenvolvedores. Terminada esta etapa, o código exemplo foi estudado e constatou-se que poderia ser otimizado. Depois de feitas as modificações para adaptar o código a este caso de estudo, a performance melhorou significativamente. Esta otimização permitiu passar de X *fps* para Y. É importante referir que as otimizações foram apenas relativas à estrutura do código exemplo e não à rede neuronal. O código exemplo é feito para ser compatível com o maior número de cenários possíveis, sendo de esperar que fosse passível de ser otimizado em casos específicos. De modo a demonstrar a otimização do código, duas redes do trabalho FlowNet2 foram utilizadas, a rede FlowNet2-c e a rede FlowNet2-CSS. O nome destas redes deve-se ao número de camadas e métodos utilizados em cada uma. Diferentes combinações entre os dois resultam em mais *fps* e menos precisão, ou vice-versa. Utilizando o código exemplo, na rede FlowNet2-c foram obtidos 3.04 *fps* e na rede FlowNet2-CSS foram obtidos 0.40 *fps*. Depois da otimização do código foram obtidos 8.48 *fps* e 2.57 *fps* nas mesmas redes. Isto corresponde a um aumento de *fps* de 279% no primeiro caso e 642,5% *fps* no segundo caso. O aumento de rendimento entre as duas redes é distinto pois tendo uma arquitetura diferente, exigem do processador e da placa gráfica de forma especifica. O *hardware* que define o rendimento máximo para a rede *FlowNet2-c* é o processador pois esta não utiliza todo o poder de processamento da placa gráfica, enquanto que a rede FlowNet2-CSS utiliza todo o poder gráfico da

placa sendo esta que limita o rendimento. As Figuras 21 e 22 mostram os testes realizados com o código exemplo e as Figuras 23 e 24 mostram os testes realizados com o código modificado.

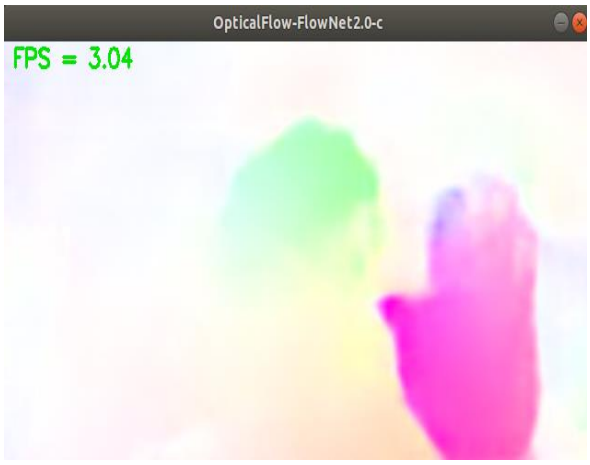


Figura 21-Rede FlowNet2-c com código exemplo

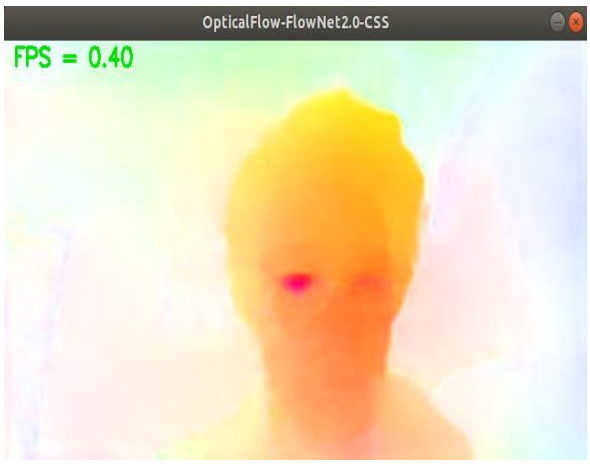


Figura 22-Rede FlowNet2-CSS com código exemplo

Figura 23-Rede FlowNet2-c com código modificado

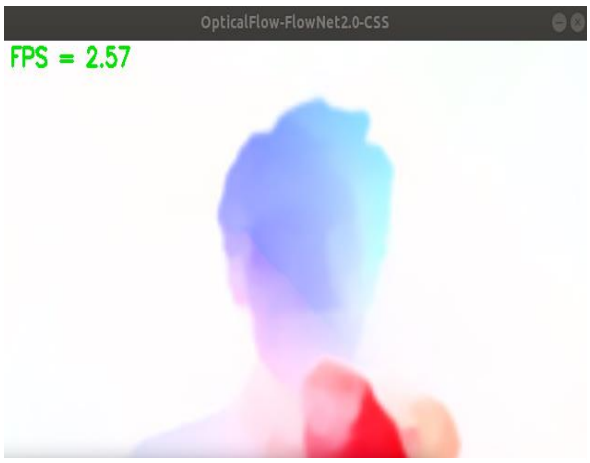


Figura 24-Rede FlowNet2-CSS com código modificado0

Todos os testes que se seguem foram realizados com o código modificado. Dentro do trabalho FlowNet2 foram testadas 13 redes distintas e além destas foi testada a rede LiteFlowNet. A tabela 1 mostra todas as redes testadas e a quantidade de *fps* máximos obtidos durante os testes.

Tabela 1-Quantidade de fps máximos obtidos para cada rede

Nome da rede	Quantidade de *fps* máximo
FlowNet2	——
LiteFlowNet	2,19
FlowNet2-css-ft-sd	2,6
FlowNet2-CSS	2,67
FlowNet2-CS	3,5
FlowNet2-C	5
FlowNet2-css-ft-sd	5,3
FlowNet2-css	5,5
FlowNet2-SD	5,7
FlowNet2-cs	6,7
FlowNet2-S	7,8
FlowNet2-ss	8
FlowNet2-c	8,5
FlowNet2-s	10,4

Analisando a tabela 1, percebemos que a rede principal FlowNet2 não tem um valor. Esta rede não foi passível de ser testada pois não existe memória gráfica dedicada suficiente para carregar os parâmetros da rede neuronal. Perante o *hardware* disponível não foi possível realizar mais testes com a rede principal, no entanto, foram testadas outras redes desse projeto. A rede LiteFlowNet foi a que apresentou o valor mais baixo de *fps*, mas o *hardware* foi capaz de suportar esta rede. Embora não seja possível retirar conclusões práticas sobre a diferença computacional entre as duas redes, a segunda foi passível de testar, o que prova que é menos pesada computacionalmente. Embora seja a rede que apresenta o menor rendimento, é das que apresenta uma melhor estimativa e com pouco ruido para baixas velocidades (Figura 25).

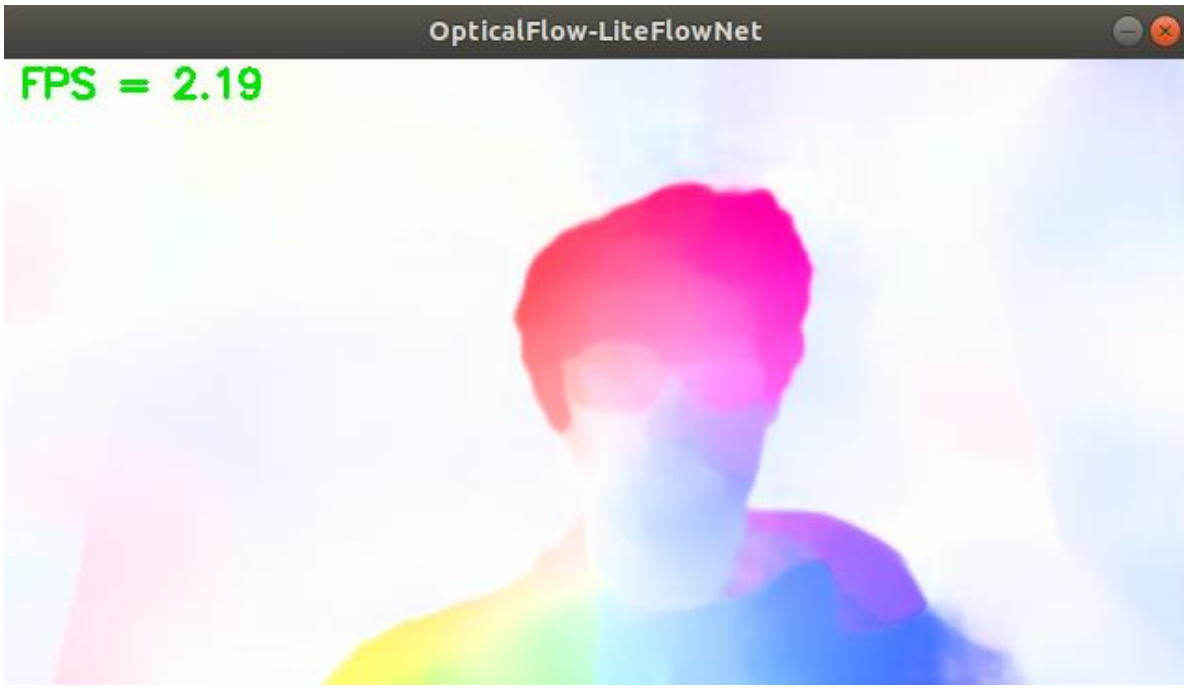


Figura 25-Estimativa de um movimento lento (LiteFlowNet)

As redes neuronais foram aplicadas diretamente sem nenhuma afinação. Como os movimentos realizados com a Kinect são de baixa velocidade, uma afinação para esse tipo de movimento iria reduzir os erros de estimação. No entanto, o custo computacional seria o mesmo. As redes que apresentam um número de *fps* mais alto, quando os movimentos são de baixa velocidade, apresentam uma estimativa com maior ruído, as redes com um custo computacional mais alto apresentam uma melhor estimativa. Para demonstrar estas características foram testadas duas redes diferentes para movimentos de baixa velocidade. A primeira rede testada apresenta um custo computacional alto, a FlowNet2-css-ft-sd, a segunda é a que apresenta o maior número de *fps*, a FlowNet2-s. Observando a Figura 26, que correspondente à primeira rede, é possível verificar uma estimativa percetível e sem grande ruído. A Figura 27, relativa à segunda rede, apresenta um ruído muito maior.

Pela observação das imagens, torna-se claro que existe um compromisso grande entre custo computacional e a qualidade dos resultados. É de salientar que embora algumas redes exijam mais do *hardware* do que outras, caso alguma delas fosse escolhida, mesmo que a mais leve computacionalmente, isso iria sobrecarregar o sistema. É pertinente relembrar que a utilização do fluxo ótico estimado não substitui o algoritmo final anteriormente proposto. Seriam por isso processados em simultâneo. Além disso, depois de estimado o fluxo ótico teriam de ser aplicados cálculos para estimar os movimentos observados. O *hardware* disponível não apresenta capacidade para processar a quantidade de informação necessária. Depois da realização dos vários testes

referidos e retiradas as suas conclusões, foi decidido não se englobar nenhuma destas redes no algoritmo final.

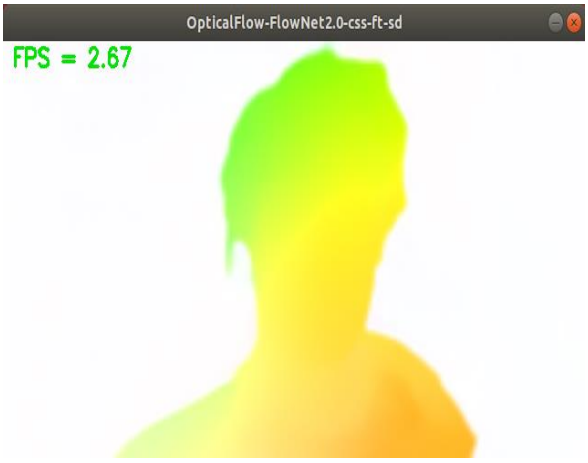


Figura 26-Estimativa de um movimento lento (FlowNet2-css-ft-sd)

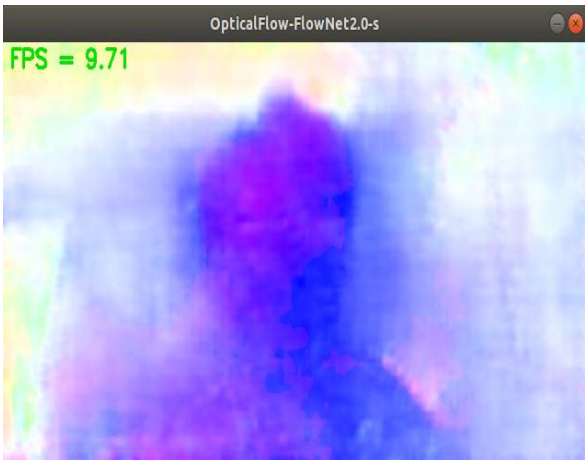


Figura 27-Estimativa de um movimento lento (rede FlowNet2-s)

Capítulo 4 – Testes e resultados

Neste capítulo são descritos os testes e resultados do sistema proposto neste projeto. São ainda apresentadas algumas sugestões e implementações que melhorariam os resultados.

O ponto mais crítico deste trabalho é a odometria visual. A construção do mapa utiliza as nuvens de pontos provenientes da Kinect na escala correta, mas a sua colocação no ambiente virtual depende da posição estimada até esse instante. Os primeiros testes realizados apenas utilizavam a rotação estimada para a criação do mapa. Este teste foi realizado mantendo a câmara sempre na mesma posição e rodando-a. A Figura 28 retrata o ambiente 3D criado utilizando a técnica mencionada. Nesta figura é retratada parte de uma divisão de uma habitação, onde a cena captada corresponde a duas paredes perpendiculares. A parede esquerda corresponde à Figura 29 e a parede direita corresponde à Figura 30.

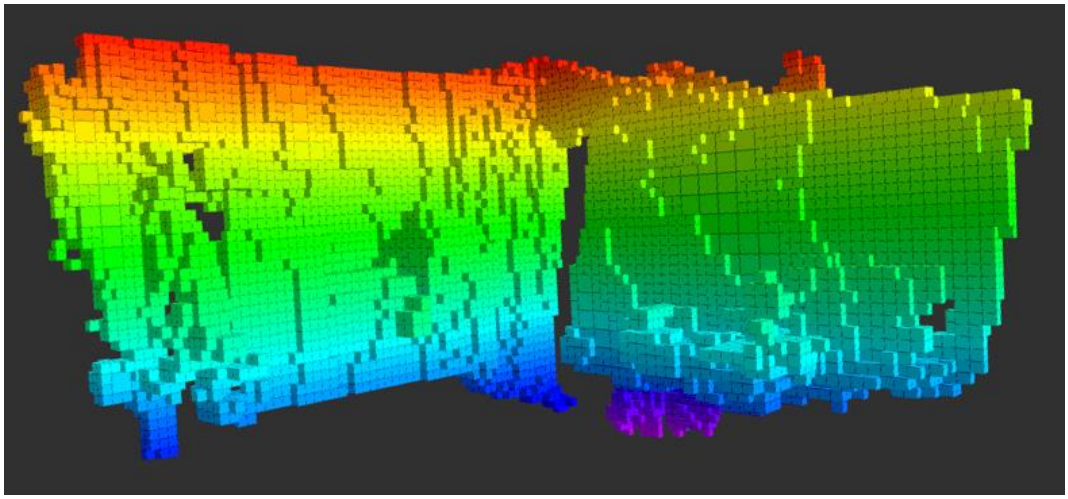


Figura 28-Ambiente 3D criado utilizando apenas a rotação

Figura 29-Imagem de parte do ambiente utilizado para a criação do mapa (perspetiva 1)

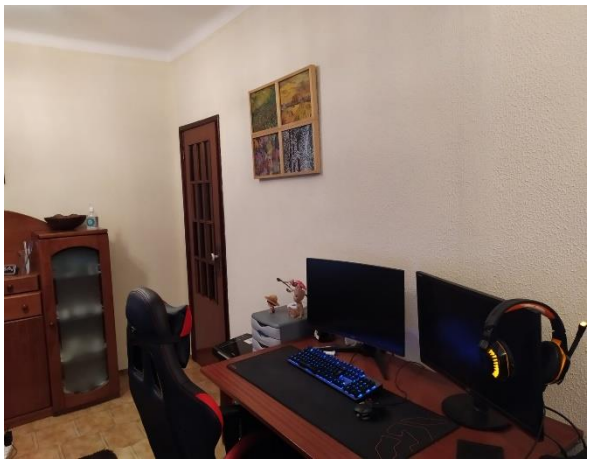


Figura 30-Imagem de parte do ambiente utilizado para a criação do mapa (perspetiva 2)

A ferramenta *rviz* permite medir distâncias dentro do mapa criado. Foram produzidos cinco mapas do mesmo ambiente (utilizando apenas a rotação). Uma secção de parede de 250 centímetros (medida real) foi utilizada como referência e a mesma secção foi medida no *software*. De seguida, foi realizada uma média dos valores obtidos. O erro médio introduzido pelo sistema foi aproximadamente 2,5 centímetros em 250 centímetros. Este valor é apenas relativo à rotação.

O passo seguinte passou por utilizar o algoritmo completo, para isso foi utilizada a mesma divisão da habitação para a construção do mapa com o algoritmo completo. Entre duas *frames* é calculada a escala respetiva a esse movimento que por sua vez é utilizada para calcular a translação nesse instante. Sempre que existe um erro de cálculo, seja no cálculo da escala, seja na estimativa da direção do movimento, esse erro é acumulado. Não existe, no presente trabalho, nenhuma forma de calcular a posição absoluta para corrigir possíveis erros nas estimativas, nem nenhum mecanismo de *loop-closure,* já mencionados no estado da arte, para reconhecer locais já visitados e refinar o mapa. Todos os erros acumulados contribuem para uma má estimativa da posição. D*rift* é o conjunto de todos os erros e retrata "deslizes" da estimativa. A Figura 31 representa o ambiente 3D que resultou do mesmo cenário apresentado anteriormente na Figura 28, mas, neste caso, utilizando o algoritmo completo. As Figuras 32, 33 e 34 representam o mesmo ambiente visto de diferentes perspetivas, sendo possível identificar várias zonas nas imagens onde ocorrem erros na estimativa da posição. Estas zonas foram destacadas para as tornar mais fáceis de visualizar. A razão principal que explica o erro observado no exemplo mencionado já foi abordada na secção3.2.4. Na realização deste ambiente 3D foram realizados movimentos laterais que levaram a uma má estimativa da odometria.

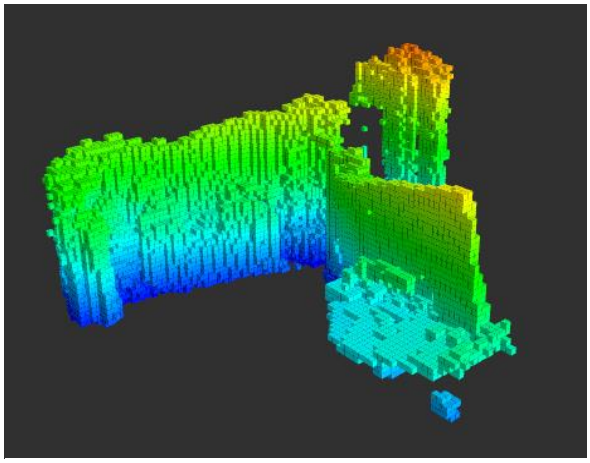


Figura 31- Ambiente 3D criado utilizando o algoritmo completo (perspetiva 1)

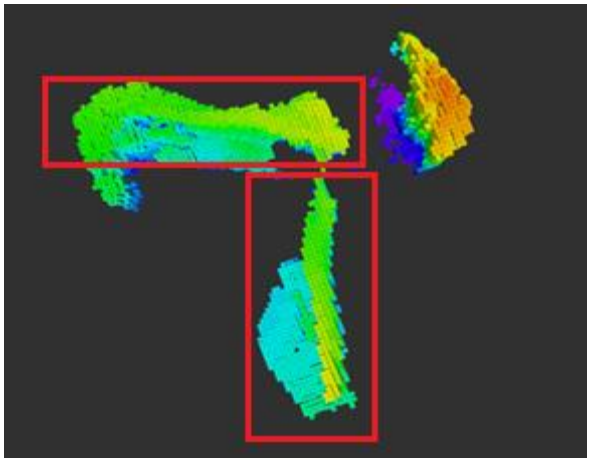


Figura 32-Ambiente 3D criado utilizando o algoritmo completo (perspetiva 2)

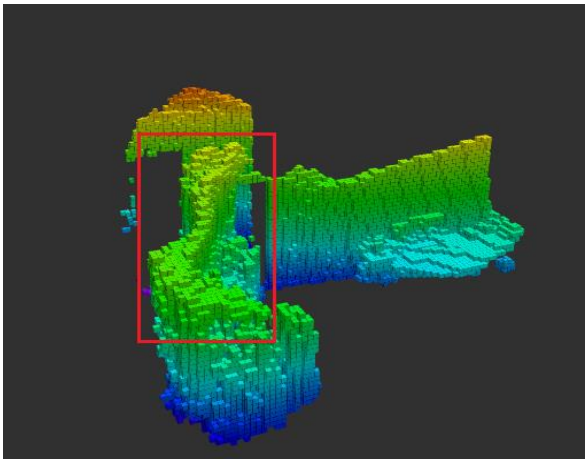


Figura 33-Ambiente 3D criado utilizando o algoritmo completo (perspetiva 3)

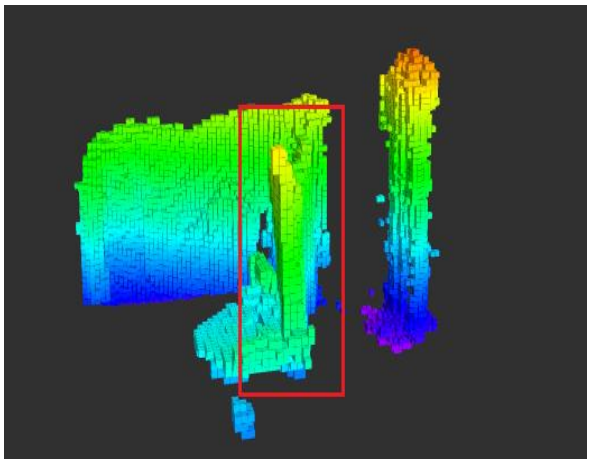


Figura 34-Ambiente 3D criado utilizando o algoritmo completo (perspetiva 4)

O teste seguinte passou por perceber as limitações do sistema perante diferentes intensidades de luz. Neste teste, metade da sala encontrava-se iluminada artificialmente e a outra não. Quando o nível de luz era demasiado baixo, não era possível ao sistema identificar os pontos-chave, isto levou a uma falha no programa. A Figura 35 mostra a reconstrução da divisão até o momento de rutura do programa (canto inferior direito da figura). As zonas destacadas mostram zonas onde existiu *drift.*

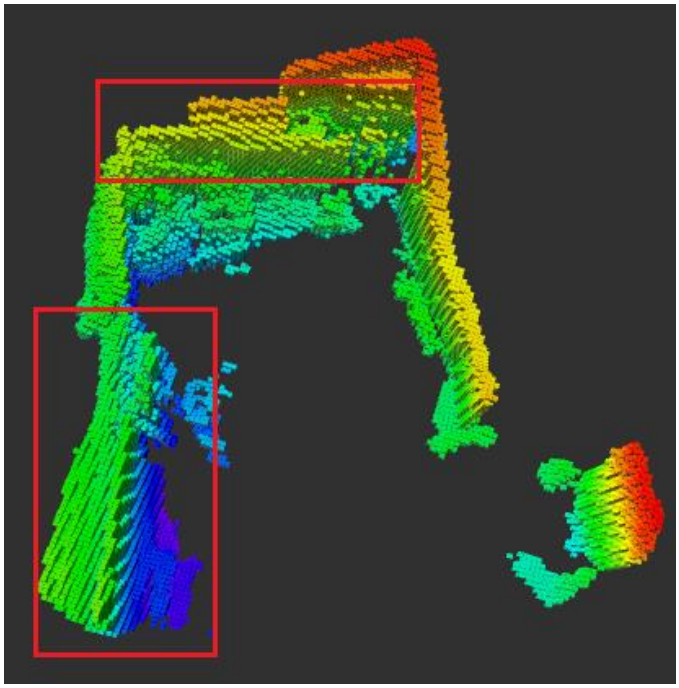


Figura 35-Reconstrução do mapa perante diferentes níveis de luz

Depois de testada essa limitação, a iluminação foi regularizada nas duas zonas da divisão. A Figura 36 mostra o resultado da reconstrução de toda a área da sala. A vermelho estão destacadas as regiões onde a estimativa apresentou maior erro.

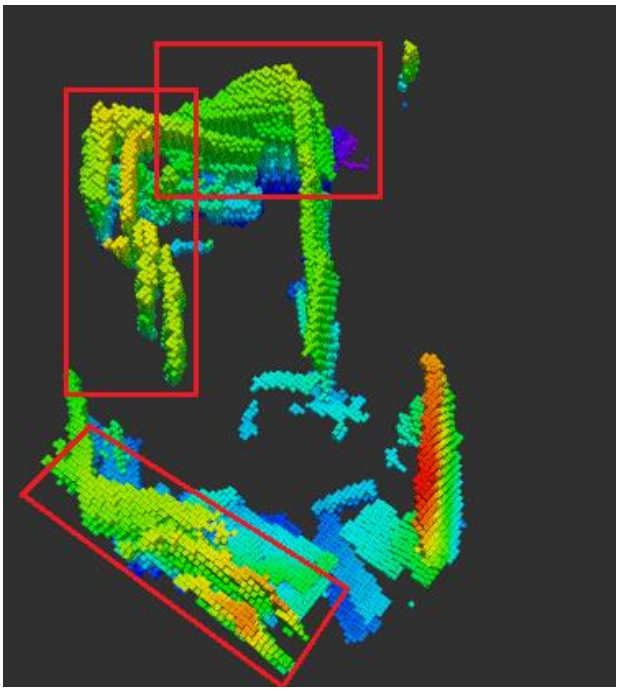


Figura 36-Reconstrução de toda a área da sala

Pela simples observação da imagem, podemos concluir que perante movimentos aleatórios lentos com a câmara, o erro na estimativa da odometria é demasiado elevado tornando o mapa impercetível. Perante movimentos deste tipo, o sistema não é robusto o suficiente para ser usado para reconstruir ambientes.

Conhecendo a limitação dos movimentos foi experimentada uma abordagem diferente. O algoritmo, como já referido, apresenta problemas com deslocações laterais ou verticais. Se a câmara for movimentada com movimentos semelhantes aos movimentos de um carro, maioritariamente para frente ou para trás, com rotações progressivas, as deslocações laterais e verticais mencionadas, em teoria, não existiriam. A Figura 37 demonstra o mapa construído pelo sistema utilizando esta técnica. A Kinect foi segurada em mãos sendo impossível cumprir a técnica proposta na perfeição, contudo, pela observação da figura podemos ver uma melhoria significativa na reconstrução da cena relativamente à figura anterior, ainda assim, existem erros nas zonas destacadas. Na construção do mapa, além do *drift* em x e y, também ocorreu em z. As Figuras 38, 39 e 40 destacam a principal zona onde ocorreu, e estas imagens representam o mesmo mapa de perspetivas diferentes. Nesta região, a altura das paredes deveria ser constante, pois a divisão é plana, mas, por erro de cálculo da odometria, parte da cena reconstruída não se encontra no mesmo nível.

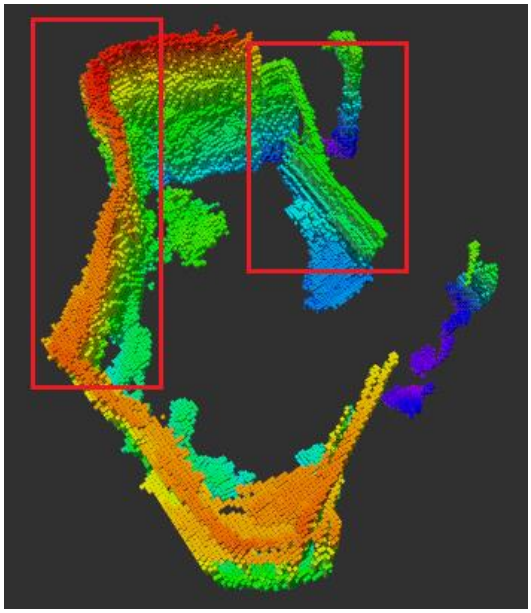


Figura 37-Reconstrução do ambiente movimentando a câmara de forma semelhante à movimentação de um carro

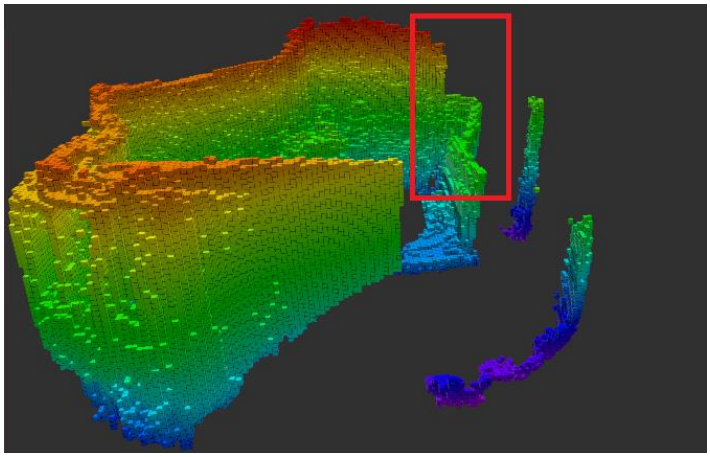


Figura 38-Drift em z (perspetiva 1)

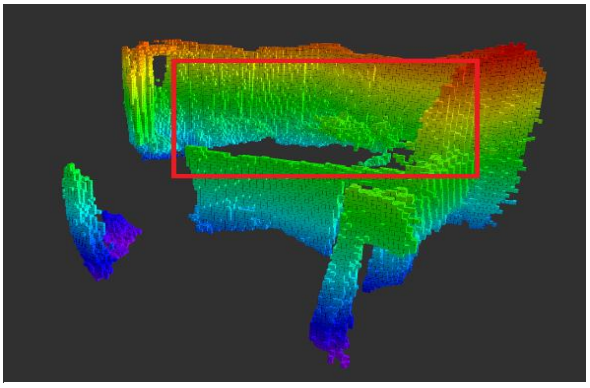


Figura 39-Drift em z (perspetiva 2)

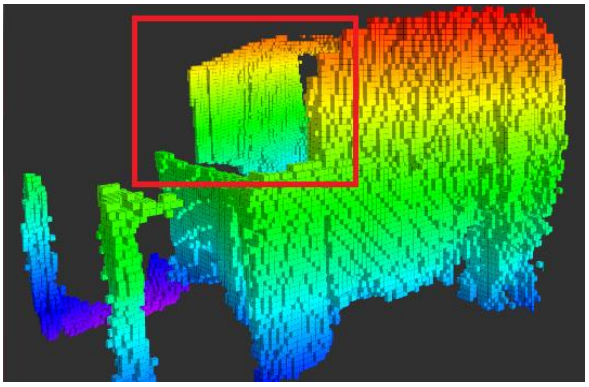


Figura 40-Drift em z (perspetiva 3)

O *drift* em z não foi encontrado nos testes anteriores, mas este exemplo prova que é um problema a resolver. Para tentar calcular o erro obtido enquanto o programa estava em funcionamento, foi feita uma medição, mais uma vez utilizando a ferramenta rviz, entre o local onde a reconstrução foi iniciada e o local onde terminou. As Figuras 41 e 42 mostram, respetivamente, o local onde foi iniciada a reconstrução e o local onde terminou. A Figura 43 mostra o local onde foi feita a medição do *drift* total.

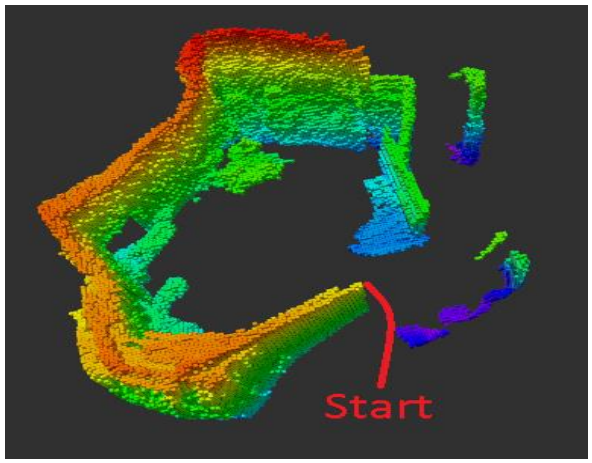


Figura 41-Local do fim da reconstrução

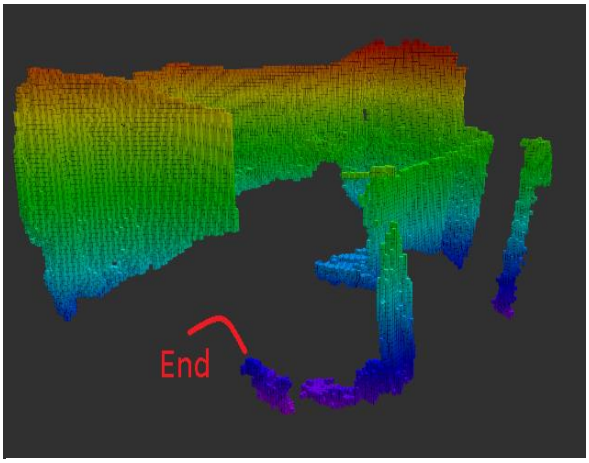


Figura 42-Local de início da reconstrução

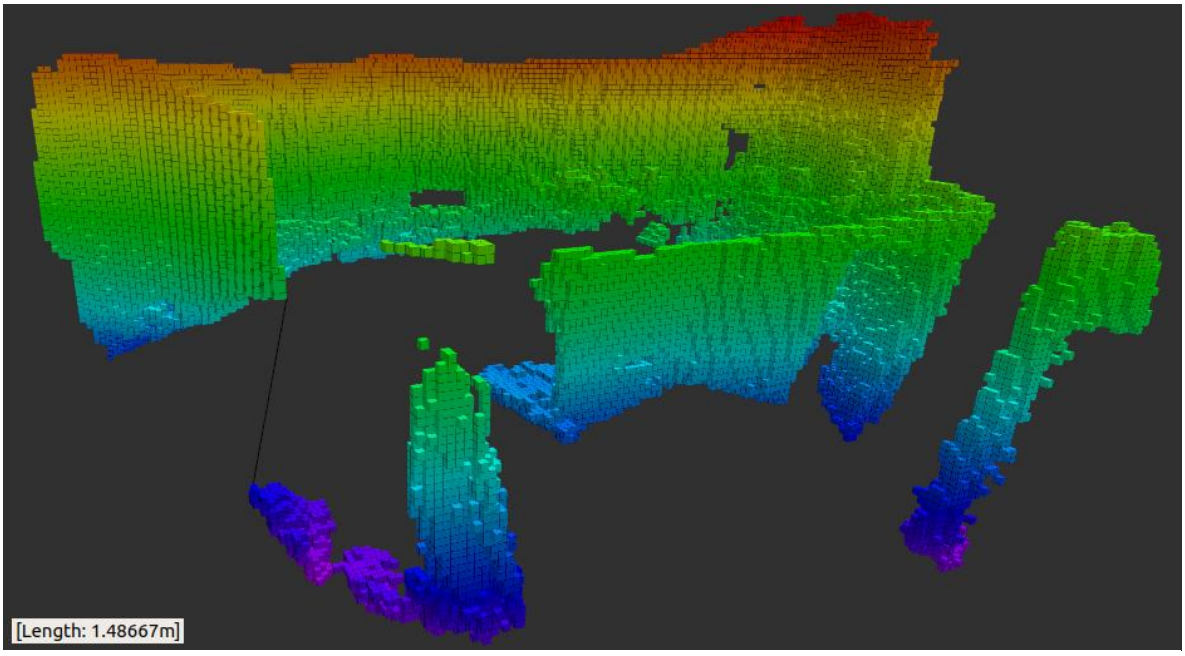


Figura 43- Medida direta do drift total

O erro obtido foi de 1,49 metros. Esta medida corresponde ao erro total englobando os três eixos. Realizando várias vezes a reconstrução do mesmo ambiente os resultados seriam diferentes e eventualmente menos previsíveis, sendo o resultado obtido apenas uma referência.

Isto deve-se ao facto de o programa estar dependente do tipo de movimento realizado pela câmara para apresentar melhores ou piores resultados, o algoritmo proposto é pouco preciso perante os movimentos já referidos. A Figura 44 mostra a planta do compartimento da casa utilizado com as medidas reais em metros, a Figura 45 mostra o mapa criado pelo sistema anteriormente apresentado, sem o destaque das zonas onde ocorreram erros na estimativa da odometria. Apesar da ocorrência de *drift* as paredes foram medidas, utilizando o mesmo software, e comparadas com as medidas reais dessas paredes. Nas zonas, do ambiente criado, onde as paredes não são direitas, as medidas foram realizadas entre as extremidades das paredes. Os resultados podem ser comparados na tabela 2. Dentro da tabela existem duas paredes virtuais sem medida porque, uma delas não apresenta informação concreta (sendo uma esquina a nuvem de pontos não reuniu informação necessária), e a outra porque a parede está dividida. Esta divisão marca o início e o fim da construção do mapa.

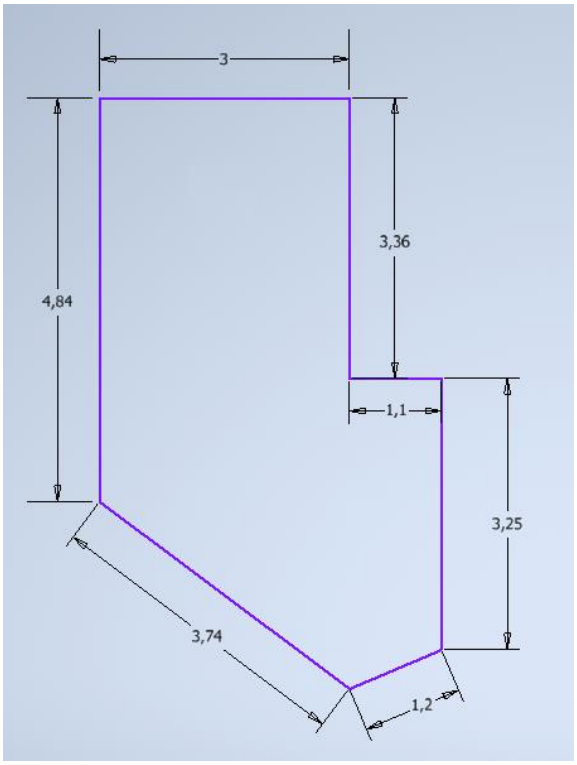


Figura 44-Planta da divisão da habitação

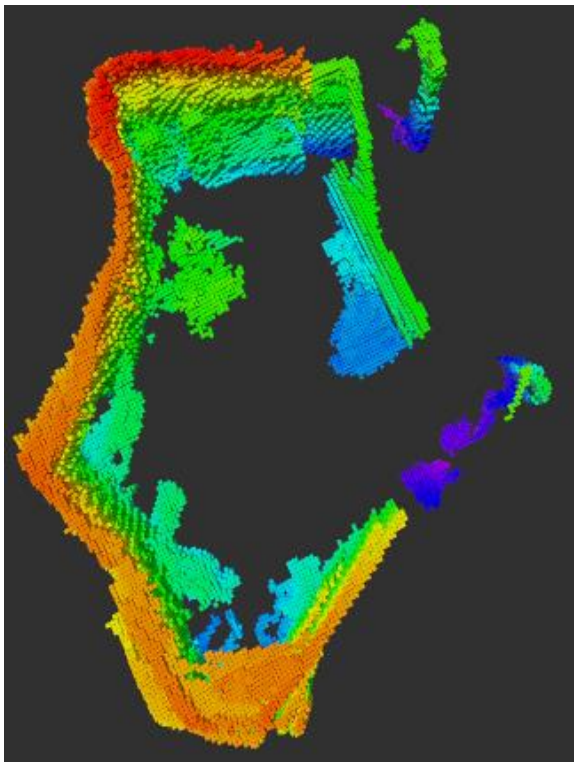


Figura 45-Mapa resultante (vista de cima)

Tabela 2-Comparação entre as medidas reais das paredes e as medidas estimadas

Medida real da parede	Medida dentro do ambiente criado	Erro
3,36 m	3,22 m	**0,14 m**
3 m	2,95 m	**0,05 m**
4,84 m	4,56m	**0,28 m**
3,74 m	3,61 m	**0,13 m**
1,2 m	0,95 m	**0,25 m**
3,25 m	-	**-**
1,1 m	-	**-**

Colocando as figuras lado a lado, torna-se mais claro que a planta obtida é bastante diferente da esperada. Algumas paredes apresentam uma boa aproximação em termos de medidas, mas, como já referido, estão dependentes do tipo de movimento realizado pela câmara, tornando os resultados apenas uma referência.

Foi obtido um erro alto e apenas foi reconstruída uma divisão de uma habitação. Para o presente algoritmo, o mapeamento de uma habitação completa com dados fiáveis, é impossível. O sistema proposto é pouco robusto e imprevisível não apresentando precisão suficiente para competir com outros sistemas da literatura. Se existir algum erro de cálculo que não tenha sido eliminado em alguma das etapas de verificações, irá persistir enquanto o sistema estiver em funcionamento. Estas razões tornam o sistema proposto inviável para uma aplicação prática.

No entanto, perante a melhoria observada com a mudança do movimento, existem algumas possibilidades que podem melhorar a precisão do sistema. No presente trabalho é exigido do sistema que calcule as deslocações e rotações em todos os eixos. Se a Kinect for montada num robô que apresente uma estrutura mecânica semelhante à de um carro será possível exigir menos deduções ao sistema, reduzindo assim o erro. Num robô deste tipo, as deslocações laterais não aconteceriam e, sendo estes movimentos os mais críticos para o sistema, o erro diminuiria. Além disso, estando a câmara aplicada numa base fixa, a sua altura seria constante, deixando de ser necessária a sua estimativa, no entanto, isto limitaria o sistema a operar em superfícies planas e niveladas. Perante o cenário descrito, o algoritmo resultante, seria suscetível a menos erros pois estaria a estimar menos variáveis.

Outra alternativa seria englobar no presente sistema outros sensores como por exemplo um IMU, que levaria a uma estimativa da posição mais precisa. O algoritmo proposto é amigável na introdução de novos sensores. Estes poderiam ajudar a reduzir os principais problemas, sendo estes o cálculo da escala e a estimativa da translação.

Capítulo 5 – Conclusão

O objetivo desta dissertação passava por desenvolver um sistema capaz de fazer um mapeamento e auto-localização em simultâneo, por outras palavras, um sistema SLAM. Para o realizar foi desenvolvido um algoritmo que utiliza uma Kinect como *hardware*. A câmara RGB é utilizada para prever o movimento e a câmara de profundidade é utlizada para estimar a escala e reconstruir o mapa.

O sistema foi desenvolvido em Python com ligação a ROS, utiliza pontos-chave do ambiente identificados com o algoritmo FAST. Os pontos-chave são seguidos ao longo do tempo e, através das transformações que sofrem devido ao movimento, são usados para estimar a matriz essencial. Através desta matriz, é deduzida a direção e orientação do movimento realizado entre duas *frames* consecutivas. Posto isto, a escala é calculada utilizando a distância dos pontos-chave à câmara. Utilizando a escala, a direção e orientação do movimento, é reconstruído o caminho percorrido pela câmara. Ao mesmo tempo que o percurso é estimado, é construído o mapa com as nuvens de pontos provenientes da Kinect. Todo o processo acontece em tempo real.

O sistema desenvolvido pode ser considerado um sistema SLAM completo pois consegue estimar a odometria e construir um mapa em simultâneo. Contudo, o sistema proposto apresenta dificuldades perante alguns tipos de movimento que levam a uma má estimativa da odometria, que por sua vez, leva a uma má construção do mapa. Os testes efetuados foram realizados numa área pequena e apresentaram um *drift* elevado. Estes sistemas, têm como objetivo cartografar toda a área de, por exemplo, uma habitação. O sistema, no presente estado, não é capaz de realizar esta tarefa. O maior problema deste sistema é a estimativa da odometria visual, esta tarefa é complexa e este sistema não é o primeiro a apresentar maus resultados contudo, na literatura, existem diversos sistemas SLAM com uma metodologia diferente que apresentam resultados mais consistentes do que o sistema desenvolvido.

Existem versões mais modernas de câmaras RGB-D, a introdução de alguma delas no sistema traria melhorias logo à partida. Não foi possível realizar mais testes com as redes neuronais propostas pois a placa gráfica e o processador do computador não eram computacionalmente capazes, no entanto, a sua implementação, poderia trazer melhorias à performance do sistema. O sistema proposto é amigável no que toca à implementação de novos sensores ou metodologias o que o torna uma excelente plataforma para desenvolver um sistema com maior precisão.

O objetivo proposto foi cumprido embora os resultados não sejam muito positivos. O trabalho realizado é uma boa base para a construção de um sistema mais robusto e fiável.

5.1 Trabalho futuro

Para melhorar este sistema poderia ser realizada uma calibração ainda mais precisa, poderiam ser englobados outros sensores como, por exemplo, IMU's ou LIDAR's. Os IMU's seriam uma boa opção com um custo computacional moderado, já os LIDAR's têm uma excelente precisão mas um custo computacional mais elevado. Além disso, existem métodos de *loop-closure* que poderiam ser implementados no sistema para fazer uma retificação do mapa calculado.

Referências

[1] Davide Scaramuzza and Friedrich Fraundorfer (2011, Dezembro). *Visual OdometryPart I: The First 30 Years and Fundamentals.* IEEE ROBOTICS & AUTOMATION MAGAZINE

[2] Mohammad O. A., Mohammad H. M., M. Iqbal and Napsiah (2016). *Review of visual odometry: types, approaches, challenges, and applications.* SpringerPlus *volume 5, Article number: 1897*

[3] J. Borenstein, H. R. Everett, and L. Feng *(*1996, abril). *Where am I? Sensors and Methods forMobile Robot Positioning.* D&D Robotics Technology Development Program and Robotics Technology Development Program.

[4] Aboelmagd N., Tashfeen B., Jacques Georgy. *Fundamentals of Inertial Navigation, Satellite-based Positioning and their Integration.* Springer Science & Business Media.

[5] Daobin W., Huawei L., Hui Z. e Shuai Z (2014). *A Bionic Camera-Based Polarization Navigation Sensor. Sensors*, *14*(7), 13006-13023.

[6] Othman Maklouf e Ahmed Adwaib (2014). *Performance Evaluation of GPS | INS Main Integration Approach. International Journal of Aerospace and Mechanical Engineering*, *8*(2), 476-484.

[7] Ribas et al. *SLAM using an Imaging Sonar for Partially Structured Underwater Environments.* In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 5040-5045). IEEE.

[8] Sanchez et al. (2012). *Autonomous indoor ultrasonic positioning system based on a low-cost conditioning circuit. Measurement*, *45*(3), 276-283.

[9] Horn and Schmidt (1995). *Continuous localization of a mobile robot based on 3D-laser-range-data, predicted sensor images, and dead-reckoning. Robotics and Autonomous Systems*, *14*(2-3), 99-118.

[10] Takeshi Takahashi (2007*). 2D localization of outdoor mobile robots using 3D laser range data. Robotics Institute Carnegie Mellon University Pittsburgh, Pennsylvania, 15213.*

[11] Lingemann et al. (2004, dezembro). *High-speed laser localization for mobile robots. Robotics and autonomous systems*, *51*(4), 275-296.

[12] Słowak, Pawel, Kaniewski, Piotr (2020, fevereiro). *LIDAR-based SLAM implementation using Kalman filter.* In *Radioelectronic Systems Conference 2019* (Vol. 11442, p. 114420N). International Society for Optics and Photonics.

[13] Ramezani et al. *Online LiDAR-SLAM for Legged Robots with Robust Registration and Deep-Learned Loop Closure. International Conference on Robotics and Automation (ICRA)* (pp. 4158-4164). IEEE.

[14] Oliver J. Woodman (2007). *An introduction to inertial navigation.* University of Cambridge, Computer Laboratory.

[15] Fang Fang, Xudong Ma and Xianzhong Dai (2005). *A Multi-sensor Fusion SLAM Approach for Mobile Robots.* In *IEEE International Conference Mechatronics and Automation, 2005* (Vol. 4, pp. 1837-1841). IEEE.

[16] Schleicher et al. (2009). *Real-Time Hierarchical Outdoor SLAM Based on Stereovision and GPS Fusion. IEEE Transactions on Intelligent Transportation Systems*, *10*(3), 440-452.

[17] Shin et al. *Direct Visual SLAM using Sparse Depth for Camera-LiDAR System. IEEE International Conference on Robotics and Automation (ICRA)* (pp. 5144-5151). IEEE.

[18] Qin et al. (2019). *A General Optimization-based Framework for Local Odometry Estimation with Multiple Sensors. arXiv preprint arXiv:1901.03638*.

[19] Shashi Poddar, Rahul Kottath, Vinod Karar. *Evolution of Visual Odometry Techniques. arXiv preprint arXiv:1804.11142*.

[20] J. Engel, V. Koltun and D. Cremers (2016). *Direct sparse odometry. IEEE transactions on pattern analysis and machine intelligence*, *40*(3), 611-625.

[21] F. Labrosse (2006). *The visual compass: Performance and limitations of an appearance-based method, Journal of Field Robotics*, *23*(10), 913-941.

[22] R. Gonzalez et al. (2011). *Combined visual odometry and visual compass for offroad mobile robots localization*. *Robotica*, *30*(6), 865-878.

[23] Lovegrove et al. (2011). *Accurate Visual Odometry from a Rear Parking Camera. IEEE Intelligent Vehicles Symposium (IV)* (pp. 788-793). IEEE.

[24] Andrew Comport, Ezio Malis, Patrick Rives (2010). *Real-time Quadrifocal Visual Odometry. TheInternational Journal of Robotics Research*, 29, pp.245-266.

[25] Srinivasan el al. (1991). *Range perception through apparent image speed in freely flying honeybees. Visual neuroscience*, *6*(05), 519-535.

[26] Horn and Schunck (1981). *Determining Optical Flow. Artificial intelligence*, *17*(1-3), 185-203.

[27] Campbell et all. (2005). *A Robust Visual Odometry and Precipice Detection System Using Consumer-grade Monocular Vision. IEEE International Conference on robotics and automation* (pp. 3421-3427). IEEE.

[28] S.Y.Pattar (2015). *Study of Corner Detection Algorithms and Evaluation Methods. International Journal of Innovative Research in Science, Engineering and Technology . Vol. 4, Issue 5*

[29] B. Kitt, A. Geiger and H. Lategahn (2010). *Visual odometry based on stereo image sequences with ransac-based outlier rejection scheme. ieee intelligent vehicles symposium* (pp. 486-492). IEEE.

[30] Tardif, Pavlidis e Daniilidis (2008). *Monocular Visual Odometry in Urban Environments Using an Omnidirectional Camera. IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 2531-2538). IEEE.

[31] Lee et al. *Motion Estimation for Self-Driving Cars With a Generalized Camera. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 2746-2753).

[32] Persson et all. (2005). *Robust Stereo Visual Odometry from Monocular Techniques. IEEE Intelligent Vehicles Symposium (IV)* (pp. 686-691). IEEE.

[33] Konolige, Agrawal e Solà (2011). *Large-Scale Visual Odometry for Rough Terrain. Robotics research* (pp. 201-212). Springer, Berlin, Heidelberg.

[34] David Nistér. An Efficient Solution to the Five-Point Relative Pose Problem. *IEEE transactions on pattern analysis and machine intelligence*, *26*(6), 756-770.

[35] Feng, Kan e Wu (2005). *AN IMPROVED METHOD TO ESTIMATE THE FUNDAMENTAL MATRIX BASED ON 7-POINT ALGORITHM. Journal of Theoretical and Applied Information Technology*, *46*(1), 212-217.

[36] Richard I. Hartley. *In Defence of the 8-point Algorithm. IEEE Transactions on pattern analysis and machine intelligence*, *19*(6), 580-593.

[37] John Oliensis e Michael Werman. *Structure from Motion using Points, Lines, and Intensities.* In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition. CVPR 2000 (Cat. No. PR00662)* (Vol. 2, pp. 599-606). IEEE.

[38] Scaramuzz et al. *Closing the Loop in Appearance-Guided Omnidirectional Visual Odometry by Using Vocabulary Trees. Robotics and Autonomous Systems*, *58*(6), 820-827.

[39] Silva et al. *Probabilistic Egomotion for Stereo Visual Odometry. Journal of Intelligent & Robotic Systems*, *77*(2), 265-280.

[40] Tateno et al. (2017). *CNN-SLAM: Real-time dense monocular SLAM with learned depth prediction. IEEE Conference on Computer Vision and Pattern Recognition* (pp. 6243-6252).

[41] Yan Loo et al. (2018). *CNN-SVO: Improving the Mapping in Semi-Direct Visual Odometry Using Single-Image Depth Prediction. International Conference on Robotics and Automation (ICRA)* (pp. 5218-5223). IEEE.

[42] Zhan et al. (2019). *Visual Odometry Revisited: What Should Be Learnt?. IEEE International Conference on Robotics and Automation (ICRA)* (pp. 4203-4210). IEEE.

[43] Smolyanskiy, kamenev e Birchfield. (2020) *On the Importance of Stereo for Accurate Depth Estimation: An Efficient Semi-Supervised Deep Neural Network Approach. IEEE Conference on Computer Vision and Pattern Recognition Workshops* (pp. 1007-1015).

[44] Zhao et al. (2018). *Learning monocular visual odometry with dense 3D mapping from dense 3D flow. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (pp. 6864-6871). IEEE.

[45] Fiala e Ufkes (2011). *Visual Odometry Using 3-Dimensional Video Input. Canadian Conference on Computer and Robot Vision* (pp. 86-93). IEEE.

[46] Yang et al. (2015). *Visual SLAM using Multiple RGB-D Cameras. IEEE International conference on robotics and biomimetics (ROBIO)* (pp. 1389-1395). IEEE.

[47] Whelan el all. *Robust Real-Time Visual Odometry for dense RGB-D Mapping. IEEE International Conference on Robotics and Automation* (pp. 5724-5731). IEEE.

[48] Cheng Zhao et al. (2018). *Dense RGB-D Semantic Mapping with Pixel-Voxel Neural Network. Sensors*, *18*(9), 3099.

[49] Franz Steidler(2007). Reality-based 3D city models from aerial and satellite data. In *Urban Data Management: Urban Data Management Society Symposium* (p. 47).

[50] Marek Strassenburg-Kleciak (2007). *Photogrammetry and 3D Car Navigation*. *51st Photogrammetric Week*, 309-314.

[51] Benjamin Ummenhofer e Thomas Brox. Global. *Dense Multiscale Reconstruction for a Billion Points. IEEE International Conference on Computer Vision* (pp. 1341-1349).

[52] Fuersattel et al. *OCPAD – Occluded Checkerboard Pattern Detector.* IEEE Winter Conference on Applications of Computer Vision (WACV).

[53] Placht et al. (2014). *ROCHADE: Robust Checkerboard Advanced Detection for Camera Calibration. European conference on computer vision* (pp. 766-779). Springer, Cham.

[54] Rosten e Drummond (2006). *Machine Learning for High-Speed Corner Detection. European conference on computer vision* (pp. 430-443). Springer, Berlin, Heidelberg.

[55] Tomasi e Kanade (1991). *Detection and Tracking of Point Features. Int. J. Comput. Vis*, 137-154.

[56] Martin A. Fischler e Robert C. Bolles (1981). *Random Sample Consensus: A Paradigm for Model Fitting with Apphcatlons to Image Analysis and Automated Cartography*. *Communications of the ACM*, *24*(6), 381-395.

[57] Tianzhu Qiao and Huaping Liu (2014). *Improved Least Median of Squares Localization for Non-Line-of-Sight Mitigation. IEEE communications letters*, *18*(8), 1451-1454.

[58] H.C Longuet-Higgins (1981). *A computer algorithm for reconstructing a scene from two projections. Nature*, *293*(5828), 133-135.

[59] Hornung et al. (2012). *OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees. Autonomous robots*, *34*(3), 189-206.

[60] Ilg et al. (2016). FlowNet 2.0: *Evolution of Optical Flow Estimation with Deep Networks.* In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 2462-2470).

https://github.com/lmb-freiburg/flownet2

[61] Hui et al. (2018). *LiteFlowNet: A Lightweight Convolutional Neural Network for Optical Flow Estimation.* In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 8981-8989).

https://github.com/twhui/LiteFlowNet

Anexos

Anexo A)

http://doc-ok.org/?p=313&fbclid=IwAR1S2ZYKg9_z0-ES7t5F1v6Em4kPt-6j9ECYIRtToL6pkLuSM6_zNBkaSuU

```
image_width: 640
image_height: 480
camera_name: rgb_A00366820987045A
camera_matrix:
  rows: 3
  cols: 3
  data: [523.91014,   0.     , 333.26039,
           0.     , 522.88189, 263.20687,
           0.     ,   0.     ,   1.     ]
camera_model: plumb_bob
distortion_coefficients:
  rows: 1
  cols: 5
  data: [0.181302, -0.333911, 0.002576, 0.001319, 0.000000]
rectification_matrix:
  rows: 3
  cols: 3
  data: [1., 0., 0.,
         0., 1., 0.,
         0., 0., 1.]
projection_matrix:
  rows: 3
  cols: 4
  data: [603.19,   -0.18721     , 307.147,0.,
           0.     , 601.744, 236.147,0.,
           0.     ,   0.     ,   1.,0. ]
```

I-Dados da calibração de fábrica

Os coeficientes de distorção são compostos por cinco elementos: k1, k2e k3 são os coeficientes do polinómio da distorção radial simétrica, t1e t2 que são os parâmetros de distorção tangencial.

$$\text{Distortion\_coefficients} = (k1, k2, t1, t2, k3)$$

A matriz da câmara projeta os pontos 3D da cena real para pixéis 2D. Com base nos dados utilizados para a calibração é calculado o ponto principal (cx e cy) e a distância focal (fx e fy).

$$\boldsymbol{camera\_matrix} = \begin{pmatrix} fx & 0 & cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{pmatrix}$$

A matriz de retificação só se aplica para sistema que utilizam mais do que uma câmara, o que não se aplica neste projeto.

A matriz de projeção, por convenção, representa a matriz da câmara e ainda duas variáveis Tx e Ty. Tx e Ty está relacionado com sistemas estéreo câmaras e representam o centro ótico da segunda câmara. Para sistemas mono câmara estas variáveis têm o valor 0. Esta matriz é calculada de forma diferente da matriz de projeção por isso é normal serem ligeiramente diferentes.

$$\boldsymbol{projection\_matrix} = \begin{pmatrix} fx & 0 & cx & Tx \\ 0 & fy & cy & Ty \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

http://docs.ros.org/en/api/sensor_msgs/html/msg/CameraInfo.html

Anexo B)

http://doc-ok.org/?p=289&fbclid=IwAR0FCtx0NQk2noy3WnnKXlzK4NP2KUjwJdPORPmHUXyrsw7h2gtJzi0V84M

```
image_width: 640
image_height: 480
camera_name: rgb_A00366820987045A
camera_matrix:
  rows: 3
  cols: 3
  data: [595.542603.19,   0.254782     , 307.806,
           0.     , 593.767, 238.752,
           0.     ,   0.     ,   1.     ]
camera_model: plumb_bob
distortion_coefficients:
  rows: 1
  cols: 5
  data: [0.181302, -0.333911, 0.002576, 0.001319, 0.000000]
rectification_matrix:
  rows: 3
  cols: 3
  data: [1., 0., 0.,
         0., 1., 0.,
         0., 0., 1.]
projection_matrix:
  rows: 3
  cols: 4
  data: [535.17706,   0.     , 334.6272 ,   0.     ,
           0.     , 535.62744, 264.07924,   0.     ,
           0.     ,   0.     ,   1.     ,   0.     ]
```

II-Dados da calibração da tabela semitransparente

Anexo C)

```
image_width: 640
image_height: 480
camera_name: rgb_A00366820987045A
camera_matrix:
  rows: 3
  cols: 3
  data: [515.83235,   0.     , 330.27965,
           0.     , 515.40825, 259.16179,
           0.     ,   0.     ,   1.     ]
camera_model: plumb_bob
distortion_coefficients:
  rows: 1
  cols: 5
  data: [0.186108, -0.396983, 0.002316, -0.002322, 0.000000]
rectification_matrix:
  rows: 3
  cols: 3
  data: [1., 0., 0.,
         0., 1., 0.,
         0., 0., 1.]
projection_matrix:
  rows: 3
  cols: 4
  data: [522.87585,   0.     , 329.75689,   0.     ,
           0.     , 526.70758, 260.04628,   0.     ,
           0.     ,   0.     ,   1.     ,   0.     ]
```

```
D = [0.1431152894540376, -0.2931806164035487, -0.001687120675399574, 0.004326035083442124, 0.0]
K = [513.6138371974026, 0.0, 333.00594170668484, 0.0, 513.607133951253, 260.66148986681077, 0.0, 0.0, 1.0]
R = [1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0]
P = [519.01611328125, 0.0, 336.54573679174064, 0.0, 0.0, 523.5571899414062, 260.2395245402622, 0.0, 0.0, 0.0, 1.0, 0.0]
```

IV- Dados da *câmara* RGB utilizando ROS

```
D: [0.0, 0.0, 0.0, 0.0, 0.0]
K: [575.8157348632812, 0.0, 319.5, 0.0, 575.8157348632812, 239.5, 0.0, 0.0, 1.0]
R: [1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0]
P: [575.8157348632812, 0.0, 319.5, 0.0, 0.0, 575.8157348632812, 239.5, 0.0, 0.0, 0.0, 1.0, 0.0]
```

III-Dados por defeito da câmara IV (ROS)