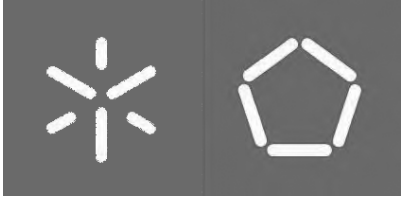
Universidade do Minho
Escola de Engenharia

Rui Pedro Lacerda Saraiva

Reconstrução tridimensional em tempo real a partir de um dispositivo móvel

dezembro de 2019



Universidade do Minho
Escola de Engenharia

Rui Pedro Lacerda Saraiva

Reconstrução tridimensional em tempo real a partir de um dispositivo móvel

Dissertação de Mestrado
Mestrado Integrado em Engenharia Eletrónica Industrial e Computadores

Trabalho efetuado sob a orientação do
Professor Doutor Agostinho Gil Teixeira Lopes

dezembro de 2019

DIREITOS DE AUTOR E CONDIÇÕES DE UTILIZAÇÃO DO TRABALHO POR TERCEIROS

Agradecimentos

A realização desta dissertação aqui apresentada não teria sido possível sem o apoio e a contribuição, direta e indireta de algumas pessoas, às quais quero e devo agradecer.

Os primeiros agradecimentos vão para o meu orientador o Doutor Agostinho Gil Lopes que me deu a oportunidade de realizar um trabalho de dissertação numa área de investigação que gosto particularmente. Agradeço também o apoio que sempre me prestou em primeiro lugar nas questões científicas associadas a este projeto de investigação, bem como, na parte psicológica associada a um trabalho desta natureza.

Em seguida, e a quem dedico este trabalho, aos meus pais, Ivone Saraiva e Acácio Saraiva pelo que fizeram e continuam a fazer por mim. Obrigado pela compreensão e por sempre me apoiarem a seguir os meus sonhos, dando-me sempre todas as condições para isso.

À minha namorada, Rebeca Moniz, que me acompanhou ao longo desta caminhada académica, agradeço a paciência e a motivação prestada e que me permitiram ultrapassar as diferentes fases deste trabalho. Obrigado pelo suporte nos momentos menos bons, a ela também dedico este trabalho. Agradeço também pela ajuda na revisão final relativamente à escrita desta dissertação.

Quero deixar também umas palavras, para as pessoas que se cruzaram comigo ao longo do meu percurso académico e principalmente umas palavras de agradecimento para os meus amigos que se cruzaram comigo no Laboratório de Automação e Robótica. Um obrigado ao Tiago Maia, ao Hélder Ribeiro, ao André Gomes.

Por fim, as últimas palavras vão para os meus colegas de trabalho, e para o meu coordenador, que me permitiram nesta última fase uma maior dedicação para finalizar esta dissertação.

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.

Resumo

Reconstrução tridimensional em tempo real a partir de um dispositivo móvel.

A abordagem computacional do problema da perceção tridimensional humana é uma tarefa bastante complexa e exige uma elevada capacidade de processamento por parte dos sistemas de visão artificial. Esta tarefa é conhecida na literatura como reconstrução 3D e tem influenciado inúmeras investigações na área da visão por computador. A necessidade de conteúdos 3D é transversal a diversas áreas, como por exemplo, arquitetura, computação gráfica, realidade virtual, bem como, a tarefas de arqueologia e preservação e restauro. No mercado já existe um número diversificado de dispositivos de baixo custo capazes de realizar captura tridimensional, como por exemplo, a *Microsoft Kinect*, mas são ainda pouco intuitivos para a maioria dos utilizadores, além de serem dependentes de computadores de tamanho elevado, dificultando a sua portabilidade. Surge assim a necessidade de introduzir novos dispositivos capazes de realizar a reconstrução 3D em tempo real. A capacidade atual dos dispositivos móveis aliada à sua portabilidade e à sua grande capacidade de aquisição de imagens, torna-os uma excelente opção para realizar as tarefas de reconstrução tridimensional.

O objetivo principal desta dissertação foi colmatar esta necessidade e, desta forma, criar um solução em tempo real, focada para dispositivos móveis e que permitisse realizar a reconstrução tridimensional de um objeto alvo. Nesta dissertação foram, então, desenvolvidas duas aplicações gráficas que partilham entre si uma camada de *software* que permite realizar o processo de reconstrução tridimensional baseada na técnica passiva *Strucuture-from-Motion*. Uma das aplicações é focada em dispositivos móveis, mais precisamente para a plataforma Android. A segunda é uma aplicação gráfica que permitiu estudar os algoritmos aplicados no processo de reconstrução tridimensional.

Palavras-chave: Dispositivos móveis, Reconstrução 3D, *Structure-from-Motion*, Visão por Computador.

Abstract

Real-time three-dimensional reconstruction from a mobile device.

The computational approach to the three-dimensional human perception problem is a very complex task and requires a processing capacity on the part of artificial vision systems. This task is known in the literature as 3D reconstruction and has influenced numerous investigations in the area of computer vision. The need for 3D contents is transversal to several areas, such as architecture, computer graphics, virtual reality, as well as archaeology and preservation and restoration tasks. On the market already exists a diverse number of low cost devices capable of performing three-dimensional capture, such as Microsoft Kinect, but are still not intuitive for most users, besides being dependent on computers of high size, making their portability difficult. The need arises to introduce new devices capable of performing 3D reconstruction in real time. The current capacity of mobile devices combined with their portability and their great capability of image acquisition makes them an excellent option to accomplish the tasks of three-dimensional reconstruction.

The main objective of this dissertation was to fill this need and, in this way, create a real-time solution, focused for mobile devices that would allow the three-dimensional reconstruction of a target object. In this dissertation, two graphic applications were developed that share among themselves a layer of software that allows to carry out the process of three-dimensional reconstruction based on the passive technique Structure-from-motion. One of the applications is focused on mobile devices, more precisely for the Android platform. The second is a graphical application that allowed studying the algorithms applied in the three-dimensional reconstruction process.

Keywords: Computer Vision, Mobile Devices, Structure-from-Motion, 3D Reconstruction.

Índice

Lista de Tabelas

Lista de Figuras

Acrónimos

BRIEF Binary Robust Independent Elementary Features

CMM Coordinate Machine Manufacturing

DoG Difference of Gaussian

FAST Features from Accelerated Segment Test

GPU Graphics Process Unit

IDE Integrated Development Environment

ICP Iterative Closest Point

ORB Oriented FAST and Rotated BRIEF

PCL Point Cloud Library

PLY Polygon File Format

RANSAC Random Sample Consensus

SCC Sistema Coordenado da Câmara

SCI Sistema Coordenado da Imagem

SCM Sistema Coordenado do Mundo

SCP Sistema Coordenado do Pixel

SfM Structure-from-motion

SIFT Scale Invariant Features Transform

SURF Speeded Up Robust Features

Capítulo 1

Introdução

Neste capítulo será apresentada uma visão geral dos tópicos incluídos nesta dissertação e do problema que está na sua origem, bem como, o seu enquadramento e motivação. Por fim, serão enunciados os objetivos que se propõem a atingir no final do projeto de dissertação. É ainda importante salientar que esta dissertação inclui-se na área da visão por computador, mais precisamente no âmbito da reconstrução tridimensional.

1.1 Enquadramento e Motivação

Vivemos numa era tecnológica, onde as sociedades modernas estão cada vez mais dependentes de sistemas inteligentes. Seja na indústria, na medicina ou simplesmente para tarefas do dia-a-dia, existe uma enorme necessidade de desenvolver soluções computacionais capazes de auxiliar e simplificar a realização de determinadas tarefas. Seja porque estes sistemas são mais precisos e ágeis ou simplesmente para facilitar em tarefas que são fisicamente mais exigentes para os humanos, a verdade é que estão incluídos em quase todas as tarefas do nosso dia-a-dia.

Independentemente da área de aplicação, o que normalmente diferencia estes sistemas é a sua capacidade de perceção e interpretação do meio envolvente. Dado isto, existe desde sempre a necessidade de equipar estes sistemas com características idênticas às que existem no complexo sistema visual humano. A visão tem um papel fundamental na perceção humana, onde uma das suas principais características é a sua capacidade de perceber o mundo em seu redor de forma tridimensional, permitindo-nos ter uma constante noção de localização, bem como uma elevada perceção tridimensional das formas dos objetos. Estas características são também essenciais em sistemas inteligentes. A área que se debruça sobre o desenvolvimento de tecnologias de visão artificial para estes sistemas inteligentes é a visão por computa-

dor. Esta é uma área abrangente que usa técnicas de processamento de imagem de forma a facilitar a extração de informação útil de imagens.

Deste modo, este trabalho enquadra-se numa das disciplinas da visão por computador, mais precisamente na reconstrução tridimensional, que usa técnicas de processamento de imagem fundidas com geometria para inferir a informação tridimensional de um objeto ou cena. Inicialmente, os sistemas de reconstrução 3D eram apenas utilizados em ambientes industriais, em tarefas bastante específicas, como por exemplo, em tarefas direcionadas à engenharia inversa [1, 2] ou em atividades de inspeção de qualidade de peças, permitindo, assim, melhorias na qualidade do produto final, evitando erros devido à subjectividade de alguns processos industriais e, acima de tudo, possibilitar uma redução de custos [3]. Na área médica também são utilizados sistemas de reconstrução tridimensional, como por exemplo, sistemas de ressonância magnética ou sistemas de tomografia computadorizada.

As investigações na área da reconstrução tridimensional têm o seu foco no desenvolvimento de técnicas e algoritmos cada vez mais eficientes, que permitam a construção de modelos 3D mais precisos e com melhor qualidade final. A capacidade destes sistemas processarem e apresentarem em tempo real *feedback* ao utilizador é uma característica fundamental, sendo que isto permite também uma melhoria significativa no resultado final, além de ser muito mais prático para o utilizador final.

O aparecimento de sistemas de aquisição de baixo custo, como por exemplo, a *Microsoft Kinect* ou a *Asus Xition Pro Live* têm possibilitado o desenvolvimento de novas aplicações e algoritmos na área da reconstrução 3D em tempo real [4, 5]. Contudo, apesar dos bons resultados obtidos com sistemas baseados nestes novos sensores, a realidade é que ainda requerem uma elevada capacidade de processamento e o consequente uso de um computador externo, tornando estes sistemas pouco práticos e intuitivos para a maioria dos utilizadores.

Com isto, surge a necessidade de desenvolver soluções mais intuitivas que permitam, de uma maneira geral, que um maior número de utilizadores tenha acesso a estes sistemas de reconstrução tridimensional. Tendo isto em conta, os dispositivos móveis, com todas as suas características atuais, tornam-se excelentes plataformas para o desenvolvimento de sistemas de visão por computador. Os dispositivos móveis deixaram de ser equipamentos exclusivos para receber e realizar chamadas, uma vez que atualmente têm processadores com elevada capacidade de processamento e estão normalmente equipados com um conjunto de excelentes câmaras no campo da fotografia digital, capazes de gravar vídeos em 4K com elevadas taxas de reprodução. Além disso, vêm munidos de um conjunto de sensores, inclusive sensores de movimentos. Tudo isto permite aumentar as suas potencialidades para tarefas no âmbito da visão por computador, mais precisamente em tarefas de realidade virtual ou reconstrução tridimensional.

Dado isto, a ideia de utilizar dispositivos móveis para a reconstrução tridimensional é uma boa solução, mesmo tendo em conta, apenas as características base dos dispositivos móveis atuais. As suas capacidades de aquisição de imagens aliado aos sensores de movimento e à sua crescente capacidade de processamento pode contribuir para o desenvolvimento de soluções de reconstrução 3D baseadas em imagens e que não necessitem de câmaras de profundidade.

1.2 Objectivos

Esta dissertação tem como principal objetivo o desenvolvimento de um sistema de reconstrução 3D baseado em dispositivos móveis, capaz de realizar a leitura tridimensional de um objeto em tempo real. Um dos pontos fundamentais é que a solução seja de baixo custo, para isso deve ser baseada apenas nas características base das plataformas móveis atuais, como por exemplo, as câmaras e os sensores inerciais, evitando, desta forma, sistemas baseados em sensores de profundidade.

Com base neste objetivo geral, é possível especificar alguns outros objetivos secundários que permitirão realizar o objetivo final. Em primeiro lugar deverá ser estudado o estado da arte das técnicas e métodos de reconstrução tridimensional, de forma a inferir qual a que mais se adequa às especificidades do objetivo final pretendido. Neste caso, deve permitir o uso exclusivamente das características dos dispositivos móveis atuais.

Por outro lado, devem ser também estudados os algoritmos associados ao método de reconstrução tridimensional escolhido e, desta forma, definir quais se aplicam melhor às capacidades de processamento dos dispositivos móveis.

É objetivo o desenvolvimento de uma aplicação Android, constituída por uma interface gráfica intuitiva, que permita ao utilizador visualizar em tempo real o processo de reconstrução tridimensional, bem como a visualização final do objeto obtido. Por outro lado, esta interface também deve permitir a troca de modos de funcionamento, bem como a alteração dos parâmetros associados aos algoritmos utilizados. Pode ser também pertinente o desenvolvimento de ferramentas de suporte ao desenvolvimento dos algoritmos de processamento de imagem.

1.3 Estrutura da dissertação

O presente documento tem como principal objetivo descrever todo o percurso realizado ao longo deste projeto de dissertação. É então composto por sete capítulos, onde é apresentado e contextualizado

o problema, descrito o estado da arte, bem como apresentada a fundamentação teórica que sustenta a implementação prática realizada. É finalizado com a apresentação e discussão dos resultados obtidos.

No primeiro capítulo, o presente, é contextualizado o problema, onde é abordada a abrangência e o enquadramento dos sistemas de reconstrução tridimensional. Por fim, são também clarificados os objetivos para esta dissertação.

No segundo capítulo é realizada a apresentação do estado da arte, onde serão apresentados, de uma forma geral, os métodos e técnicas pertinentes encontrados na literatura da área. Com isto, foi dado um maior foco aos métodos baseados em processamento de imagem. Neste capítulo, será também realizado uma apresentação das soluções existentes no mercado, para a tarefa de reconstrução tridimensional e relacionadas com as técnicas e métodos apresentados.

No terceiro capítulo é apresentada a técnica escolhida, bem como toda a fundamentação teórica no qual se baseia. Sendo também apresentados os algoritmos inseridos nesta dissertação.

No quarto capítulo são apresentadas as bibliotecas utilizadas ao longo deste trabalho, onde é dado ênfase à plataforma Android.

No quinto capítulo é apresentada a implementação propriamente dita, onde são descritos todos os passos importantes durante o desenvolvimento.

Por fim, o sexto e sétimo capítulo correspondem, respetivamente, à apresentação dos resultados obtidos e à apresentação da conclusão e apresentadas propostas de trabalho futuro.

Capítulo 2

Estado da Arte

Neste capítulo serão apresentados os principais métodos e técnicas encontrados na literatura da reconstrução tridimensional, bem como algumas soluções comerciais. Sendo o tema desta dissertação relacionado com a área da reconstrução tridimensional, torna-se necessário apresentar e analisar os seguintes tópicos: (i) Sistemas e técnicas de Reconstrução 3D e (ii) Soluções comerciais de Sistemas de Reconstrução 3D. No primeiro são apresentados as técnicas e métodos de uma forma geral relacionados com a reconstrução tridimensional. Já no segundo são apresentadas algumas soluções existentes no mercado para a solução do problema da reconstrução tridimensional.

2.1 Sistemas e Técnicas de Reconstrução 3D

O processo de reconstrução tridimensional tem como principal objetivo a criação de modelos geométricos precisos de ambientes ou objetos. Como já foi referido anteriormente, as suas aplicações abrangem um elevado número de áreas, entre outras, arquitetura [6], arqueologia [7], medicina [8], robótica [9], realidade virtual [10]. Dado isto, áreas diversificadas impõem problemas distintos que, por sua vez, exigem soluções diferentes. Assim sendo, são diversas as técnicas e métodos que têm sido desenvolvidas ao longo dos anos. Estas são normalmente agrupadas tendo em consideração algumas características chave [11], tais como, as tecnologias utilizadas ou a forma de interação entre o sistema de digitalização tridimensional e os objetos ou cenas. Segundo os trabalhos apresentados em [11, 12] os sistemas de reconstrução tridimensional podem ser divididos em dois grandes grupos distintos, são estes, sistemas baseados em contacto e sistemas baseados em não contacto. Na Figura 2.1, está representado um organograma representativo do métodos de reconstrução tridimensional abordados neste capítulo, organizados nos respetivos grupos.

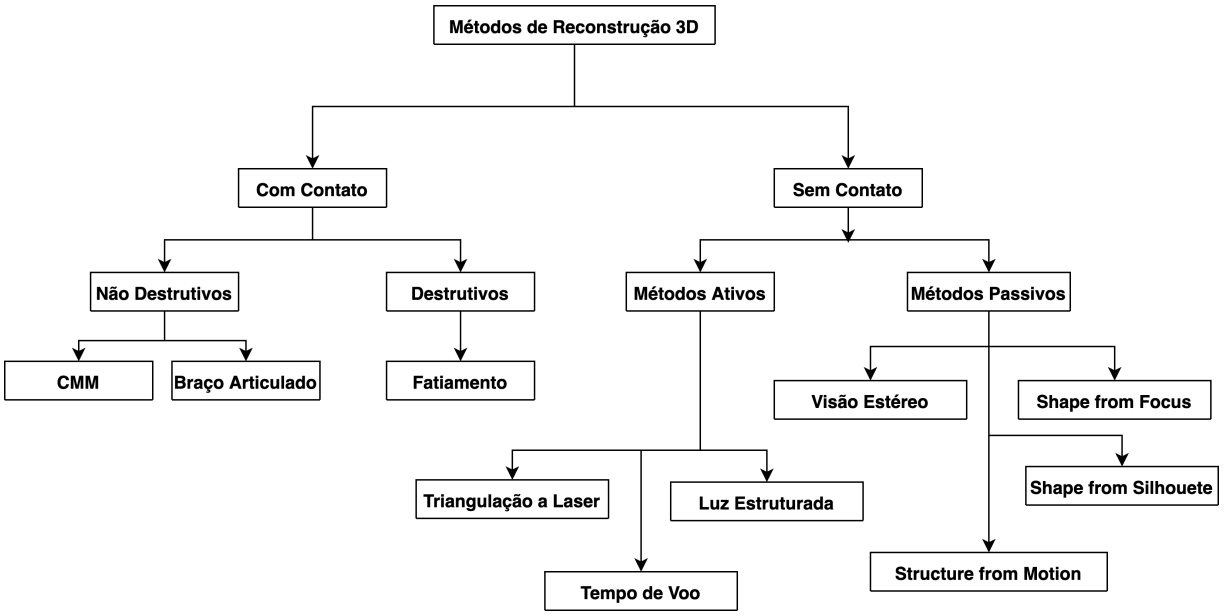


Figura 2.1: Diagrama de Técnicas de Reconstrução 3D.

Cada um dos métodos que serão apresentados possuem as suas características e, consequentemente, vantagens e desvantagens. Tendo em consideração que neste trabalho foi pré-definido a utilização de dispositivos móveis para a realização do levantamento tridimensional dos objetos e considerando também que estes nas suas configurações normais são equipados apenas com câmaras RGB, as técnicas a utilizar deverão ser baseadas essencialmente em métodos ópticos. Contudo, de forma a conhecer o estado da arte é importante apresentar de forma resumida os métodos existentes.

2.1.1 Métodos de Contacto

Como o nome indica, os métodos de contacto são métodos ou técnicas que exigem um contacto físico entre o sistema de digitalização e a superfície do objeto a ser reconstruído. A sua classificação pode também ser sub-dividida em métodos destrutivos ou métodos não destrutivos, dependendo se há ou não destruição do objeto durante o processo de reconstrução tridimensional.

Os métodos não destrutivos são caracterizados pelo uso de uma sonda de contacto, que acopladas a braços articulados, a máquinas de medição de coordenadas (CMM) ou máquinas de comando numérico (CNC) conseguem de forma muito precisa adquirir ponto a ponto as coordenadas tridimensionais da superfície do objeto [13]. Estes sistemas funcionam geralmente em dois modos distintos, um deles é o modo automático onde o sistema, usualmente uma CMM ou uma CNC, percorre um trajeto gerado pelo software de controlo, muita das vezes com o apoio de sistemas de visão artificial acoplados. Já no modo manual são mais utilizados os braços articulados, neste modo o operador tem que manualmente percorrer

a superfície dos objetos. Na Figura 2.2 são apresentados exemplos dos dois modos de funcionamento.

(a) (b)

Figura 2.2: Sistemas CMM: a) sem sistema de Visão 3D (**Imagem de:** Greg Vojtko[1]) b) com sistema de braço Articulado (**Imagem de:** U.S. Navy[2]) **Fonte:** Departamento de Defesa dos Estados Unidos.

Os métodos destrutivos, ao contrário de todos os outros, implicam uma destruição efectiva do objeto sobre análise. O processo baseia-se numa fase inicial no corte fatia a fatia do objeto, com posterior levantamento tridimensional de cada fatia individualmente, com cerca de 25 μm de espessura [14]. Por fim, via software, a informação tridimensional de cada fatia é fundida, resultando no modelo 3D final do objeto.

De uma forma geral, os métodos de contacto partilham o mesmo conjunto de vantagens e desvantagens com algumas particularidades que podem fazer a diferença na escolha do método a utilizar. Os sistemas baseados em CMM são mais precisos que os sistemas baseados em braços articulados, por sua vez têm menos graus de liberdade, o que dificulta na aquisição de superfícies mais complexas. Os sistemas de braços articulados tornam-se menos indicados para a reconstrução de objetos com extensas superfícies, uma vez que são controlados manualmente e a tarefa de reconstrução torna-se mais árdua para o operador. Ambos os sistemas, inclusive os métodos destrutivos, têm a grande desvantagem de necessitarem de contacto direto com o objeto, o que em objetos mais frágeis, como por exemplo, em artefactos históricos ou em controlo de qualidade de peças que vão para o cliente final tornam-se inviáveis, uma vez que podem danificar os objetos. Por sua vez, dado serem sistemas extremamente precisos, tornam-se mais complexos e, por conseguinte, são mais caros e especializados para um alvo específico. É ainda importante referir que são amplamente utilizados em tarefas de controlo de qualidade em várias indústrias e também em tarefas de engenharia reversa.

[1] `https://media.defense.gov/2016/Aug/26/2001616981/-1/-1/0/160502-N-HW977-279.JPG`
[2] `https://media.defense.gov/2016/Jan/14/2001335067/-1/-1/0/151202-N-DI674-003.JPG`

2.1.2 Métodos Sem Contacto

Os métodos sem contacto, como o nome indica, são métodos de reconstrução tridimensional, que não exigem qualquer contacto entre o objeto a reconstruir e o sistema de reconstrução. A sua classificação é feita tendo em consideração a fonte de energia utilizada, sendo geralmente dividida em métodos acústicos, magnéticos e ópticos. Estes três grupos utilizam a análise da propagação de três fontes de energia distintas, são elas som, campo magnético e luz [11]. Neste trabalho será dado foco aos métodos de reconstrução ópticos, com isto a apresentação do estado da arte dos métodos Sem Contacto será dividida em métodos ópticos ativos e métodos ópticos passivos.

A sua principal distinção está relacionada com a fonte e forma de luz utilizada. Enquanto que nos métodos passivos é analisada a reflexão da luz natural presente na superfície dos objetos no momento da aquisição, nos métodos ativos são utilizadas outras fontes de luz, como por exemplo, a luz infravermelha ou a luz laser. O método escolhido para a implementação desta dissertação é conhecido como Structure-from-Motion e inclui-se nos métodos ópticos passivos, este será apresentado na secção 3.2, no Capítulo 3. Dado isto, serão agora apresentados outros métodos sem contacto importantes na literatura da área.

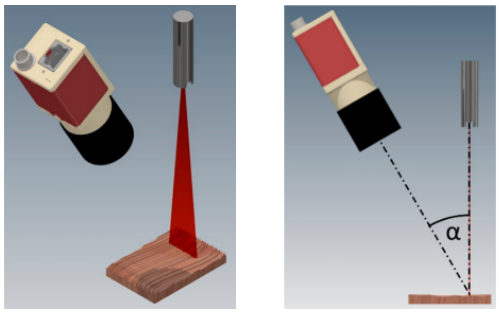
2.1.2.1 Métodos Ópticos Ativos

As técnicas ativas para a aquisição da forma tridimensional de um objeto, são caracterizadas pela projeção direta e controlada de energia sobre a cena, como por exemplo, sob a forma de vários tipos de luz, infravermelha, laser ou padrões de luz estruturada. A energia refletida é detetada por sensores colocados especificamente na cena, sendo que os dados recolhidos fornecem direta ou indiretamente informações sobre a distância da cena, que posteriormente são utilizados para o processo de reconstrução.

2.1.2.1.1 Triangulação a Laser

Os métodos de triangulação a laser são frequentemente utilizados em sistemas de reconstrução tridimensional, estes como o próprio nome indica, baseiam-se no princípio da triangulação e são normalmente constituídos por três pontos distintos que se posicionam de forma triangular, onde cada elemento se posiciona num dos vértices desse triângulo. Estes três elementos são uma fonte de luz, um equipamento com um sensor fotossensível e o objeto alvo. O seu funcionamento baseia-se na emissão de um feixe de luz normalmente na gama do laser ou infravermelho, na forma de ponto ou linha, que ao incidir na superfície do objeto é refletida e, posteriormente, observada pelo equipamento fotossensível [15]. Devido à perspetiva formada pelo posicionamento dos três constituintes do sistema, é possível analisar a informa-

ção tridimensional de um determinado objeto. No caso de ser emitida uma linha laser ou infravermelha o sistema analisará a sua deformação ao longo da superfície do objeto [16]. Um sistema de triangulação a laser é exemplificado nas imagens da Figura 2.3. Normalmente a estes sistemas de triangulação está associado um *software* proprietário de processamento de imagem que permite a análise das imagens adquiridas através do equipamento de visão que, por sua vez, transforma os dados em informação tridimensional acerca da superfície do objeto. Este método é bastante utilizado e muito robusto, contudo a sua precisão está diretamente relacionada à exatidão no posicionamento do sistema emissor-câmara. Podem ser encontrados sistemas de triangulação a laser com precisão na gama μm, tipicamente variando entre 20-200μm e com alcance entre 40-400mm [17]. A principal desvantagem relacionada a este tipo de sistema é só permitir levantamento tridimensional a objetos estáticos e constituídos com superfícies nem demasiado reflexivas nem completamente opacas.



(a) (b)

Figura 2.3: Ilustração[3]de um sistema de triangulação a laser a) conjunto dos três objetos constituintes do sistema; b) representação da geometria triangular do sistema [16].

2.1.2.1.2 Luz Estruturada

Esta técnica de reconstrução tridimensional é também baseada no princípio da triangulação, sendo que o seu princípio de funcionamento é baseado na iluminação ativa da cena através da projeção de um padrão bem definido sobre a superfície do objeto alvo, seguindo-se uma posterior análise através de um *software* de apoio. Estes sistemas de reconstrução tridimensional são, normalmente, constituídos por três elementos que se posicionam de forma triangular, dado se basearem também no princípio da triangulação. Comparativamente ao método de triangulação a laser, que é constituído por um emissor laser e uma câmara, no método de luz estruturada um projetor especial substituí o emissor laser.

[3]Sem alterações e sob Licença CC BY 4.0-`https://creativecommons.org/licenses/by/4.0/`

Neste método a câmara é utilizada para adquirir uma imagem da cena que, por sua vez, está iluminada pela luz estruturada. No caso da superfície do objeto alvo ser planar, sem variações, o padrão apresentado na imagem adquirida através da câmara é igual ao padrão de luz estruturada emitido. Por sua vez, caso a superfície do objeto seja irregular, a forma geométrica da superfície distorcerá o padrão de luz estruturada emitido pelo projetor [18], como é representado pela imagem presente na Figura 2.4.

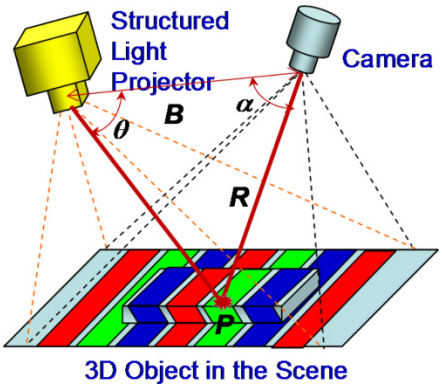


Figura 2.4: Ilustração[4]de um Sistema de Luz Estruturada [18].

A relação geométrica entre a câmara, o projetor de luz estruturada e a superfície do objeto é representada pelo princípio da triangulação presente na imagem da Figura 2.4, sendo conseguida através da aplicação de trigonometria, e expressa pela seguinte equação [18]:

$$R = B\frac{\sin(\theta)}{\sin(\alpha+\theta)} \tag{2.1.1}$$

Onde R representa a distância da câmara à superfície do objeto, B a distância entre o projetor e a câmara e α e θ são respetivamente os ângulos formados pelo centro ótico da câmara e do projetor com a superfície do objeto, representando, desta forma, dois dos ângulos presentes no triângulo formado pelo posicionamento destes equipamentos.

Atualmente, existe um conjunto diverso de técnicas específicas para o método de luz estruturada, sendo cada uma indicada para uma determinada tarefa, onde devem ser considerados os constrangimentos específicos. As técnicas de luz estruturada podem ser divididas em duas categorias distintas, são elas *múltiplas aquisições* e *uma aquisição*. A primeira produz resultados mais precisos em objetos estáticos enquanto a segunda produz melhores resultados a partir de objetos em movimento. As técnicas baseadas em uma única aquisição podem ainda ser classificadas tendo em consideração o tipo de padrão utilizado durante o processo. Por exemplo, existem técnicas que utilizam padrões contínuos

[4]Licença indicada em rodapé no pdf presente no link, somente para uso não Comercial - `http://imagebank.osa.org/getExport.xqy?img=LmFvcC0zLTItMTI4LWcwMDE=&xtype=pdf&article=aop-3-2-128-g001`

variáveis, outras utilizam padrões baseados em listas ou baseados em redes.

Por fim, é importante referir que a principal vantagem das técnicas baseadas em luz estruturada é a sua velocidade de aquisição de dados e consequente diminuição no tempo total de reconstrução. Ao contrário da técnica de triangulação a laser, que necessita de um varrimento sequencial de todo o objeto, os sistemas baseados em técnicas de luz estruturada têm a capacidade de adquirir o padrão de uma só vez. Deste modo, estão apenas limitados ao campo de visão do sistema ótico utilizado.

2.1.2.1.3 Tempo de Voo

O tempo de voo mais conhecido na literatura da área como *Time-of-Flight* é uma técnica ativa que permite obter a forma tridimensional de um objeto ou cena. Estes são sistemas compostos essencialmente por um sistema ótico e por um emissor, que permite enviar sinais modulados pulsados ou contínuos, usualmente com comprimentos de onda na faixa do infravermelho ou laser, de forma a não serem detetados pelo olho humano. Têm como principal característica permitirem estimar de forma muito precisa a profundidade a que cada pixel se encontra de um determinado ponto no espaço tridimensional presente no seu campo de visão.

De uma forma geral, estes métodos baseiam o seu princípio de funcionamento na análise do comportamento que um determinado sinal emitido pelo sistema de tempo de voo mantém desde a sua emissão até ser recebido novamente no sistema através da reflexão numa superfície. Em [16, 19] o autor, define que os sistemas de tempo de voo podem ser divididos em sistemas de tempo de voo direto e sistemas de tempo de voo indireto. Esta distinção está relacionada com as próprias características dos dois sistemas, que variam e estão diretamente relacionadas com a forma como é calculada a distância que se encontra cada pixel.

O primeiro calcula a profundidade de forma direta, quer isto dizer, através da medição do tempo que um determinado sinal demora no seu percurso de ida e volta desde o emissor até à superfície de um determinado objeto ou cena. Uma vez que estes sistemas trabalham com ondas electromagnéticas, é conhecida *à priori* a sua velocidade de propagação, a velocidade da luz e, consequentemente, é possível calcular a distância percorrida pelo sinal através da seguinte equação:

$$v = \frac{d}{t} \tag{2.1.2}$$

Onde *v* representa a velocidade em m/s, *d* representa a distância percorrida em *metros* e *t* representa o tempo em $segundos$. Dado que o objetivo dos equipamentos baseados nos princípios de tempo

de voo é a criação de um mapa de profundidade com a informação da distância de cada pixel da imagem, a equação anterior pode ser reorganizada de forma a evidenciar o cálculo da distância, da seguinte forma:

$$d = \frac{tc}{2} \tag{2.1.3}$$

Aqui c representa a velocidade da luz, que é uma constante aproximadamente igual a $3.0\text{x}10^8\ m/s$. A distância é metade da distância percorrida pelo impulso de luz, uma vez que, o método no cálculo do tempo considera o tempo total desde a emissão do impulso até a sua recepção novamente.

Já os sistemas de tempo de voo indiretos baseiam-se na análise da fase entre o sinal modulado enviado e o recebido após a reflexão numa determinada superfície [20], como representado pela imagem da Figura 2.5.

São normalmente utilizados sinais sinusóides, mas podem ser utilizados outros sinais periódicos. Para a medição da fase, cada pixel coleta a quantidade de luz refletida pela cena quatro vezes em intervalos iguais em cada período (m_o, m_1, m_2, m_3). Esta técnica de demodulação de sinal é conhecida como *four-bucket* [20] e permite calcular a profundidade. Em primeiro lugar, deve ser calculada a fase entre o sinal enviado e o recebido, através da seguinte equação:

$$\varphi = \text{arctan}(\frac{m_3 - m_1}{m_0 - m_2}) \tag{2.1.4}$$

Após o cálculo da fase é possível estimar a distância aplicando a próxima equação:

$$D = \frac{c}{2 * f_m}\frac{\varphi}{2\pi} \tag{2.1.5}$$

onde c é a velocidade da luz no vácuo e f_m a frequência de modulação do sinal enviado.

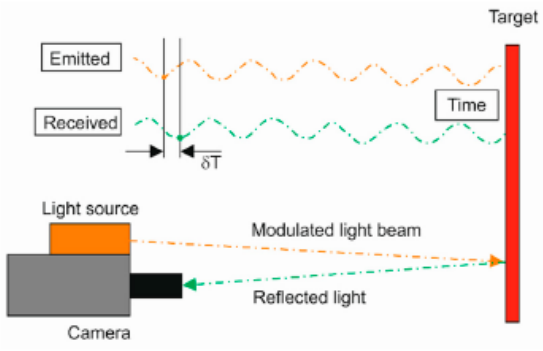


Figura 2.5: Principio de funcionamento de um Sistema de Tempo de Voo [5][16].

[5]Sem alterações e sob Licença CC BY 4.0-https://creativecommons.org/licenses/by/4.0/

Ainda segundo [20], é apresentada uma distinção entre os métodos de tempo de voo, luz estruturada e visão estéreo. Onde o primeiro sobressai por não necessitar de calibração nem iluminação externa e não estar dependente de algoritmos de correspondência de pontos de interesse. Obtém uma boa precisão para objetos com poucas texturas e permite alcances, dependendo do sistema, entre 0,3 e 7,5 metros. Apesar de tudo, estes sistemas de forma geral são mais caros do que maior parte dos sistemas de reconstrução tridimensional, devido à parte eletrónica a si associada para a demodulação dos sinais.

2.1.2.2 Métodos Ópticos Passivos

As técnicas passivas para a aquisição da forma tridimensional de um objeto baseiam-se, essencialmente, na utilização de luz visível para iluminar a cena. São, normalmente, sistemas baseados em estereoscopia, ou seja, utilizam uma ou mais câmaras para capturarem imagens de pontos de vista diferentes de um objeto. Outros métodos, como o *Shape from Silhouette* ou *Shape from Focus* são também utilizados para processos de reconstrução tridimensional e serão apresentados de seguida.

2.1.2.2.1 Visão Estéreo

O método da visão estéreo é um método de reconstrução passivo e surge da comparação com o sistema visual humano. O nosso cérebro estima a profundidade através da utilização simultânea dos dois olhos, bem como do conhecimento da distância entre eles. Isto cria uma disparidade, informação esta que é fundamental no processo de perceção tridimensional.

Aplicando esta ideia nos sistemas de visão por computador surgiu o método de visão estéreo, neste são normalmente utilizadas duas câmara posicionadas na horizontal e com um distância fixa previamente definida [21], como é ilustrado pela imagem da Figura 2.6. Cada uma destas tem a função de adquirir uma imagem de uma perspectiva diferente de um determinado objeto, isto permite criar a disparidade que acontece no sistema visual humano. Após a aquisição, cada uma das imagem é processada de forma a serem evidenciados os seus pontos fortes, pontos estes que têm alguma características que os faz sobressair dos pontos vizinhos pertencentes à mesma imagem. Além do mais, estes pontos devem ser comuns às imagens vizinhas, adquiridas de diferentes perspectivas. Chama-se a isto o *problema da correspondência* e tem como principal objetivo determinar pares de pixéis relativos ao mesmo ponto, mas em diferentes imagens. Por fim, o processo de utilizar um par de pontos comuns pertencentes a duas imagens diferentes e daí calcular um ponto único no espaço tridimensional é denominado *Triangulação*. É importante referir que para este processo é fundamental conhecer previamente algumas informações

do sistema ótico em causa, são elas os parâmetros intrínsecos e extrínsecos, conseguidos através de um processo prévio de calibração [19].

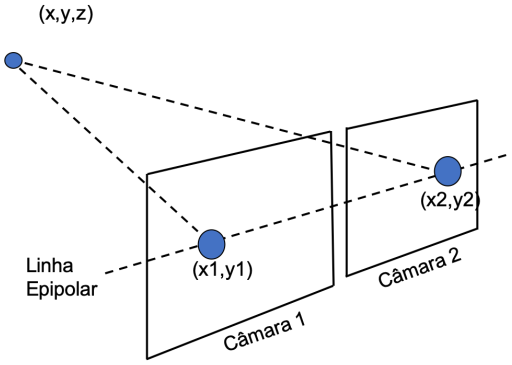


Figura 2.6: Ilustração do método de visão estéreo.

A principal vantagem deste método é o seu baixo custo, uma vez que é baseado na utilização de câmaras de imagem. Por sua vez, a principal desvantagem é necessitar de objetos com texturas bem definidas para uma boa precisão do resultado final. Este método, de todos apresentados neste capítulo, é o que se assemelha mais ao método utilizado nesta dissertação, a estrutura a partir do movimento. Isto porque os dois métodos baseiam-se essencialmente nos mesmo princípios de funcionamento.

2.1.2.2.2 Shape from Silhouette

O *Shape from Silhouette* é um método ótico passivo para estimar a forma tridimensional de um objeto. A ideia inicial surgiu em 1974 no trabalho desenvolvido por Baumgart durante a sua tese de doutoramento[22], onde expôs o conceito e o comprovou apresentando o exemplo da reconstrução tridimensional de um cavalo de brincar e de um menino.

O método foca-se na aquisição de várias imagens do mesmo objeto, mas adquiridas de diferentes pontos de vista, com o principal intuito de analisar as silhuetas formadas em cada momento. O conceito de silhueta na área da visão por computador, identicamente a outras áreas, é a forma ou contorno criado por um determinado objeto. O método considera que para cada imagem adquirida num determinado ponto de vista, a silhueta forma um cone de raios projetados, como é possível visualizar na imagem da Figura 2.7. O resultado da interceção de todos os cones de todas as imagens representa aproximadamente a forma tridimensional do objeto em estudo. Esta forma tridimensional é intitulada na literatura da área como *Visual Hull*, este termo surgiu no trabalho apresentado por Laurentini [23]. O *Visual Hull* é a interseção de cones de visualização gerados a partir do centro da câmara e que intersetam as silhuetas do objeto [24].

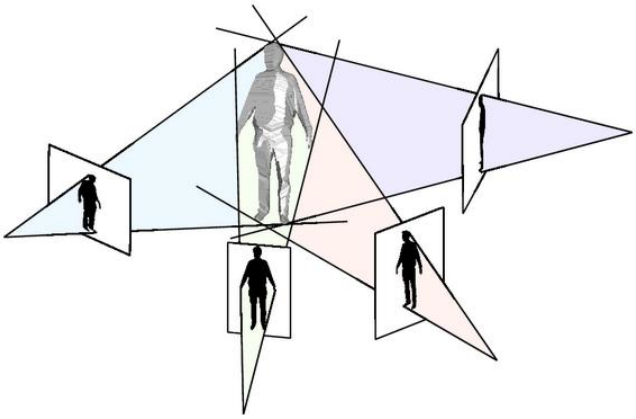


Figura 2.7: Ilustração [6]da técnica *Visual Hull* em uma pessoa a partir de quatro vistas [24].

A principal vantagem deste método é a sua relativa facilidade de implementação, quer isto dizer, a nível de *hardware* não exige máquinas com elevadas capacidades de processamento, uma vez que a deteção e análise de silhuetas a nível de processamento de imagem não é uma tarefa muito exigente. Dado isto, podem ser produzidos sistemas de *Shape from Silhouette* de baixo custo, com apenas uma câmara ou com um conjunto de várias câmaras que agilizará o processo de aquisição de imagens. Por outro lado, com este método obtêm-se resultados melhores quando o processo é realizado em ambientes interiores com luz bem controlada e principalmente em objetos estáticos.

2.1.2.2.3 Shape from Focus

O método *Shape from Focus* ou *Depth from Focus*, contrariamente aos outros métodos apresentados anteriormente, permite obter como resultado final um mapa de profundidade e não uma reconstrução tridimensional propriamente dita. É frequentemente utilizada em sistemas de auto foco passivos incorporados em câmaras digitais, bem como, em microscópios.

O método foca-se no problema de reconstruir a profundidade de uma cena mudando ativamente as definições ópticas da câmara até o ponto de interesse ficar em foco [25]. Trazer o ponto de interesse para o plano focal significa fazer variar alguma característica óptica, como por exemplo, a distância focal, a abertura ou a distância ao objeto.

Para entender melhor este método é importante recorrer à seguinte equação [26]:

$$\frac{1}{f} = \frac{1}{d_o} = \frac{1}{d_i} \tag{2.1.6}$$

[6]Sob Licença CC BY 3.0-`https://creativecommons.org/licenses/by/3.0/`

Esta equação traduz a relação entre a distância focal da lente, a distância do objeto e a distância ao plano de formação da imagem, representadas respetivamente por f, d_o e d_i.

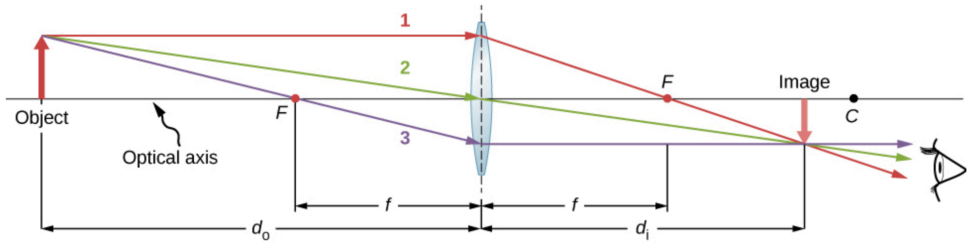


Figura 2.8: Modelo das lentes finas[7] [26].

A Figura 2.8 representa o processo geométrico implícito na formação de uma imagem. Todos os raios luminosos provenientes do objeto são projetados no plano da imagem, após sofrerem uma refração aquando da passagem pela lente. É possível considerar que uma imagem ou parte dela está desfocada quando o plano do sensor não coincide com o plano da imagem. Quer isto dizer, o objeto representado no esquema anterior para estar focado, o plano da imagem formado pela sua projeção deveria ser coincidente com o plano do sensor, representado pela letra F, ou seja, o objeto deveria estar a uma distância d_o=f. É importante referir que o grau do desfoque é tanto pior quanto mais longe o objeto estiver da distância focal [27]. É possível chegar à conclusão que uma só imagem não contem informação suficiente para inferir o mapa de profundidade com exatidão. Para isso, o método propõe a aquisição de pelo menos duas imagens distintas, após isso o problema resume-se à análise do desfoque em cada ponto da cena nas duas imagens.

Para a análise do desfoque são utilizados algoritmos e operadores que permitam medir o nível de desfoque de uma determinada imagem ou de um conjunto de pontos [27].

2.2 Exemplos de Sistemas de Reconstrução 3D

Finalizada a apresentação dos métodos e técnicas de reconstrução tridimensional considerados mais pertinentes no âmbito deste trabalho, é importante agora apresentar algumas soluções já existentes no mercado que se baseiam nas técnicas apresentadas anteriormente. As soluções incluem sistemas de *hardware* de reconstrução tridimensional, bem como soluções de software.

[7] Sob Licença CC BY-NC-SA 3.0 US -https://creativecommons.org/licenses/by-nc-sa/3.0/us/

2.2.1 Microsoft Kinect

A *Microsoft Kinect* [28], anunciada em 2010 pela *Microsoft* em parceria com a *Prime Sense* , foi lançada com o intuito de adicionar ao seu ecossistema de jogos electrónicos um equipamento capaz de detetar movimento, gestos e comandos de voz do utilizador e, desta forma, levar os jogos para um nível diferente no que toca à interação entre utilizador e o jogo. Dado o sucesso do primeiro sistema, a *Microsoft* apresenta uma atualização, lançando em meados de 2013 a *Microsoft Kinect 2.0*, aquando da apresentação da sua nova consola de jogos.

Os dois sistemas são equipados com uma câmara a cores, um sistema emissor de infravermelhos e uma câmara de infravermelhos, bem como um conjunto de microfones. O novo equipamento permitiu uma atualização dos sistemas ópticos embebidos, permitindo maiores resoluções, maior alcance e maior *Field of View*, como é possível verificar na comparação apresentada na Tabela 2.1.

	Kinect v1	**Kinect v2**
Tecnologia	Triangulação com luz estrutura	Time-of-flight
Camara a Cores	640x480 30fps 1280x960 12fps	1920x1080 30fps
Camara de Infravermelhos	640x480 30fps 320x240 30fps 80x60 30fps	512x424 30fps
Field of View	43° Vertical 57° Horizontal	>43° Vertival 70° Horizontal
Alcance de Profundidade	0.4m-4m	0.5m-8m

Tabela 2.1: Comparação das principais características das duas versões da Kinect.

Não é objetivo principal deste trabalho apresentar uma comparação detalhada dos dois sensores, para isso, podem ser encontrados na literatura alguns artigos, com por exemplo, o apresentado em [29]. Mas é importante referir o principal aspeto que as difere, que é a tecnologia de reconstrução tridimensional implícita.

Enquanto o primeiro sistema é baseado no príncipio de funcionamento da triangulação com luz estruturada, onde um conjunto emissor e receptor infravermelhos é responsável pela emissão e recepção de um padrão luminoso que é deformado consoante a superfície do objeto, o segundo sistema é baseado no princípio de funcionamento de *Time-of-Flight*. Ou seja, é emitido um pulso de luz, no caso infravermelho, que por reflexão volta ao sensor, o cálculo da profundidade de cada ponto é obtido através da diferença de tempo desde a emissão até à receção do feixe.

Num mercado super competitivo em quase todas as áreas, não sendo exceção no mercado dos sensores de profundidade e das câmaras inteligentes, têm sido desenvolvidos outros equipamentos com

características semelhantes às da *Microsoft Kinect* ou baseados noutros princípios e técnicas de reconstrução tridimensional. Por exemplo, o *Asus Xtion Pro* [30], lançado pela empresa ASUS, com características semelhantes às da *Microsoft Kinect*. A *Intel* com a tecnologia *Intel Real Sense* [31] tem um portfólio de equipamentos capazes de entender o mundo de forma tridimensional, baseados em princípios de funcionamento, como triangulação com luz estruturada, *Time-of-Flight* ou visão estéreo, inclusive embebidos em computadores portáteis para as tarefas de autenticação biométrica, o que também acontece desde a versão *X* do *iPhone*, onde a *Apple* desenvolveu tecnologia proprietária para a sua câmara frontal, utilizando métodos e técnicas de reconstrução 3D para realizar reconhecimento facial.

É importante referir também a *LMI Technologies* que produz câmaras inteligentes, bastante focadas para o mercado industrial e para a pesquisa e desenvolvimento, as *Gocator* [32], baseados nos princípios de funcionamento acima referidos, com a particularidade de não necessitarem de um computador a controlar o processo de reconstrução. O processo é feito internamente nos equipamentos e os resultados são apresentados ao utilizador via um *web server* interno. Para tarefas mais complexas, a empresa desenvolveu um *SDK* que permite a interação com os seus equipamentos.

(a) (b)

Figura 2.9: Sistemas Kinect: a) Kinect v1[8]b) Kinect v2[9].

2.2.2 Structure Sensor

Este equipamento foi desenvolvido pela empresa *Occipital*, que em 2014, apresenta o projeto no conhecido site de financiamento coletivo, *Kickstarter*, com o objetivo de recolher fundos suficientes para lançar o projeto. O sensor foi criado com o mesmo intuito da *Microsoft Kinect*, mas com o alvo específico dos dispositivos móveis, funcionando como um equipamento adicional. Os dispositivos móveis da *Apple* foram os alvos principais, sobretudo o iPad, onde a própria empresa investiu na criação de

[8]Imagem retirada de: `https://upload.wikimedia.org/wikipedia/commons/f/fe/KinectSensor.png`

[9]Imagem retirada de: `https://upload.wikimedia.org/wikipedia/commons/f/f6/Xbox-One-Kinect.jpg`

algumas aplicações, como por exemplo, *Scanner* [33], que permite a reconstrução tridimensional de objetos ou a aplicação *Canvas* [33], que permite a digitalização de espaços físicos e a posterior criação de ficheiros CAD.

Atualmente, devido à criação do *Structure SDK* [34], por parte da *Occipital*, já é possível desenvolver aplicações e sistemas para várias plataformas, incluindo *Android*, *Linux*, *Mac OS* e *Windows*, inclusive para alguns motores de desenvolvimento de jogos, como o *Unity*. Além do mais, o *SDK*, agiliza a fusão dos dados adquiridos pelo equipamento com outros sensores presentes nos dispositivos móveis, como por exemplo, as câmaras ou os acelerómetros.

O seu princípio de funcionamento é baseado em técnicas de triangulação e em técnicas de luz estruturada através da luz infravermelha, para isso, está equipado com dois *leds* infravermelhos, um projetor de luz estruturada infravermelha e uma câmara infravermelha. Ou seja, padrões infravermelhos são emitidos pelos emissores de energia, onde depois são detetados pela câmara após sofrerem reflexão no objeto.

Mais recentemente, a empresa lançou um sensor específico para a utilização em aplicações robóticas.

(a) (b)

Figura 2.10: Sistemas Structure Sensor: a) Aplicação iPad[10] b) Aplicado num iPad[11].

2.2.3 Project Tango

O *Project Tango* foi um projeto desenvolvido pela *Google*, pela equipa ATAP, *Advanced Technology and Projects Group*, em parceria com algumas universidades e laboratórios de investigação nas áreas da visão por computador e da robótica. O objetivo principal passava por agregar num dispositivo móvel o conhecimento gerado em anos de pesquisa nas áreas acima referidas, em temas como a re-

[10]Cortesia da Occipital: `https://structure.io/static/occipital/images/product_photos/jessy_scanning_phil.jpg`

[11]Cortesia da Occipital: `https://structure.io/static/occipital/images/product_photos/alysha_holding_ipad.jpg`

construção tridimensional, SLAM(*Simultaneous Localization and Mapping*) ou em algoritmos para realidade aumentada. Segundo o artigo [35], o objetivo do *Project Tango* é dotar aos dispositivos móveis a capacidade de perceber o espaço tridimensional e com isto ajudar as pessoas a interagir com os ambientes de forma diferente do que fazem habitualmente, além do mais, oferecer uma nova ferramenta de trabalho que permitisse agilizar a prototipagem de alguns objetos ou cenas, poupando horas de trabalho aos profissionais, uma vez que as ferramentas nessas áreas têm um elevado custo. Assim sendo, o *Project Tango* foi um projeto que permitiu criar um conjunto de dispositivos móveis, como por exemplo, o representado na Figura 2.11, baseados no sistema operativo *Android*, que agregavam um conjunto de câmaras e sensores de visão, permitindo, desta forma, perceber o mundo em redor em três dimensões e em tempo real. A primeira versão do equipamento estava equipado com uma câmara traseira de 4 megapixéis *RGB-IR*, uma câmara frontal 1 megapixéis um conjunto de sensores, como por exemplo, uma câmara de baixa resolução de *tracking* de movimento e um sensor de profundidade [35]. Além disto, está também equipado com um processador dedicado de baixo consumo para lidar com a grande quantidade de dados gerada nas tarefas de visão por computador.

A *Google* em parceria com a *Lenovo* ainda fez uma tentativa de comercializar o produto baseado no *Project Tango*, lançando o Lenovo Phab 2, mas devido à competitividade no mercado dos dispositivos móveis e ao pequeno nicho de mercado que este dispositivo agradaria acabou por cancelar o projeto. Contudo, a *Google* lançou uma API de realidade Aumentada, *ArCore*, onde é aplicada muito do conhecimento adquirido durante o projeto.

Figura 2.11: Project Tango *Development Kit*[12].

[12]Imagem retirada de:`https://commons.wikimedia.org/wiki/File:Google_ATAP%27s_Project_Tango_tablet_(15387052663).jpg`

Capítulo 3

Fundamentação Teórica

Neste capítulo serão apresentados os conceitos teóricos que suportam a fase de implementação deste trabalho de dissertação. Numa primeira fase é descrito o modelo geométrico implícito nas câmaras atuais, onde é apresentado o sistema de coordenadas, modelados os parâmetros intrínsecos e extrínsecos, bem como os parâmetros de distorção. De seguida, são abordados os métodos de calibração associados à reconstrução tridimensional, onde surge a descrição do Método de Zhang. É também descrita em pormenor a técnica de reconstrução tridimensional utilizada, no caso, *Structure-from-Motion*.

3.1 Calibração da Câmara

Atualmente, as câmaras são complexos sistemas ópticos, compostos normalmente por um sensor eletrónico de imagens, CCD ou CMOS, em conjunto com um sistema poderoso de lentes que permite aumentar e direcionar a quantidade de luz que chega ao sensor ótico. Contudo, a inclusão das lentes nos sistemas ópticos implica a adição de distorções a um sistema já por si complexo. Estas distorções são, normalmente, compensadas através de processos de calibração. Além disso, estes processos são importantes para criar uma relação entre as medidas na câmara (em unidades de pixéis) e as medidas tridimensionais reais (milímetros, metros).

Sendo assim, a compreensão do funcionamento interno de um sistema ótico, mais precisamente o processo de formação de uma imagem, é de extrema importância em tarefas relacionadas com processamento de imagem, inclusive em tarefas de reconstrução tridimensional. Nestas, existe uma grande necessidade de extrair das imagens informações úteis que permitam caracterizar de um forma precisa as estruturas presentes no espaço tridimensional.

Dado isto, é essencial a compreensão da geometria envolvida no processo de formação de imagens,

onde estão implícitos modelos matemáticos que são utilizados para descrever as transformações geométricas impostas pelo sistema ótico presente numa câmara. Nas secções seguintes serão abordadas estas transformações geométricas de forma a apresentar um modelo final onde é modelado o comportamento de formação de uma imagem. Os fundamentos matemáticos apresentados ao longo desta secção são baseados maioritariamente em três trabalhos presentes na literatura da área [36, 37, 38].

3.1.1 Modelação Matemática da Câmara

De forma a compreender as transformações internas de uma câmara, são normalmente utilizados modelos que descrevem de forma aproximada o processo de formação de uma imagem, como por exemplo, o modelo *pinhole* [36, 38]. Este é uma aproximação utilizando um sistema ótico simples, de forma a modelar o comportamento de sistemas ópticos mais complexos, como por exemplo uma câmara. É frequentemente utilizado, visto que apresenta uma aproximação bastante aceitável às transformações geométricas que acontecem durante o processo de formação de imagens numa câmara.

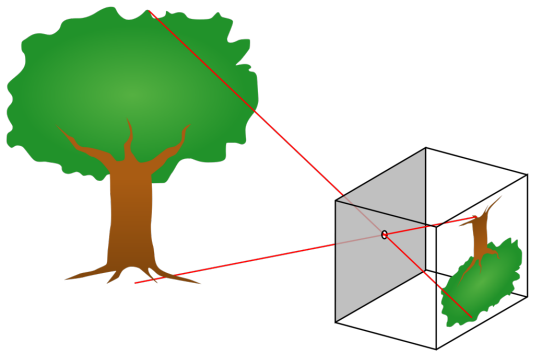


Figura 3.1: Representação do modelo da câmara *pin-hole*[1].

Neste modelo a câmara é representada como sendo uma caixa preta fechada, com um pequeno orifício num dos seus lados, que representa a abertura da lente, como representado na Figura 3.1. Os raios de luz que entram passam através deste orifício e são direcionados para o sensor ótico, onde a imagem aparece de forma invertida [38]. Posto isto, pode ser dito que o modelo *pinhole* é a primeira aproximação no mapeamento de um ponto no plano tridimensional para o mesmo ponto na imagem bidimensional.

3.1.2 Sistema de Coordenadas

Na modelação matemática do processo de formação de imagens é essencial uma pré-definição dos sistemas de coordenas implícitos no sistema. Desta forma, é possível modelar matematicamente as transformações geométricas que definem um sistema ótico. Em [37], a autora define, muito sucintamente e de forma clara, um conjunto de sistemas coordenados presentes no processo de formação de uma imagem, são eles:

- Sistema Coordenado do Mundo (SCM) - Este sistema coordenado representa o mundo tridimensional, onde estão realmente situados os objectos capturados pela câmara.
- Sistema Coordenado da Câmara (SCC) - Sistema coordenado que representa as coordenadas da câmara. A sua origem é no centro ótico da câmara e está definido de maneira que o eixo Z seja perpendicular ao plano da imagem e os eixos X e Y sejam paralelos.
- Sistema Coordenado da Imagem (SCI) - Sistema coordenado que representa o plano da imagem, com origem no ponto de projeção do centro ótico da câmara sobre o plano da imagem.
- Sistema Coordenado do Pixel (SCP) - Sistema coordenado que define a posição de um ponto na imagem, em relação à matriz de pixéis. Normalmente assume-se como origem deste sistema o canto superior esquerdo da imagem.

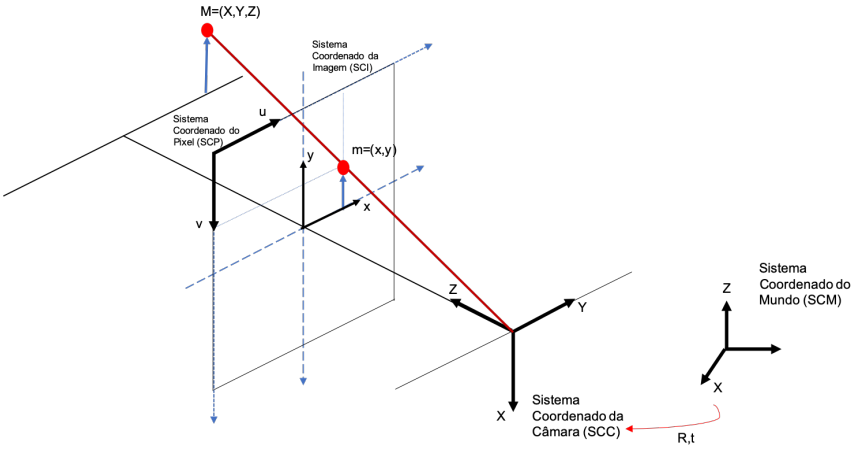


Figura 3.2: Representação dos sistemas coordenados presentes num sistema Mundo-Câmara.

Na Figura 3.2 é apresentado um esquema que traduz o funcionamento de um sistema ótico, onde é realizada uma projeção de um ponto no plano da imagem, representado por m, no plano tridimensional,

[1]Imagem retirada de: `https://upload.wikimedia.org/wikipedia/commons/3/3b/Pinhole-camera.svg`

representado por M. Além disto, estão também definidos os quatro sistemas coordenados apresentados anteriormente. A transformação entre o SCM e o SCC é uma transformação geométrica entre dois sistemas coordenados no plano tridimensional, logo é caracterizada por uma rotação, R, e por uma translação , t, e define os parâmetros extrínsecos da câmara. Os parâmetros intrínsecos definem as características internas das câmaras e são obtidos através da modelação das transformações geométricas entre os sistemas coordenados SCC, SCI e SCP [37].

3.1.3 Modelação dos Parâmetros Intrínsecos

Os parâmetros intrínsecos, são um conjunto de parâmetros que caracterizam o funcionamento interno da câmara. A sua definição está diretamente relacionada com as transformações geométricas entre os sistemas coordenados internos da câmara.

Desta forma, para a definição dos parâmetros intrínsecos podem ser definidas duas transformações entre os sistemas coordenados da câmara apresentados anteriormente, são elas:*(i) Transformação entre o SCC e o SCI* e *(ii) Transformação entre o SCI e o SCP*. Estas duas transformações definem o processo de formação de uma imagem, relativamente ao referencial da câmara [37].

3.1.3.1 Transformação entre o SCC e o SCI

A aproximação realizada pelo modelo *pinhole* permite definir as relações geométricas presentes na conversão entre um ponto no espaço (3D) e o seu correspondente no plano da imagem (2D), este mapeamento geométrico entre o plano e o espaço baseia-se no conceito de *transformação perspectiva*.

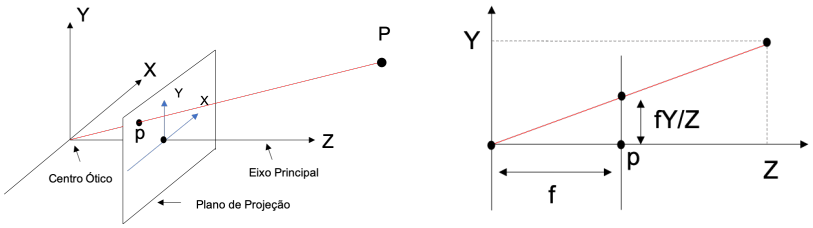


Figura 3.3: Modelo Geométrico *Pinhole*.

Assim, um ponto **P**=(X, Y, Z) pertencente ao espaço tridimensional é projetado num ponto **p**=(x, y) no plano da imagem, como é representado na Figura 3.3a. Além disto, podem ser também verificados, os seguintes elementos: o centro ótico C, o eixo principal Z, eixo este perpendicular ao plano de projeção. A sua interseção com o plano de projeção é denominada por ponto principal. Por

fim, a distância entre o centro da câmara e o plano de projeção é denominada de distância focal *f*[36]. É importante salientar que o modelo *pinhole* considera que a imagem aparece invertida no sensor ótico, apresentado na Figura 3.1. Para ultrapassar esta dificuldade a imagem deve ser invertida, o que corresponde a admitir que o plano da imagem está localizado à frente do centro ótico da câmara (Figura 3.3b).

Analisando a Figura 3.3b e tendo em conta o conceito de semelhança entre triângulos, é obtida a seguinte expressão:

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

Ou seja, podem ser resolvidas da seguinte maneira:

$$x = f\frac{X}{Z} \tag{3.1.2}$$

$$y = f\frac{Y}{Z} \tag{3.1.3}$$

A relação entre as equações 3.1.2 e 3.1.3 define uma *transformação perspetiva*, transformação esta não linear. Dado isso, para efeitos de simplificação, podem ser representadas na forma matricial, **x**=P**X**, em coordenadas homogéneas, através da seguinte equação:

$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}^{C} \tag{3.1.4}$$

Onde $\lambda = Z$ é um fator de escala homogéneo.

3.1.3.2 Transformação entre o SCI e o SCP

Depois de consideradas as transformações geométricas entre o sistema coordenado da câmara e o sistema coordenada da imagem, é necessário agora considerar as transformações geométricas que acontecem no plano da imagem. Na equação 3.1.4, assume-se que a origem no sistema coordenado da imagem é o ponto principal.

Entretanto, na maioria dos sistemas de imagem, é considerado normalmente que o sistema coordenado do pixel é definido como tendo a sua origem no canto superior esquerdo da imagem. Posto isto, é necessária uma conversão entre o sistemas coordenado da imagem e do pixel. Utilizando como base a

equação 3.1.4 e tendo em conta que um ponto no sistema de coordenado do pixel é representado por $(u, v)^T$, então a conversão entre um referencial e o outro é dada por:

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & p_x & 0 \\ 0 & f & p_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}^C \tag{3.1.5}$$

em que $(p_x, p_y)^T$ são as coordenadas do ponto principal.

Um importante factor a ter em consideração e que não está contemplado no modelo representado pela equação 3.1.5 é o sistema métrico utilizado para representar as coordenadas em cada referencial, uma vez que o sistema coordenado da câmara e do pixel representam as coordenadas utilizando unidades diferentes, metros e unidade de pixel, respetivamente para cada referencial.

Além disso, outra característica que deve ser considerada é a *razão de aspecto* dos pixéis, ou seja, a razão entre a sua largura e a sua altura. Normalmente, estes são quadrados (*razão de aspeto: 1:1)*, mas em câmaras que adotam um sensor de imagem do tipo CCD, existe a possibilidade de não serem. Sendo assim, e considerando que as coordenadas da imagem são medidas em pixéis e o número de pixéis por unidade de distância nas coordenadas de imagem são m_x e m_y, então a equação 3.1.5 pode ser representada da seguinte forma:

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha_x & s & x_0 & 0 \\ 0 & \alpha_y & y_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}^C \tag{3.1.6}$$

onde $\alpha_x = fm_x$, $\alpha_y = fm_y$, $x_0 = m_x p_x$ e $y_0 = m_y p_y$ representam respetivamente as distâncias focais e as coordenadas do ponto principal, representados em termos de unidade de pixel. Por fim, outro parâmetro que pode ser considerado é o *skew*, representado no modelo por *s* e está relacionado com o ângulo entre os eixos da imagem. Na maioria das câmaras este parâmetro é considerado nulo.

Desta forma, a equação que relaciona um ponto no sistema coordenado da câmara **X** com um ponto

no sistema coordenado do pixel **x** em coordenadas homogéneas, é dada pela seguinte equação:

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha_x & s & x_0 & 0 \\ 0 & \alpha_y & y_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}^C = K[I|0]X \tag{3.1.7}$$

onde K é a matriz dos parâmetros intrínsecos e é igual a:

$$K = \begin{bmatrix} \alpha_x & s & x_0 & 0 \\ 0 & \alpha_y & y_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{3.1.8}$$

3.1.4 Modelação dos Parâmetros Extrínsecos

Ao contrário dos parâmetros intrínsecos da câmara que estão relacionados com as características internas da câmara, os parâmetros extrínsecos estão relacionados com a posição e a orientação da câmara em relação ao sistema coordenado do mundo. Como já foi abordado anteriormente, na secção 3.1.2, os sistemas coordenados da câmara e do mundo, pertencem ambos ao espaço tridimensional, com a diferença que o sistema coordenado da câmara tem a sua origem no centro ótico da câmara e o sistema coordenado do mundo numa localização e orientação arbitrária. A transformação que caracteriza a relação entre estes sistemas coordenados é denominada de *transformação de corpo rígido* e é definida pela seguinte equação

$$X^C = RX^M + T \tag{3.1.9}$$

onde X^c e X^M representam respetivamente um ponto no sistema coordenado da câmara e do mundo e relacionam-se tendo em consideração uma rotação, representada pela matriz *R* e por uma translação representada pelo vetor *T*. A transformação de corpo rígido, apresentada na equação 3.1.9, pode ser expressa em coordenadas homogéneas:

$$\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}^C = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}^M \tag{3.1.10}$$

Por fim, e tendo em consideração a modelação dos parâmetros intrínsecos apresentada na secção 3.1.3 e a modelação dos parâmetros extrínsecos apresentada nesta secção, e observando as equações 3.1.7 e 3.1.10, é possível definir o modelo que permite converter um ponto no sistema coordenado do mundo, num ponto no sistema coordenado do pixel, através da seguinte expressão:

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}^{P} = K[R|t] \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}^{M} \tag{3.1.11}$$

3.1.5 Modelo de Distorção da Lentes

Os sistemas ópticos atuais têm incluídos um conjunto de lentes com o principal objetivo de aumentar e direcionar a quantidade de luz que entra para a câmara. Em contrapartida, a adição destes conjuntos de lentes adiciona às imagens um conjunto de distorções que devem ser consideradas, principalmente em sistemas que realizem medições em imagens, como por exemplo, em sistemas de reconstrução tridimensional. Nestes sistemas é essencial conhecer as distorções introduzidas pelas lentes, uma vez que, uma má modelação destas é a principal causa de imprecisões no modelo 3D final. Por exemplo, uma distorção frequente, é o aparecimento de uma linha curva no plano da imagem, que representa uma linha recta no plano tridimensional.

O processo de fabrico das câmaras e das lentes é a maior fonte de problemas que, posteriormente, causam distorções na imagem, principalmente em sistemas ópticos de baixo custo. Normalmente, são consideradas apenas a *distorção radial* e a *distorção tangencial* no processo de modelação matemática das distorções.

A distorção radial na prática é o deslocamento do ponto principal p_x e p_y da matriz dos parâmetros intrínsecos, representado na equação 3.1.8. Esse deslocamento pode ser positivo ou negativo e está relacionada à não coincidência entre o eixo ótico da câmara e o centro físico da imagem, resultando em dois tipos de distorção radial [39], representados na Figura 3.4.

O deslocamento radial negativo, denominado de *pincushion*, é caracterizado pela concentração dos pontos das margens no centro da imagem. De forma, contrária no deslocamento radial positivo, ou distorção *barrel*, os pontos da periferia da imagem estendem-se. A modelação da distorção radial imposta pelas lentes é conseguida através do mapeamento dos pontos antigos *(x,y)* em pontos corrigidos

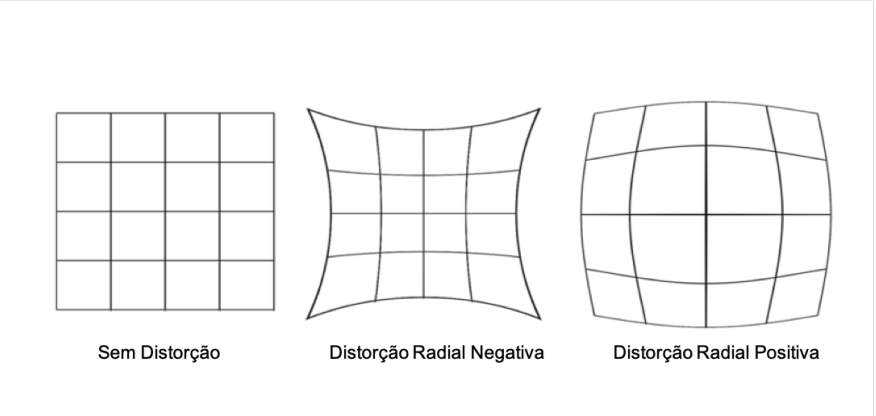


Figura 3.4: Efeito da distorção radial[2].

(x_r, y_r), segundo o modelo matemático [37]:

$$\begin{aligned} x_r &= x + x.r \\ y_r &= y + y.r \\ r &= [k_1(x^2+y^2) + k_2(x^2+y^2) + ... + k_n(x^2+y^2)^n] \end{aligned} \tag{3.1.12}$$

onde k_i são os coeficientes de índice i da distorção radical considerada.

Por fim, outra distorção considerada normalmente é a distorção tangencial, esta está relacionada com a orientação angular entre o plano focal e o o plano sensorial, que, na perfeição, deveriam ser completamente paralelos [39]. Este defeito é causado normalmente por problemas na processo de fabrico das câmaras.

Assim, e tendo em consideração os coeficientes de distorção radial k_1 e k_2, os coeficientes de distorção tangencial p_1 e p_2 e o modelo representado na equação 3.1.12, os efeitos da distorção tangencial podem ser corrigidos, segundo o seguinte modelo [37]:

$$\begin{aligned} x_r &= x(1 + k_1 r^2) + 2p_1 xy + p_2(r^2 + 2x^2) \\ y_r &= y(1 + k_1 r^2 + k_2 r^4) + p_1(r^2 + 2y^2) + 2p_2 xy \end{aligned} \tag{3.1.13}$$

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

3.1.6 Métodos de Calibração de Câmaras

A calibração de uma câmara é um passo essencial em qualquer sistema de reconstrução tridimensional, uma vez que permite caracterizar o processo de formação de imagem, facilitando posteriormente

[2]Imagem retirada de: `https://commons.wikimedia.org/wiki/File:Distorton_barrel_and_pincushion.png`

a obtenção de informações métricas acerca do espaço tridimensional. Por outras palavras, a calibração nada mais é que um processo que permite estimar de forma robusta um conjunto de parâmetros que caracterizam o sistema ótico. Estes parâmetros já foram descritos anteriormente neste documento, são eles, os parâmetros intrínsecos, extrínsecos e os de distorção. Estes permitem criar uma relação matemática entre as coordenadas de um ponto no espaço tridimensional e as coordenadas bidimensionais do mesmo ponto numa imagem, bem como, inferir as distorções impostas pelas lentes.

No trabalho apresentado em [40] o autor classifica os métodos de calibração em dois grupos distintos, o primeiro grupo denomina-o de *Calibração Fotométrica* e inclui os métodos que se baseiam na observação de um determinado padrão. Estes padrões são constituídos por formas bem definidas e geometria precisa, normalmente quadrados ou círculos apresentados num plano branco. As sua formas bem definidas contribuem para um rápida deteção pelos sistemas de visão por computador, já a sua geometria precisa contribui para a qualidade do modelo tridimensional final. Por sua vez, no segundo grupo estão incluídos os métodos de *Auto Calibração*, como o nome já sugere, as técnicas inseridas neste grupo não necessitam de um padrão de calibração, focando-se essencialmente na análise das características da cena ou do objeto. Os pontos em comum em imagens distintas, com aplicação de matemática, permitem inferir os parâmetros da câmara. Estes métodos partem do pressuposto que as imagens são adquiridas a partir da mesma câmara e, consequentemente, devem ter os mesmos parâmetros intrínsecos. Alguns trabalhos na literatura, podem ser encontrados [41, 42, 43].

3.1.6.1 Método de Calibração de Zhang

A técnica proposta por Zhang [40] implica a utilização de um padrão de calibração coplanar, o qual deve ter uma estrutura geométrica previamente conhecida e bastante precisa. Usualmente, é utilizado um padrão da calibração equiparado a um tabuleiro de xadrez, onde a geometria dos quadrados é bem conhecida. O autor também refere no trabalho que este padrão deve ser adquirido, pelo menos, segundo duas orientações distintas e que durante o processo de aquisição de imagens, as características internas da câmara não podem ser alteradas. Isto corresponde a dizer que os parâmetros intrínsecos devem manter-se constantes durante a aquisição, onde apenas os parâmetros extrínsecos podem ser variados, o que corresponde a movimentar o padrão ou o sistema ótico. Ao utilizar um padrão de calibração coplanar na prática está a assumir-se que $Z=0$ na equação 3.1.11, assim o mapeamento entre os pontos

tridimensionais e bidimensionais, simplifica-se, obtendo-se:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \cong K[R|t] \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \overset{Z=0}{\rightarrow} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \cong \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} \tag{3.1.14}$$

O algoritmo baseia-se no conceito de homografia, uma vez que a restrição ao plano Z=0 define uma transformação projetiva. Sendo que as homografias entre imagens podem ser obtidas através de uma equação de minimização do tipo não linear e que pode ser resolvida através da aplicação do algoritmo *Levenberg-Marquardt*. A apresentação matemática do método de *Zhang* é exposta no trabalho por ele publicado, relativo a este método de calibração [40].

Para a solução do problema da calibração, neste trabalho em específico, foram utilizadas as funções de calibração presentes na biblioteca *OpenCV* que, por sua vez, se baseiam nos métodos apresentados no trabalho proposto por Zhang [40].

O algoritmo proposto pela biblioteca OpenCV, baseado no método de Zhang, é aplicado através da chamada das funções *findChessboardCorners()*, *calibrateCamera()* e segue os seguintes passos:

1. Adquirir várias imagens do padrão de calibração segundo pontos de vistas distintos movendo a câmara ou o plano de calibração. Desta forma, alteram-se os parâmetros extrínsecos.

2. Extração das coordenadas dos vértices que constituem o padrão de calibração, se for utilizado um padrão tipo tabuleiro xadrez, os vértices correspondem aos pontos de interceção entre os diferentes quadrados. Para a deteção dos vértices pode ser utilizada a função *findChessboardCorners()*.

3. Estimar os parâmetros intrínsecos, extrínsecos e os coeficientes da distorção radial através da função *CameraCalibrate()*.

3.2 Estrutura a partir do movimento

A estrutura a partir do movimento, ou mais conhecida na literatura da área, pelo seu termo em inglês, *Structure from Motion*, é uma técnica de reconstrução tridimensional que utiliza a informação de um conjunto de imagens adquiridas de diferentes pontos de vista para recuperar a estrutura tridimensional de um objeto ou cena. Os diferentes pontos de vista podem ser conseguidos a partir de aquisições de uma única câmara móvel ou através da aquisição a partir de um conjunto de diferentes câmaras.

Paralelamente, permite também estimar a orientação da câmara para cada uma das imagens adquiridas, podendo esta técnica ser aplicada em algoritmos para estimar o percurso que uma determinada câmara realizou.

Posto isto, o método pode ser dividido em dois grupos distintos, no qual se incluem as técnicas que se focam na reconstrução de um conjunto de imagens sequenciais adquiridas a partir da mesma câmara e um segundo grupo onde se incluem as técnicas que se focam na reconstrução de um conjunto de imagens não ordenadas, adquiridas ou não pelas mesmas câmaras. O método *Structure from Motion* está ilustrado na Figura 3.5.

Nas últimas décadas tem aparecido na literatura um elevado número de trabalhos que abordam esta técnica, em diversas áreas que abordem a reconstrução tridimensional, a partir de satélites [44], veículos aéreos [45], carros autónomos [46] ou mesmo em dispositivos móveis [47, 48].

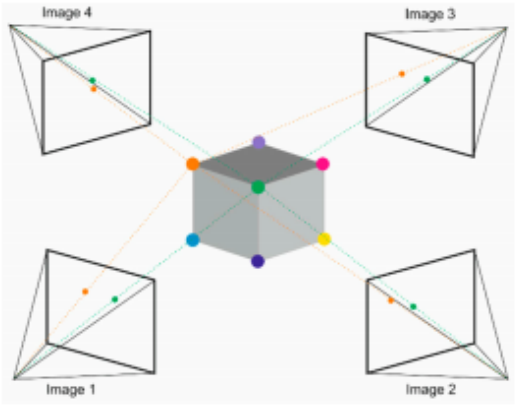


Figura 3.5: Ilustração do Método *Structure from Motion*[3][16].

O algoritmo utilizado ao longo desta dissertação é baseado no trabalho [36], e é apresentado de seguida:

O resultado final deste algoritmo permite a construção de uma nuvem de pontos representativa do objeto alvo, bem como a orientação da câmara para cada uma das imagens adquiridas. Esta orientação é composta pelas matrizes de rotação e translação. Além do mais, para cada par de imagens obtidos são detetados os pontos de interesse, bem como os respetivos emparelhamentos entre pontos de interesse semelhantes em imagens distintas. É importante referir que o ponto 4 não foi implementado nesta dissertação. De seguida, serão apresentados os algoritmos e conceitos associados a este método.

[3]Sem alterações e sob Licença CC BY 4.0-`https://creativecommons.org/licenses/by/4.0/`

Algoritmo 1 Algoritmo Estrutura a partir do movimento

1. Emparelhar e rastrear pontos de interesse em todas as imagens da sequência.
2. Inicializar a recuperação da estrutura e do movimento.
   (a) Escolher duas imagens adequadas para a inicialização.
   (b) Emparelhar os respetivos pontos de interesse entre cada uma das duas imagens escolhidas.
   (c) Reconstruir a estrutura inicial referente às duas imagens escolhidas.
3. Para cada imagem adicional:
   (a) Inferir correspondências relativas à estrutura tridimensional já existente.
   (b) Calcular a orientação da câmara utilizando um algoritmo robusto.
   (c) Refinar a estrutura existente.
   (d) Inicializar a nova estrutura tridimensional contendo os novos dados.
4. Refinar a estrutura e movimento através do *bundle adjustment*

3.2.1 Deteção de Pontos de Interesse

O primeiro passo nos processos de reconstrução tridimensional, baseados em técnicas ópticas passivas, como é o caso dos algoritmos de SfM, é o rastreamento de pontos interesses ou *features*. De uma forma resumida, o processo de rastreamento consiste, numa primeira fase, na deteção de pontos de interesse numa imagem e, numa segunda fase, no seu emparelhamento com os pontos correspondentes numa segunda imagem. Estes pontos de interesse são normalmente caracterizados por algum padrão que os distingue em relação à sua vizinhança próxima. Normalmente, estas diferenças estão relacionadas com mudanças de cor, intensidade e textura. Ao longo dos anos, foram sendo desenvolvidos alguns algoritmos que oferecem uma grande confiabilidade e robustez quando se trata de deteção e emperelhamento de pontos de interesse.

3.2.1.1 SIFT

O SIFT, apresentado por David Lowe [49], é um algoritmo recorrentemente utilizado em sistemas de visão por computador e tem como principal objetivo detetar e emparelhar pontos de interesse em diferentes imagens. Foi desenvolvido com o intuito de apresentar um método que fosse invariante às escalas e rotação das imagens, cobrindo assim algumas lacunas de outros métodos, que se focavam mais no reconhecimento da forma e aparência do que propriamente em características locais da imagem.

Com o objetivo de fazer a deteção e a descrição dos pontos de interesse da imagem, o algoritmo SIFT segue quatro etapas, segundo [49]:

1. Deteção de Extremos no Espaço de escalas.
2. Localização Precisa dos Pontos Chave.
3. Definição da Orientação.
4. Descrição dos Pontos Chave.

3.2.1.1.1 Deteção de Extremos no Espaço de Escalas

A primeira fase do algoritmo consiste na criação de uma pirâmide Gaussiana. Para isso, é utilizada a imagem de origem, em tons de cinza. A construção de pirâmide é conseguida através de dois conceitos fundamentais, o espaço de escalas e a oitava. O espaço de escalas é formado por um conjunto de imagens com um progressivo nível de desfoque, obtido através da aplicação de um filtro Gaussiano. As imagens com diferentes níveis de desfoque são representadas pela seguinte equação:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y) \tag{3.2.1}$$

onde **L** representa a imagem com desfoque e é conseguida através da operação de convolução de uma imagem **I**, com um função Gaussiana **G**, e uma variância σ.

A função Gaussiana *G* é representada pela seguinte equação:

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2} \tag{3.2.2}$$

Já cada oitava é formada pelo espaço de escalas, ou seja, cada oitava é formada por um conjunto de imagens com um progressivo desfoque, consequência da aplicação de um filtro Gaussiano. Cada oitava difere da anterior, devido ao redimensionamento da imagem para metade, deste modo, as imagens da primeira oitava têm o tamanho original, já na segunda o tamanho é metade e assim sucessivamente. Após a construção da pirâmide Gaussiana é aplicado um filtro DoG às imagens obtidas. Este filtro consiste, no cálculo da diferença Gaussiana, entre duas imagens, de duas escalas consecutivas, pertencentes à mesma oitava. Matematicamente a diferença Gaussiana é conseguida através da aplicação da seguinte equação:

$$D(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, k)) * I(x, y) = L(x, y, k\sigma) - L(x, y, \sigma) \tag{3.2.3}$$

onde **D**, representa a imagem final e *k* representa o índice do nível de escala.

Na Figura 3.6, está representada a pirâmide Gaussiana, onde estão implícitos os conceitos de oitava, de espaço de escalas e da diferença Gaussiana.

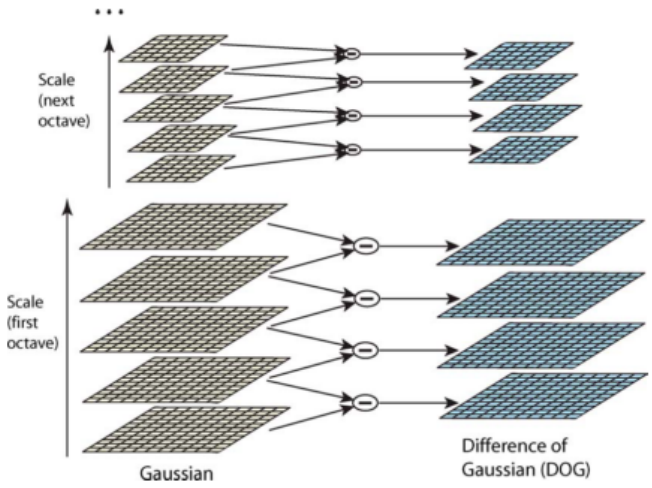


Figura 3.6: Construção do espaço de escalas no algoritmo SIFT [49] (Courtesia do Autor).

A invariância à escala é conseguida pela utilização do filtro DoG, uma vez que, a diferença Gaussiana proporciona uma aproximação a $\sigma^2\nabla^2 G$. Lindeberg [50] mostrou que é necessária uma normalização laplaciana com um fator σ^2 para uma invariância à escala. Em Mikolajczyk [51], numa comparação experimental, demostrou que encontrando o máximo e mínimo de $\sigma^2\nabla^2 G$ conseguem-se encontrar pontos de interesse mais estáveis, comparativamente à utilização de outros métodos, como Hessian ou Harris.

Segundo [49], a relação entre **D** e $\sigma^2\nabla^2 G$, fica compreendida pela seguinte equação:

$$\sigma^2\nabla^2 G = \frac{\partial G}{\partial \sigma} \approx \frac{G(x, y, k\sigma) - G(x, y, \sigma)}{k\sigma - \sigma} \tag{3.2.4}$$

onde,

$$G(x, y, k\sigma) - G(x, y, \sigma \approx (k-1)\sigma^2\nabla^2 G \tag{3.2.5}$$

Por fim, a localização dos potencias pontos chaves é conseguida através da pesquisa dos máximos e mínimos de $D(x, y, \sigma)$. Cada suposto ponto de interesse será comparado com os seus 8 vizinhos na mesma imagem e com os seus 9 vizinhos nas imagens imediatamente acima e abaixo, só serão selecionados se forem maior ou menores que todos os seus vizinhos comparados.

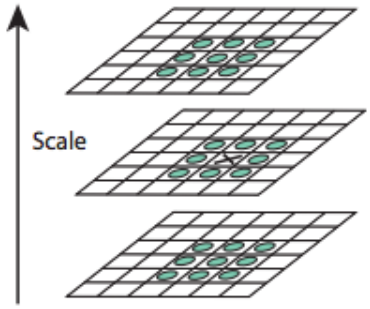


Figura 3.7: Construção do espaço de escalas no algoritmo SIFT [49] (Courtesia do Autor).

3.2.1.1.2 Localização Precisa dos Pontos Chave

Após a localização dos potenciais pontos de interesse, um elevado número de pontos são encontrados. De forma a tornar o algoritmo mais eficiente, muitos deles devem ser eliminados. Para isso, realiza-se um refinamento desses valores através da aplicação da Série de Taylor, de forma a obter resultados mais robustos e confiáveis. Após isto, são eliminados os pontos de baixo contraste (mais instáveis a ruído), como também os pontos localizados sobre uma aresta.

3.2.1.1.3 Definição da Orientação

Uma vez conhecida a localização dos pontos de interesse, o passo seguinte passa por determinar a sua orientação(θ) e magnitude (m). Esta fase do algoritmo é essencial uma vez que, permite que este seja invariante à rotação. Conhecida a localização e a escala, é possível selecionar uma imagem L suavizada, a partir da pirâmide Gaussiana criada anteriormente. Dado isto, as equações para o cálculo da magnitude e da orientação, são dados por:

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y)^2 + L(x,y+1) - L(x,y-1)^2)} \tag{3.2.6}$$

$$\theta(x,y) = \tan^{-1}(L(x,y+1) - L(x,y-1)/L(x+1,y) - L(x-1,y)) \tag{3.2.7}$$

Após o cálculo da orientação e da magnitude, esta informação é organizada num histograma de orientações, formado por 36 campos, o que cobre uma faixa de 360 graus de orientações. Relativamente às magnitudes dos gradientes, estas são ponderadas tendo em consideração uma janela circular Gaussiana, com uma variância, σ, 1.5 vezes superior à da escala do ponto de interesse. Posteriormente, é

identificada e atribuída ao ponto de interesse a orientação mais proeminente, que corresponde à maior barra do histograma de orientações. Todos as orientações, acima de 80% são também convertidas em pontos de interesse.

3.2.1.1.4 Descrição dos Pontos de Interesse

A última etapa do algoritmo consiste na criação dos descritores. Para isso, uma janela 16x16 pixéis é criada na vizinhança de um ponto de interesse. Esta é dividida em 16 blocos de janelas 4x4, onde para cada bloco são calculadas as magnitudes e orientações dos gradientes. Esta informação é armazenada num histograma de 8 barras, para cada uma das janelas 4x4, representado na Figura 3.8, o comprimento de cada seta corresponde à soma da magnitude dos gradientes nesse sentido. No total, para cada ponto de interesse é obtido um vetor descritor de 128 posições.

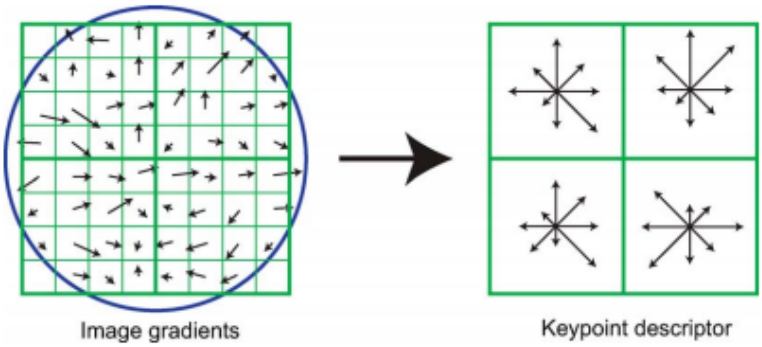


Figura 3.8: Processo de descrição de pontos de interesse do algoritmo SIFT [49] (Courtesia do Autor).

3.2.1.2 SURF

O *SURF*, *Speeded Up Robust Features*, é também um algoritmo robusto para a deteção e descrição de pontos de interesse numa imagem. Foi apresentado por Herbert Bay et al na conferencia ECCV em 2006, sendo publicado posteriormente em 2008 [52]. O algoritmo *SURF* foi inspirado no algoritmo *SIFT* mantendo algumas características, como por exemplo, a invariância à escala, bem como à rotação. Segundo os seus autores este consegue encontrar pontos de interesse mais significativos e mais rápido em comparação ao antecessor *SIFT*. Para isso, o algoritmo faz uso do conceito de imagens integrais [69] de forma a reduzir o tempo de computação, pois estas possibilitam uma rápida computação de convoluções baseadas em filtros de caixa.

O conceito de imagem integral consiste em aplicar a seguinte equação a uma imagem de entrada:

$$I_{\sum}(x,y) = \sum i = 0^{x} \sum_{j=0}^{y} I(i,j) \tag{3.2.8}$$

A função anterior na prática consiste em criar uma segunda imagem, a partir da imagem inicial, no qual cada pixel é o somatório de todos os pixéis antecessores em x e y, conceito este ilustrado pela Figura 3.9.

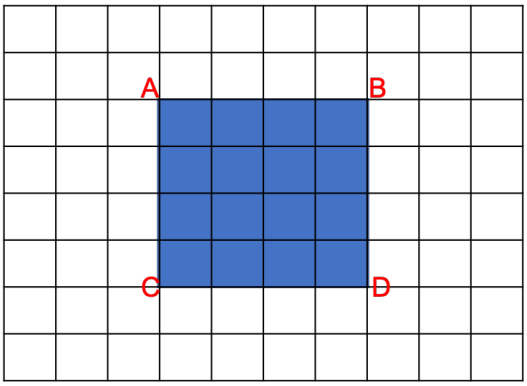


Figura 3.9: Ilustração do Conceito de Imagem Integral.

Após o cálculo da imagem integral de uma determinada imagem de entrada, com apenas quatro operações de leitura e três cálculos é possível calcular a área retangular para qualquer tamanho da imagem, como demonstra a seguinte equação:

$$I_{\sum}(ABCD) = I_{\sum}(A) + I_{\sum}(D) - I_{\sum}(B) - I_{\sum}(C) \tag{3.2.9}$$

Outro conceito importante em visão por computador e bastante utilizado neste algoritmo, é o de filtro de caixa, também conhecido por máscara ou *kernel*, consiste na aplicação de uma determinada matriz, que pode variar consoante o objetivo desejado, esta é aplicada através da operação de convolução, permitindo realçar alguma característica numa imagem original.

Segundo o trabalho apresentado em [52] o algoritmo *SURF* é dividido em duas fases distintas:

1. Deteção dos pontos de interesse, através do *Fast-Hessian Detector*

2. Descrição dos Pontos de Interesse.

Serão agora detalhadas as duas fases do algoritmo.

3.2.1.2.1 Deteção do Pontos de Interesse

A primeira fase do algoritmo *SURF* consiste na identificação do pontos de interesse, para isso baseia-se no cálculo da matriz de *Hessian*. Para uma dada função arbitrária *f(x,y)*, o determinante da matriz de *Hessian* é dado pela seguinte equação:

$$det(f(x,y)) = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix} = \frac{\partial^2 f}{\partial x^2}\frac{\partial^2 f}{\partial y^2} - (\frac{\partial^2 f}{\partial x \partial y})^2 \tag{3.2.10}$$

O determinante da matriz de *Hessian* é o produto dos seus *eigenvalues* que, consoante o seu sinal, permite classificar o ponto como sendo um valor extremo ou não. Se o determinante for negativo, quer dizer que os *eigenvalues* têm sinais diferentes, logo o ponto não é um extremo, se por outro lado for positivo, significa que os *eigenvalues* têm valores iguais e então o ponto é um valor extremo [**?**].

No âmbito do processamento de imagem, e mais precisamente no algoritmo *SURF*, a equação *f(x,y)* é substituída pela função *I(x,y)* que representa a intensidade de um determinado pixel numa imagem. As derivadas de segunda ordem representadas na equação 3.2.10, no caso específico do *SURF* são conseguidas aplicando um filtro Gaussiano normalizado de segunda ordem à imagem. Isto é conseguido através de operações de convolução e dos conceitos de filtros de caixa e imagens integrais, que permitem acelerar o processo. Assim a matriz de *Hessian* para um ponto *x=(x,y)* a uma escala σ é representada da seguinte forma:

$$H(x,\sigma) = \begin{bmatrix} L_{xx}(x,\sigma) & L_{xy}(x,\sigma) \\ L_{xy}(x,\sigma) & L_{yy}(x,\sigma) \end{bmatrix} \tag{3.2.11}$$

Onde $L_{xx}(x,\sigma)$ representa a convolução da derivada de segunda ordem $\frac{\partial^2}{\partial x^2}g(\sigma)$ com a imagem I num ponto x, o mesmo para $L_{xy}(x,\sigma)$ e $L_{yy}(x,\sigma)$. Assim, pode ser calculado o determinante da matriz de *Hessian* para cada pixel e utilizar o valor para encontrar os pontos de interesse. Contudo, as derivadas de segunda ordem, devem ser discretizadas e cortadas antes de serem aplicadas [52], como representa a Figura 3.10.

Dado isto, o determinante da matriz de *Hessian* utilizando as aproximações Gaussianas e os filtros de caixa representados anteriormente, é apresentado pela seguinte equação:

$$det(H) = D_x x D_y y - (0.9 D_x y)^2 \tag{3.2.12}$$

Calculando o determinante de *H* e encontrando o máximo, este pode corresponder a ponto de interesse. Posto isto, é utilizado o conceito de espaço de escalas e de pirâmide para encontrar os pontos de

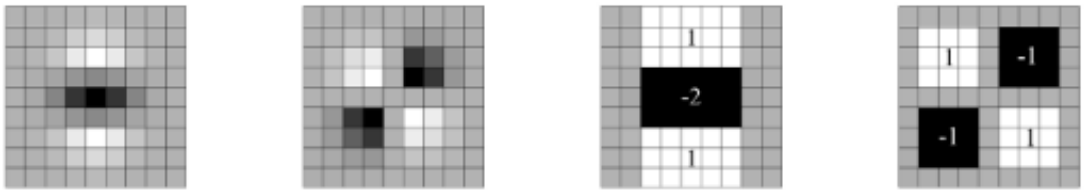


Figura 3.10: Da esquerda para a direita: A primeira metade representa a derivada parcial gaussiana de segunda ordem na direção *y* e *xy*, respetivamente. A segunda metade representa as suas aproximações, respetivamente, utilizando filtros de caixa.[52] (Courtesia do Autor).

interesse invariantes à escala. O mesmo conceito já era utilizado no algoritmo *SIFT*, apresentado anteriormente, onde o tamanho da imagem era variado e aplicado um filtro Gaussiano. Contrariamente ao que acontece no algoritmo SIFT em que a pirâmide é conseguida fazendo variar o tamanho da imagem e posteriormente aplicando um filtro Gaussiano na própria, obtendo-se assim uma pirâmide em que a base é formada pela maior imagem e o topo pela menor. No algoritmo SURF o método é ligeiramente diferente, sendo utilizados filtros 9x9, onde consecutivamente vai sendo aumentado, criando um *upscaling* à imagem. Formando, desta forma, uma pirâmide onde a base é referente à imagem mais pequena e o topo à imagem maior.

Após definida a pirâmide de escalas, segue-se a deteção efetiva dos pontos de interesse para isso é aplicado um *threshold* adequado para frisar os pontos de interesse mais fortes. Após estes pontos de interesse considerados mais fortes serem referenciados, serão comparados com os seus 26 vizinhos, os 8 da mesma escala e os 18 da escala superior e inferior.

3.2.1.2.2 Descrição dos Pontos de Interesse

Para obter a invariância à rotação, o algoritmo *SURF* associa a cada ponto de interesse uma orientação dominante. Para determinar esta orientação, é utilizado o cálculo da resposta da transformada de *Haar*, que na prática consiste em aplicar os filtros de *HaarWavelet*, representados na Figura 3.11, numa dada imagem. Estes filtros permitem encontrar gradientes em ambas as direções dos eixos coordenados *x* e *y* de uma dada imagem. Por sua vez, associados ao conceito de imagem integral permite diminuir bastante os custos de computação do algoritmo.

O primeiro passo na deteção da orientação dominante é então o cálculo da resposta ao filtro *Haar wavelet* para cada pixel compreendido numa área circular com raio máximo de 6σ. O filtro utilizado tem tamanho de 4σ, onde σ representa a escala no qual foi encontrado o ponto de interesse. Cada ponto

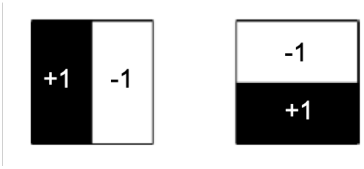


Figura 3.11: Ilustração dos Filtros *Haar Wavelets*. O da esquerda calcula a resposta na direção *x*, enquanto que o da direita calcula da direção y.

é ponderado, baseado na sua distância ao ponto de interesse, utilizando uma função Gaussiana. Para o cálculo final da orientação é utilizada uma janela deslizante de tamanho $\frac{\pi}{3}$ a percorrer a totalidade do círculo. Para cada intervalo de janela deslizante são somadas todas as respetivas respostas criando, assim, um novo vetor. A orientação do vetor mais longo, torna-se a orientação do ponto de interesse.

Para a descrição final dos descritores, após a determinação da orientação dominante, é construída uma janela quadrada à volta dos pontos de interesse com orientação igual à determinada. A janela é sub-dividida em 4x4 regiões onde são calculadas as transformadas de *Haar* para 25 pontos de amostragem, obtendo-se:

$$v_{subregiao} = [\sum dx, \sum dy, \sum |dx|, \sum |dy|] \tag{3.2.13}$$

O descritor obtido é composto por um vetor com 64 (4x4x4) elementos, uma vez que, cada sub-região contribui com 4 valores para o descritor final. Além do mais é invariante às rotações e à escala, bem como às diferenças de brilho e contraste.

3.2.1.3 ORB

O algoritmo ORB surge como objetivo de ser uma alternativa ao SIFT e ao SURF, propondo melhor desempenho computacional [56]. Foi pensado para ser utilizado em aplicações em tempo real, principalmente com o seu foco para dispositivos com baixo poder de processamento e sem aceleração por GPU. Deste modo, este algoritmo é uma fusão entre o detetor FAST [57] e o descritor BRIEF [58].

Numa primeira fase, é utilizado o detetor FAST para localizar os pontos de interesse, e, em seguida, aplicado um filtro de cantos de Harris para separar os melhores pontos encontrados. Além disto, o algoritmo utiliza um esquema de pirâmide para tratar das questões da escala. Contudo, a grande desvantagem do detetor FAST é a não invariância às rotações, o que significa que não suporta variações na rotação da imagem.

Na fase final, para computar os descritores dos pontos de interesse é utilizado o descritor BRIEF.

3.2.2 Geometria Epipolar

Um dos principais conceitos a ter em consideração quando se abordam técnicas baseadas em visão estéreo é o termo de *geometria epipolar*. Como já foi abordado anteriormente, as técnicas baseadas em visão estéreo, têm como principal característica a análise de imagens da mesma cena, mas adquiridas de perspectivas diferentes. Assim, a *geometria epipolar* [36, 38] é a relação geométrica da interseção entre os planos das imagens com o conjunto de *planos epipolares*, tendo a *baseline* (linha que une os centros ópticos das câmaras) como eixo.

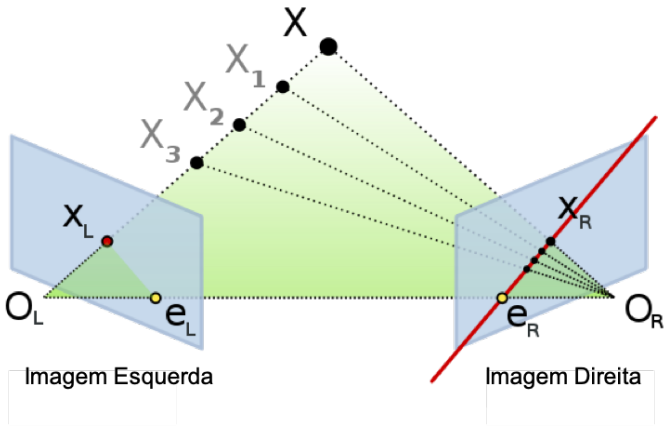


Figura 3.12: Representação da Geometria Epipolar[4].

Na Figura 3.12 estão representados todos os elementos relacionados com a *geometria epipolar*.

- Os epipolos e_L e e_R são os pontos de interseção entre a *baseline* e os planos da imagens e representam respetivamente a projeção dos centros ópticos, O_L e O_R, na imagem contrária.
- A interseção entre a *baseline* e os raios ópticos formados pela projeção do ponto *X* nos planos da imagem, respetivamente nos pontos X_L e X_R é denominado de plano epipolar, e está representado a verde.
- As linhas epipolares são formadas pela interseção do plano epipolar com os planos da imagens direita e esquerda, respetivamente, no caso específico da Figura 3.12, está representada a vermelho na imagem Direita.

As linhas epipolares, na prática, representam mais do que apenas a interseção entre o plano epipolar e o plano da imagem. A linha epipolar é a projeção do raio ótico formado entre o ponto tridimensional e o centro ótico da câmara, no plano da imagem contrário. Desta forma, é possível afirmar que a projeção do ponto no outro plano da imagem encontra-se localizado ao longo da linha epipolar contrária.

Resumindo, a geometria epipolar estabelece a relação geométrica entre um ponto de uma imagem e uma linha em outra imagem, o que permite, além de facilitar a recuperar da informação de profundidade, também diminuir a área de pesquisa de um par de pontos homólogos, uma vez que, eles têm que respeitar a restrição. Os pontos só são considerados correspondentes *se e só se* estiverem contidos nas linhas epipolares correspondentes no plano de imagem contrário, ou seja, o ponto tridimensional *X*, projetado nos planos de imagem por X_L e X_R estão contidos respetivamente nas linhas epipolares. A este conceito chama-se *restrição epipolar*. Por fim, todas as linhas epipolares intersetam-se no epipolo de cada plano de imagem, e_L e e_R respetivamente para a imagem esquerda e direita.

3.2.3 Matriz Essencial

A matriz essencial é uma matriz de tamanho 3x3 e descreve as propriedades relativas à correspondência de pontos em um conjunto de imagens estéreo assumindo que as câmaras utilizadas satisfazem o modelo *pin-hole*. Esta matriz é uma especialização da matriz Fundamental que, por sua vez, é uma representação algébrica da geometria epipolar, que representa um mapeamento projetivo de pontos correspondentes de uma imagem para linhas epipolares de outra [61]. Por outras palavras, permite validar o conceito de restrição epipolar.

A geometria epipolar através do uso da matriz essencial é descrita através da seguinte equação [62]:

$$[p_2; 1]^T = K^{-T} E K^{-1} [p_1; 1] = 0 \tag{3.2.14}$$

Onde *E* representa a matriz essencial, p_1 e p_2 representam respetivamente os pares de pontos correspondentes referentes a duas imagens distintas, e *K* representa a matriz dos parâmetros intrínsecos da câmara no qual foram adquiridas as imagens.

Relativamente à biblioteca OpenCV, esta contem uma função para o cálculo da matriz essencial. Para isso, faz uso do conjunto de pares de pontos correspondentes, referente a um par de imagens, e da matriz de calibração. Esta função pertence ao módulo *Calib3d* e é denominada *findEssentialMat()* e está representada no Algoritmo 2. A confiança é um parâmetro da função que permite iterar o cálculo da matriz essencial enquanto a confiança desejada não for atingida. Após calculada a matriz essencial é possível obter a matriz dos parâmetros extrínsecos, caracterizada por uma matriz de rotação e um vetor de translação.

[4] Imagem retirada de:`https://upload.wikimedia.org/wikipedia/commons/1/14/Epipolar_geometry.svg`

Algoritmo 2 Algoritmo Cálculo da matriz essencial (<=Confiança)

1. Selecionar aleatoriamente 5,7 ou 8 pares de pontos de interesse.
2. Calcular a matriz essencial
3. Determinar os pontos *inliers* e *outliers* através do algoritmo RANSAC.
4. Refinar a matriz essencial considerando todos os pontos *inliers*.

3.2.4 Algoritmo RANSAC

Os autores *Fichler and Bolles* propuseram um método iterativo usado para estimar os parâmetros de um determinado modelo matemático, relativo a um conjunto de dados que contem *outliers*, denominado *RANSAC* [59]. O seu principal objetivo passa por descobrir os parâmetros de determinado modelo matemático, válidos para a maioria dos pontos pertencentes a um conjunto total e, desta forma, descartar os pontos que contêm ruído. No Algoritmo 3 [60] é apresentado de forma resumida este algoritmo.

Algoritmo 3 Algoritmo RANSAC

1. Selecionar aleatoriamente um número de pontos necessários para determinar os parâmetros do modelo.
2. Resolver o modelo tendo em consideração os pontos escolhidos.
3. Determinar quantos pontos do conjunto inicial se ajustam ao modelo tendo em consideração uma tolerância predefinida.
4. Se a fração do número de *inliers* sobre o número total dos pontos do conjunto exceder um limite predefinido, devem ser re-estimados os parâmetros do modelo novamente, tendo em consideração *inliers* identificados.
5. Repetir as etapas 1 a 4 (0 máximo de N vezes)

Este algoritmo é utilizado em duas funções da biblioteca OpenCV utilizadas regularmente ao longo deste trabalho, são elas *findEssentialMat()* e *solvePnpRansac()*.

3.2.5 Algoritmo ICP

O algoritmo *Iterative Closest Point* [55] permite determinar iterativamente uma transformação, caracterizada por uma rotação e por uma translação, entre duas nuvens de pontos. Este algoritmo está assim também associado à função *solvePnpRansac* pertencente à biblioteca OpenCV. De forma, a melhorar os cálculos das correspondência é também associado o algoritmo *RANSAC* à respetiva função

OpenCV.

Dados os pontos, representados por:

$$X = x_1, ..., x_n \tag{3.2.15}$$

e,

$$P = p_1, ..., p_n \tag{3.2.16}$$

é possível obter a transformação, representada pela translação t e rotação R que minimiza a soma dos erros quadráticos entre os pontos, através da equação [54]:

$$E(R,t) = \frac{1}{N_p} \sum_{i=1}^{N_p} ||x_i - R_{p_i} - t||^2 \tag{3.2.17}$$

onde x_i e p_i representam os pontos correspondentes.

Sendo o cálculo das matrizes de rotação e translação muito úteis ao longo do processo de reconstrução tridimensional, uma vez que, permite posteriormente o cálculo das matrizes de projeção associadas, que por sua vez, permite avançar com o processo de triangulação.

3.2.6 Triangulação

Uma vez conhecidos os parâmetros extrínsecos da câmara (*R* e *t*), calculados através da decomposição da matriz essencial, e conhecido também a matriz dos parâmetros intrínsecos(K), calculada através do processo de calibrarão, é possível triangular um par de pontos através das suas coordenadas no plano da imagem. Nas equações seguintes estão representadas as matrizes de projeção:

$$P_1 = K[I|0] \tag{3.2.18}$$

$$P_2 = K[R|t] \tag{3.2.19}$$

onde I é uma matriz identidade, 0 é um vetor coluna , R é uma matriz que representa a rotação relativa entre a primeira e a segunda câmara, e t é um vetor coluna que representa a translação relativa entre câmaras. As equações 3.2.18 e 3.2.19 representam a triangulação referente ao primeiro par de imagens de uma sequência, devido a isso, é utilizada a matriz identidade I e o vetor de translação a 0.

Para a realização de uma triangulação referente aos pontos correspondentes de um determinado par de imagens é utilizada a função *triangulatePoints()* presente na biblioteca OpenCV. A função recebe

como parâmetros de entrada as respetivas matrizes de projeção, bem como os pontos correspondentes em cada uma das imagens. O processo matemático implícito no processo de triangulação está devidamente descrito no trabalho [38].

Capítulo 4

Bibliotecas utilizadas

Como já foi referido anteriormente, a aplicação a desenvolver terá como base o sistema operativo Android, logo torna-se fundamental compreender os conceitos básicos de desenvolvimento/funcionamento de uma aplicação baseada neste sistema operativo. Além disto, são também apresentadas outras bibliotecas que foram utilizadas na fase de implementação desta dissertação.

4.1 Plataforma Android

O Android é um sistema operativo (SO) para dispositivos móveis desenvolvido originalmente por uma empresa denominada de *Android Inc.*. Posteriormente comprada pela *Google*, que em 2007 criou a *Open Handset Alliance*, que envolvia um consórcio de empresas em que o seu foco de trabalho eram as tecnologias *mobile*. Neste mesmo ano foi realizada a apresentação pública do Android OS e lançada a versão beta do Android SDK. Em outubro de 2008 foi lançado comercialmente o primeiro telemóvel a correr o Android OS (versão 1.0), sendo este dispositivo móvel um T-Mobile's G1 HTC Dream [63]. Atualmente, o Android encontra-se na versão 10.0.

O sistema operativo rapidamente cresceu, sendo adotado por um elevado número de marcas, que o colocaram nos seus dispositivos móveis. Inicialmente foi desenvolvido exclusivamente para o uso em telemóveis, mas tendo em conta a sua grande aceitação e também a sua natureza personalizável, permitiu que este sistema operativo tenha vindo a ser portado para outros tipos de plataformas. Atualmente, existem versões do Android utilizadas nas mais diversas plataformas, como por exemplo, em televisões, relógios inteligentes, carros e até em placas de desenvolvimento na área da IoT. Tendo inclusivamente dado origem a novas vertentes do sistema operativo, tais como, Wear OS [64], para relógios inteligentes, Android TV [65], para televisões, Android Auto [66], para carros ou Android Things [66], para IoT.

4.1.1 Arquitetura do Sistema Operativo Android

O sistema operativo Android tem como base o kernel do Linux e tem a particularidade de ter implementada uma máquina virtual que permite executar aplicações nativas em Java. Dado isto, a sua linguagem oficial para o desenvolvimento de aplicações é linguagem Java.

A arquitetura do sistema operativo Android [68] está organizada num conjunto de camadas distintas, (1) *Linux Kernel*; (2) *Hardware Abstraction Layer (HAL)* (3) Bibliotecas C/C++ Nativas e *Android runtime* (4) *Java API Framework*; e (5) Aplicações do Sistema, como é representado na Figura 4.1.

A camada base é onde está localizado o sistema operativo propriamente dito, este é baseado no *Kernel* do *Linux*, como já foi referido anteriormente. Esta camada é responsável pelos serviços de baixo nível, como por exemplo, a gestão de memória e processos ou a interface com o *hardware* através da implementação das *drivers*.

A camada de abstração de *hardware* permite expor as capacidades de *hardware* do dispositivo para a camada *Java API Framework* de forma transparente, desta forma é possível ao Android ser utilizado em plataformas com *hardware* distinto.

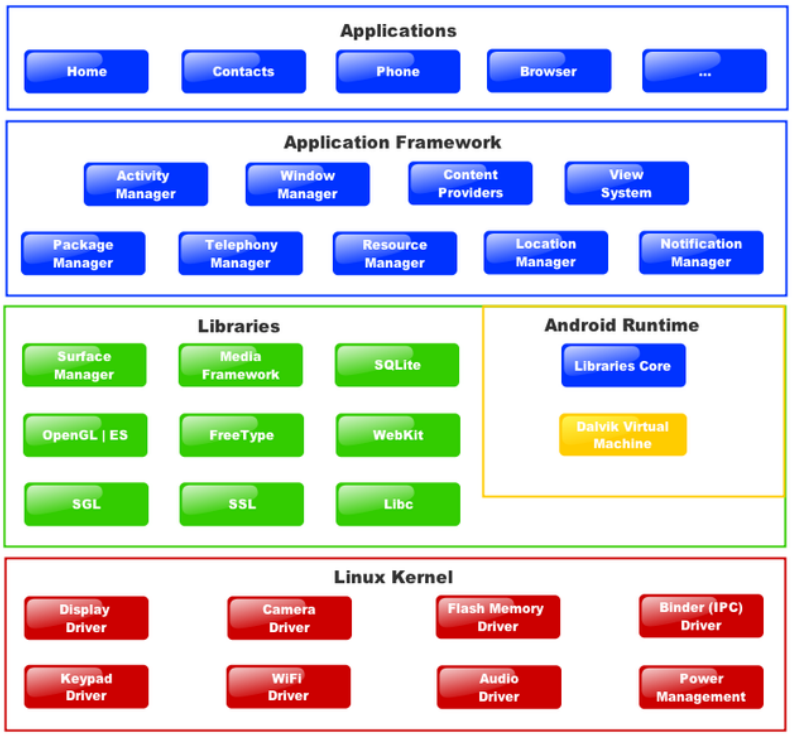


Figura 4.1: Arquitetura do sistema operativo Android [1].

[1]Imagem retirada de: `https://upload.wikimedia.org/wikipedia/commons/a/a5/Diagram_android.png`

Na terceira camada estão implementadas as bibliotecas nativas que são implementadas em código C/C++, como por exemplo, algumas bibliotecas de suporte para diversos tipos de formatos de áudio e vídeo, bem como a biblioteca OpenGL/ES. Além disso, nesta camada está também implementada o código referente às máquinas virtuais Java. A partir da versão 5.0 do Android começou a ser utilizado o sistema *ART* invés do *Dalvik* para a máquina virtual Java. É importante ter em conta, que é possível acessar às bibliotecas nativas presentes nesta camada em aplicações desenvolvidas na última camada. Para isso, é utilizado o *Android NDK*.

Na camada *Java API Framework* estão implementadas as APIs de desenvolvimento Android, que são utilizadas pelas aplicações na última camada. Estas APIs implementam a leitura de sensores, o acesso a serviços do sistema operativo através da linguagem Java. Algumas bibliotecas nativas C/C++ podem ser acessadas de através da sua implementação nesta camada.

A última camada representa a camada das aplicações, onde estão presentes as aplicações de uma forma geral, quer sejam elas nativas ou desenvolvidas por terceiros.

4.1.2 Requisitos para Desenvolvimento

A primeira coisa a ter em consideração no desenvolvimento de uma aplicação, independentemente da plataforma para qual se esteja a desenvolver, é a preparação do ambiente de desenvolvimento.

No caso específico de desenvolvimento de aplicações para dispositivos Android é necessário ter em conta a instalação de alguns pacotes básicos. Tendo isto em consideração, a *Google* fornece um conjunto de pacotes que preparam o ambiente de trabalho para o desenvolvimento de aplicações nativas para dispositivos Android. Um destes pacotes é o *Android SDK*, ou *Android Software Development Kit*, este contem um conjunto de bibliotecas e ferramentas para o desenvolvimento e compilação de aplicações nativas, utilizando a linguagem Java. Além disso, fornecem um conjunto de emuladores que permitem que as aplicações sejam testadas sem necessidade do uso de um dispositivo Android físico. É necessário ter em conta que este pacote é de uso obrigatório, sendo que não é possível o desenvolvimento de aplicações Android sem a sua instalação.

Outro pacote que ao contrário do anterior não é de uso obrigatório, mas sim dependente da necessidade da aplicação é o *Android NDK* ou *Android Native Development Kit*. Este pacote engloba um conjunto de ferramentas que permitem o desenvolvimento e compilação de bibliotecas ou partes de código que sejam implementados na linguagem C/C++. Isto permite, um elevado nível de reutilização, uma vez que podem ser compiladas bibliotecas desenvolvidas em C/C++ para outras plataformas.

Apesar de não ser obrigatório, visto que o pacote *Android SDK* disponibiliza todas as ferramentas

para a compilação através linha de comando, existe uma necessidade de utilizar um IDE (*Integrated Development Enviroment*) específico para o desenvolvimento de aplicações Android, de forma a agilizar todo processo de desenvolvimento. Existem então duas alternativas: o pacote *Android DeveloperTools* (ADT) ou o *Android Studio*. No caso específico desta dissertação foi utilizado o *Android Studio*.

4.1.3 Componentes básicos de uma Aplicação

As aplicações Android são compostas normalmente por um ou mais tipos de componentes. Normalmente são considerados quatro tipos de componentes básicos no desenvolvimento de aplicações Android, sendo eles:

- *Activity*
- *Service*
- *BroadcastReceiver*
- *ContentProveider*

De uma forma geral, as *atividades* são responsáveis pela interface gráfica para com o utilizador; os *services* implementam operações com execução em *background*; os *BroadcastReceivers* respondem a eventos do sistema; e os *ContedProviders* armazenam e/ou fornecem dados para a aplicação.

Outro componente fundamental, presente em todas as aplicações para o sistema operativo *Android* é o ficheiro manifesto ou *AndroidManifest.xml*. Este é um arquivo XML, onde estão declarados todos os componentes e requisitos de uma aplicação, bem como qualquer configuração de *hardware*. É importante também incluir aqui os *Fragmentos* e os *DialogFragmetos* permitem o desenho de interfaces gráficas mais versáteis.

4.2 Qt

O Qt [70] é uma framework de desenvolvimento de aplicações multiplataforma para desktop, sistemas embebidos e dispositivos móveis e é suportado em plataformas, como Linux, OS X, Windows, Android, iOS. É principalmente utilizado no desenvolvimento de interfaces gráficas e tem como principal característica o poder de compilar códigos para diversas plataformas distintas. O Qt não é considerado uma linguagem de programação, nativamente ele utiliza o C++ padrão e várias macros adicionais para o desenvolvimento

de código. O pré-processador MOC, analisa antes da etapa da compilação os arquivos escritos em *Qt-extended* C++ e gera código C++. Assim pode ser compilado por qualquer compilador C++ padrão, tais como GCC, MinGW e MSVC. A plataforma Qt utiliza o *IDE Qt Creator* para o desenvolvimento de código, bem como o *Qt Desing* para o desenvolvimento de interfaces gráficas. Permite também a criação de interfaces gráficas, a partir de uma linguagem própria, denominada *QML*, *Qt Meta Language*, sendo esta uma linguagem de programação declarativa.

4.3 OpenCV

A biblioteca OpenCV (*Open Source Computer Vision Library*) [71] foi desenvolvida inicialmente pela Intel com o objetivo de fornecer uma estrutura comum para o desenvolvimento de aplicativos de visão por computador e acelerar, assim, o crescimento da perceção em máquinas através do desenvolvimento de aplicações comerciais. A biblioteca é composta por mais de 2500 algoritmos optimizados, incluíndo funções que, de forma geral, cobrem as principais tarefas relacionadas ao estado da arte da visão por computador e *machine learning*. Os algoritmos são normalmente utilizados em tarefas, como por exemplo, reconhecimento facial, identificação de objetos , seguimento de objetos em movimentos, criação de nuvens de pontos tridimensionais, entre outras tarefas relacionadas a áreas acima referidas. Atualmente, é disponibilizada sob uma licença BSD, *Berkley Software Distribution (BSD)*, por isso, é livre para uso académico e comercial. Atualmente possui interfaces para as linguagens C, C++, Python e Java e está disponível para as plataformas Windows, Os X e Linux, bem como para Android. Esta biblioteca possui módulos de processamento de imagem, video I/O, estrutura de dados, álgebra linear e interface gráfica com o utilizador, entre outros. Permite a compilação individual de cada um dos módulos, facilitando assim a sua utilização em plataformas com recursos mais limitados. Atualmente, a biblioteca OpenCV já permite o processamento paralelo a partir da GPU, *Graphics Processing Unit*, através de biblioteca específicas, como por exemplo, CUDA e OpenCL.

A versão utilizada nesta dissertação é a 3.4.1, sendo que a versão mais atual é a 4.1.1. É importante frisar que os algoritmos *SIFT* e *SURF* utilizados neste projeto fazem parte de um módulo extra, não incluído na versão principal, este deve ser compilado separadamente e posteriormente incluído à versão base. Este módulo é denominado OpenCV Contrib.

4.4 PCL

A PCL [72] é uma biblioteca *open-source*, desenvolvida pela *Willow Garage*, que contém um conjunto de algoritmos que permitem o processamento de nuvens de pontos e geometrias tridimensionais na área da visão por computador. Esta foi desenvolvida em C++ e é distribuída sob os termos da licença da *Berkley Software Distribution (BSD)*. Está disponível para a grande maioria dos sistemas operativos presentes atualmente no mercado, como por exemplo, *Windows*, *MacOS*, *Linux* ou *Android*.

A biblioteca PCL está divida num conjunto de módulos, os quais podem ser compilados individualmente, permitindo o seu uso em plataformas com recursos computacionais mais limitados. A biblioteca é constituída pelo seguintes módulos, *common*, *features*, *filters*, *io*, *kdtree*, *keypoints*, *registration*, *sample_consensus*, *segmentation*, *surface* e *visualization*. No caso especifico desta dissertação foram utilizados os módulos *common* e *io*. O primeiro contém entre outras as estruturas de dados que representam os pontos tridimensionais, como por exemplo, *PointXYZRGB* ou *PointXYZ*. Já o módulo *io* é responsável pelas funções que gerem a importação e exportação das nuvens de pontos para os formatos de ficheiros suportados pelo PCL, no caso, o PCD e o PLY.

4.5 OpenGL

O OpenGL [73] é uma biblioteca com funções utilizadas para a computação gráfica, com o intuito de agilizar o desenvolvimento de aplicações gráficas, quer para conteúdos 2D como para conteúdos 3D. Esta biblioteca tem implementações para a grande maioria do sistemas operativos presentes no mercado, sendo os principais, Windows, Mac, Linux, Android e iOS. Atualmente, tem interface para as seguintes linguagens de programação C, C++, Fortran, Ada e Java. O OpenGL fornece um conjunto de funções que permite auxiliar na construção de matrizes de visualização, projeção e no desenho das superfícies 3D, tirando partido também das GPUs incluídas nas plataformas no qual está a correr, e, desta forma, permitir uma aceleração gráfica nas suas renderizações.

Capítulo 5

Implementação do Sistema

Neste capítulo será apresentado o trabalho desenvolvido ao longo deste projeto, bem como, descritas detalhadamente as diferentes fases da sua implementação. De uma forma resumida, ao longo deste projeto, foi desenvolvida uma aplicação Android com foco na reconstrução tridimensional e uma aplicação gráfica com o intuito de verificar os algoritmos de processamento de imagem. Além disso, foi também construída uma plataforma rotativa de apoio à aplicação Android.

5.1 Arquitetura do Sistema

A implementação deste projeto foi dividida em duas fases distintas, a primeira, com foco na implementação dos algoritmos de processamento de imagem, sustentados pela fundamentação teórica da reconstrução tridimensional, apresentada anteriormente. Já a segunda fase, focou-se essencialmente no desenvolvimento de uma aplicação para a plataforma Android e na consequente integração dos algoritmos desenvolvidos na primeira fase.

Dado isto, torna-se fulcral abordar dois outros aspetos que justificaram a criação de uma segunda aplicação gráfica. O primeiro aspeto está relacionado com as linguagens de programação utilizadas no desenvolvimento, quer da aplicação Android, quer dos algoritmos de processamento de imagem. O desenvolvimento para a plataforma Android, é, maioritariamente, utilizada a linguagem Java, dado ser a mais documentada e suportada. No entanto, e apesar de existirem bibliotecas de processamento de imagem nativas para a plataforma Android, foi decidido utilizar a biblioteca OpenCV, desenvolvida em C/C++. A escolha desta biblioteca deve-se ao facto de ser amplamente utilizada em projetos de processamento de imagem e consequentemente ter ao dispor uma boa base de documentação e suporte.

O segundo aspeto, e também importante para a criação de uma segunda aplicação, deve-se prin-

cipalmente à necessidade de agilizar a implementação dos algoritmos de reconstrução tridimensional e facilitar o seu respectivo *debug*. Isto deve-se ao facto do IDE de desenvolvimento, *Android Studio*, não permitir de forma eficiente o desenvolvimento de grandes quantidades de código C/C++, visto ser mais focado para o desenvolvimento em Java. Para finalizar, é importante referir que a aplicação gráfica C/C++ foi desenvolvida através da IDE de desenvolvimento Qt.

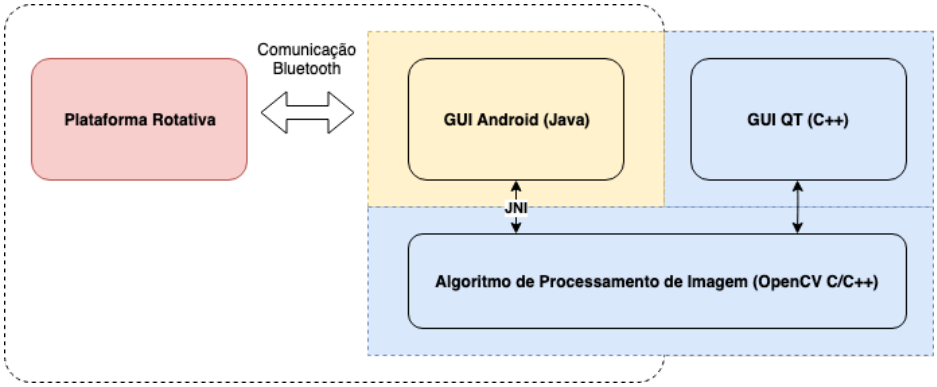


Figura 5.1: Vista Geral do Sistema Desenvolvido.

O sistema é então composto por uma aplicação móvel, desenvolvida para a plataforma Android que partilha com a aplicação gráfica desenvolvida em Qt a parte de processamento de imagem e algoritmos de reconstrução tridimensional. A aplicação Android tem a particularidade de poder comunicar via Bluetooth com uma plataforma rotativa.

Visto que se optou pela implementação de duas aplicações de alto nível distintas, a principal exigência na fase de desenvolvimento focou-se, essencialmente, na facilidade de portar o código de baixo nível de uma aplicação para a outra. Resumidamente, em cada aplicação existem duas camadas diferentes, em que uma gere a parte relacionada com a interface gráfica e interação com o utilizador e a outra foca-se principalmente na parte de implementação dos algoritmos de processamento de imagem que, por sua vez, é partilhada pelas duas aplicações, como é possível observar na Figura 5.1.

5.2 Aplicação Gráfica Qt

A primeira fase da implementação deste projeto consistiu na aplicação dos algoritmos de processamento de imagem que sustentam a reconstrução tridimensional. Para isso, como já foi referido anteriormente, optou-se inicialmente pelo desenvolvimento de uma aplicação gráfica, desenvolvida através do IDE Qt, na linguagem C\C++. Esta é baseada na biblioteca de visão por computador OpenCV, bem como na biblioteca PCL para manipulação de nuvens de pontos e na biblioteca OpenGL para a respetiva

visualização.

De seguida são apresentados alguns pontos considerados relevantes e que referem algumas tarefas que a aplicação permite fazer, com foco nas tarefas de *debug* e complemento da aplicação Android, são eles:

- Permite importar e exportar modelos tridimensionais, representados por nuvens de pontos, construídas em ambas as aplicações ou em aplicações de terceiros. Suportando assim, a importação e exportação nos formatos *PCD* e *PLY*.

- É possível visualizar e interagir com as nuvens de pontos criadas ou importadas. Para isso, utilizou-se a biblioteca *OpenGL*, de forma a apresentar os modelos tridimensionais criados ao utilizador. Foi escolhida esta biblioteca em específico, devido à sua fácil portabilidade entre diferentes plataformas, com o intuito de o código ser facilmente portado para a plataforma Android.

- Na aplicação gráfica em Qt é possível importar um conjunto de imagens, por exemplo, adquiridas através da aplicação Android e posteriormente serem utilizadas durante o processo de reconstrução.

- A aplicação permite importar parâmetros de calibração de câmaras. Facilitando desta forma, por exemplo, a realização do processo de calibração na aplicação Android e a posterior importação dos parâmetros para a aplicação Qt.

- Permite ao utilizador variar alguns algoritmos associados ao processo de reconstrução tridimensional, bem como, alguns parâmetros a si associados.

Deste modo, esta secção, apresentará a camada referente à parte da interface gráfica, bem como tarefas associadas. A parte da descrição da camada de processamento de imagem, camada esta comum à aplicação Android será apresentada na próxima secção.

5.2.1 Interface Gráfica

A aplicação representada na imagem da Figura 5.2 é composta por um conjunto de elementos de interação, que permitem ao utilizador interagir com o programa em si e, ao mesmo tempo, visualizar o que se está a passar em tempo real com o processo de reconstrução tridimensional. Esta é composta por uma única janela e foi desenvolvida através da ferramenta *Qt Designer* do próprio *Qt Creator*, a

qual tem uma função de *Drag-and-Drop* para construção de interfaces gráficas de forma mais rápida e intuitiva.

O resultado final é um ficheiro *xml* no qual fica armazenado o *layout*. No caso concreto desta aplicação, o ficheiro é denominado *mainwindow.ui* e é associado à classe com o mesmo nome aquando da sua inicialização. Como é possível observar, ainda na Figura 5.2, a aplicação é organizada em seis áreas distintas, cada uma das quais, dedicada a uma tarefa específica.

A primeira é a janela OpenGL e é responsável por renderizar os elementos tridimensionais, para além da nuvem de pontos representativa do objeto tridimensional alvo, permite também a representação dos eixos coordenados x, y e z, bem como uma *grid* coplanar ao eixo x. O menu 2 está ainda relacionado com a gestão das funções OpenGL, permitindo, desta forma, importar e exportar nuvens de pontos previamente criadas nos formatos PLY e PCD. Neste menu é possível também desabilitar a visualização da *grid* e dos eixos coordenados, bem como através das *slidebars* rodar alguns dos eixos coordenados permitindo, assim, a visualização numa perspectiva diferente.

O menu 3 é responsável pelas funções de gerir as tarefas gerais da aplicação. Permite definir o número de imagens que serão usadas durante o processo de reconstrução tridimensional, bem como a sua localização no disco. Contem também uma *flag* que permite habilitar uma visualização maior das imagens e respetivos pontos de interesse, facilitando em tarefas de depuração. Por fim, permite também a importação de novos parâmetros de calibração através do botão *GetCalibrationParameters*, sendo que o quadrado colorido indica ao utilizador se já foram ou não importados os respetivos parâmetros de calibração. O botão *StartProcess* inicia o processo de reconstrução, após todos os estados estarem cumpridos, sendo eles os parâmetros de calibração importados, bem como, a seleção de um teste de filtragem através do menu 6.

O menu 4, é também um menu de visualização, apresentando a cada momento duas imagens referentes ao par analisado pelo algoritmo, bem como, uma imagem com os pontos de interesse e respetivos emparelhamentos referentes às duas imagens.

Por fim, os menus 5 e 6 são compostos por um conjunto de elementos que permitem gerir os algoritmos implícitos no processo de reconstrução tridimensional. O menu 5, mais precisamente, é focado nos algoritmos de deteção e emparelhamento de pontos de interesse, dado isto, é composto por uma *combobox* que permite escolher os algoritmos, entre eles, *SURF*, *SIFT* e *ORB*. Relativamente aos dois primeiros é também possível a modificação dos parâmetros referentes às variáveis de entrada das respetivas funções na biblioteca OpenCV. O menu 6 é responsável pelas restantes funções relativas ao processo de reconstrução tridimensional, onde é possível definir o método de filtragem, através da escolha entre

ratioTest e *SymmetryTest*. É possível também definir alguns parâmetros nas funções de cálculo da matriz essencial, assim como, da função *solvePnPRansac* da biblioteca OpenCV e utilizadas ao longo do processo de reconstrução tridimensional.

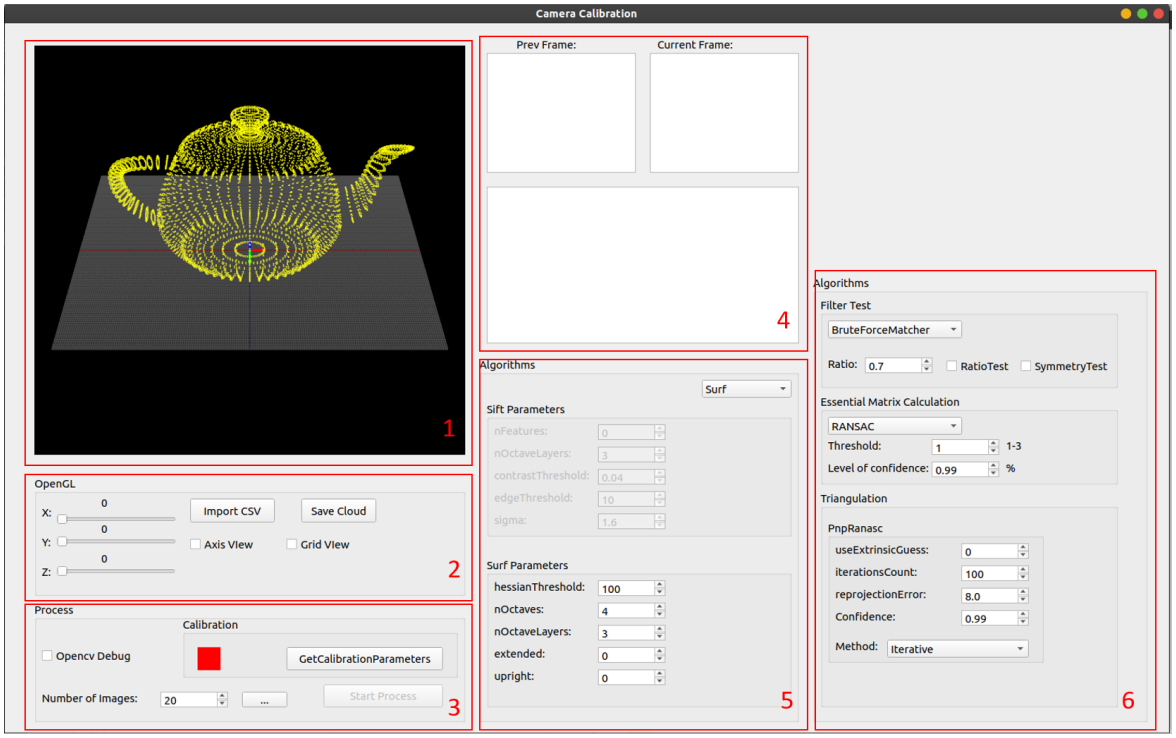


Figura 5.2: Interface Gráfica da Aplicação QT.

5.2.1.1 Arquitetura de Software

A classe MainWindow é a classe principal e gere a interface gráfica com o utilizador. Associada à interface gráfica está também a classe MyGLWidget onde são implementados os métodos responsáveis pela chamada das funções da biblioteca OpenGL. Por sua vez, as classes CameraCalibrator e SfM têm contido a implementação dos algoritmos de processamento de imagem, respetivamente para tarefas de calibração e reconstrução tridimensional. A classe FileInterface é responsável pela gestão e manipulação de ficheiros externos. É importante referir que estas três classes são também partilhadas pela aplicação Android. Na Figura 5.3 está representado um diagrama de classes simplificado onde estão definidas as classes presentes nesta aplicação gráfica.

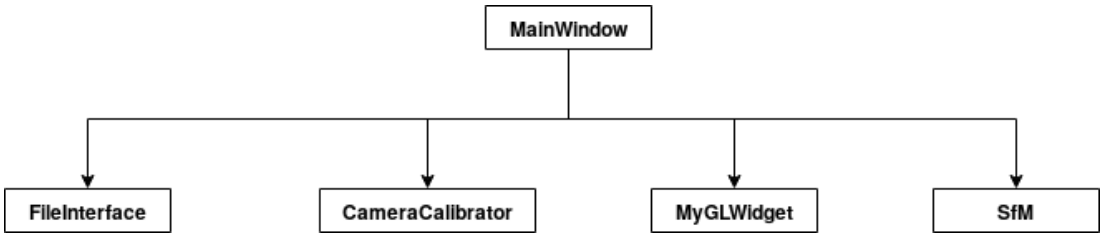


Figura 5.3: Diagrama de Classes simplificado

5.2.1.2 OpenGL Widget

O *OpenGLWidget*, como o próprio nome indica, é uma ferramenta do tipo *widget* que pode ser associada ao *layout* de uma interface gráfica sendo responsável por fazer a ponte entre o Qt e a biblioteca OpenGL. Este é implementado através da classe *QGLWidget* presente nativamente na *framework* Qt. Esta classe fornece três funções virtuais essenciais para a renderização de cenas em OpenGL, *initializeGL()*, *paintGL()* e *resizeGL()*.

Dado isto, foi então criada uma nova classe, denominada *MyGLWidget*, com o objetivo de gerir todas as tarefas associadas à biblioteca OpenGL. Esta nova classe, é uma subclasse da *QGLWidget* e implementa as funções herdadas da classe mãe. A função *initializeGL()* é responsável pelas tarefas de inicialização, necessárias para renderizar uma cena através de OpenGL, o que envolve normalmente a definição de cores, direção da luz e outras propriedades consideradas importantes. Neste caso específico, é utilizada para definir a cor do *background* da janela, isto é conseguido através da chamada da função *glClearColor()*, onde os seus parâmetros de entrada correspondem a um valor normalizado entre 0 e 1 e representam respetivamente um valor no espaço de cores RGB, o quarto parâmetro está relacionado com níveis de transparência. O *background* por defeito está definido na cor preta, o que corresponde a chamar a função *glClearColor(0.0f, 0.0f, 0.0f, 1.0f)*.

Outra função é o *resizeGL()*, que tem o propósito de gerir os parâmetros de renderização gráfica sempre que alguma alteração é feita no tamanho do *widget* na interface gráfica, por exemplo, quando um ajuste do tamanho da janela é realizado. Nesta é chamada a função *glviewport()* onde é definida a área dentro do *widget* no qual o OpenGL pode trabalhar. Para isso, recebe como parâmetros de entrada o comprimento e largura da *widget* em questão. Posteriormente, é chamada a função *glMatrixMode(GL_PROJECTION)*, seguida da função *glLoadIdentity()*. A primeira indica que as próximas transformações geométricas serão aplicadas na matriz de projeção, a segunda função copia para a matriz de projeção, uma matriz identidade com o mesmo tamanho. Por fim, é chamada a *glFrustum()*, que gere a transformação perspetiva aplicada na janela OpenGL, de forma a definir o volume de trabalho da própria.

A função *paintGL()* é responsável por gerir o que é renderizado na janela OpenGL presente na interface gráfica. O Fluxograma da Figura 5.4 descreve o funcionamento em específico desta função. Inicialmente, e como é possível observar pelo Fluxograma, é necessário aplicar um conjunto de transformações geométricas, através da aplicação de uma translação, função *glTranslate()*, e três rotações, função *glRotatef()*, em cada um dos eixos coordenados. Isto é necessário, devido a possibilitar a interação entre o utilizador e a janela *OpenGL*. Por outras palavras, o utilizador tem a possibilidade de interagir com os modelos apresentados através das *slidebars* presentes na interface gráfica, bem como, através do uso do *scroll* do rato. Posto isto, o movimento dos modelos apresentados na janela OpenGL, implica a aplicação de transformações geométricas na matriz do modelo, o uso desta matriz faz-se com a chamada da função *glMatrixMode(GL_MODELVIEW)*, seguida da chamada da matriz identidade através da função *glLoadIdentity()*. Foram também implementadas as funções *setRotationX()*, *setRotationY()*, *setRotationZ()* e *zoomManage()* que são chamadas quando um evento numa dessas ferramentas é gerado. As primeiras três funções recebem como parâmetro de entrada o respetivo ângulo enviado diretamente pela função, que lê o valor das *slidebars* na aplicação gráfica, sendo guardadas em variáveis locais na classe *MyGLWidget*, *rotate_x*, *rotate_y* e *rotate_z*. Estas variáveis são respetivamente o parâmetro de entrada para as três funções *glRotatef()* aplicando, desta forma, transformações geométricas nos três eixos coordenados. Por sua vez, a função o *zoomManage()* recebe uma variável do tipo booleano que indica a direção do movimento, se for 1, então o *zoom* é multiplicado por uma escala de 1.1, senão é dividido por uma escala do mesmo valor. Estes valores são guardados em variáveis locais na classe *MyGLWidget* , de forma a permitir manter o estado anterior e só alterar na janela OpenGL quando o utilizador efetivamente tenha mexido em algumas das ferramentas.

No caso específico desta aplicação, são desenhados três elementos em simultâneo na janela OpenGL, são eles, a direção dos eixos coordenados no referencial do OpenGL, representados cada um por um segmento de reta, respetivamente a vermelho, verde e azul, para os eixos, x, y e z. Uma *grid* desenhada na interceção dos eixos e coplanar ao eixo x e, por fim, o modelo tridimensional, no caso, representado por uma nuvem de pontos. Para a sua representação são usadas respetivamente, as funções *drawAxis()*, *drawGrid()* e *drawPointCloud()*, implementadas na classe *MyGLWidget*. Através de duas *checkboxs* na interface gráfica do utilizador, este tem a possibilidade de habilitar ou desabilitar individualmente a apresentação quer dos eixos coordenados, quer da *grid* na janela de renderização do OpenGL. Resumidamente, nas funções de desenho são utilizadas os elementos básicos de desenho da biblioteca OpenGL, como por exemplo, a função *glVertex3f()* para desenhar linhas ou pontos e a função *glColor3f()* para definir a respetiva cor. A função *glVertex3f()* tem a particularidade de ser utilizada para desenho de

linhas ou pontos, para isso é chamada da função *glBegin()*, onde através do parâmetro de entrada é definido o desenho de linha ou pontos, respetivamente pela chamada das *keywords GL_LINES* ou *GL_POINTS*. Já especificamente na função *drawPointCloud()*, a matriz *cloud_xyzrgb* guarda a informação de todos os pontos do modelo tridimensional, bem como, a informação sobre a sua cor, no espaço de cores RGB. Finalizado o processo de reconstrução tridimensional, a nuvem de pontos é guardada numa variável local na classe *MyGLWidget*, para a sua apresentação na janela OpenGL, um ciclo percorre a matriz, a partir do qual os valores são lidos, sendo colocados no *widget* através da chamada da função *glVertex3f()*.

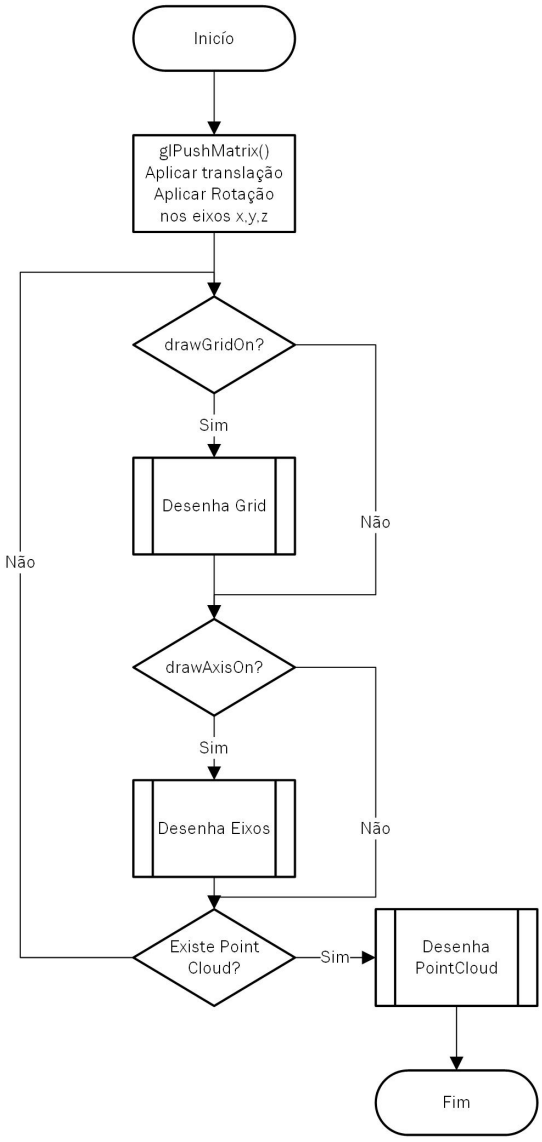


Figura 5.4: Fluxograma referente ao processo de renderização.

5.3 Modos de Funcionamento

De seguida, serão apresentados os modos de funcionamento presentes na camada de baixo nível quer da aplicação gráfica Qt, quer da aplicação para a plataforma Android. Os modos são: *modo de inicialização*, *modo de calibração* e *modo de reconstrução*.

5.3.1 Modo de Inicialização

O modo de inicialização é responsável por verificar se a calibração da câmara já se encontra realizada, aquando do início do programa. Para isso, verifica a existência de um ficheiro em memória, mais precisamente o ficheiro *parameters.xml*. Caso este exista, são carregados os respetivos parâmetros para o programa principal e, consequentemente, o modo de reconstrução é ativo. Pelo contrário, se o ficheiro não existir, é ativo o modo de calibração, isto na aplicação Android, uma vez que, a aplicação gráfica Qt não permite o modo de calibração.

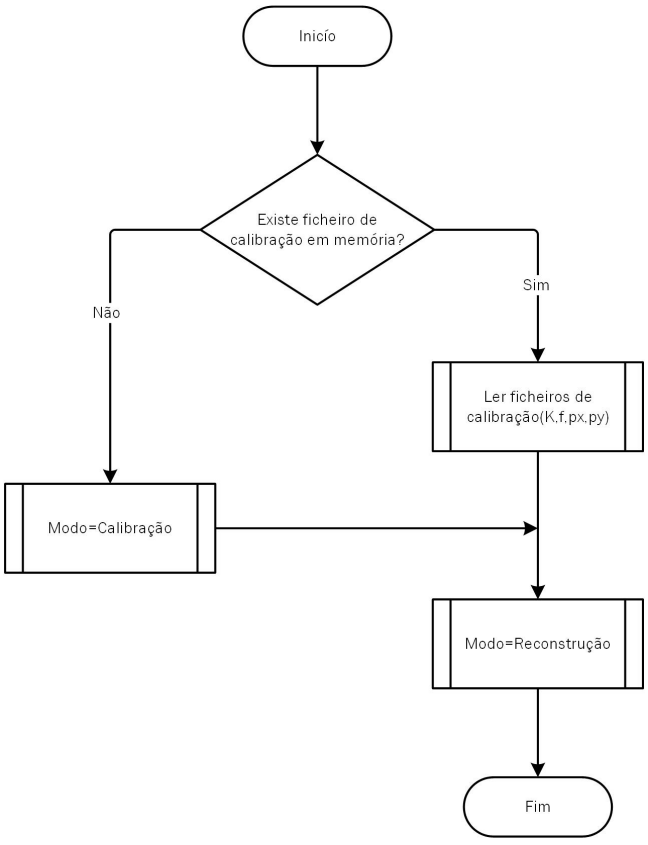


Figura 5.5: Fluxograma Modo de Inicialização.

Para a gestão dos ficheiros optou-se por criar a classe *FileInterface* que gere a leitura e escrita de ficheiros das aplicações. Esta é baseada na classe *FileStorage* da própria biblioteca OpenCV que

permite, de forma fácil, manipular em ficheiro as estruturas complexas da biblioteca OpenCV (Matrizes, Vetores), bem como, gravar em diversos formatos de ficheiros (XML, YAML). A classe *FileInterface* implementa apenas dois métodos, um referente à escrita, *SaveParameters*, e outra referente à leitura de ficheiros, *ReadParameters*.

No Fluxograma da Figura 5.5 está representado o processo de inicialização implementado. O bloco de decisão deste Fluxograma representa o momento em que a aplicação vai verificar a existência ou não do ficheiro de parâmetros na diretoria previamente definida. Este mecanismo é conseguido através da função *ReadParameters()* que em caso de existência ou não do ficheiro retorna um ou zero, respetivamente.

5.3.2 Modo de Calibração da Câmara

Como já foi abordado anteriormente, a calibração é um processo fundamental em qualquer sistema ótico que englobe processamento de imagem e que exija como resultado final alguma medida física do meio. O processo de calibração tem como principal objetivo inferir os parâmetros da câmara: intrínsecos, extrínsecos e de distorção. O modo de calibração implementado neste projeto é baseado nas funções de calibração propostas pela biblioteca OpenCV, que por sua vez, se baseiam no algoritmo de Zhang, já apresentado anteriormente na Secção 3.1.6. Para a calibração foi utilizado um padrão planar monocromático com aspeto de um tabuleiro de xadrez e com a geometria bem conhecida. Outro ponto importante e que deve ser frisado é que o processo de calibração desenvolvido só está disponível para a aplicação aplicação Android.

O processo de calibração proposto pela biblioteca OpenCV, resume-se à aplicação dos três seguintes passos:

1. Deteção dos pontos de interceção interiores do padrão de calibração de várias imagens adquiridas, sendo que para isto é utilizada a função *findChessboardCorners()*, que guarda estes pontos num vetor de pontos 2D.

2. Utilizar a geometria previamente conhecida do padrão para criar um vetor tridimensional, onde X e Y são as distâncias reais entre cada quadrado do padrão e Z assume-se que é igual a zero.

3. Aplicar a função *calibrateCamera()* que recebe como parâmetros de entrada os vetores calculados nos pontos anteriores e retorna de entre outras coisas os parâmetros intrínsecos e de distorção.

Dado isto, foi criada uma classe denominada *CameraCalibrator* que tem associada um conjunto de métodos que permitem gerir todo o processo de calibração. Destes métodos é importate fazer referência

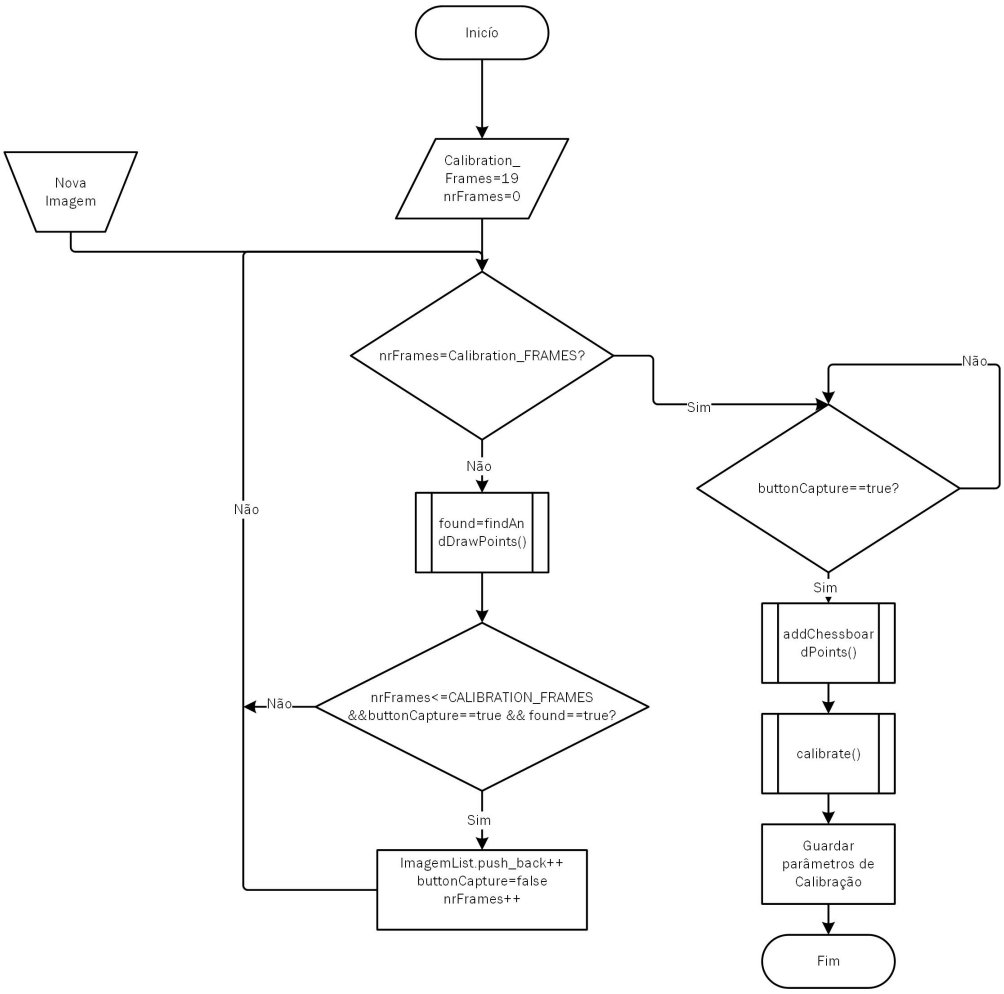


Figura 5.6: Fluxograma Processo de Calibração.

ao *findAndDrawPoints()*, *addChessboardPoints()* e *calibrate()*, pois estes são a base do processo de calibração.

O processo é representado pelo Fluxograma presente na Figura 5.6. Numa primeira fase, quando a aplicação entra no modo de calibração, o utilizador é guiado a tirar várias imagens, de diferentes perspetivas, ao padrão xadrez. O número de imagens está definido como 20, é importante salientar que segundo o método *Zhang* são necessárias pelo menos duas imagens, mas o algoritmo beneficia com a entrada de um maior número.

O bloco definido como *Nova Imagem*, no Fluxograma presente na Figura 5.6, representa a aquisição de uma nova imagem. No caso específico da aplicação Android esta aquisição é gerida através da API da câmara e será posteriormente explicada na Secção referente à aplicação Android.

A cada nova imagem adquirida é utilizada a função *findAndDrawPoints()* da classe *CameraCalibrator*, para detetar e apresentar em tempo real ao utilizador os pontos interiores do padrão. Esta função retorna uma resposta, positiva ou negativa, consoante a deteção ou não do mesmo, sendo guardado

esse estado na variável *found*. Para gerir a aquisição destas imagens é utilizado um contador que vai incrementando quando a aplicação obedece a três condições, a primeira exige que o número de imagens já adquiridas seja inferior às inicialmente definidas, no caso 20. A segunda é referente ao valor da variável *found*, que deve estar com o estado a *true* e, por fim, a terceira está relacionada com o evento do toque do botão de captura, que torna a variável *buttonCapture true* quando é tocado. A cada nova imagem analisada, as coordenadas bidimensionais referentes aos pontos interiores são guardados num vetor.

Quando o número de imagens adquiridas e processadas atinge o número desejado o processo de calibração entra na segunda fase, ou seja, o utilizar é informado e guiado a pressionar o botão para seguir no processo. Quanto isto acontece, é chamada a função *calibrate()*, que por sua vez, retorna os parâmetros intrínsecos, extrínsecos e de distorção, bem como, grava a informação pertinente no ficheiro *parameters.xml*.

5.3.3 Modo de Reconstrução

O modo de reconstrução, como o nome indica, é o modo onde são realizadas todas as tarefas relacionadas com o processo de reconstrução tridimensional. Este trabalho baseia-se na técnica passiva *Structure-from-motion*, mais precisamente no Algoritmo 1 apresentado na Secção 3.2 deste trabalho. Desta forma, o modo de reconstrução está dividido em três fases distintas, extração e descrição de pontos de interesse, emparelhamento de pontos de interesse e triangulação de pontos de interesse, resultando numa nuvem de pontos representativa do objeto original.

Posto isto, foi criada uma nova classe denominada *SfM* que encapsula todos os métodos necessários para o processo de reconstrução tridimensional. Os métodos *setFeatureDetector()*, *setDescriptorExtractor()*, *computeKeypoints()* e *computeDescriptors()*, *ratioTest()* e *SymmetryTest()* estão implementados nesta classe e foram adaptados do código de exemplo presente na página do *Github*[1]do próprio OpenCV. Aqui é importante referir que as primeiras duas funções são genéricas e permitem inicializar o objeto referente ao algoritmo de deteção e descrição dos pontos de interesse, para isso, os objetos representativos desses algoritmos são passados à função a partir da interface gráfica, utilizando para isso uma variável do tipo *cv::Feature2D*. As outras funções são respetivamente para cálculo dos pontos de interesse, cálculo dos descritores, implementação do teste de rácio e teste de simetria.

Além disto, a classe *SfM* tem implementados os métodos que fazem a chamada das funções OpenCV, referentes ao processo de reconstrução tridimensional.

[1] `https://github.com/opencv/opencv/blob/master/samples/cpp/tutorial_code/calib3d/real_time_pose_estimation/src/RobustMatcher.cpp`

5.3.3.1 Extração e Descrição de Pontos de Interesse

Na prática, para este modo ser ativo, são necessárias duas condições serem cumpridas, a primeira está relacionada com o processo de calibração, o qual deve estar finalizado, uma vez que, os parâmetros resultantes são necessários durante o processo de reconstrução tridimensional. Por sua vez, a outra condição, depende do utilizador e está relacionada com o evento de toque no botão de *start* que inicia o processo propriamente dito. Na Figura 5.7 está representado o Fluxograma que representa a sequência da primeira fase deste modo, sendo que esta foca-se fundamentalmente na aquisição e, posterior, deteção e descrição dos pontos de interesse em cada uma das imagens.

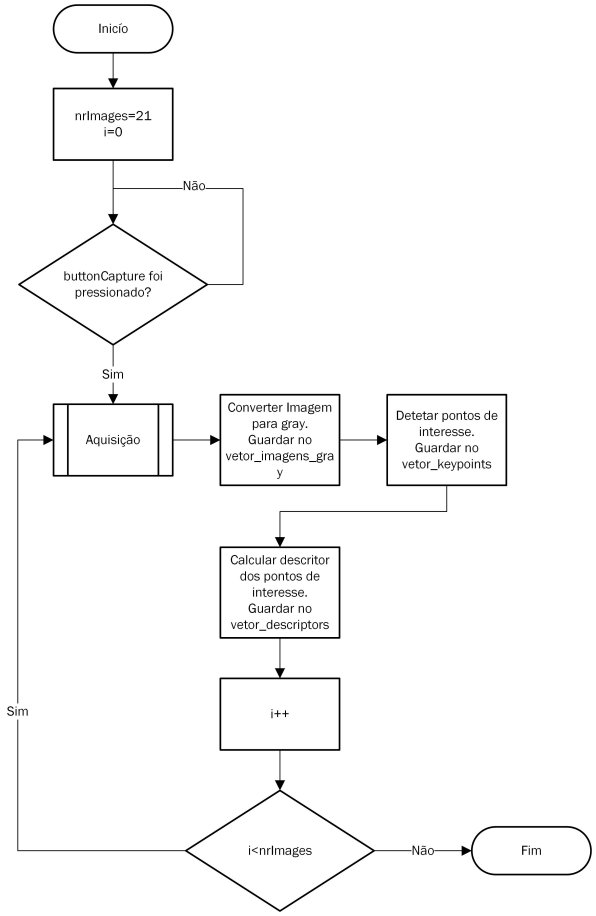


Figura 5.7: Fluxograma referente ao processo de Aquisição, deteção e descrição de pontos de interesse.

Durante o processo são utilizadas vinte e uma imagens obtidas de diferentes perspectivas à volta da superfície do objeto. No caso da aplicação gráfica Qt são lidas a partir do ficheiro, uma vez que, já foram previamente adquiridas através da aplicação desenvolvida para a plataforma Android. Um ciclo de vinte e uma iterações é iniciado, onde cada imagem individualmente é adquirida e convertida para a escala de *gray* através da função *cvtColor()* onde, por fim, é armazenada num vetor de imagens *gray*. Através das funções *computeKeyPoints()* e *computeDescriptors()* são detetados os pontos de in-

teresse e os respetivos descritores para cada uma das imagens, os quais são também armazenados em vetores, respetivamente *vetor_keypoints* e *vetor_descriptors*. Os índices destes vetores, correspondem exatamente à ordem no qual as imagens foram adquiridas. Aqui é importante referir que a função *computeKeyPoints()* recebe como parâmetros de entrada a respetiva imagem em tons de cinza e a função *computeDescriptors()* recebe como parâmetros de entrada os pontos de interesse calculados a partir da função *computeKeyPoints()*. Estas funções são genéricas uma vez que usam uma variável do tipo *cv::Features2D* que armazena o objeto do algoritmo de deteção e descrição de pontos de interesse inicialmente escolhido pelo utilizador através da interface gráfica. Por sua vez, estes objetos referentes aos algoritmos têm implementados os métodos utilizados nas funções *computeKeyPoints()* e *computeDescriptors()*, respetivamente os métodos *detect* e *compute*.

5.3.3.2 Emparelhamento de Pontos de Interesse

Encontrados os pontos de interesse e os respetivos descritores para cada uma das vinte e uma imagens, é necessário agora encontrar o par, composto por um ponto de interesse numa imagem e pelo seu correspondente na imagem seguinte, esse processo denomina-se *match* e é feito através de um *descriptor matcher*. No caso específico deste trabalho, foram utilizados dois distintos, são eles o *Brute-force matcher descriptor* e *Flann matcher descriptor*. O primeiro utiliza um descritor de uma imagem e compara-o individualmente com todos os descritores da segunda imagem, sendo que normalmente o critério de comparação é feito através da comparação da distância euclidiana entre pontos de interesse em imagens distintas, que deve ser mínima. Já o segundo, *Flann matcher descriptor*, tem também uma implementação na biblioteca OpenCV, e é baseada na biblioteca *Fast Library for Approximate Nearest Neighbors*[2]. Por sua vez, esta contem a implementação de um conjunto de algoritmos optimizados para a pesquisa de pontos de interesse, focados para extensos conjuntos de dados. A interface gráfica desenvolvida, permite ao utilizador escolher qual o *descriptor matcher* que pretende utilizar. É importante referir que durante a inicialização da interface gráfica é realizada uma chamada à função *setMatcherMode()* implementada na classe *SfM*, permitindo desta forma, a definição do método a utilizar durante o processo de emparelhamento dos pontos de interesse.

No Fluxograma presente na Figura 5.8 está representado o processo de emparelhamento entre pontos de interesse. Nesta fase, um ciclo de vinte iterações permite encontrar os pares de pontos de interesse correspondentes entre imagens consecutivas. Para isso, foi implementado o método *getMatches()* na classe *SfM*, este recebe como parâmetros de entrada o conjunto de pontos de interesse e descritores

[2] `https://www.cs.ubc.ca/research/flann/`

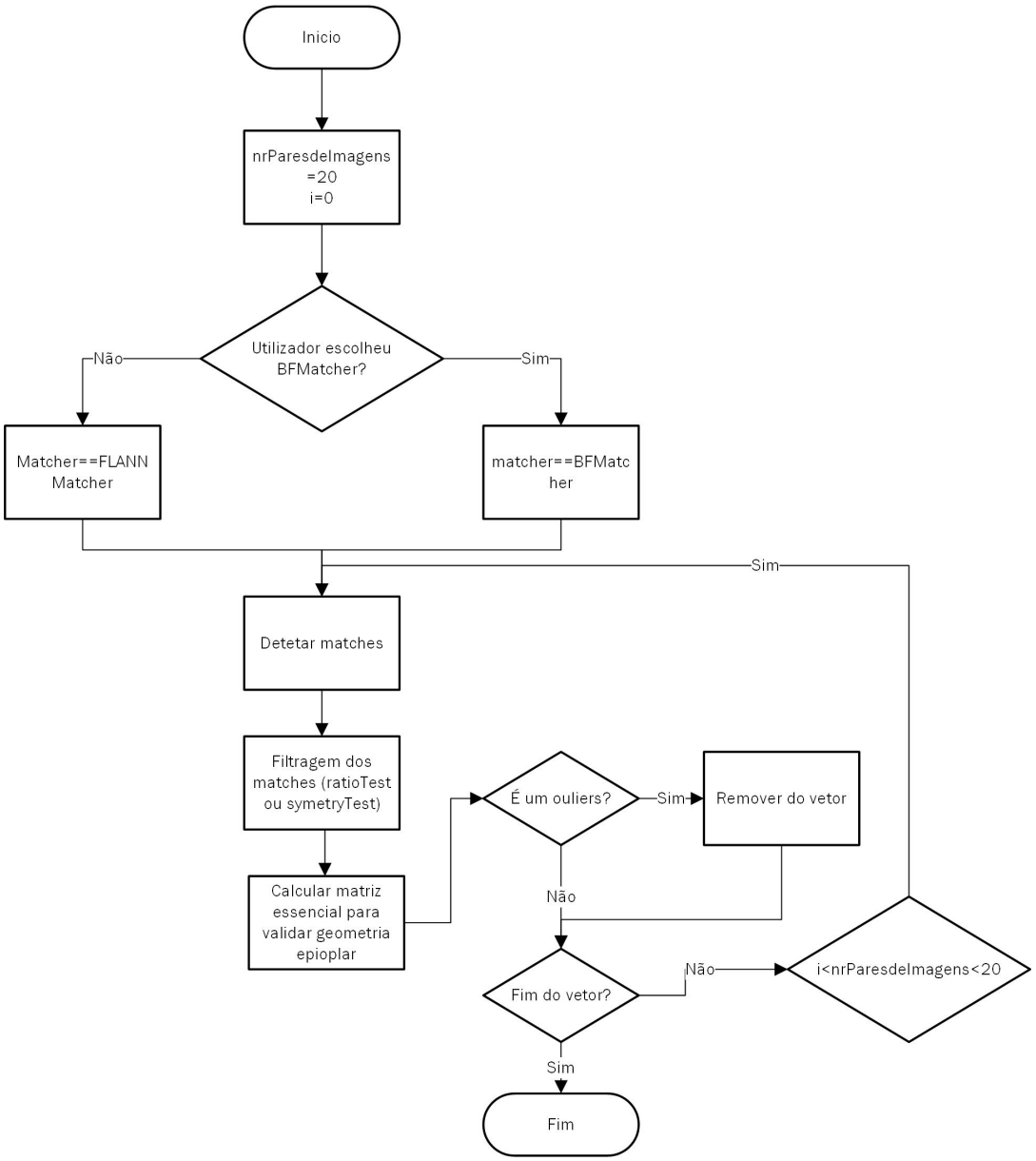


Figura 5.8: Fluxograma do processo de filtragem de pontos de interesse e remoção de *outliers*.

respectivos a uma imagem, bem como, os pontos de interesse e descritores referentes à imagem seguinte. Como retorno, obtém-se o vetor de *matches* correspondentes. Este método, por sua vez, faz a chamada do método *knnMatch()* da classe *DescriptorMatcher*, nativa da biblioteca OpenCV, que está associada a um dos descritores *matcher* escolhido. É importante referir que este método recebe como parâmetros de entrada os descritores referentes a uma imagem e os descritores respetivos da imagem seguinte e, permite definir um número máximo de correspondência para um dado descritor na imagem seguinte. No caso, esse parâmetro foi definido para k=2, o que representa que o algoritmo de *match* para um descritor na primeira imagem poderá encontrar até dois descritores na segunda imagem. Para reduzir os pares de pontos de interesse no qual a correspondência não ficou bem realizada por parte do algoritmo,

é pertinente aplicar algum tipo de filtro, de forma a eliminar estes pontos dos vetores. Para isso, são utilizados os métodos *ratioTest()* e *symmetryTest()*, utilizados diretamente da pasta do Github do OpenCV, como já foi referido em cima.

Os dois métodos de filtragem de correspondências estão representados pelos Fluxogramas da Figura 5.9. Como é possível observar no Fluxograma da Figura 5.9a, referente ao método *ratioTest*, é utilizado um iterador para percorrer o vetor de correspondências, referente às correspondências entre os descritores da primeira imagem e os descritores correspondentes na segunda imagem. O filtro elimina à partida os pares que não tenham duas correspondências associadas na segunda imagem, por sua vez, numa segunda fase um rácio entre as distâncias entre a correspondência na primeira imagem e as duas correspondências respetivas na segunda imagem é calculado, caso seja superior a um valor predefinido são também eliminadas. Já no caso do método *symmetryTest*, representado pelo Fluxograma da Figura 5.9b, são calculadas as correspondências entre os descritores da primeira imagem e os da segunda, e o processo contrário também, ou seja, são obtidos dois vetores de correspondências um referente ao sentido, primeira-segunda imagem e o outro referente ao sentido segunda-primeira imagem. Dois iteradores são utilizados para percorrer os índices dos vetores de correspondências, onde as correspondências de um vetor são comparadas com a correspondência respetiva do outro vetor, caso sejam diferentes são eliminadas do vetor. O vetor resultante após a chamada da função *getMatches()* é do tipo *DMatch*, sendo esta uma estrutura de dados referente à biblioteca OpenCV. Esta estrutura não armazena propriamente as coordenadas dos pontos de interesses, mas sim referencia-os, quer isto dizer, para cada emparelhamento guardado na estrutura, implicitamente estão guardadas a distância entre descritores e o índice correspondente no qual estão armazenadas as coordenadas do ponto de interesse em questão, quer para a primeira imagem, quer para a segunda. Resumidamente, estão guardadas na estrutura de dados, para todos os descritores encontrados entre o par de imagens analisado a cada momento, as seguintes variáveis: *float distance*, *int index_queryIdx*, *int index_trainIdx*, respetivamente a distância e índices no qual estão armazenados os respetivos pontos de interesse para a primeira imagem, e para a segunda imagem respetivamente no vetor original. Dado isto, e como os passos seguintes assim o exigem, dois novos vetores de pontos de interesse com as respetivas coordenadas devem ser criados, para a primeira imagem e para a segunda. Para isso, através de um ciclo é possível correr o vetor de emparelhamento e daí através da informação dos índices é possível recuperar as coordenadas respetivas para os novos vetores. São também criados dois vetores com os índices de forma a ser mais fácil acessar em operações futuras. São criados, então, vetores tipo *vector<Keypoint>*, respetivamente *good_points_first* e *good_points_last* para armazenamento dos pontos de interesse após processo de emparelhamento e filtragem e dois ve-

tores do tipo *vector<int>*, respetivamente *good_points_first_idx* e *good_points_last_idx* com a informação referente aos índices dos pontos de interesse no vetor original.

Finalizada a procura, o emparelhamento e o filtro das correspondências, uma última etapa pode e deve ser realizada para um refinamento mais preciso no emparelhamento das correspondências. Voltando à parte final do Fluxograma presente na Figura 5.8, é possível verificar uma referência ao cálculo da matriz essencial, matriz esta já abordada na secção 3.2.3. Neste caso, a matriz essencial é utilizada para validar a geometria epipolar, segundo o apresentado na secção 3.2.2, um ponto de interesse encontrado numa primeira imagem tem o seu correspondente sobre a linha epipolar numa segunda imagem. Uma vez que os algoritmos de deteção de pontos de interesse não têm em consideração a geometria da formação da imagem é necessário um método para estimar essa conformidade dos pontos de interesse, dado isto, utiliza-se a matriz essencial. Para o respetivo cálculo é utilizada a função *findFundamentalMat()*, função nativa do OpenCV, que permite estimar a matriz essencial através dos pontos de interesse encontrados em duas imagens distintas, bem como dos parâmetros de calibração referentes à câmara a partir do qual foram adquiridas. A função faz uso do algoritmo *RANSAC* para estimação dos *outliers*, a precisão pode ser gerida através do parâmetro *threshold* que permite definir a distância máxima em pixéis que o algoritmo *RANSAC* considera que o ponto de interesse está fora da linha epipolar.

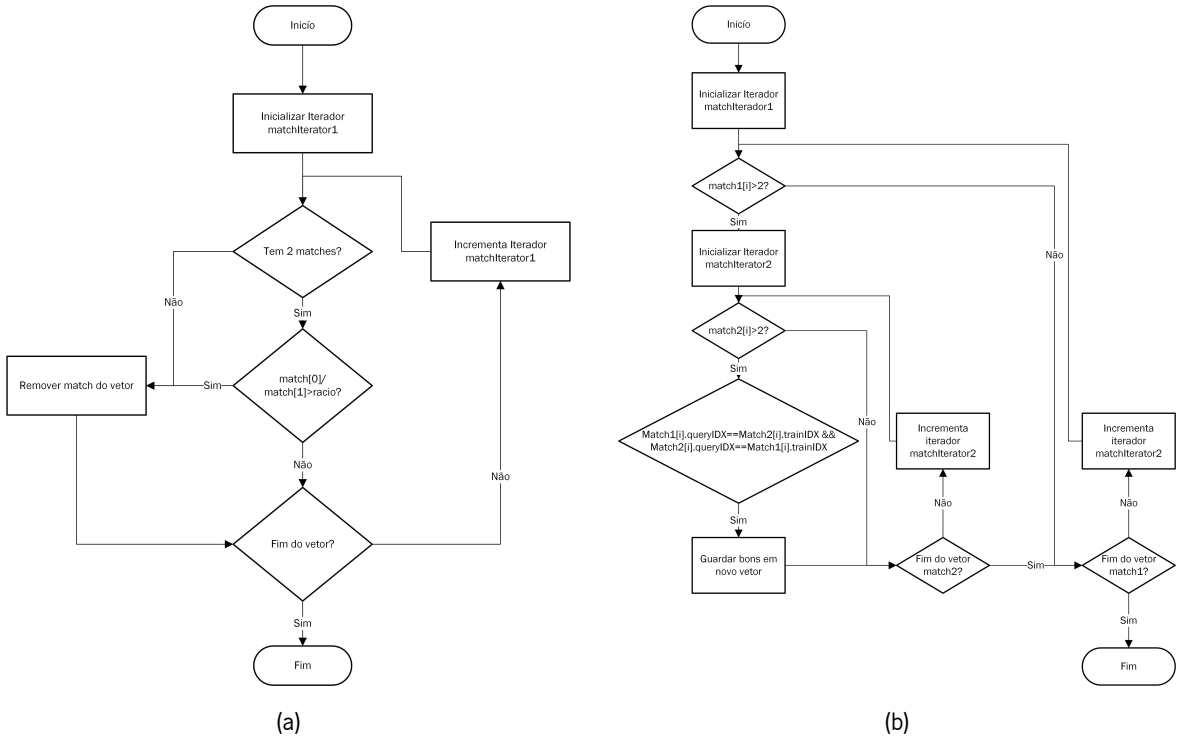


Figura 5.9: a) Fluxograma Método *ratioTest* b) Fluxograma Método *symmetryTest*.

5.3.3.3 Triangulação de Pontos de Interesse

A triangulação dos pontos de interesse, nesta dissertação, é a última fase do processo referente à reconstrução tridimensional. No final desta fase, uma nuvem de pontos tridimensionais representativa do objeto em análise é obtida. Uma vez emparelhados e filtrados os pares de pontos correspondentes é possível obter a sua informação tridimensional através do processo de triangulação, isso é conseguido através da aplicação das equações apresentadas na Secção 3.2.6 deste documento, respetivamente a equação 3.2.18 e a 3.2.19. É importante referir ainda, que as equações são uma versão simplificada, da equação 3.1.11 presente na Secção 3.1.4, deste documento que, por sua vez, representa a geometria implícita na formação de uma imagem numa câmara, onde são considerados os seus parâmetros intrínsecos e extrínsecos.

Um conjunto de pontos tridimensionais iniciais são necessários para se iniciar o processo de reconstrução completo, com o objetivo de triangular todos os pares de pontos correspondentes presentes no conjunto total das imagens adquiridas. Isto deve-se ao facto de serem necessários para a utilização da função *solvePnPRansac()*, da biblioteca OpenCV, que é a base desta fase de triangulação e será abordada novamente ao longo desta secção. Dado isto, a triangulação inicial é feita a partir dos pares de pontos correspondentes referentes às duas primeiras imagens da sequência. O Fluxograma presente na Figura 5.10 representa a primeira fase deste processo de triangulação.

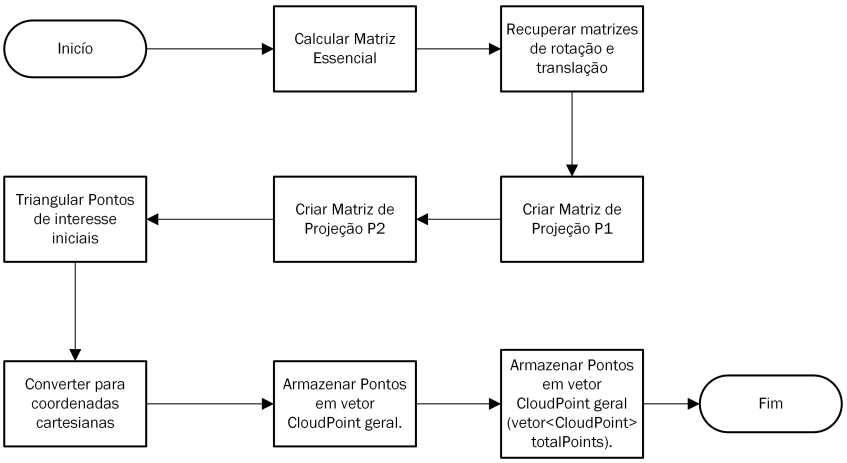


Figura 5.10: Fluxograma do processo de triangulação inicial, aplicado aos pares de pontos correspondentes pertences à primeira e segunda imagem da sequência total.

Para aplicar as equações 3.2.18 e a 3.2.19 é necessário o conhecimento prévio da matriz dos parâmetros extrínsecos que descreve o movimento entre a aquisição da primeira imagem e da segunda. De forma a recuperar esta informação, é necessário a chamada da função *recoverPose()* da biblioteca OpenCV, uma vez que, esta vai permitir obter a matriz de rotação e o vetor de translação respetivos

desse mesmo movimento. Para isso, a função necessita como parâmetros de entrada do vetor de pontos correspondentes para cada uma das imagens, bem como da matriz essencial, obtida através da função *findEssentialMat()*. Posto isto, segundo o Fluxograma da Figura 5.10, o primeiro passo consiste no cálculo da matriz essencial e, posteriormente, na recuperação da matriz de rotação e do vetor de translação, que se consegue através da chamada da função *recoverPose()*. Obtida a matriz dos parâmetros extrínsecos, é já possível criar respetivamente a matriz *P1*, referente à equação 3.2.18 e a matriz *P2* referente à equação 3.2.19, ambas matrizes de tamanho 3x4. Calculadas estas matrizes, são então triangulados os pares de pontos correspondentes a partir da chamada da função *triangulatePoints()*, que recebe como parâmetros de entradas ambas as matrizes de projeção, respetivamente *P1* e *P2*, bem como, os vetores dos pares emparelhados para cada uma das imagens. Como *output* é possível obter um vetor onde estão contidos os pontos triangulados no espaço tridimensional em coordenadas homogéneas. Para finalizar, os pontos são convertidos para coordenadas cartesianas e em seguida armazenadas num vetor do tipo *CloudPoint*, que contem todos os pontos triangulados no plano tridimensional, declarado como vetor<CloudPoint> totalPoints. Este vetor é do tipo *CloudPoint* e a sua estrutura de dados é constituída por uma variável que contem as suas coordenadas tridimensionais para os três eixos coordenados, uma variável que contem a informação do índice correspondente à localização dos pontos de interesse e descritores nos seus respetivos vetores e por último contém também uma variável onde fica armazenada a informação acerca da cor do ponto. Isto permite nas iterações seguintes acessar facilmente aos pontos de interesse e descritos referentes aos pontos contidos no vetor na nuvem de pontos.

Obtida a triangulação inicial dos primeiros pontos, correspondentes aos emparelhamentos referentes ao par inicial de imagens, é necessário agora realizar um procedimento equivalente para os restantes pares de imagens. No Fluxograma da Figura 5.11 está representado o processo final de triangulação que tem como resultado a nuvem de pontos final. À semelhança do que acontece para o primeiro par de imagens, para a triangulação de um dado par de pontos é necessário saber as respetivas matrizes de projeção, para isso é necessário o conhecimento prévio da matriz de parâmetros extrínsecos, constituída pela matriz de rotação e translação entre cada imagem. Para o cálculo da matriz dos parâmetros extrínsecos é utilizada a função *solvePnPRansac()*. Esta permite estimar a pose de um determinado objeto através de um dado conjunto de pontos seus no espaço tridimensional, bem como as suas correspondentes projeções no plano da imagem. É importante referir que é também necessário a matriz de calibração. Para cada um dos pontos triangulados referentes ao primeiro par de imagens, no caso já armazenados no vetor<CloudPoint> totalPoints, é necessário recuperar a informação referente aos seus pontos de interesse, bem como aos seus descritores. A estrutura de dados *CloudPoint*, como já foi referido, é constituída por uma variável

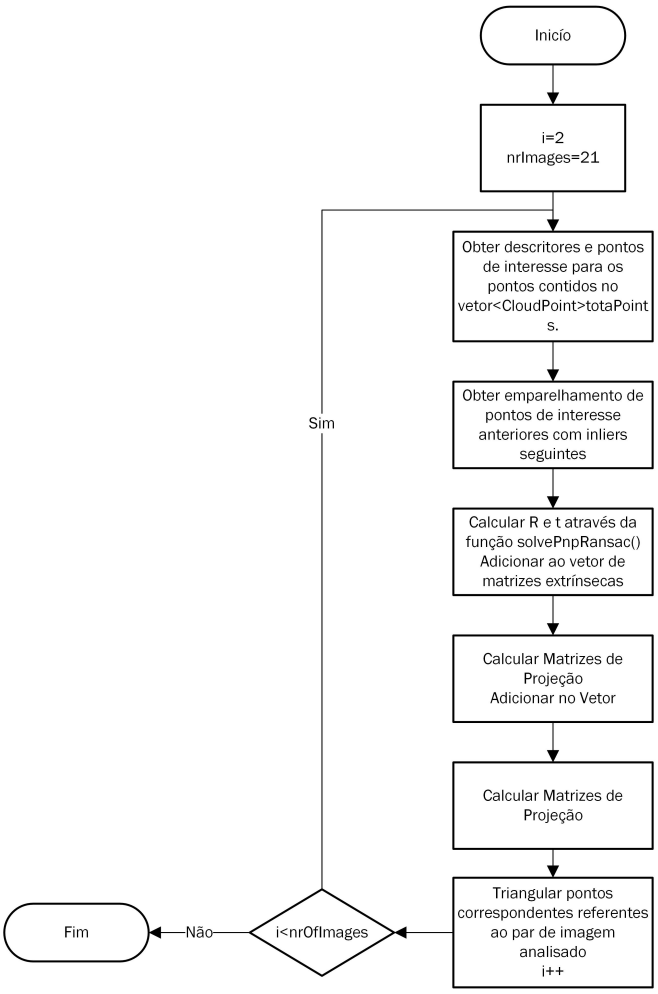


Figura 5.11: Fluxograma referente ao processo de triangulação final.

que contem a informação da localização dos pontos de interesse e respetivos descritores nos vetores gerais onde foram armazenados durante o processo de filtragem de *outliers*.

Após isto, um emparelhamento entre os pontos de interesse e respetivos descritores é realizado com os pontos de interesse e descritores referentes aos *inliers* da imagem seguinte. Este emparelhamento é realizado com a chamada da função *getMatches()*, usada anteriormente no processo de emparelhamento inicial. Obtidos os emparelhamentos é ja possível chamar a função *solvePnPRansac()*, com parâmetros de entrada, os pontos tridimensionais, os emparelhamentos e a respetiva matriz de calibração, por sua vez obtêm-se as matrizes de rotação e translação. Posto isto, são calculadas as matrizes de projeção, à semelhança do que foi feito na triangulação inicial, onde são guardadas num vetor que armazena todas as matrizes de projeção. Isto permite a cada nova iteração ir buscar o valor da matriz atual e da anterior. A última tarefa do processo é a chamada da função *triangulatePoints* que permite triangular os respetivos pontos de interesse no plano tridimensional. Estes pontos são adicionados ao vetor<CloudPoint> totalPoints que contem todos os pontos tridimensionais. O processo repete-se até ser

percorrido todo o vetor de *inliers* referente aos pares de imagens analisados.

5.4 Aplicação Android

A aplicação móvel, desenvolvida no âmbito desta dissertação é denominada *3DReconstruction* e foi desenvolvida maioritariamente na linguagem *Java* através do IDE de desenvolvimento *Android Studio*. Foi também utilizado o *Android NDK* para realizar o interface com a biblioteca na linguagem C++, responsável pelos algoritmos de reconstrução tridimensional.

De uma forma geral, a aplicação é composta por uma interface gráfica que permite gerir alguns parâmetros do processo de reconstrução tridimensional, bem como gerir a comunicação *Bluetooth* responsável por comunicar com a plataforma rotativa. Por sua vez, permite também apresentar as imagens da câmara do dispositivo móvel, bem como acessar a informação proveniente dos sensores inerciais. A aplicação Android desenvolvida é composta por uma atividade principal, denominada *Camera2GLActivity*, que gere um conjunto de Fragmentos e *DialogFragments* que permitem implementar uma interface gráfica com o utilizador, como é o exemplo das classes, *LeftButtonsFragment*, *RightButtonsFragment*, *MyGLSurfaveViewFragment*, *BluetoothDialogFragment*, *ImuDialogFragment* que representa respetivamente os Fragmentos e os *DialogFragments*. Para a implementação da interface *Bluetooth*, a comunicação com os sensores inerciais e a as tarefas de aquisição e processamento de imagem foram implementadas nomedamente as classes, *BlueoothControl*, *SensorFusion*, *Camera2Renderer* e *NativePart*, bem como algumas funções em C++ para a comunicação entre a parte desenvolvida em Java e a biblioteca implementada em C++. Nos próximos tópicos deste capítulo será melhor apresentado o funcionamento da aplicação Android desenvolvida.

5.4.1 Atividade Camera2GLActivity

Quando o utilizador inicia a aplicação *3DReconstruction* é apresentada a atividade principal, denominada *Camera2GLActivity* e é representada pela Figura 5.12. Esta atividade é responsável pela gestão de todos os processos intrínsecos a esta aplicação, nomeadamente na gestão da interface gráfica com o utilizador, em processos internos como a gestão da interface *Bluetooth*, da câmara e da leitura de sensores do dispositivo móvel, bem como a interface com a biblioteca de visão por computador desenvolvida em C/C++. Permite também a alternância entre modos de funcionamento, como por exemplo, *Modo de Calibração* e *Modo de Reconstrução*, além de permitir a escolha dos algoritmos de processamento de imagem a utilizar, à semelhança do acontece na aplicação gráfica em Qt.

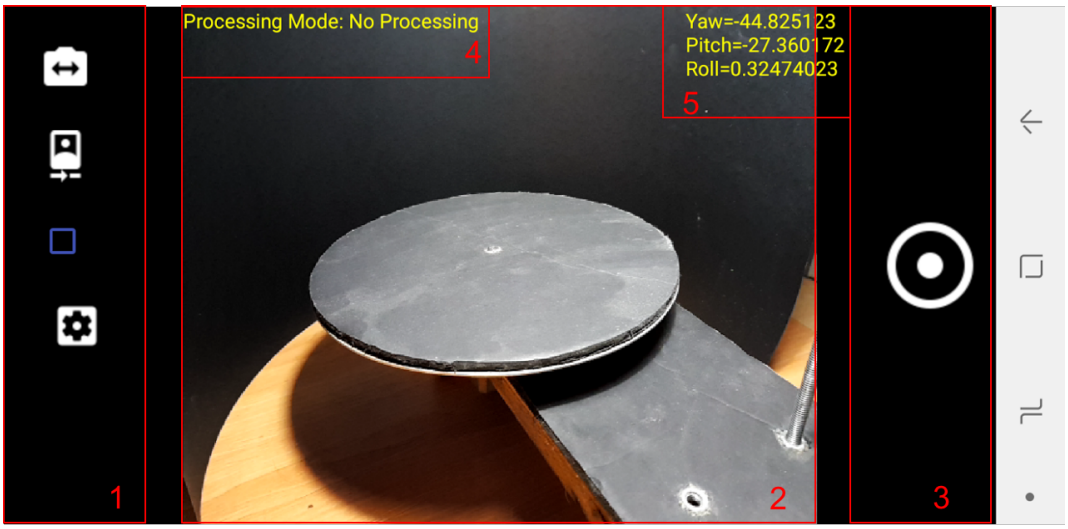


Figura 5.12: Atividade Principal Aplicação Android.

Para a construção da interface gráfica da atividade *Camera2GLActivity* foram criados três fragmentos, *LeftButtonsFragments*, *MyGLSurfaceViewFragment* e o *RightButtonsFragments*, representados repetivamente pelas áreas 1, 2 e 3 na imagem da Figura 5.12. Como já foi referido anteriormente, os fragmentos são grupos de interface do usuário que estão associados a uma *atividade*, tendo o seu próprio ciclo de vida. Uma *atividade* pode ter associados mais do que um fragmento e permite a troca em tempo real de um fragmento por outro. Posto isto, o uso de fragmentos permite uma maior modularidade para a aplicação e também uma maior flexibilidade, uma vez que permite diferentes *layouts* dependendo do tamanho do display do dispositivo em causa. A interface gráfica permite também apresentar ao utilizador o modo em que o processo se encontra representada pelo grupo 4 na imagem da Figura 5.12 além de apresentar a leitura do *Yaw*, *Pitch* e *Roll* do acelerómetro do dispositivo móvel, apresentado no grupo 5 da imagem da Figura 5.12.

5.4.1.1 Fragmento *LeftButtonsFragments*

Este fragmento é composto por quatro botões (*buttonProcessingMode*, *buttonChange*, *cbCalibrationProcess*, *buttonSettings*), para tarefas como mudar de modo de funcionamento, alternar entre a câmara frontal e a câmara traseira, escolher os algoritmos de processamento de imagem a utilizar e gerir a interface do *Bluetooth* do dispositivo. Para a criação dos botões é utilizada a ferramenta *ImageButtons* que permite associar a cada botão uma imagem e, assim, tornar a aplicação graficamente mais atraente.

É importante referir que este fragmento gere exclusivamente a interface gráfica e o evento associado ao clique dado pelo utilizador quando escolhe um determinado botão. Assim sendo, para cada

evento de botão está associada uma *callback*, que por sua vez tem uma *interface* associada na atividade principal. No Fluxograma da Figura 5.13 é apresentada a sequência dos eventos associados aos botões pertencentes ao fragmento *LeftButtonsFragments* e as funções da atividade principal associadas a cada evento. As funções na atividade principal associados aos botões *buttonProcessingMode*, *buttonChange*, *cbCalibrationProcess*, *buttonSettings* são respetivamente *showPopUp()*, *changeCamera()*, *showPopoKeypoints()* e *showPopoUpSettings()*.

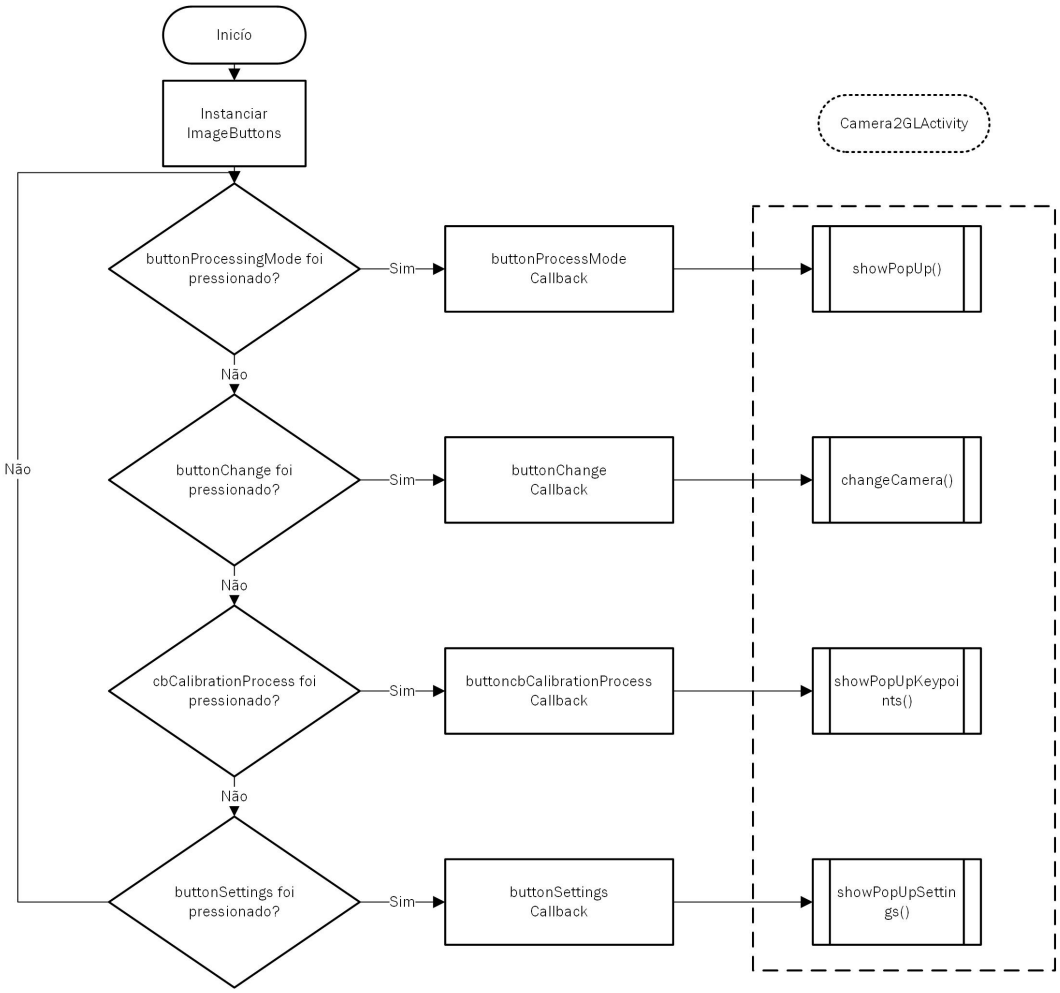


Figura 5.13: Fluxograma referente aos eventos associados ao *LeftButtonsFragments*.

A função *showPopUp()* apresenta ao utilizador uma janela *pop up*, onde o utilizador deve escolher entre os dois modos de funcionamento, são eles, *Reconstruction Mode* e *Calibration Mode*. Um terceiro modo é utilizado nesta aplicação, quando esta inicia por defeito, vem no modo *No Processing Mode*, quer isto dizer, que nenhum processamento é aplicado às imagens adquirias pela câmara.

A função *showPopoKeypoints()* à semelhança da função *showPopUp()* gera uma janela de *Pop Up* na aplicação *Android*. Esta apresenta uma lista com os respetivos algoritmos de deteção de pontos de interesse numa imagem implementados, sendo eles, *ORB*, *SIFT* e *SURF*. Por sua vez, quando o

utilizador escolhe um dos algoritmos da lista, o *index* referente à sua posição na lista é enviado para a camada de baixo nível da aplicação, responsável pela parte de processamento de imagem. Para isso, é chamado o método *detectorAlgorithm(int algorithm)* pertencente à classe *NativePart* onde o parâmetro de entrada representa o respetivo *index* do algoritmo escolhido pelo utilizador.

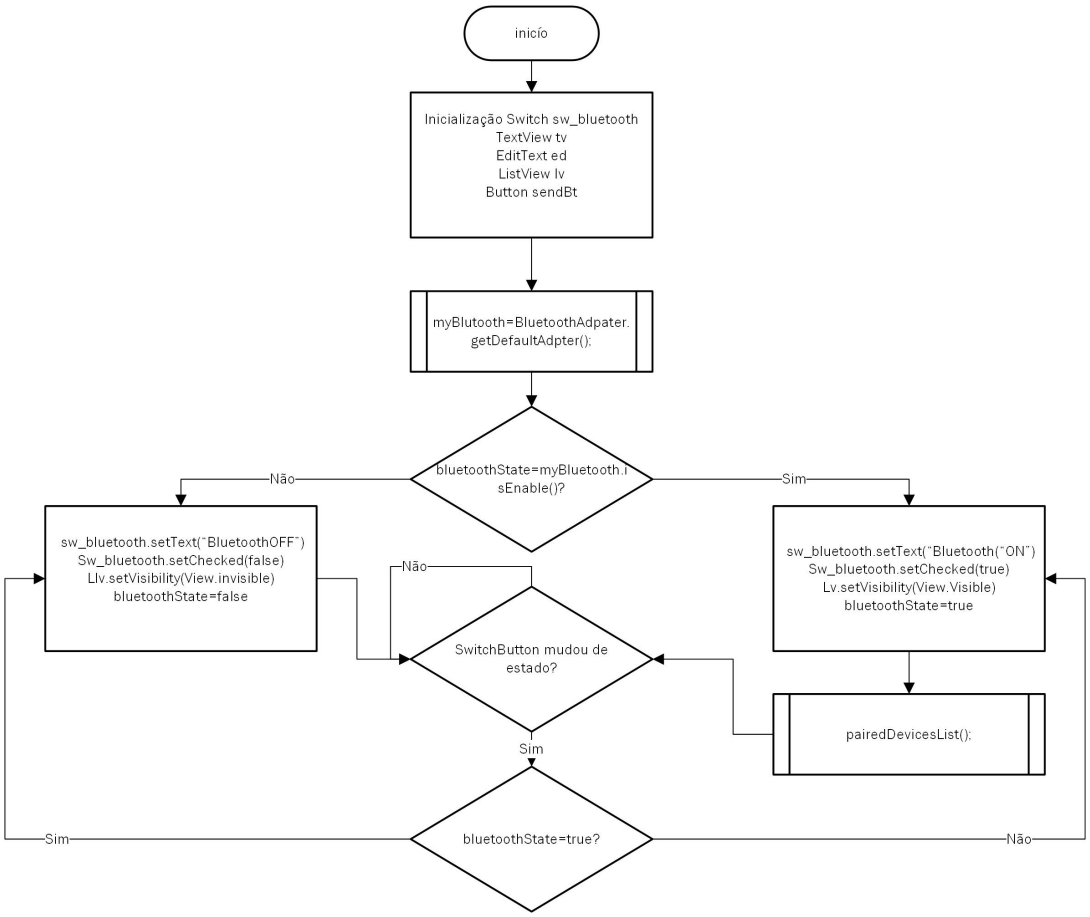


Figura 5.14: Fluxograma referente à interface *Bluetooth*.

Já a função *showPopoUpSettings()* apresenta uma janela de *Pop Up* ao utilizador com um menu onde este pode gerir as interfaces disponíveis, como por exemplo, a interface *Bluetooth* e os sensores do dispositivo móvel. Quando o utilizador escolhe um elemento da lista na janela *Pop Up* um evento é gerado e, consequentemente, é aberto um *Dialog Fragment*. Este, como já foi abordado anteriormente, é uma caixa de diálogo que permite adicionar algumas ferramentas de interface gráfica, à semelhança do fragmento que tem um ciclo de vida próprio e necessita de ser chamado a partir de um *host*, como por exemplo, uma Atividade ou mesmo um *fragmento*. Foi então criado um *Dialog Fragments* para gerir respetivamente as interfaces de *Bluetooth*.

Relativamente ao *DialogFragment* que gere a interface *Bluetooth* é composto por um *Switch-Button*, uma *TextView*, uma *ListView*, um *EditText* e um botão. Excluindo o *SwitchButton* que está

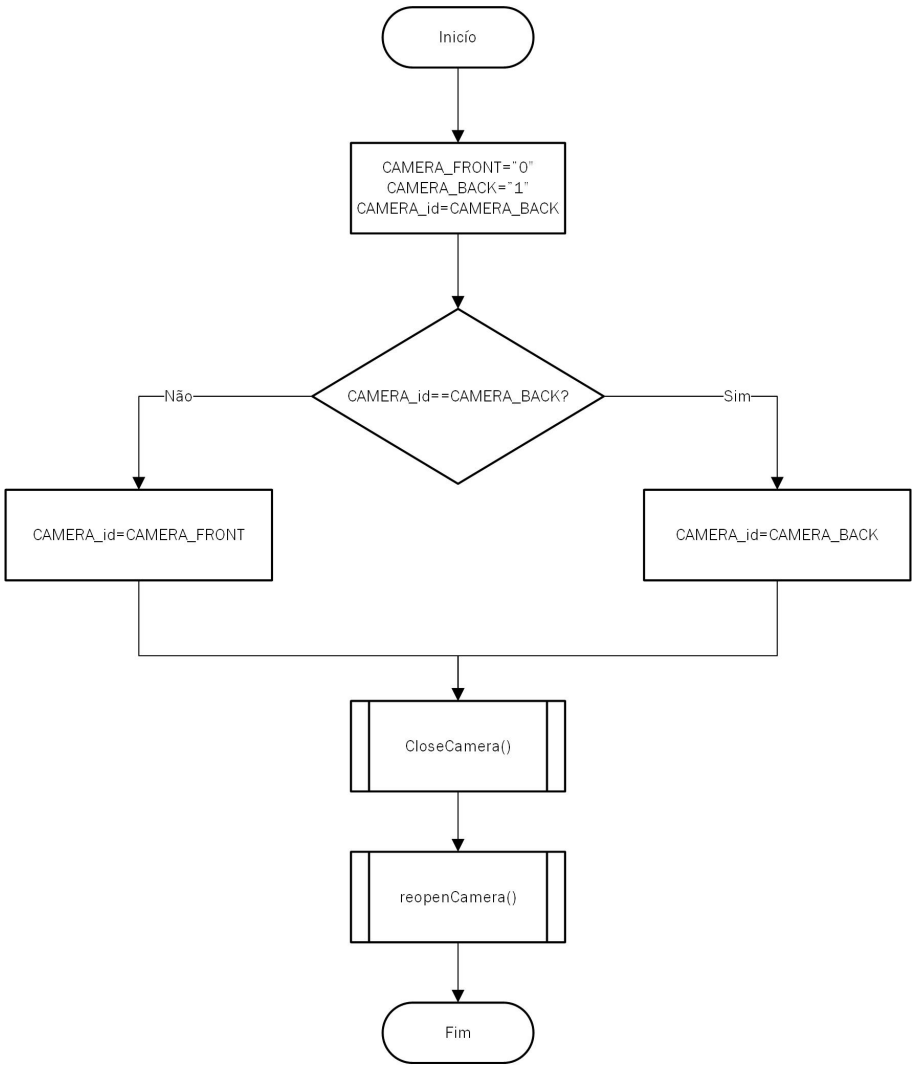


Figura 5.15: Fluxograma referente ao processo de Troca de Câmara.

sempre visível para o utilizador, as outras ferramentas têm o seu estado de visibilidade dependente do estado geral da interface *Bluetooth*, estando visíveis somente quando esta está ativa. A gestão deste estado é feito através do *SwitchButton* que permite ao utilizador ativar e desativar o *Bluetooth* sem ter que sair da aplicação. Dado isto, é necessária uma validação inicial à interface *Bluetooth* quando é inicializado o *DialogFragment*. É possível validar o estado da interface *Bluetooth* instanciando um objeto da classe *BluetoothAdpater*, através do método *getDefaultAdapter()* que representa o módulo *Bluetooth* local e permite gerir algumas das tarefas relacionadas com esta comunicação. Para validar se a comunicação está ou não ativa, é chamado o método *isEnable()*. Após a validação do estado ativo e caso se confirme que está ativo é chamada a função *pairedDevicesList()*, que é responsável por verificar os equipamentos emparelhados ao dispositivo móvel e apresentá-los na *ListView* presente no *DialogFragment*. O Fluxograma da Figura 5.14 representa esta implementação.

Por fim, a função *changeCamera()* permite alternar entre a câmara frontal e a câmara traseira. No

Fluxograma da Figura 5.15 está representada a implementação da função *changeCamera()*. É utilizada uma variável global *CAMERA_id* para registar em cada momento qual a câmara ativa. Por defeito, está no estado zero que indica que está a ser utilizada a câmara traseira. Quando é pressionado o botão *changeCamera()*, é realizada uma validação a esta variável. Consoante a validação, esta toma o valor contrário à câmara ativa no momento em que o botão foi pressionado. Após isso, a câmara ativa é fechada e é reaberta a outra câmara.

5.4.1.2 Fragmento *RightButtonsFragments*

O fragmento *RightButtonsFragments* é composto por um só botão de ação, o *buttonCapture* que é responsável por dar a ação para adquirir imagens da câmara. Mantém o mesmo paradigma do *LeftButtonsFragments*, gerindo exclusivamente a interface gráfica e o evento relacionado com o clique do botão, tendo uma *callback* associada e a respetiva *interface* implementada na atividade principal. Por sua vez, a função implementada na atividade principal faz a chamada de um método implementado na classe *NativePart*, que informa a camada de baixo nível que houve um evento no botão.

5.4.1.3 Fragmento *MyGLSurfaceViewFragment*

O fragmento *MyGLSurfaceViewFragment* é responsável por gerir o processo referente à câmara e respetiva visualização associada à interface gráfica da aplicação Android. A sequência de inicialização utilizada é a sugerida pela documentação Android relativamente à API da câmara[3]e está representada no Fluxograma presente na Figura 5.16. O processo é iniciado pela verificação do *hardware* existente no dispositivo móvel em específico, seguindo pela chamada da função *Camera.Open()* que inicia a captura de imagens. A nível de interface gráfica é criado um *FrameLayout* para armazenar cada nova imagem adquirida. Antes de apresentar a nova imagem ao utilizador, é chamada a função *processFrame()* implementada na classe *NativePart*, que será apresentada na próxima secção. Esta, por sua vez, faz a chamada das funções de processamento de imagem na camada de baixo nível, referentes ao modo no qual a aplicação se encontra a cada momento. Finalizado o processamento de aquisição e processamento de imagem, esta é colocada no *FrameLayout* para apresentação ao utilizador.

5.4.1.4 Classe NativePart

Como já foi referido e apresentado anteriormente, neste documento o código de processamento de imagem foi implementado com recurso à biblioteca de processamento de imagem *OpenCV* desenvolvida

[3]`https://developer.android.com/guide/topics/media/camera#java`

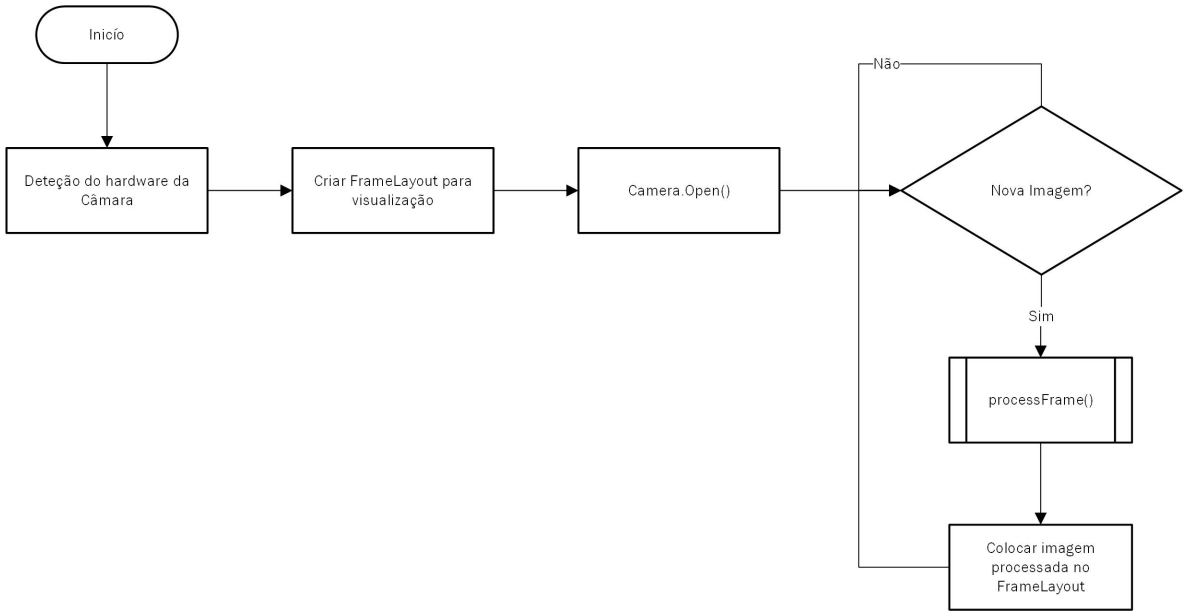


Figura 5.16: Fluxograma referente ao processo de Aquisição de imagem.

na linguagem C++. Desta maneira, foi necessário criar uma classe em Java que mediasse a interação entre o código desenvolvido nesta linguagem e o código nativo em C++ da biblioteca de processamento de imagem propriamente dita. Com esse objetivo, foi implementada a classe *NativePart*, desenvolvida na linguagem Java e composta por um conjunto de métodos que permitem aceder às funções implementadas na linguagem C++.

Os métodos nesta classe são implementados usando a *keyword native* definida antes da declaração do tipo de retorno da função, como por exemplo, *public void native tipo funcName()*. Esta *Keyword* é aplicada para indicar que o método é implementado em código nativo usando o JNI, *Java Native Interface*, por outras palavras, indica ao compilador que os métodos definidos como *native* estão implementados nativamente na linguagem C++. É importante salientar que a implementação dos métodos na linguagem C++ é realizada no ficheiro *JNIProcessor.cpp*.

Os métodos declarados nesta classe são os seguintes, *processFrame()*, *onClickButton()* e *detectorAlgorithm()* e serão apresentados de seguida. O método *detectorAlgorithm(int algorithm)* tem como parâmetro de entrada uma variável do tipo inteiro que representa o algoritmo de deteção de pontos de interesse que será utilizado durante o processo de deteção de pontos de interesse, estão implementados os seguintes algoritmos: o *ORB*, *SIFT* e o *SURF*, à semelhança do do que acontece na aplicação gráfica Qt. Cada um destes é identificado pelo valor da variável *int algorithm* que tem o valor compreendido entre 0 e 2 consoante o algoritmo escolhido.

Por sua vez, método *processFrame()*, pode ser considerado o mais importante implementado na

camada de baixo nível da aplicação, uma vez que é responsável por controlar a máquina de estados que gere os modos de funcionamento da aplicação, bem como, a chamada de todas as funções de processamento de imagem. Os modos de funcionamento são exatamente os mesmos, dos que os apresentados na Secção 5.3, *modo de inicialização*, *modo de calibração* e *modo de reconstrução*. Quando é iniciada a aplicação é realizada a primeira chamada do método *processFrame()* que entrará em modo de inicialização. É realizada a verificação da existência ou não do ficheiro de calibração, caso não exista, a aplicação entrará em modo de calibração. O utilizador será guiado através de mensagens no ecrã para iniciar o processo de calibração através do botão *buttonCapture*. Finalizado o processo de calibração é iniciado o processo de reconstrução tridimensional. Caso o utilizador escolha através dos menus ir diretamente para o modo de reconstrução sem o modo de calibração estar efetuado receberá uma mensagem a avisar que deve realizar o processo de calibração primeiro.

O método *onClickButton()* é responsável por realizar a interface entre a camada gráfica da aplicação e a camada de baixo nível, mais precisamente, relativa ao botão presente na interface gráfica. Quando o evento associado ao toque do botão de captura é desplotado a função *onClickButton()* é chamada, que por sua vez chama uma função na camada baixo nível que controla o estado de uma variável booleana. O estado desta variável é utilizado posteriormente para gerir algumas tarefas, como por exemplo, para iniciar o processo de calibração ou o de reconstrução, bem como, para adquirir imagens. É importante ainda referir que quando a aplicação está conectada à plataforma rotativa, o botão deixa de ter tantas funcionalidades, uma vez que o processo é realizado de forma automática. Assim sendo e para manter a coerência com a implementação desenvolvida, a função é chamada sempre que seja necessário, por exemplo, uma nova aquisição. Por exemplo, quando a plataforma rotativa manda uma mensagem a dizer que já está no ponto seguinte de aquisição, isso fará o programa internamente mudar o estado da variável *buttonCapture* para permitir uma nova aquisição e processamento sem tocar propriamente no botão de captura de imagem.

5.4.1.5 Interface Bluetooth

A interface *Bluetooth* nesta aplicação em específico foi desenvolvida com o intuito de permitir uma comunicação direta entre o dispositivo móvel e uma plataforma rotativa, permitindo, desta forma, a aquisição de imagens de pontos de vista diferentes em todos os 360° da superfície de um objeto alvo, de forma bastante precisa e controlada.

Para isso, foi implementada a classe *BluetoothControl*, composta esta por um conjunto de métodos que permitem gerir toda a comunicação *Bluetooth*. Os métodos *sendToBluetooth*, *DisconnectBlu-*

etooth() e *getStatusBluetoothConnection()* permitem enviar, desconectar e verificar o estado da comunicação *Bluetooth*. Por sua vez, o método *setBluetoothAdress(String address)* é responsável pela interface entre a atividade principal, *Camera2GLActivity*, e a classe que gere a comunicação *Bluetooth* e tem como principal objetivo permitir a conexão a um equipamento *Bluetooth* identificado por um endereço MAC, nesta caso em específico a plataforma rotativa. Após chamado este método, um conjunto de funções são chamadas para efetivamente consumar a conexão ao dispositivo em causa. A função *ConnectBluetooth()* é uma delas e é responsável por efetivar esta comunicação, para isso faz uso de algumas classes da API *Bluetooth* da plataforma *Android*, como é o caso da classe *BluetoothAdapter*, que permite instanciar um *BluetoothDevice* usando o endereço MAC, parâmetro de entrada do método *setBluetoothAdress()*. Por fim, é inicializada a comunicação através da criação de um *socket*. O método *setBluetoothAdress(String address)* é também responsável por iniciar uma *thread* que corre em *background* e permite a aplicação ir recebendo tramas da plataforma rotativa e desta forma proceder com algumas tarefas. Como por exemplo, o dispositivo móvel manda a base rodar e esta responde com um comando de confirmação. Este comando é recebido pela função *run thread* que continuamente está à escuta no *socket*.

5.5 Plataforma Rotativa

A criação de modelos tridimensionais, tendo em consideração as técnicas passivas, como por exemplo, a técnica de estrutura a partir do movimento, utilizada neste trabalho, baseia-se essencialmente na análise de imagens de um objeto, adquiridas de pontos de vista diferentes. Contudo, apesar do objetivo principal deste trabalho de investigação ser focado na reconstrução tridimensional baseado em dispositivos móveis, foi decidido utilizar inicialmente uma plataforma rotativa. Desta forma, foi possível obter um espaço de trabalho num ambiente mais controlado, bem como, o conhecimento exato do deslocamento do objeto a cada imagem adquirida, o que permitiu o melhor estudo dos algoritmos desenvolvidos. Na Figura seguinte, é possível observar a estrutura mecânica da plataforma desenvolvida.

5.5.1 Componentes Utilizados

A plataforma utilizada foi construída com a utilização de componentes baratos e de fácil manuseamento. Uma vez que, o objetivo era ter uma plataforma de teste, que fosse barata e precisa o suficiente para testar os algoritmos desenvolvidos durante a sua implementação no dispositivo móvel. Para a construção foram utilizados os seguintes materiais:

Figura 5.17: Plataforma Rotativa Construída.

- 1 Arduino Uno
- 1 Motor de Passo Bipolar
- 1 Fonte de Alimentação
- 1 Estrutura de Madeira
- 1 Acoplador Mecânico
- 1 Módulo Bluetooth (HC-05)
- 1 Circuito Integrado Ponte H (L293D)

5.5.2 Arduino Uno

Atualmente, no mercado existem um elevado número de plataformas, baseadas em microcontroladores. Na construção desta plataforma rotativa, em específico, foi escolhido o *Arduino Uno* [74], pois apresenta uma ótima relação custo-benefício. Além disso, têm uma biblioteca de desenvolvimento bastante abrangente e de fácil aprendizagem, baseada na linguagem C/C++, suportada por uma enorme comunidade *online*. O seu custo benefício, aliado à sua biblioteca de desenvolvimento, torna esta plataforma ideal para o desenvolvimento de projetos de baixa complexidade e que necessitem de uma rápida implementação.

O *Arduino Uno* é baseado no microcontrolador *ATmega328P*, produzido pela ATMEL, possui 14 pinos de entrada/saída, dos quais 6 podem ser utilizados como saídas PWM (*Pulse Width Modulation*), possui também 6 entradas analógicas. A placa pode ser alimentada através da conexão USB, bem como através de uma fonte externa de energia. Para a programação do microcontrolador foi utilizado o *Arduino IDE* [75], que por sua vez, permite a programação numa linguagem de alto nível semelhante e baseada em C++. O *IDE* é desenvolvida na linguagem Java.

5.5.3 Módulo Bluetooth

Para a comunicação com a aplicação Android foi utilizada a tecnologia *Bluetooth*. Posto isto, e uma vez que, o Arduino Uno não possui integrado nenhum módulo que permita a comunicação através desta tecnologia, optou-se então pela utilização do módulo *HC-05* [76]. Este módulo é um conversor *Bluetooth* para comunicação série e tem um custo bastante reduzido, sendo por isso, uma opção bastante económica para se adicionar uma conexão *Bluetooth* a um projeto.

Relativamente à sua integração com a placa Arduino, é relativamente simples, uma vez que, é ligada diretamente a uma porta série física disponível na placa no microcontrolador. É necessário na primeira utilização definir previamente no próprio módulo *HC-05* o *baudrate* da comunicação série. Feito isto, a comunicação série no Arduino deve ser configurada para comunicar ao *baudrate* definido no módulo.

5.5.4 Motor de Passo Bipolar

Um motor de passo é um transdutor que converte pulsos elétricos em movimento mecânico de rotação. Ao contrário dos motores elétricos convencionais, os motores de passo são caracterizados por uma rotação do eixo do motor muito mais precisa. Cada rotação completa do eixo do motor é caracterizada por um número bem definido de passos individuais, onde cada passo, corresponde a um ângulo de rotação preciso. Este motores são frequentemente divididos em dois grupos, sendo eles, os motores de passo unipolares e os motores de passo bilopares.

No caso especifico, desta plataforma rotativa, foi utilizado um motor de passo bipolar, visto que têm a grande vantagem de oferecer mais torque. Normalmente, exigem circuitos eletrónicos mais complexos para o seu controlo, fisicamente têm enrolamentos separados, sendo necessário uma polarização reversa durante a operação para se consumar o passo. O motor de passo utilizado nesta plataforma rotativa, permite efetuar 200 passos por evolução, o que corresponde a 1,8° por passo.

5.5.5 Circuito Eletrónico para Controlo do Motor de Passo

Apresentada a estrutura mecânica da plataforma, bem como, os seus principais componentes eletrónicos constituintes, é de seguida apresentado uma ilustração do esquema elétrico final, representado na imagem da Figura 5.18.

É importante voltar a referir que o módulo *bluetooth* se conecta ao Arduino na única porta série disponível nesta placa, ou seja, o pino *TX0* do Arduino conecta-se ao pino *RX* do módulo, por sua vez, o pino *RX0* do Arduino liga-se ao pino *TX* do módulo.

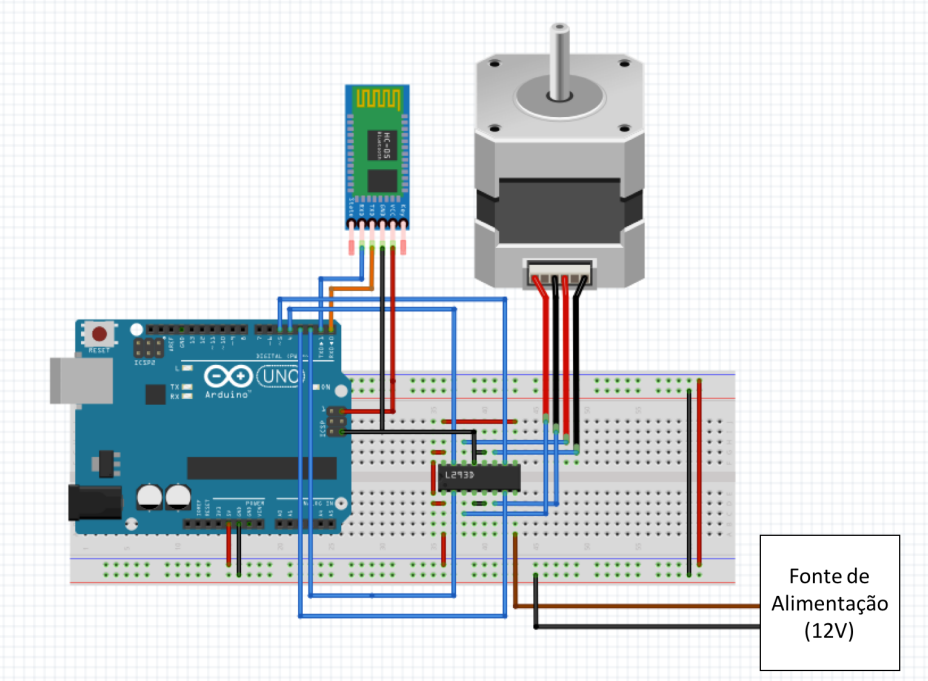


Figura 5.18: Ilustração do esquema elétrico de controlo de rotação da plataforma rotativa.

Relativamente à ligação entre o microcontrolador e o motor de passo, é utilizado o circuito integrado *L293D* [77], que por sua vez, é um circuito *Ponte H* e permite controlar até 2 motores DC ou um motor de passo bipolar. A utilização deve-se à incapacidade das portas lógicas do Arduino debitarem a corrente necessária para alimentar corretamente as bobines presentes no motor de passo.

5.5.6 Software para Controlo de Motor de Passo

Para finalizar, é apresentado na Figura 5.19 o Fluxograma referente ao código implementado no Arduino para controlo do motor de passo. Foi utilizada a biblioteca *Stepper*[4]do Arduino para o interface com o motor de passo.

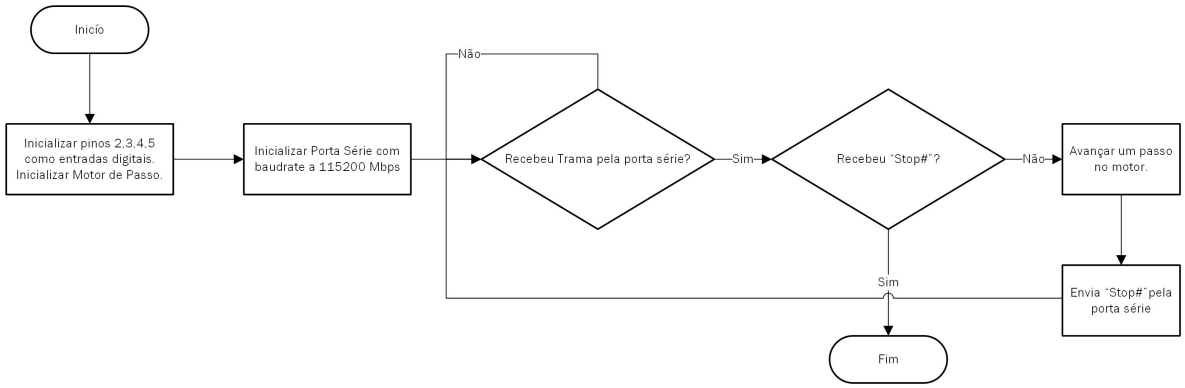


Figura 5.19: Fluxograma Controlo de Motor de Passo.

Resumidamente, após as inicializações dos quatro pinos que controlam o motor de passo, definidos com saídas digitais e após a inicialização do objeto *Stepper* e da porta série, o programa fica à espera de uma trama através da porta série. Todo o processo é controlado através da aplicação Android, ou seja, o microcontrolador não sabe quantos passos o motor já andou. A aplicação manda a informação para o motor avançar, o Arduino descodifica essa informação e manda o motor avançar um passo. Finalizado o movimento é enviada de volta para a aplicação uma trama a dizer *"Stop#"* que indica à aplicação que o movimento já foi finalizado e pode adquirir uma nova imagem. Caso a aplicação Android envie a mensagem *"Stop#"* significa que o processo de aquisição foi finalizado.

[4]Biblioteca Stepper: `https://www.arduino.cc/en/reference/stepper`

Capítulo 6

Resultados

Neste capítulo serão apresentados os resultados práticos obtidos ao longo desta dissertação, focada na reconstrução tridimensional e nos algoritmos a si associados. Dado isto, de modo a apresentar os resultados de forma clara e o mais descritiva possível, um ciclo completo a todo o processo será realizado, desde a aquisição das imagens até à geração da nuvem de pontos. Serão apresentados dois casos de estudo, onde serão utilizados dois objetos de estudos diferentes.

6.1 Processo de Calibração

A primeira fase do processo de reconstrução tridimensional, como já foi referido anteriormente no Capítulo 5, consiste na calibração do equipamento que realizará a aquisição das imagens utilizadas durante todo o processo. No caso específico, foi utilizado um *smartphone One Plus One* da marca *One Plus*, equipado com uma câmara de 13 megapixéis, f/2.0 e uma abertura 1/3. Como já foi referido, o processo de calibração, permite obter os parâmetros intrínsecos da câmara, mais precisamente, a distância focal, o ponto principal, os coeficientes de distorção radial e os coeficientes de distorção tangencial.

O processo consiste na aquisição de vinte imagens de um padrão de calibração ao estilo de um tabuleiro de xadrez. A sua geometria deve ser bem conhecida, ou seja, devem ser conhecidas as distâncias entre os diferentes quadrados do respetivo padrão de calibração. Para realizar o processo de calibração são utilizadas as funções do OpenCV que estão incluídas na classe *CameraCalibrator*, explicada na Secção 5.3.2. A imagem presente na Figura 6.1 representa o processo de calibração realizado através da aplicação Android desenvolvida.

Pela observação ainda da Figura 6.1, é possível verificar a deteção dos 66 pontos inferiores presentes no padrão de calibração utilizado. Neste caso específico, foi utilizado um padrão de 12 colunas por

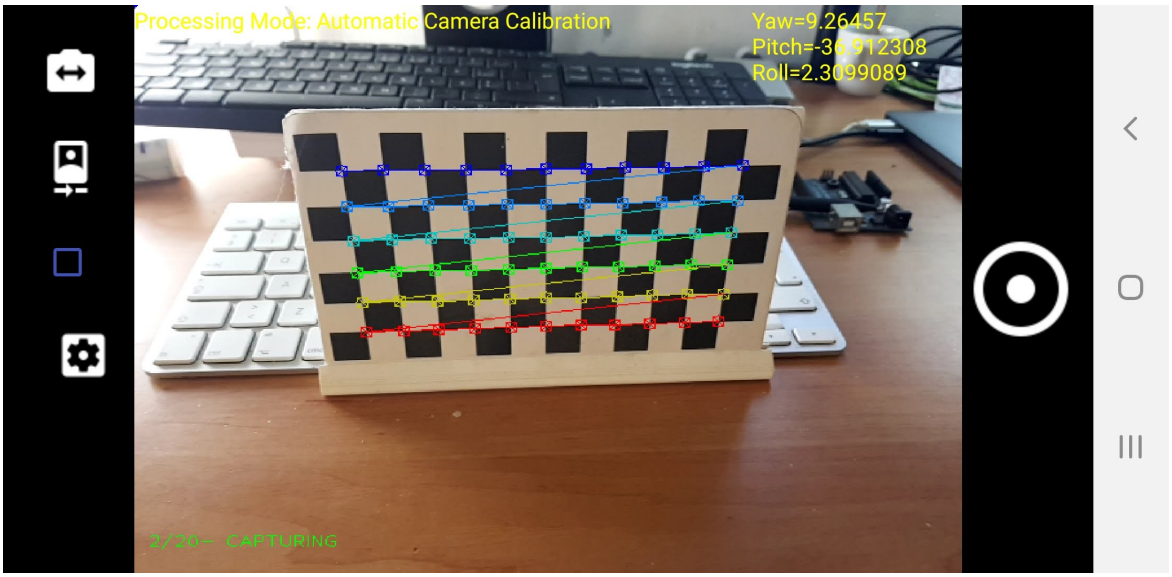


Figura 6.1: Exemplo de processo de calibração na aplicação Android desenvolvida

7 linhas. É importante referir que o *software* só permite a apresentação visual dos pontos interiores detetados, quando a função *findAndDrawPoints()* retorna um valor positivo.

Entre cada aquisição de uma nova imagem, vão sendo armazenadas num vetor as coordenadas dos pontos interiores detetados e calculados as projeções tridimensionais correspondentes, considerando Z=0. Após a deteção de todos os pontos interiores de todas as imagens e calculadas as respetivas projeções, é chamada a função do OpenCV *calibrateCamera()*, a qual retorna a matriz dos parâmetros extrínsecos, bem como, os coeficientes de distorção calculados.

A informação obtida após o processo de calibração é guardada internamente no programa para ser posteriormente utilizada ao longo do processo de reconstrução tridimensional. Por sua vez, automaticamente finalizado o processo de calibração o ficheiro *parameters.xml* é gerado. Desta forma, é possível a sua importação na aplicação gráfica Qt. A tabela 6.1 contem os valores dos parâmetros intrínsecos obtidos no final do processo de calibração e que serão utilizados nos casos de estudo apresentados na próxima secção.

Parâmetros Intrínsecos	
Distância Focal (f_x e f_y)	910,36 e 913,82
Ponto Principal (c_x e c_y)	310,18 e 242,70

Tabela 6.1: Valores referentes à matriz de calibração, após finalização processo de calibração.

6.2 Casos de Estudo

Nos casos de estudos apresentados de seguida são utilizadas vinte e uma imagens adquiridas previamente pela aplicação *Android*, com o auxílio da plataforma rotativa, completando uma volta de 360 graus pela superfície do objeto alvo. Cada imagem adquirida tem uma resolução *640x480* pixéis. Posteriormente, foram processadas através da aplicação gráfica QT. É também importante ter em consideração que foi utilizado um computador fixo equipado com um processador Ryzen 5 2400G 3.66GHz com 16GB de Memória DDR4 3200MHz, a correr o Ubuntu 18.04 LTS. Para a apresentação dos resultados serão analisadas as seguintes fases: deteção e emparelhamento de pontos de interesse, remoção de *outliers* e triangulação e geração de nuvens de pontos tridimensionais. Estas fases foram descritas detalhadamente ao longo da Secção 5.3.3 deste trabalho, referente à descrição da implementação. É também realizada a apresentação dos tempos de processamento para cada um dos estudos em questão. É importante ainda referir que apesar das imagens colocadas neste documento estarem apresentadas no espaço de cores RGB, todo o seu processamento é realizado numa imagem monocromática, numa escala de cinza.

6.2.1 Caso de Estudo 1

Neste caso de estudo em particular, foi utilizado um objeto com forma cilíndrica, apresentado na imagem da Figura 6.2. Foram analisadas as vinte e uma imagens, o que corresponde a vinte pares de imagens analisados. De forma a não ficar muito exaustivo, devido ao elevado número de imagens, optou-se por apresentar os dados relativos a apenas um par, para cada uma das fases consideradas pertinentes no processo de reconstrução tridimensional.

Para o caso de estudo seguinte, foram utilizadas as seguintes condições nos respetivos algoritmos, nomeadamente, foi utilizado o algoritmo *SURF* para a deteção e descrição dos pontos de interesse, para a filtragem dos *outliers* foi utilizado o teste de rácio, bem como, uma confiança de 0.99 no cálculo da matriz essencial.

Figura 6.2: Objeto de estudo Caso de Estudo 1.

6.2.1.1 Deteção e Emparelhamento de Pontos de Interesse

Após a aquisição das imagens, o processo entra na fase de deteção e emparelhamento de pontos de interesse e respetivos descritores. Como já foi abordado anteriormente, um ponto de interesse é um pixel que contem uma característica que o faz distingir dos demais pixéis que, pelo contrário, não são considerados pontos de interesse. Estes são caracterizados a partir da sua localização no plano da imagem através das suas coordenadas x e y. Contrariamente, os descritores permitem caracterizar os pontos de interesse, tendo em consideração outras características além da sua posição. Dependendo do algoritmo utilizado para os detetar são invariantes à escala e também à rotação, o que quer dizer que independentemente da escala da imagem ou da sua orientação o ponto de interesse deve ser detetado da mesma forma. Ou seja, para comparar dois pontos de interesse, são necessários os seus descritores, uma vez que estes contêm as características que os permite comparar.

Figura 6.3: Emparelhamento inicial através do *Brute Force Matcher*.

Na Figura 6.3 está representado o resultado referente à deteção, caracterização e emparelhamento de pontos de interesse. No caso em específico, estão representadas a imagem 1 e 2 da sequência das vinte e uma adquiridas. Na imagem 1 foram encontrados 1100 pontos fortes, enquanto que na imagem 2 foram encontrados 993 pontos fortes. No final desta primeira fase de emparelhamento, para cada ponto de interesse encontrado na imagem 1 são emparelhados dois pontos de interesse na segunda imagem. Isto corresponde a 2200 emparelhamentos realizados entre a primeira e segunda imagem, uma vez que os 1100 pontos da primeira imagem têm dois correspondentes na segunda imagem. Foi utilizado o *BruteForce descriptor matcher* para o processo de emparelhamento. Neste caso, para cada ponto de interesse encontrado na primeira imagem, o seu descritor foi comparado individualmente com cada descritor correspondente a um ponto de interesse na segunda imagem. Como é possível observar pela imagem da Figura 6.3, muitas correspondências não são coerentes entre a primeira e a segunda imagem, dado ter sido adquirido com a câmara parada na mesma posição. Assim, era expectável que a posição

em y dos pontos de interesse pares fosse bastante semelhante, o que não se verifica em algumas casos na imagem da Figura 6.3. Posto isto, é necessário filtrar todos os pontos de interesse e, desta forma, validar quais os emparelhamentos que estão realmente corretos. Como método de filtragem foi utilizado o teste de rácio, neste exemplo em específico. São eliminadas correspondências que não sejam duplas e as quais tenham um rácio entre distâncias correspondentes não superiores a um determinado valor, que no caso foi definido para 0.7. No fim da filtragem, através do método referido, para este primeiro par, foram removidas 636 correspondências, obtendo-se no final 464 correspondências bem sucedidas. Na Figura 6.4 é apresentado o estado do processo após a aplicação do filtro do rácio para o primeiro par de imagens. A verde estão representados os pontos de interesse na primeira e na segunda imagem, bem como, a sua correspondência. Por outro lado, a vermelho, estão representados os pontos de interesse removidos. Resumidamente, na primeira imagem foram encontrados 1100 pontos de interesse, na segunda imagem 993 pontos de interesse, sendo que foram removidos 626, obtendo, desta forma, 464 correspondência para o primeiro par de imagens.

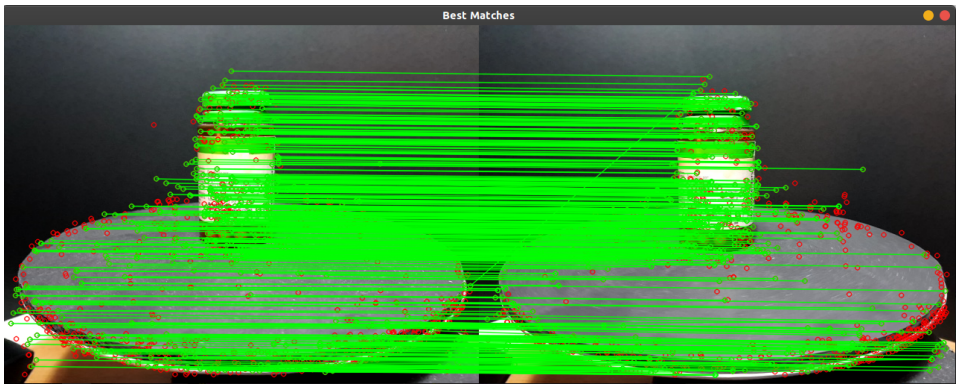


Figura 6.4: Emparelhamentos de pontos de interesse após aplicação do teste de rácio. A verde estão representados os *inliers* e a vermelho os *outliers*.

6.2.1.2 Remoção de Outliers

Antes de finalizar o processo de deteção e emparelhamento de pontos de interesse, uma última validação é necessária, uma vez que o objetivo final é uma nuvem de pontos no espaço tridimensional. Deste modo, é necessário validar as correspondências, tendo em consideração o processo matemático inerente ao processo de formação de imagens. Como já foi abordado na Secção 3.2.2 deste documento, de forma resumida, uma correspondência para poder ser triangulada, o ponto correspondente de interesse deve estar contido na linha epipolar na imagem par. Na prática, isto é conseguido através do cálculo da matriz essencial e da aplicação da matriz dos parâmetros intrínsecos da câmara, daí advém a necessidade

prévia do processo de calibração. Após o cálculo da matriz essencial é possível obter o número de emparelhamentos corretos e incorretos, segundo a geometria epipolar, os chamados *inliers* e *outliers*. Neste caso específico, das 464 correspondências após o processo de filtragem 357 foram consideradas *inliers* e 107 consideradas *outliers*. O processo de deteção, emparelhamento e filtragem de pontos de interesse foi realizado para os vinte pares de imagens, sendo que os resultados estão resumidos para cada par de imagens no gráfico da Figura 6.5. Cada conjunto de gráficos de barras é correspondente a um par de imagens, a primeira barra (cor azul marinho) e a quarta (cor cinza) são, respetivamente, o número de pontos de interesse detetados na primeira e segunda imagem de cada par. O segundo (cor laranja) e terceiro (cor amarela) representam, respetivamente, o número de pontos após a aplicação do emparelhamento e teste de rácio. No quinto gráfico (cor azul) estão representados o número de pontos removidos através do processo de filtragem. Por fim, nos últimos dois gráficos de barras (cor verde e azul escuro) são apresentados, respetivamente, os *inliers* e os *outilies* após o cálculo da matriz essencial. Para a próxima fase do processo de reconstrução tridimensional são utilizados os *inliers* representados pelas barras de cor verde e o número de pontos de interesse das imagens, representado pela barra cinza, isto para cada um dos vinte pares de imagens.

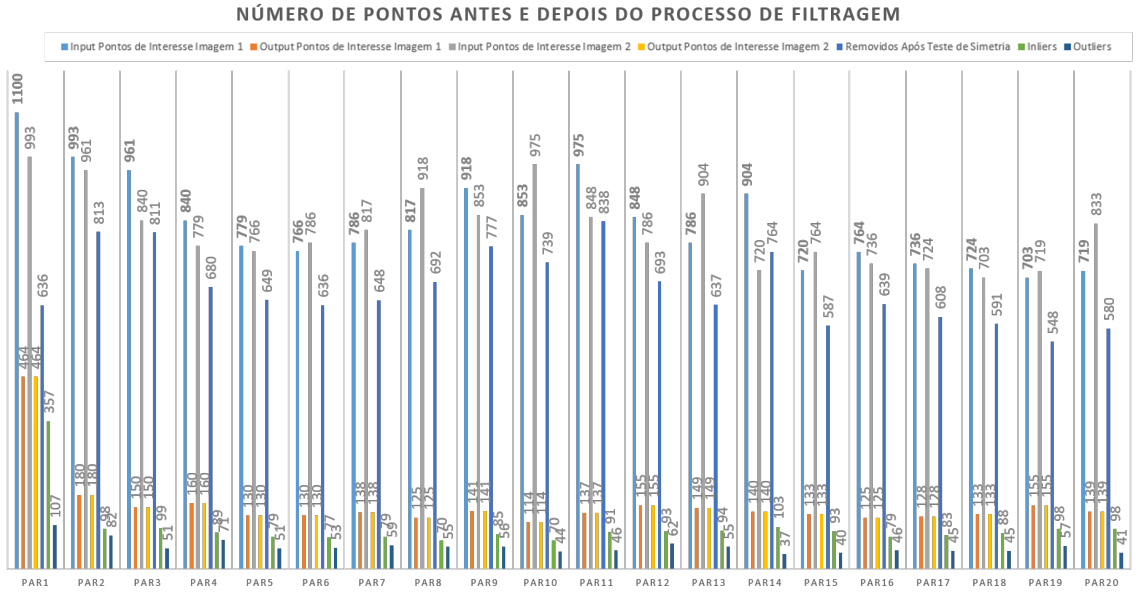


Figura 6.5: Número de pontos de interesse relativos a todos os pares de imagem durante o processo de deteção e emparelhamento.

6.2.1.3 Triangulação e Geração de Nuvem de Pontos

A triangulação é última fase deste processo de reconstrução tridimensional, sendo que a sua implementação é explicada na Secção 5.3.3.3 e os resultados para o Caso de Estudo 1 são apresentados pelo gráfico de barras presente na Figura 6.6. Resumidamente, esta fase consiste no cálculo das coordenadas tridimensionais dos *inliers* obtidos no processo anterior, o que resulta numa nuvem de pontos final constituída por 2023 pontos.

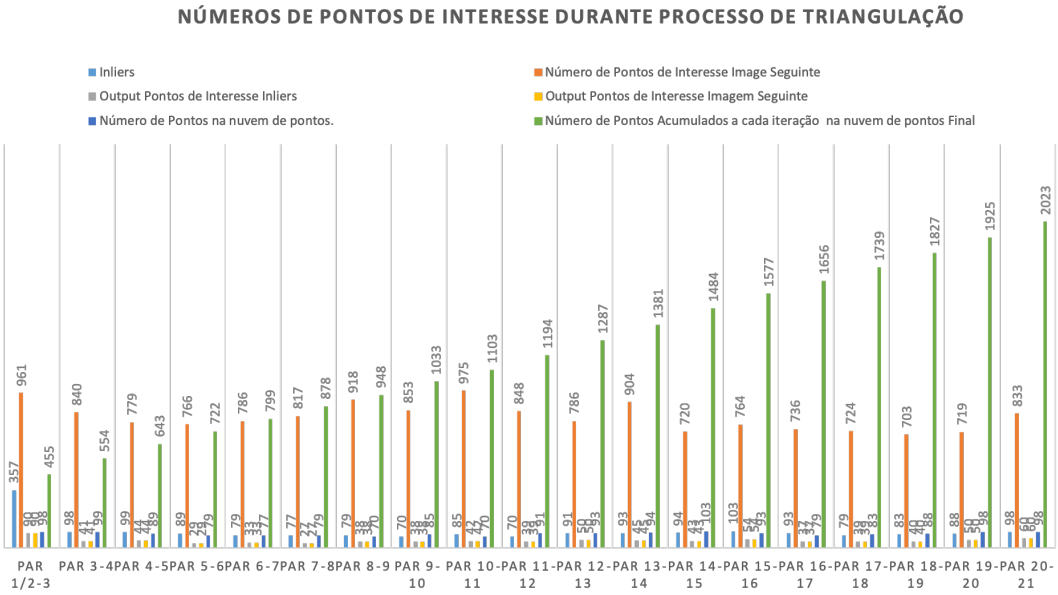


Figura 6.6: Números de pontos de interesse e número de pontos na nuvem de pontos final durante o processo de triangulação

O processo inicia-se por uma triangulação inicial aos *inliers* referentes ao par 1. Dos 464 emparelhamentos encontrados entre a imagem 1 e a imagens 2, após a validação da geometria epipolar através do cálculo da matriz essencial restaram 357 emparelhamentos, dos quais, 107 foram rejeitados. Estes 357 pontos são classificados como *inliers*. De forma a triangular estes 357 pontos foi necessário, em primeiro lugar, obter as matrizes de projeção correspondentes. Isto equivale a calcular as matrizes de rotação e translação que caracterizam o movimento entre as duas imagens constituintes do par 1. Para o cálculo destas matrizes recorreu-se ao cálculo da matriz essencial , que por sua vez, é aplicada à função *recorverPose()*. Obtidas estas matrizes, é possível aplicar as equações 3.2.18 e 3.2.19. Após isto, os 357 pontos foram triangulados a partir da chamada da função *triangulatePoints*, originando a primeira nuvem de pontos tridimensionais deste processo. Estes pontos foram armazenados num vetor geral que representa a nuvem de pontos final. Por outro lado, esta nuvem de pontos inicial é utilizada na próxima iteração, que permite triangular os *inliers* referentes ao segundo par de imagem, ou seja, constituídos pela

imagem 2 e 3. Durante as próximas iterações foi utilizada a função *solvePnPRansac()* para o cálculo das matrizes de rotação e translação, de forma a criar as respetivas matrizes de projeção que permitem triangular os pontos no espaço tridimensional. Como já foi referido, a função *solvePnPRansac()* necessita de um conjunto de pontos tridimensionais do objeto e respetivas correspondências desses pontos com os pontos de interesse da imagem seguinte para retornar a matriz de rotação e translação. Dado isto, é necessário obter a informação dos pontos de interesse e respetivos descritores referentes à nuvem de pontos formada pelos 357 pontos. Após conseguido essa informação, é iniciado um processo de emparelhamento entre esses pontos e os pontos de interesse e respetivos descritores da imagem seguinte, onde também um teste de rácio é aplicado, de forma a encontrar os melhores pares de pontos. Finalizado isto, estes emparelhamentos referentes ao par 2, bem como, a nuvem de pontos tridimensional são aplicados à função *solvePnPRansac()* que, por sua vez, retorna as respetivas matrizes de rotação e translação que permitem calcular as matrizes de projeção. Por fim, a partir das matrizes de projeção é possível triangular os correspondentes *inliers*. No gráfico de barras presente na Figura 6.6, é possível observar este processo para todos os pares de imagem. Como exemplo, é possível observar o primeiro conjunto de barras, onde os 357 pontos iniciais são emparelhados com os 961 pontos referentes à terceira imagem. Este emparelhamento resulta após aplicação do teste de rácio em 90 correspondências, que por sua vez, são utilizados para o cálculo das matrizes de rotação e translação que constituem as matrizes de projeção. Posto isto, são triangulados os 98 *inliers* deste par de imagens. É também possível observar através da barra verde o número de pontos presentes na nuvem de pontos a cada nova iteração. Por exemplo, no fim desta iteração a nuvem contem 455 pontos, que corresponde à soma dos 357 pontos triangulados inicialmente mais os 98 pontos triangulados na segunda iteração. A representação da nuvem de pontos final é apresentada pela imagem da Figura 6.7. Esta foi retirada da janela OpenGL da aplicação gráfica QT. Este exemplo em específico é composto pelos 2023 pontos triangulados durante o processo de triangulação dos pontos de interesse classificados como *inliers* na fase de emparelhamento e filtragem.

Na Figura 6.8 é representado a nuvem de pontos de um ponto de vista lateral, desta forma é possível observar a profundidade do objeto presente na nuvem de pontos.

Pela observação das duas figuras, respetivamente a Figura 6.7 e Figura 6.8 é possível verificar que se obteve formas semelhantes à forma original do objeto de estudo utilizado. Outra observação importante é relativa à cor do objeto, e que também corresponde às cores originais do objeto, onde é possível verificar as camadas realizadas pelas diferentes cores.

Outro teste foi também realizado e está representado na Figura 6.9. Este teste foi realizado com as mesmas condições do anterior, ou seja, utilizando o algoritmo SURF e o teste de rácio para filtragem dos

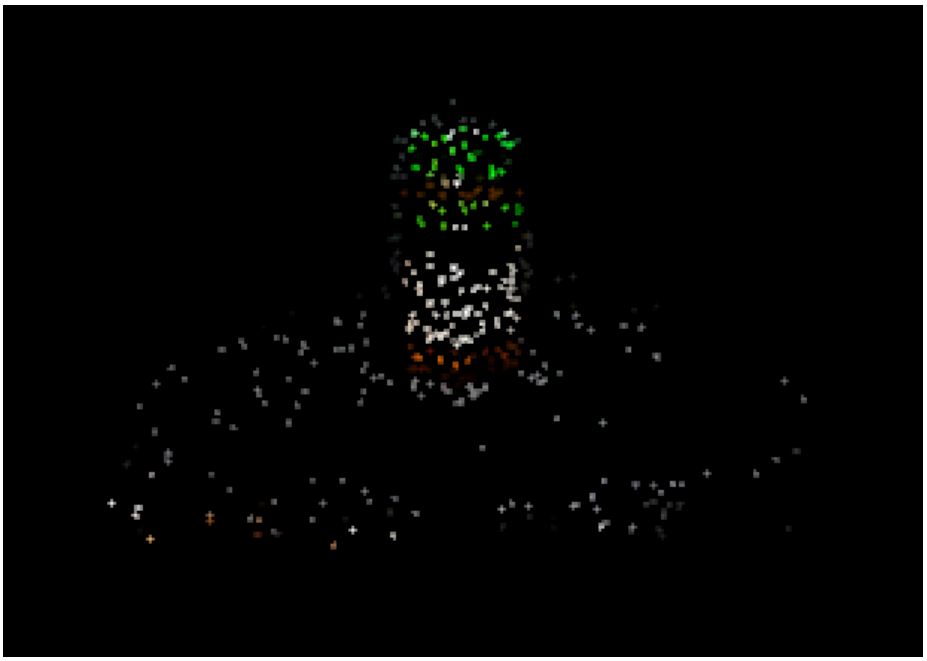


Figura 6.7: Nuvem de pontos obtida para o caso de estudo 1, apresentada de uma vista frontal.

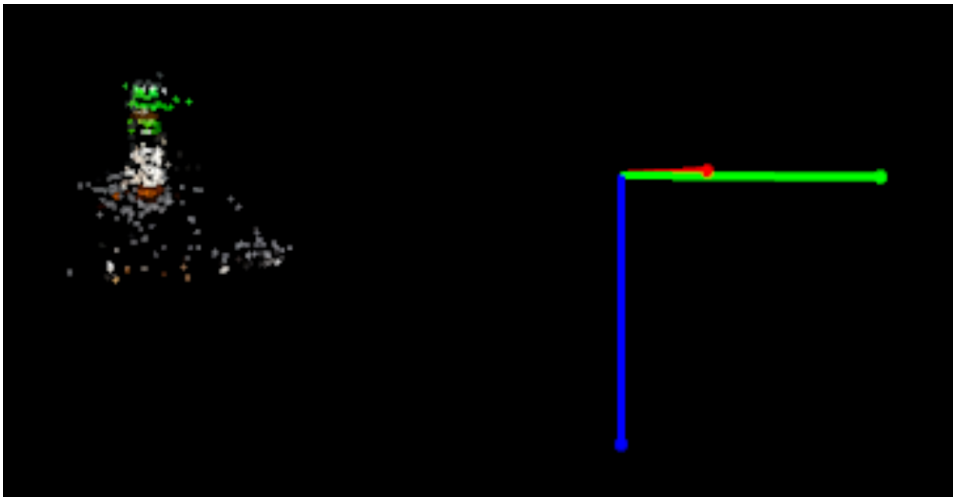


Figura 6.8: Nuvem de pontos obtida para o caso de estudo 1, apresentada de uma vista lateral.

pontos de interesse, mas com a particularidade de ter sido utilizado 0.8 para a confiança no cálculo da matriz essencial, contrariamente aos 0.99 utilizados no caso anterior.

É possível observar pela imagem, que com a diminuição do nível de confiança no cálculo da matriz essencial, por um lado o número de pontos aumentou uma vez que foram obtidos 2752 ao contrário dos 2093 obtidos no caso anterior. Por outro lado verifica-se uma alteração considerável na forma obtida, continuando a perceber que se trata do mesmo objeto. O cálculo da matriz essencial, como já foi abordado, é responsável numa primeira instância pela classificação dos *inliers* e *outliers*, e uma vez que, foi diminuída a confiança no seu cálculo é expectável um maior número de pontos ser classificado como

inliers.

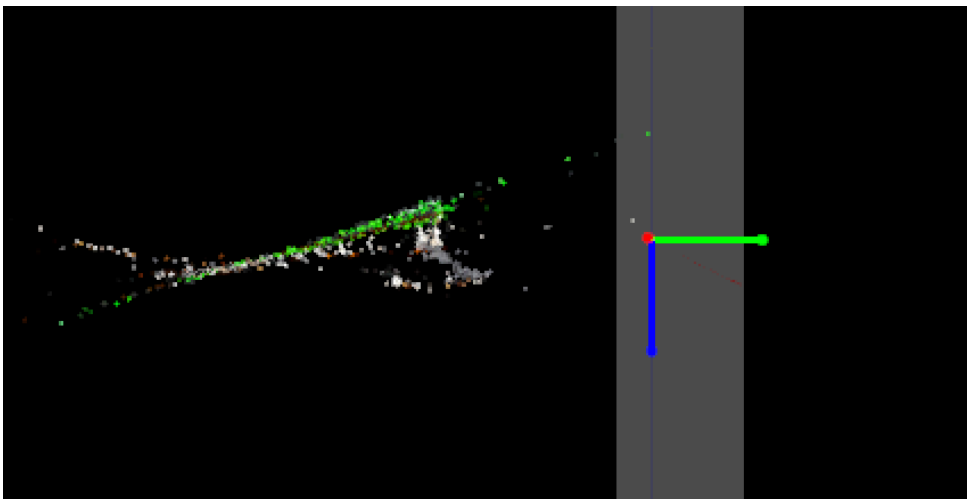


Figura 6.9: Teste realizado com 0.8 de confiança no cálculo da matriz essencial, apresentado de uma vista lateral.

6.2.1.4 Tempos de Processamento

Analisados os números de pontos obtidos em cada uma das fases do processo de reconstrução tridimensional, é importante agora analisar os tempos de processamento. Na Tabela 6.2, são apresentados os tempos para cada uma das fases apresentadas anteriormente. O tempo total desde a fase de deteção de pontos de interesse até à fase de triangulação foi aproximadamente 1,9 segundos, correspondente ao somatório dos tempos de cada uma das fases individualmente. A etapa mais demorada é a primeira, onde são realizados a deteção, emparelhamento e filtragem dos pontos de interesse encontrados nas vinte e uma imagem adquiridas, sendo que esta fase demora aproximadamente 1,149 segundos. A segunda fase é referente à computação dos respetivos *inliers* e *outliers* para cada par de imagens, demorando cerca de 0,615 segundos. Por fim, a fase da triangualção demora aproximadamente 0,112 segundos a completar o processo nos 2023 pontos constituintes da nuvem de pontos final.

É importante realizar uma análise mais profunda à primeira fase do processo, visto ser a mais demorada. Foi utilizado neste caso específico o algoritmo SURF para realizar a deteção e descrição dos pontos de interesse. No gráfico da Figura 6.10 são apresentados os tempos de processamento necessários para detetar e descrever todos os pontos nas vinte e uma imagens da sequência. As barras a azul representam os tempos necessários para detetar os pontos de interesse nas respetivas imagens, enquanto as barras a laranja representam o tempo para descrever os pontos de interesse para cada uma das imagens. Resumidamente, são necessários 0,344 segundos para detetar todos os pontos em todas as imagens da

Fases do Algoritmo	**Tempo**
Deteção e Emparelhamento de Pontos de Interesse	1,149 segundos
Remoção de *Outliers*	0,615 Segundos
Triangulação	0,112 Segundos
Tempo Total	1,876 Segundos

Tabela 6.2: Tempos de processamento para cada uma das fases do algoritmo de reconstrução tridimensional

sequência, enquanto são necessários 0,646 segundos para descrever os mesmo pontos nas imagens. Os restantes 0,159 segundos são utilizados pelos teste de rácio para filtrar os emparelhamentos mal sucedidos. Por fim, em média são necessários 0,164 segundos para detetar os pontos em cada uma das imagens e 0,308 segundos para os descrever.

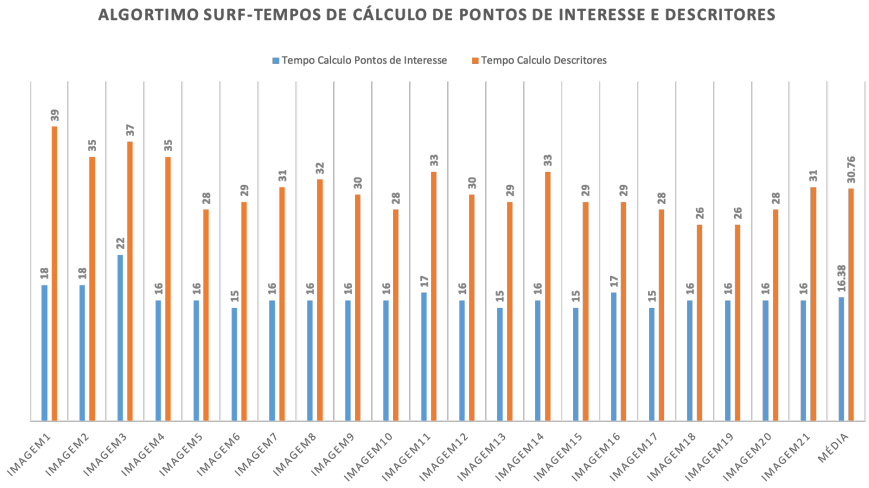


Figura 6.10: Tempos de deteção e descrição de pontos de interesse no caso de estudo 1, utilizando o algoritmo SURF.

De forma a obter dados para comparação, são apresentados mais dois casos de estudo. Um onde é utilizado o algoritmo SIFT para realizar a deteção e descrição dos pontos de interesse e o outro onde é utilizado o algoritmo ORB. Relativamente ao primeiro caso de estudo, os resultados são apresentados através do gráfico de barras presente na Figura 6.11. São obtidos um tempo total de processamento na primeira fase de 1,544 segundos, onde 0,787 são correspondentes à deteção dos pontos de interesse e 0,757 correspondem à respetiva descrição. Dados estes valores, é possível concluir que algoritmo

SIFT demora mais tempo para detetar todos os pontos de interesse em cada uma das imagens. Outro ponto a ter em consideração é que o número de pontos de interesse encontrados em cada uma das imagens é normalmente menor ao encontrado através do algoritmo SURF, o que ainda piora os tempos de processamento. Comparando os 1100 pontos encontrados na primeira imagem, o algoritmo SURF demorou cerca de 18 milisengundos, enquanto o algoritmo SIFT para encontrar 501 pontos de interesse na mesma imagem demorou cerca de 44 milissegundos.

Por fim, foi também realizado um teste utilizando o algoritmo ORB onde em média o algoritmo demora 3 milisegundos para detetar os pontos de interesse numa imagem e 4,24 milisengundos para os descrever. Na primeira imagem o algoritmo encontrou 2041 pontos de interesse em 4 milisegundos.

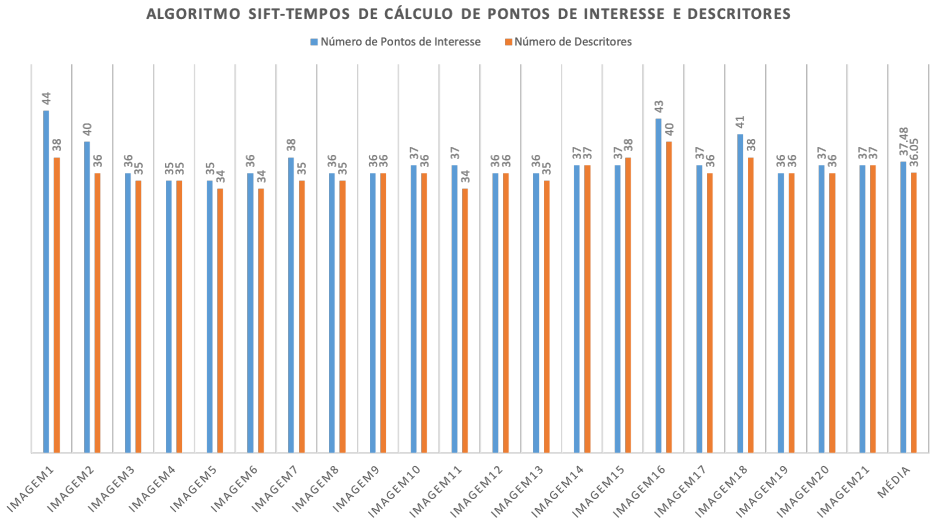


Figura 6.11: Tempos de deteção e descrição de pontos de interesse para o objeto do caso de estudo 1, utilizando o algoritmo SIFT

.

6.2.2 Caso de Estudo 2

No segundo caso de estudo, foi utilizado o objeto representado na Figura 6.12. O processo de aquisição de imagem foi realizado nas mesmas condições que o primeiro, resultando vinte e uma imagens para processar. Foi utilizado também o algoritmo *SURF*, mas ao contrário do que acontece no Caso de Estudo 1, em que foi utilizado a teste de rácio, neste foi utilizado o teste de simetria para filtragem dos emparelhamentos mal sucedidos. Além disto, foi também utilizada um grau menor de confiança no cálculo da matriz essencial, passando dos 0,99 para os 0,60.

Figura 6.12: Objeto de estudo Caso de Estudo 2

6.2.2.1 Deteção e Emparelhamento de Pontos de Interesse

Comparativamente ao que foi feito no primeiro caso de estudo, o processo inicia-se pela deteção e emparelhamento dos pontos de interesse em cada uma das vinte e uma imagens previamente adquiridas. A Figura 6.13 representa o resultado referente a esta primeira fase e é respetiva aos dois primeiros pares de imagens.

Enquanto no teste de rácio para cada ponto de interesse da imagem 1 são emparelhados dois pontos de interesse respetivos na imagem 2. No teste de simetria, para cada ponto de interesse na primeira imagem são também emparelhados dois pontos de interesse na segunda imagem, mas também é realizado um emparelhamento entre os pontos de interesse da segunda imagem para dois pontos de interesse respetivos na primeira imagem. No caso específico da imagem da Figura 6.13 estão representados 1020 ligações, uma vez que, foram emparelhados 610 pontos de interesse em cada uma das direcções.

Figura 6.13: Emparelhamento Inicial através do Brute Force Matcher.

Na Figura 6.14 estão representados a verde os pontos de interesse na primeira e na segunda imagem, bem como a sua correspondência. Por outro lado, a vermelho, estão representados os pontos de interesse removidos após a aplicação do teste de simetria. Resultando assim 337 pontos de interesse.

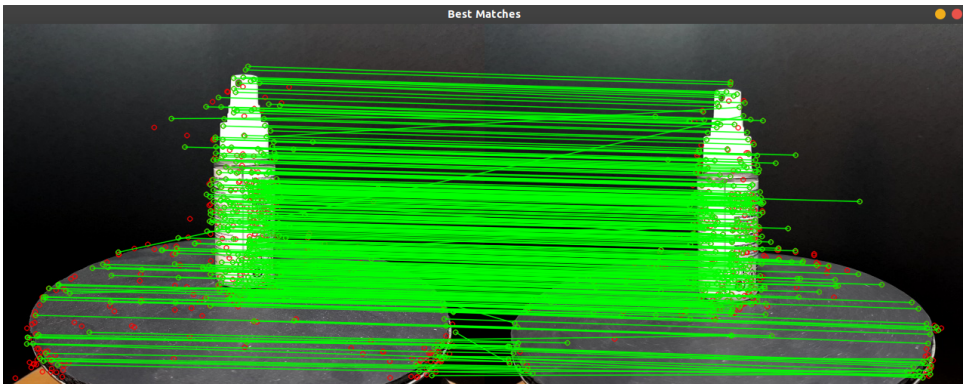


Figura 6.14: Emparelhamentos de pontos de interesse após aplicação do teste de simetria. A verde estão representados os *inliers* e a vermelho os *outliers*.

6.2.2.2 Remoção de Outliers

Finalizada a fase de deteção e emparelhamento de pontos de interesse, onde resultaram 337 pontos de interesse após a aplicação do teste de simetria, é necessário agora validar o conceito de geometria epipolar. Posto isto, é calculada a matriz essencial, onde resultam para o primeiro par de imagens 246 *inliers* e 91 *outliers*.

6.2.2.3 Triangulação e Geração de Nuvem de Pontos

Obtidos os *inliers* correspondentes aos 20 pares de imagens analisados, é possível agora iniciar o processo de triangulação. À semelhança do que se passou no Caso de Estudo 1, é necessária a triangulação inicial dos pares de pontos de interesse referentes ao primeiro par de imagens. O que corresponde obter as coordenadas tridimensionais dos 246 pares de pontos de interesse referentes ao primeiro par de imagens. Posto isto, é dado início ao processo de triangulação dos restantes *inliers* representativos dos restantes par de imagens. Numa primeira fase, é necessário calcular a matriz de rotação e de translação entre as imagens consecutivas, para isso, é necessário obter os pontos de interesse e respetivos descritores referentes aos pontos tridimensionais inseridos na nuvem de pontos referentes às iterações anteriores. Obtido isto, é realizado o emparelhamento com os pontos de interesse da imagem seguinte, onde estes são filtrados através do teste da simetria. Aqui sim, já é possível calcular a matriz de rotação e translação através da chamada da função *solvePnPRanac()*, da nuvem de pontos total obtida na iteração anterior, bem como as correspondências com os pontos de interesse na imagem seguinte. Calculadas as matrizes de projeção, é possível triangular os *inliers* seguintes e adicioná-los à nuvem de pontos final. Este processo é representado pelo gráfico de barra presente na Figura 6.15. Através da observação gráfico de barras representado pela cor verde é possível verificar a evolução do número de

pontos na nuvem de pontos final em cada iteração do processo de reconstrução tridimensional. A nuvem de pontos final é constituída por 2036 pontos tridimensionais e está representada na Figura 6.16. Pela observação da Figura 6.16 é possível validar a forma do objeto inicial apresentado para o Caso de Estudo 2, onde vemos uma nuvem de pontos com aspeto cilíndrico e a afunilar na parte superior, exatamente como o objeto de estudo utilizado. A nível das cores não é possível tirar conclusões significativas, uma vez que o objeto tem uma cor constante ao longo de toda a sua superfície.

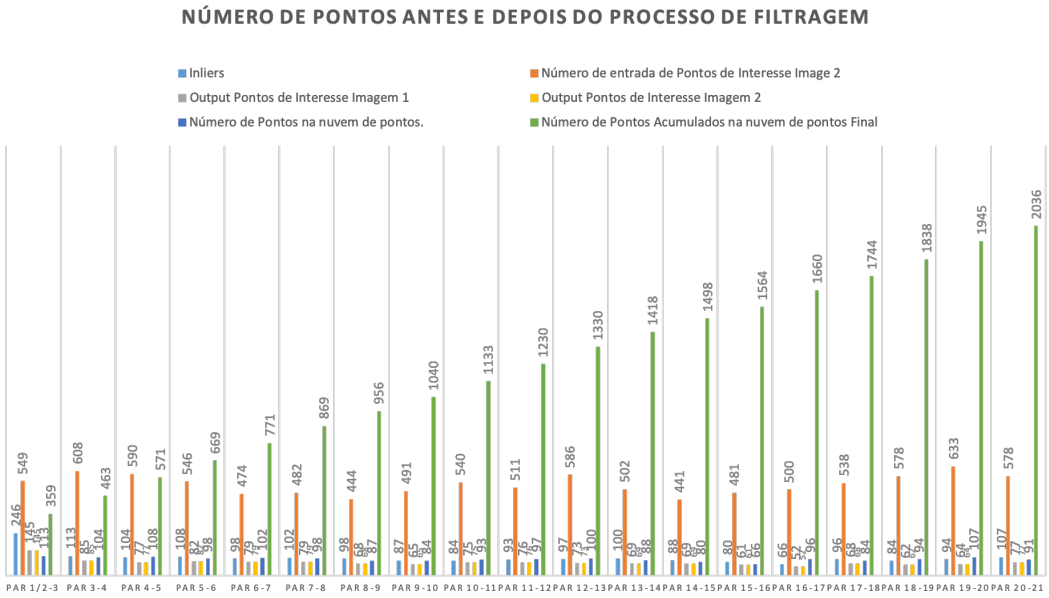


Figura 6.15: Números de pontos de interesse e números de pontos na nuvem de pontos final durante o processo de triangulação.

Figura 6.16: Nuvem de pontos obtida para o caso de estudo 2, apresentada de uma vista frontal.

A nível de tempos de processamento foram obtidos resultados bastante similares aos apresentados no Caso de Estudo 1

6.2.3 Outros Casos

Para demostrar a capacidade da aplicação no dispositivo móvel Android detetar os pontos de interesse de um determinado objeto, são apresentadas as duas Figuras seguintes, respetivamente a Figura 6.17 e a Figura 6.18.

Figura 6.17: Deteção de pontos de interesse através do algoritmo SURF na aplicação Android.

No primeiro caso foi utilizado o algoritmo SURF, enquanto que no segundo caso foi utilizado o algoritmo ORB. O algoritmo ORB à semelhança do que acontecia na aplicação gráfica Qt deteta um maior número de pontos de interesse inicialmente, não significando isto, que sejam bons pontos de interesse.

A nível de tempos de processamento não foi possível retirar qualquer valor real através da aplicação, mas é possível afirmar que o algoritmo ORB é muito mais leve a nível computacional que os algoritmos SURF e SIFT, também testados. É possível concluir isto através da usabilidade da aplicação Android em tempo real, que durante o processo de deteção de pontos de interesse, quando o uso dos algoritmos SURF e SIFT, é bastante lenta. É possível notar a demora na deteção dos pontos de interesse. Por outro lado, quando está a ser utilizado o algoritmo ORB a usabilidade melhora substancialmente, já não sendo notado qualquer atraso na deteção e no *framerate* de apresentação de imagens na aplicação.

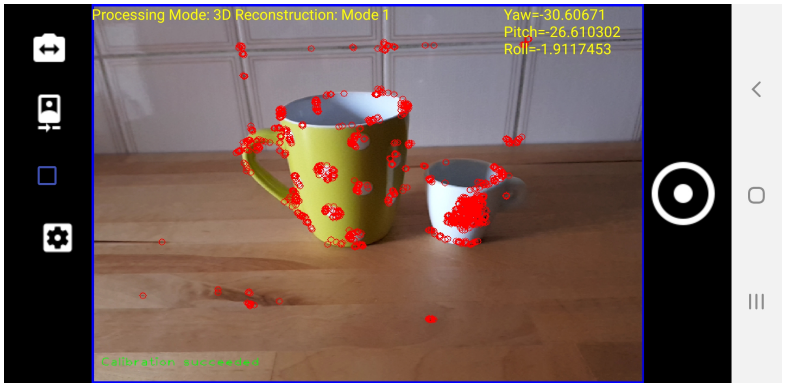


Figura 6.18: Deteção de pontos de interesse através do algoritmo ORB na aplicação Android.

6.2.4 Discução dos Resultados

Uma apreciação final deve ser feita relativamente aos resultados obtidos através dos casos de estudo apresentados. É possível afirmar que com o algoritmo SURF foram obtidas representações bastante similares ao objeto inicial, onde é possível identificar as suas formas, bem como as suas cores posicionadas no respetivo lugar. Com os outros algoritmos não foram obtidos resultados significativos, quer isto dizer, no algoritmo SIFT a pouca quantidade de pontos de interesse obtidos inicialmente, não permitia ter a triangulação do número de pontos suficientes para criar uma representação, onde fosse possível detetar claramente as formas de um determinado objeto de estudo. Já pelo contrário, com o algoritmo ORB, o número de pontos obtidos é bastante mais elevado que nos outros dois. Contudo, o resultado obtido também não foi o desejável, uma vez que, a qualidade de pontos obtidos apesar de ser elevada não permitia uma reconstrução capaz de observar um objeto com as formas bem definidas. Isto leva a crer que os pontos fortes e respetivos emparelhamentos não são detetados com a maior exatidão possível. A nível de deteção de pontos de interesse é possível afirmar tendo em conta as imagens obtidas, que o algoritmo, SURF, consegue detetar pontos de interesse correspondentes em imagens distintas com bastante exatidão, onde saltam à vista um ou outro ponto em que o emparelhamento não foi encontrado com exatidão. O SIFT pelos testes realizados é o que permite uma maior exatidão na correspondência entre pontos de interesse em imagens distintas, por sua vez, é o algoritmo que extrai o menor número de pontos de interesse, sendo que também é o mais lento. Pela análise aos tempos de processamentos, é possível concluir que isso deve-se à fase de deteção de pontos de interesse, uma vez que os tempos de descrição dos pontos de interesse são bastante similares em todos os algoritmos testados. Estas conclusões acabam por ser coincidentes com o levantamento inicial realizado e apresentado no capítulo da fundamentação teórica deste trabalho. Onde os autores do algoritmo SURF, devido ao uso do conceito de imagens integrais e filtros de caixa afirmam que o algoritmo era mais rápido que o seu antecessor, o SURF. Por sua vez, o algoritmo mais leve dos três é o ORB, que faz uso do detetor FAST e do descritor BRIEF. Aqui uma última conclusão pode ser tirada, dos três o que se obteve melhores resultados a nível de reconstrução tridimensional foi através do algoritmo SURF, por sua vez, para a aplicação num dispositivo móvel é um pouco pesado, sendo que aqui o mais indicado seria o algoritmo ORB, uma vez que a nível de processamento é bastante mais rápido que os outros dois, isso foi testado através da aplicação destes algoritmos na aplicação Android desenvolvida.

Capítulo 7

Conclusões e Trabalho Futuro

Neste capítulo serão apresentadas as conclusões referentes a este trabalho de dissertação, bem como apresentado sugestões para trabalho futuro.

7.1 Conclusões

A presente dissertação de mestrado permitiu implementar um sistema de reconstrução tridimensional baseado na técnica passiva, estrutura a partir do movimento, com o objetivo de ser aplicado a um dispositivo móvel equipado com o sistema operativo Android.

Inicialmente, realizou-se um levantamento ao estado da arte das técnicas e métodos de reconstrução tridimensional, bem como, os principais sistemas de reconstrução presentes no mercado, enquadrando, assim, o leitor, de forma geral, com o tema implícito neste trabalho. Várias técnicas e métodos foram apresentados, por exemplo, os métodos de contacto bastante precisos são frequentemente utilizados em tarefas de engenharia reversa e têm como principal desvantagem a necessidade de contacto físico com o objeto alvo, além de serem sistemas bastante caros. Por outro lado, as técnicas de não contacto ativas têm um custo-benefício bastante positivo, conseguindo, deste modo, produzir modelos tridimensionais bastante precisos. Por sua vez, exigem normalmente, sistemas complexos acoplados aos sistemas de visão, bem como em alguns casos sistemas eletrónicos complexos para demodulação de sinais. Por fim, foram também descritas algumas técnicas de não contacto passivas, no qual se inclui a técnica estrutura a partir do movimento, utilizada neste trabalho. Estas técnicas têm como principal vantagem a sua simplicidade, uma vez que, normalmente, só necessitam de um sistema ótico para aquisição de imagens, sendo focadas, posteriormente, em algoritmos de processamento de imagem. Têm algumas limitações, principalmente a sua dependência aos materiais e texturas presentes nas superfícies dos

objetos e cenas alvos.

Relativamente à componente prática, foram desenvolvidas duas aplicações distintas ao longo desta dissertação. A primeira foi desenvolvida através da *framework* Qt e foi idealizada para ser uma ferramenta de suporte à aplicação Android e, deste modo, permitir agilizar o processo de desenvolvimento dos algoritmos de reconstrução 3D, bem como facilitar a sua depuração ao longo de todo o processo de desenvolvimento. Contudo, finalizada a sua implementação obteve-se uma aplicação capaz de realizar todas as tarefas referentes ao método utilizado, como por exemplo, a seleção de métodos de filtragem, a alteração de parâmetros, assim como, visualização, interação, importação e exportação de nuvens de pontos e a possibilidade de carregar parâmetros de calibração obtidos através de outros equipamentos.

A segunda focou-se no desenvolvimento de uma aplicação Android, que foi idealizada inicialmente para permitir realizar todo o processo de reconstrução tridimensional. Nesta fase, podem ser indicadas as principais dificuldades encontradas durante a fase de implementação desta dissertação, sendo que estas estão relacionadas principalmente com o desenvolvimento para a aplicação Android. A estrutura de *software* relativa à parte de processamento de imagem foi partilhada entre a aplicação gráfica Qt e a aplicação Android e, que por sua vez, foi desenvolvida nativamente em C++, isto requereu que todas as bibliotecas utilizadas fossem compiladas para Android. Relativamente à biblioteca OpenCV base, não houve qualquer dificuldade na sua utilização para Android. As primeiras dificuldades surgiram na compilação dos algoritmos *SURF* e *SIFT* que se encontram implementados numa biblioteca extra do OpenCV, denominada OpenCV Contrib. A principal causa desta dificuldade, em primeira instância, foi a falta de experiência com a plataforma Android e, a segunda, a falta de documentação disponível para a compilação de bibliotecas externas para correr nativamente em C++ através da plataforma Android. Dado isto, foi perdido algum tempo até ter sido possível a compilação das bibliotecas *SURF* e *SIFT*. Outra dificuldade encontrada, e na mesma linha, foi a compilação da biblioteca PCL para a plataforma Android, esta ainda foi mais difícil uma vez que tem dependências de outras bibliotecas. Posto isto, relativamente aos objetivos iniciais propostos a aplicação Android cobre grande parte dos objetivos definidos. Além de permitir realizar o processo de calibração, com a respetiva exportação dos parâmetros associados, é também possível visualizar em tempo real a deteção dos pontos de interesse e escolher entre algoritmos a utilizar. Com isto, a aplicação permite a deteção e emparelhamento dos pontos de interesse em tempo real para um conjunto de imagens. Por outro lado, dos objetivos principais não foi possível concluir a fase final do processo de reconstrução tridimensional, uma vez que dependia da biblioteca PCL que só foi conseguida compilar numa fase adiantada do projeto. Consequentemente, também não é possível a visualização da nuvem de pontos na aplicação Android.

Por fim, relativamente aos resultados obtidos através da aplicação gráfica Qt, onde foram testados os algoritmos de deteção e descrição de pontos de interesse, dos quais o *SURF*, foi o que obteve melhores resultados a nível de modelo tridimensional final, através do qual foi possível visualizar um modelo tridimensional equivalente ao objeto alvo. A nível de tempos de processamento e considerando que o processo de filtragem e triangulação têm uma duração residual, a grande parte do tempo decaía sobre a tarefa de deteção e descrição dos pontos de interesse em cada uma das imagens. O algoritmo ORB é o que mais rapidamente consegue detetar e descrever os pontos de interesse e, dado isto, seria o melhor para ser utilizado na solução para dispositivos móveis, uma vez que requer uma menor capacidade de processamento.

Finalizando, relativamente aos objetivos inicialmente definidos é possível afirmar que grande parte foram concluídos, onde foi desenvolvido um módulo de reconstrução tridimensional, portável entre várias aplicações e que permite cobrir o processo de reconstrução implícito no método passivo estrutura a partir do movimento.

7.2 Trabalho Futuro

Concluído o presente trabalho de dissertação é agora importante sugerir algumas propostas para trabalho futuro.

Uma vez não ter sido possível cumprir na integra todos os objetivos inicialmente delineados no início de trabalho, a primeira proposta assenta na conclusão dos pontos que não foram completamente concluídos neste projeto de dissertação. Esses pontos focam-se essencialmente na aplicação Android, a proposta seria finalizar o processo de reconstrução tridimensional, focando na fase de triangulação que não foi completamente concluída. Por sua vez, outra proposta seria adicionar a possibilidade de visualizar em tempo real a nuvem de pontos, resultado final do processo de reconstrução, também isto na aplicação Android, bem como a fusão sensorial dos sensores do dispositivo móvel nos algoritmos de reconstrução tridimensional.

Uma segunda proposta para trabalho futuro, foca-se na aplicação de alguma técnica para refinamento da nuvem de pontos obtida, de forma a obter modelos tridimensionais mais precisos. O *bundle-adjustment* é uma técnica presente em quase todos os trabalhos relacionada com o método de reconstrução tridimensional utilizado nesta dissertação.

Uma terceira proposta foca-se na implementação dos algoritmos de processamento de imagem a correr a partir da GPU do dispositivo móvel. Como concluído, o algoritmo SURF que obteve os melhores

resultados relativamente à reconstrução tridimensional foi um pouco pesado quando testado na aplicação Android. Atualmente, já existem implementações destes algoritmos através de processamento paralelo entre os processadores e as placas de vídeo, isto permitiria aproveitar as especificações que os dispositivos móveis atuais já têm e, desta forma, acelerar os tempos de processamento.

Uma quarta proposta é focada na implementação de um algoritmo de calibração automático, que evitasse assim o uso de padrões de calibração externos.

Por fim, uma última proposta foca-se na adição da capacidade das aplicações desenvolvidas permitirem criar reconstruções densas através das nuvens de pontos inicialmente obtidas, isto quer dizer, criar realmente um modelo tridimensional, através da nuvem de pontos.

Bibliografia

[1] J. Wang, D. Gu, Z. Yu, C. Tan, and L. Zhou, “A framework for 3D model reconstruction in reverse engineering,” Comput. Ind. Eng., vol. 63, pp. 1189–1200, 2012.

[2] [1] W. Wang, “Applications of reverse engineering in manufacturing industry,” in Technical Paper - Society of Manufacturing Engineers, 2013, vol. TP13PUB47.

[3] C. A. Madrigal, J. W. Branch, A. Restrepo, and D. Mery, “A method for automatic surface inspection using a model-based 3D descriptor,” Sensors (Switzerland), vol. 17, no. 10, pp. 1–20, 2017.

[4] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J.Davison, P. Kohli, J. Shotton, S. Hodges, and A. Fitzgibbon. KinectFusion:Real-Time Dense Surface Mapping and Tracking. In ISMAR, 2011.

[5] S. Izadi et al., “KinectFusion: Real-time 3D reconstruction and interaction using a moving depth camera,” in UIST’11 - Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology, 2011, no. 11, pp. 559–568.

[6] C. Poullis and S. You, ”3D Reconstruction of Urban Areas,”2011 International Conference on 3D Imaging, Modeling, Processing, Visualization and Transmission, Hangzhou, 2011, pp. 33-40. doi: 10.1109/3DIMPVT.2011.14

[7] F. Bruno, S. Bruno, G. De Sensi, M. L. Luchi, S. Mancuso, and M. Muzzupappa, “From 3D reconstruction to virtual reality: A complete methodology for digital archaeological exhibition,” J. Cult. Herit., vol. 11, no. 1, pp. 42–49, 2010.

[8] J. A. Parker, “Image Reconstruction in Radiology,” Image Reconstr. Radiol., pp. 3–6, 2018.

[9] T. Fazakas and R. T. Fekete, ”3D reconstruction system for autonomous robot navigation,”2010 11th International Symposium on Computational Intelligence and Informatics (CINTI), Budapest, 2010, pp. 267-270. doi: 10.1109/CINTI.2010.5672236

[10] T. B. Moeslund and E. Granum, “A survey of computer vision-based human motion capture,” Comput. Vis. Image Underst., vol. 81, no. 3, pp. 231–268, 2001.

[11] T. Várady, R. R. Martin, and J. Cox, “Reverse engineering of geometric models - An introduction,” CAD Comput. Aided Des., vol. 29, no. 4, pp. 255–268, Apr. 1997.

[12] Rocchini, C., Cignoni, P., Montani, C., Pingi, P., & Scopigno, R. (2001). A low cost 3D scanner based on structured light. Computer Graphics Forum, 20(3), 299–308. https://doi.org/10.1111/1467-8659.00522

[13] D. Page, A. Koschan, S. Voisin, N. Ali, and M. Abidi, “3D CAD model generation of mechanical parts using coded-pattern projection and laser triangulation systems,” Assem. Autom., vol. 25, no. 3, pp. 230–238, 2005.

[14] D. Giri, M. Jouaneh, and B. Stucker, “Error sources in a 3-D reverse engineering process,” Precis. Eng., vol. 28, no. 3, pp. 242–251, 2004.

[15] F. Bernardini and H. Rushmeier, “The 3D model acquisition pipeline,” Comput. Graph. Forum, vol. 21, no. 2, pp. 149–172, 2002.

[16] P. Sieka ski et al., “On-line laser triangulation scanner for wood logs surface geometry measurement,” Sensors (Switzerland), vol. 19, no. 5, 2019.

[17] Blais, ”F. Review of 20 years of range sensor development,”J. Electron, Imaging 2004, 13, 231.

[18] J. Geng, “Structured-light 3D surface imaging: a tutorial,” Adv. Opt. Photonics, vol. 3, no. 2, p. 128, 2011.

[19] M. V. Theo Moons, Luc Van Gool, ”3D Reconstruction from Multiple Images Part 1 : Principles 3D Reconstruction from Multiple Images Part 1 : Principles,”2010

[20] S. Foix, G. Alenyà, and C. Torras, “Lock-in time-of-flight (ToF) cameras: A survey,” IEEE Sens. J., vol. 11, no. 9, pp. 1917–1926, 2011.

[21] R. A. Hamzah and H. Ibrahim, “Literature survey on stereo vision disparity map algorithms,” J. Sensors, vol. 2016, 2016.

[22] B. G. Baumgart, “Geometric Modeling For Computer Vision,” 1974.

[23] A. Laurentini, “The Visual Hull Concept for Silhouette-Based Image Understanding,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 16, no. 2. pp. 150–162, 1994.

[24] B. Petit et al., “Multicamera real-time 3D modeling for telepresence and remote collaboration,” Int. J. Digit. Multimed. Broadcast., vol. 2010, no. August, 2010.

[25] P. Favaro, “Shape from Focus / Defocus,” Computer (Long. Beach. Calif)., no. 1, 1989.

[26] ”LibreTexts.” [Online]. Available: `https://phys.libretexts.org/Bookshelves/University_Physics/Book%3A_University_Physics_(OpenStax)/Map%3A_University_Physics_III_-_Optics_and_Modern_Physics_(OpenStax)/2%3A_Geometric_Optics_and_Image_Formation/2.4%3A_Thin_Lenses` [Accessed: 15-Fev-2019]

[27] S. Pertuz, D. Puig, and M. A. Garcia, “Analysis of focus measure operators for shape-from-focus,” Pattern Recognit., vol. 46, no. 5, pp. 1415–1432, 2013.

[28] ”Microsoft Kinect” [Online]. Available: `https://developer.microsoft.com/pt-pt/windows/kinect` [Accessed: 24-Fev-2019]

[29] S. Paul, S. Basu, and M. Nasipuri, “Microsoft Kinect in Gesture Recognition : A Short Review,” vol. 8, no. 5, pp. 2071–2076, 2015.

[30] ”Asus Xtion Pro.” [Online]. Available: `https://www.asus.com/3D-Sensor/Xtion/` [Accessed: 25-Fev-2019]

[31] ”Intel Real Sense.” [Online]. Available: `https://www.intel.com/content/www/us/en/architecture-and-technology/realsense-overview.html` [Accessed: 25-Fev-2019]

[32] ”Gocator.” [Online]. Available: `https://lmi3d.com/products/gocator-3D-smart-sensors` [Accessed: 25-Mar-2019]

[33] ”StructureApps” [Online]. Available: `https://structure.io/structure-sensor/apps` [Accessed: 10-Mar-2019]

[34] ”Structure SDK.” [Online]. Available: `https://structure.io/developers` [Accessed: 10-Mar-2019]

[35] D. Keralia, K. K. Vyas, and K. Deulkar, "Google Project Tango – A Convenient 3D Modeling Device," Int. J. Curr. Eng. Technol., vol. 4, no. 5, pp. 3139–3142, 2014.

[36] A. Heyden M. Pollefeys. "Multiple view geometry," 2005.

[37] T. Cristina and D. S. Azevedo, "Reconstrução e Caracterização de Estruturas Anatómicas Exteriores usando Visão Activa," 2002.

[38] A. Hartley, Richard, Zisserman, "Multiple View Geometry in Computer Vision: Richard Hartley, Andrew Zisserman: 9780521540513: Amazon.com: Books," Cambridge, 2004. [Online]. Available: `https://books.google.pt/books?id=si3R3Pfa98QC&printsec=frontcover&hl=pt-PT&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false` [Accessed: 28-Mai-2018]

[39] L. E. Filho, "Extração Semi-Automática de Feições Lineares e a Calibração dos Parâmetros Extração Semi-Automática de Feições Lineares e a Calibração dos Parâmetros Intrínsecos de Câmeras," no. June, 2016.

[40] Z. Zhang, "A flexible new technique for camera calibration," IEEE Trans. Pattern Anal. Mach. Intell., vol. 22, no. 11, pp. 1330–1334, 2000.

[41] F. Espuny, "A New Linear Method for Camera Self-Calibration with Planar Motion *," J Math Imaging Vis, vol. 27, pp. 81–88, 2007.

[42] B. Buxton and R. Cipolla, "Preface," Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 1065, no. January, p. vi, 1996.

[43] O. Faugeras, L. Quan, and P. Sturm, "Self-calibration of a 1D projective camera and its application to the self-calibration of a 2D projective camera," in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 1998, vol. 1406, pp. 36–52.

[44] N. Watanabe, S. Nakamura, B. Liu, and N. Wang, "Utilization of Structure from Motion for processing CORONA satellite images: Application to mapping and interpretation of archaeological features in Liangzhu Culture, China," Archaeol. Res. Asia, vol. 11, pp. 38–50, Sep. 2017.

[45] R. Mlambo, I. H. Woodhouse, F. Gerard, and K. Anderson, "Structure from motion (SfM) photogrammetry with drone data: A low cost method for monitoring greenhouse gas emissions from forests in developing countries," Forests, vol. 8, no. 3, 2017.

[46] G. H. Lee, F. Faundorfer, and M. Pollefeys, "Motion estimation for self-driving cars with a generalized camera," Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit., pp. 2746–2753, 2013.

[47] V. A. Prisacariu, O. Kähler, D. W. Murray, and I. D. Reid, "Real-Time 3D Tracking and Reconstruction on Mobile Phones," IEEE Trans. Vis. Comput. Graph., vol. 21, no. 5, pp. 557–570, 2015.

[48] P. Tanskanen, K. Kolev, L. Meier, F. Camposeco, O. Saurer, and M. Pollefeys, "Live metric 3D reconstruction on mobile phones," 2013.

[49] D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," Int. J. Comput. Vis., vol. 60, no. 2, pp. 91–110, 2004.

[50] T. Lindeberg, "Scale-Space Theory in Computer Vision," April. 1994.

[51] K. Mikolajczyk, "Detection of local features invariant to affine transformations Application to matching and recognition," Most, p. 171, 2002.

[52] H. Bay, T. Tuytelaars, and L. Van Gool, "SURF: Speeded up robust features," in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2006, vol. 3951 LNCS, pp. 404–417.

[53] C. Evans, "Notes on the OpenSURF Library," Univ. Bristol Tech Rep CSTR09001 January, no. 1, p. 25, 2009.

[54] "ICP" [Online]. Available: `http://ais.informatik.uni-freiburg.de/teaching/ss11/robotics/slides/17-icp.pdf` [Accessed: 11-Mar-2018]

[55] Z. Zhang, "Iterative point matching for registration of free-form curves and surfaces," Int. J. Comput. Vis., vol. 13, no. 2, pp. 119–152, 1994.

[56] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: An efficient alternative to SIFT or SURF," Proc. IEEE Int. Conf. Comput. Vis., pp. 2564–2571, 2011.

[57] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," pp. 1–14, 2010.

[58] M. Calonder et al., "Brief: Binary robust independent elementary features," in European Conference on Computer Vision, Heraklion, ECCV, 2010, pp. 778-792.

[59] M. a Fischler and R. C. Bolles, “Random Sample Paradigm for Model Consensus: A Apphcatlons to Image Fitting with Analysis and Automated Cartography,” Commun. ACM, vol. 24, no. 6, pp. 381–395, 1981.

[60] G. Konstantinos, “Overview of the RANSAC Algorithm,” , 2010.

[61] M. C. dos Santos, “Revisão de conceitos em projeção, homografia, calibração de câmera, geometria epipolar, mapas de profundidade e varredura de planos,” , 2012.

[62] ”OpenCV - Essential Mat” [Online]. Available: https://docs.opencv.org/3.4/d9/d0c/group__calib3d.html#ga0c86f6478f36d5be6e450751bbf4fec0 [Accessed: 21-Jan-2019]

[63] ”Announcing the Android 1.0 SDk, release 1 | Android Developers blog, 2008” [Online]. Available: https://android-developers.googleblog.com/2008/09/announcing-android-10-sdk-release-1.html [Accessed: 11-Jan-2018]

[64] ”AndroidTV” [Online]. Available: https://www.android.com/tv/ [Accessed: 11-Jan-2018]

[65] ”WearOS” [Online]. Available: https://wearos.google.com/#hands-free-help [Accessed: 11-Jan-2018]

[66] ”AndroidAuto” [Online]. Available: https://www.android.com/auto/ [Accessed: 11-Jan-2018]

[67] ”AndroidThings” [Online]. Available: https://developer.android.com/things [Accessed: 11-Jan-2018]

[68] ”Arquitetura Plataforma Android” [Online]. Available: https://developer.android.com/guide/platform?hl=pt-BR [Accessed: 11-Jan-2018]

[69] Paul Viola and Michael Jones, ”Rapid object detection using a boosted cascade of simple features,”, 2001.

[70] ”Qt Creator.” [Online]. Available: https://www.qt.io/why-qt?utm_campaign=Navigation%202019&utm_source=Nav%202019 [Accessed: 11-Mar-2019]

[71] ”OpenCV.” [Online]. Available: https://opencv.org/about/ [Accessed: 11-Mar-2019]

[72] "Point Cloud Library." [Online]. Available: `http://pointclouds.org/about/` [Accessed: 05-Mar-2019]

[73] "OpenGL." [Online]. Available: `https://www.opengl.org/about/` [Accessed: 06-Mar-2019]

[74] "Arduino Uno." [Online]. Available: `https://store.arduino.cc/arduino-uno-rev3` [Accessed: 10-Jul-2019]

[75] "Arduino IDE." [Online]. Available: `https://www.arduino.cc/en/main/software` [Accessed: 10-Jul-2019]

[76] "HC-05." [Online]. Available: `https://www.ptrobotics.com/338-bluetooth` [Accessed: 11-Jul-2019]

[77] "L293F." [Online]. Available: `http://www.ti.com/lit/ds/symlink/l293.pdf` [Accessed: 11-Jul-2019]