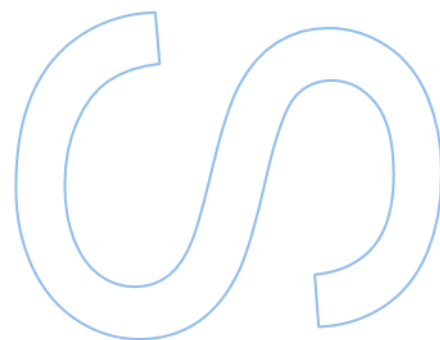
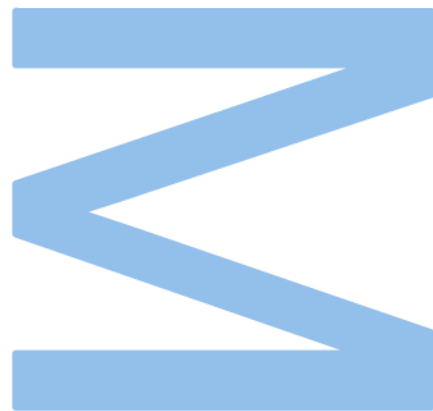


Seguimento de objetos para melhorar a solução de navegação de um sistema de mapeamento móvel com vídeo

João Carvalho

Mestrado em Engenharia Geoespacial
Departamento de Geociências, Ambiente e Ordenamento do Território
Faculdade de Ciências da Universidade do Porto
2024



Seguimento de objetos para melhorar a solução de navegação de um sistema de mapeamento móvel com vídeo

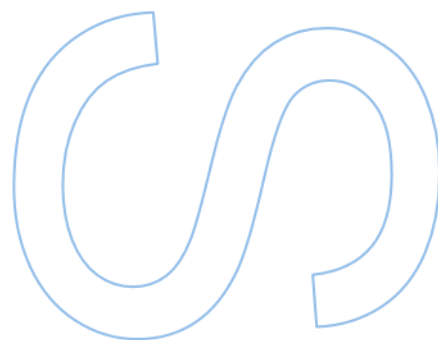
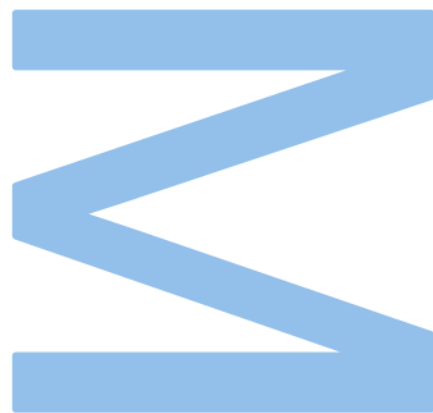
João Carvalho

Dissertação realizada no âmbito do Mestrado em Engenharia
Geoespacial

Departamento de Geociências, Ambiente e Ordenamento do Território
2024

Orientador

José Alberto Gonçalves, Professor Auxiliar, Faculdade de Ciências
da Universidade do Porto



Agradecimentos

Ao concluir esta jornada acadêmica, gostaria de agradecer a todos aqueles que contribuíram direta ou indiretamente para a realização desta tese de mestrado.

Em primeiro lugar, queria agradecer ao meu orientador, Prof. José Alberto Gonçalves, pela sua orientação incansável, apoio constante e conselhos valiosos ao longo de todo este processo. A sua experiência e dedicação foram fundamentais para o desenvolvimento deste trabalho.

Não podia também deixar de agradecer à minha família, principalmente aos meus pais e irmã que sempre apoiaram, desejaram o melhor para mim e acima de tudo, sempre acreditaram em mim mesmo nos momentos mais difíceis. À minha namorada, que sempre me apoiou em todas as etapas e me deu a coragem e força de vontade necessária para concluir este capítulo da minha vida com sucesso.

A todos que de alguma forma contribuíram para esta conquista, o meu sincero obrigado.

Resumo

Com a evolução da visão computacional em áreas como a navegação autónoma de diferentes tipos de agentes como automóveis e robôs, torna-se cada vez mais importante desenvolver sistemas capazes de estimar uma posição precisa, tendo em conta o custo e a complexidade do sistema. A utilização de recetores GNSS (Global Navigation Satellite System) por si só, não possibilita obter em todos os cenários uma posição de qualidade suficiente que a navegação autónoma necessita, pelo que se torna indispensável criar sistemas que utilizem diferentes tipos de sensores.

Esta dissertação de mestrado, pretende estudar uma abordagem que permite aumentar a precisão de posições obtidas através de um recetor GNSS em zonas em que a qualidade do sinal é baixa, nomeadamente em grandes áreas urbanas e regiões com uma elevada densidade de arvoredo. O principal objetivo, é desenvolver um método que através de uma câmara consiga reconstruir o trajeto percorrido, de forma a permitir correções de dados de posicionamento em locais de baixa precisão.

A metodologia proposta, passa pela utilização de diversos algoritmos da biblioteca OpenCV (Open Source Computer Vision) em C++, de forma a implementar um sistema de odometria visual monocular capaz de detetar pontos chave nas imagens, seguir esses mesmos pontos ao longo do percurso, calcular uma estimativa para a pose da câmara em cada ponto, e por fim, a reconstrução do trajeto percorrido. Esta trajetória é depois comparada com a referência no terreno para avaliar a sua precisão.

Numa primeira fase serão utilizados dados de um dataset chamado KITTI, desenvolvido pelo Instituto de Tecnologia de Karlsruhe e pelo Instituto Tecnológico da Toyota em Chicago. Foi criado com o objetivo de promover a pesquisa e desenvolvimento deste tipo de tecnologias em navegação autónoma. A estes dados, está também associado uma referência no terreno que permite comparar a rota obtida com a real. Na segunda fase o sistema será testado com a utilização de imagens obtidas no terreno, em Portugal, através de uma câmara GoPro fixada num carro em movimento.

A abordagem proposta apresenta uma solução com componentes de baixo custo e uma complexidade computacionalmente reduzida, que permite melhorar a navegação em regiões onde os recetores GNSS não nos fornecem uma precisão suficiente. Este trabalho tem implicações para aplicações como navegação autónoma de automóveis ou robôs, mapeamento urbano e sistemas de localização em ambientes complexos.

Palavras-chave: [C++, GNSS, OpenCV, navegação autónoma, odometria visual, visão computacional]

Abstract

With the evolution of computer vision in areas such as autonomous navigation of different types of agents like cars and robots, it is becoming increasingly important to develop systems capable of estimating an accurate position, taking into account the cost and complexity of the system. The use of GNSS (Global Navigation Satellite System) receivers alone, does not make it possible to obtain a position of sufficient quality for autonomous navigation in all scenarios, so it is essential to create systems that use different types of sensors.

This master's thesis aims to study an approach that can increase the accuracy of positions obtained from a GNSS receiver in areas where the signal quality is poor, particularly in large urban areas and regions with a high density of trees. The main objective is to develop a method that, using a camera, can reconstruct the path travelled in order to allow corrections to positioning data in low-precision locations.

The proposed methodology involves using various algorithms from a library called OpenCV in c++, in order to implement a monocular visual odometry system capable of detecting key points in the images, following those same points along the route, calculating an estimate for the camera's pose at each point and finally reconstructing the path travelled. This trajectory is then compared with a ground truth to assess its accuracy.

Initially, data from a dataset called KITTI, developed by the Karlsruhe Institute of Technology and the Toyota Institute of Technology in Chicago, will be used to promote research and development into this type of autonomous navigation technology. This data is also associated with a ground truth that allows the route obtained to be compared with the real one. In the second phase, the system will be tested using images obtained on the ground using a GoPro camera attached to a moving car.

The proposed approach presents a solution with low-cost components and reduced computational complexity that improves navigation in regions where GNSS receivers do not provide sufficient accuracy. This work has implications for autonomous car or robot navigation applications, urban mapping and localisation systems in complex environments.

Keywords: [C++, GNSS, OpenCV, autonomous navigation, computer vision, visual odometry]

Índice

Lista de Gráficos.....	vii
Lista de Figuras	viii
Lista de Abreviaturas	ix
1. Introdução.....	1
1.1. Contextualização.....	1
1.2. Motivação.....	2
1.3. Objetivos	4
1.4. Organização do Documento	4
2. Revisão Bibliográfica.....	5
2.1 Posicionamento GNSS.....	5
2.2 Visão Computacional.....	8
2.3 Odometria Visual.....	9
2.3.1 Técnicas de Odometria Visual Monocular	9
2.4 OpenCV	10
3. Metodologia.....	11
3.1 Visão Geral do Sistema.....	11
3.2 Implementação do Sistema de Odometria Visual Monocular	12
3.2.1 Importação de dados e pré-processamento	12
3.2.2 Detecção e seguimento de caraterísticas.....	13
3.2.3 Estimar movimento.....	16
4. Resultados com o dataset KITTI.....	17
4.1 Conjunto de dados KITTI.....	17
4.2 Resultados obtidos com o dataset KITTI	18
5. Resultados obtidos com GoPro	21
6. Interface gráfica.....	24
7 Conclusão.....	26
Referências Bibliográficas	27

Anexo 1 – Processamento de frames	29
Anexo 2 – Interface gráfica	36
Anexo 3 – Ciclo principal da aplicação.....	39

Lista de Gráficos

Gráfico 1 Trajeto estimado da sequência 1 utilizando o algoritmo FAST.....	18
Gráfico 2 Trajeto estimado da sequência 1 utilizando o algoritmo ORB.	19
Gráfico 3 Trajeto estimado da sequência 2 utilizando o algoritmo FAST.....	19
Gráfico 4 Trajeto estimado da sequência 2 utilizando o algoritmo ORB.	20

Lista de Figuras

Figura 1 Arquitetura do sistema de odometria visual.....	11
Figura 2 Parte de uma imagem na qual foi aplicado o algoritmo ORB.	14
Figura 3 Exemplo de vetor de fluxo ótico entre frames.	14
Figura 4 Vetor de fluxo ótico entre duas imagens consecutivas.	15
Figura 5 Configuração da carrinha utilizada para obter o dataset KITTI (https://www.cvlibs.net/datasets/kitti/setup.php).	17
Figura 6 Trajeto efetuado com a GoPro na região de Matosinhos	21
Figura 7 Trajeto estimado da sequência gravado com a GoPro em Matosinhos.....	22
Figura 8 Interface gráfica	24

Lista de Abreviaturas

BRIEF	BINARY ROBUST INDEPENDENT ELEMENTARY FEATURES
ESA	EUROPEAN SPACE AGENCY
FAST	FEATURES FROM ACCELERATED SEGMENT TEST
GNSS	GLOBAL NAVIGATION SATELLITE SYSTEM
GPS	GLOBAL POSITIONING SYSTEM
IMU	INERTIAL MEASUREMENT UNIT
KITTI	KARLSRUHE INSTITUTE OF TECHNOLOGY AND TOYOTA TECHNOLOGICAL INSTITUTE
LIDAR	LIGHT DETECTION AND RANGING
ORB	ORIENTED FAST AND ROTATED BRIEF
OPENCV	OPEN SOURCE COMPUTER VISION LIBRARY
PNG	PORTABLE NETWORK GRAPHIC
RANSAC	RANDOM SAMPLE CONSENSUS
SIFT	SCALE-INVARIANT FEATURE TRANSFORM
SURF	SPEEDED UP ROBUST FEATURES

1. Introdução

Este capítulo de introdução, apresenta as bases estabelecidas para a investigação sobre a utilização de técnicas de odometria visual monocular, para a construção de um sistema capaz de melhorar a precisão do posicionamento e navegação em ambientes onde o sinal dos recetores GNSS é fraco ou instável. De seguida, são apresentados 4 subcapítulos que permitem contextualizar, apresentar as motivações, os objetivos, e por fim, a estrutura deste documento.

1.1. Contextualização

No cenário atual de navegação e posicionamento, a precisão e confiabilidade dos sistemas globais de navegação por satélite são fundamentais para uma grande variedade de aplicações modernas especialmente na última década, com o aumento de investigação e desenvolvimento de sistemas de navegação autónoma, mapeamento urbano e sistemas de localização em ambientes complexos.

Os GNSS (Global Navigation Satellite System) revolucionaram a forma de como nos orientamos e navegamos pelo mundo, permitindo obter em condições ideais uma boa aproximação da posição em que se encontra o observador. No entanto, existem locais em que estes sistemas enfrentam limitações impossibilitando obter uma localização precisa e confiável, nomeadamente, em áreas urbanas densas e regiões com vegetação alta e densa. Um dos problemas deve-se ao efeito de multicaminho, em que o sinal proveniente dos satélites reflete em edifícios altos e estruturas naturais, provocando um aumento do tempo de receção do sinal e consequentemente um agravamento do erro do posicionamento. Esta irregularidade não permite a utilização de apenas sistemas GNSS para aplicações como navegação autónoma, que necessita especialmente de posicionamento de grande precisão para que o veículo se consiga movimentar em trajetos mais complexos, mapeamento urbano onde muitas vezes é necessário mapear pontos localizados em zonas que os sinais GNSS são fracos, operações de resgate e emergência que podem ocorrer em locais de grande complexidade onde o posicionamento preciso ajuda a salvar vidas, entre outros. Pelo que se torna indispensável criar sistemas complementares que sejam capazes de corrigir o posicionamento nestes casos.

Neste contexto a odometria visual monocular surge como uma tecnologia promissora, capaz de minimizar estes erros com uma relação custo/benefício bastante boa em relação a outras técnicas. Esta técnica utiliza sequências de imagens capturadas por

uma única câmara para reconstruir o trajeto percorrido, através da análise da diferença entre os frames é possível estimar com uma grande precisão a trajetória do agente, especialmente em curtas distâncias. A odometria visual oferece várias vantagens, entre elas o facto de não depender de sinais externos ao contrário dos sistemas GNSS, o baixo custo e simplicidade uma vez que apenas depende de uma câmara comum, poder ser utilizada em tempo real e também a precisão em curtas distâncias que pode ser complementada com os dados de satélite em zonas mais complexas.

Posto isto, esta dissertação de mestrado pretende desenvolver um sistema capaz de auxiliar o posicionamento por satélite através de odometria visual monocular, implementada utilizando a biblioteca OpenCV e a linguagem de programação C++.

1.2. Motivação

A motivação para o tema de pesquisa desta dissertação de mestrado surge de uma necessidade, presente em diversas áreas, de uma solução de navegação e posicionamento mais robusta e precisa. Entre elas:

Urbanização crescente

O rápido crescimento de áreas urbanas um pouco por todo o mundo dificulta cada vez mais a precisão dos sistemas de navegação comuns que utilizam apenas sistemas GNSS. Segundo dados recentes das Nações Unidas, é esperado que 68% da população mundial viva em áreas urbanas até 2050 (UN, 2024). Este crescimento, traz consigo dificuldades acrescidas, como o aumento da densidade e altura dos edifícios, que pioram o efeito de multicaminho, uma maior complexidade da rede rodoviária, uma maior concentração de população e uma maior procura por este tipo de sistemas no meio urbano.

Avanços na tecnologia de condução autónoma

Com a crescente pesquisa e desenvolvimento de sistemas autónomos de condução, aumenta a necessidade de soluções de posicionamento mais robustas capazes de assegurar uma localização precisa e confiável em qual quer que seja o ambiente. Esta fusão de múltiplos sensores como as câmaras, permite criar sistemas mais robustos e redundantes.

Evolução de aplicações baseadas em localização

Nos tempos que correm, cada um de nós tem um dispositivo no bolso capaz de receber sinais GNSS com uma boa precisão, no entanto, estes sofrem das mesmas dificuldades

referidas anteriormente neste tipo de sistemas. Mais uma vez, a odometria visual pode ser um aliado a estas técnicas tradicionais, possibilitando por exemplo, que aplicações de realidade aumentada obtenham a precisão necessária em todos os locais, nomeadamente meios urbanos densos.

Operações de emergência e segurança pública

Em cenários de emergência e catástrofes, é de elevada importância assegurar um sistema de posicionamento e mapeamento, em terreno que por vezes pode ser bastante complexo para os sistemas tradicionais. Uma melhoria na precisão, pode ter um impacto direto na eficácia dessas operações e, potencialmente, salvar vidas.

Aplicações em ambientes naturais

Para além dos ambientes urbanos, existe cada vez mais uma necessidade de navegação e posicionamento preciso em áreas florestais com elevada densidade de árvores. E tal como anteriormente, são zonas onde há uma grande degradação da qualidade dos sinais, e por sua vez, da precisão da localização. Esta é também uma área onde a odometria visual pode ajudar significativamente a estimar a posição com uma maior robustez, permitindo um monitoramento ambiental em florestas densas, operações agrícolas em regiões de vegetação densa, atividades de turismo e também ao combate a incêndios.

Soluções de custo reduzido

A odometria visual, além de ajudar a aumentar a precisão e robustez de sistemas de posicionamento por satélite, é uma técnica viável do ponto de vista económico, dado que são utilizadas câmaras comuns para a recolha de informação. Evitando o uso de sensores adicionais de custo elevado, como é o caso do LiDAR (Light Detection and Ranging) que utiliza feixes laser para calcular distâncias entre o emissor até um objeto. Deste modo, qualquer pessoa pode ter acesso a este tipo de tecnologia e aplicar nas mais diversas áreas.

Para concluir, esta pesquisa é motivada pela necessidade de combater as limitações dos sistemas mais tradicionais de posicionamento por satélite, especialmente em ambiente de grande densidade, tanto urbano como florestal. A odometria visual monocular apresenta-se como uma solução viável para combater essas necessidades e permitir avanços em tecnologias como a condução autónoma e realidade aumentada.

1.3. Objetivos

Esta dissertação de mestrado tem como objetivo geral o desenvolvimento de um sistema de navegação que utilize odometria visual monocular, de forma a reconstruir o trajeto percorrido por um agente. Este sistema pode ser posteriormente combinado com um GNSS, de forma a permitir a correção de dados em que o sinal esteja condicionado pelo ambiente em que se encontra. Para tal, definiram-se os seguintes objetivos específicos:

- Desenvolvimento de um sistema que de odometria visual monocular em C++ utilizando a biblioteca OpenCV;
- Desenvolvimento de uma interface gráfica em C++ de forma a facilitar a visualização e controlo do processamento;
- Teste do sistema com o dataset KITTI;
- Teste do sistema com dados obtidos no terreno com recurso a uma GoPro;
- Avaliação do desempenho do sistema.

1.4. Organização do Documento

Este documento está principalmente dividido em 6 capítulos, um primeiro onde é apresentada a contextualização do projeto, a motivação que levou à elaboração do mesmo e ao desenvolvimento de um sistema de odometria visual monocular e os objetivos propostos para a dissertação.

No segundo capítulo, é elaborada uma revisão bibliográfica onde são introduzidos os principais conceitos utilizados no desenvolvimento do sistema e serve como base para os seguintes capítulos.

No terceiro, é descrita a metodologia utilizada para desenvolver o sistema em questão.

No quarto e quinto capítulo é efetuada uma avaliação sobre a performance do sistema utilizando em primeiro um dataset chamado KITTI, e em segundo utilizando dados obtidos no terreno com recurso a uma câmara GoPro.

O sexto capítulo contém uma descrição de como foi desenvolvida a interface gráfica para análise de resultados, como está estruturada e o modo de funcionamento.

Por fim, o sétimo capítulo apresenta as conclusões gerais retiradas do desenvolvimento deste sistema, os pontos positivos, os pontos negativos e o que pode ser melhorado futuramente.

2. Revisão Bibliográfica

2.1 Posicionamento GNSS

Os sistemas globais de navegação por satélite, são utilizados diariamente por milhões de dispositivos nas mais variadas áreas, como a aviação, marinha, engenharia civil, agricultura, pesca, busca e salvamento, setor da energia, entre outros. Desde 1995, através do sistema GPS (Global Positioning System), passou a ser possível obter informação sobre o tempo, posição e velocidade de um observador, especialmente a partir do ano 2000, em que o sistema que limitava a performance do posicionamento para a população em geral foi desligado, permitindo assim a qualquer cidadão ter acesso a navegação de alta precisão. Mais tarde, surgiram outros sistemas globais de navegação por satélite, como por exemplo, GALILEO desenvolvido pela União Europeia e pela Agência Espacial Europeia (ESA) e GLONASS operado pelo Ministério da Defesa da Rússia.(Tran *et al.*) Existe uma interoperabilidade e compatibilidade entre os diferentes sistemas, para que o utilizador receba a melhor qualidade de dados no local em que se encontra, de forma a promover a segurança e a conveniência do seu uso (Kornhauser, 2006). Estão divididos principalmente em três segmentos: segmento espacial da qual fazem parte as constelações de satélites, segmento de controlo que é responsável pela gestão de todos os satélites e os seus sistemas, e por fim o segmento do utilizador que engloba todos os tipos de recetores dos utilizadores (Kornhauser, 2006). A ideia fundamental do posicionamento por satélite baseia-se na trilateração, um método para determinar a posição de um observador através da medição da distância, a um mínimo de três pontos de coordenadas conhecidas, no entanto para este tipo de posicionamento, são necessárias quatro pseudodistâncias a quatro satélites. A pseudodistância refere-se à diferença entre o tempo de transmissão do sinal a partir do satélite e o tempo da receção do sinal registado pelo recetor multiplicado pela velocidade da luz. Ao contrário da medição de uma distância normal, nas pseudodistâncias são incluídos os erros de relógio (Blewitt, 1997). Codificadas nestes sinais, vêm as mensagens de navegação que são posteriormente lidas pelos recetores, a partir destas é possível obter os parâmetros orbitais que permitem calcular as coordenadas do satélite, possibilitando a utilização do método da trilateração (Blewitt, 1997).

2.1.1. Erros associados ao posicionamento GNSS

Existem diversos tipos de erros que afetam a qualidade da posição calculada pelo recetor, sejam erros provocados pela propagação dos sinais pela atmosfera, erros do próprio satélite ou erros relacionados com o recetor (Kornhauser, 2006).

Erros do relógio de satélite

A bordo dos satélites encontram-se relógios bastante precisos, no entanto, ao longo do tempo estes acabam por sofrer ligeiros desvios em relação ao tempo padrão GNSS, que levam à perda de precisão das medições. O segmento de controlo está continuamente a monitorizar estes sistemas, mas por vezes existe um intervalo até que a anomalia seja detetada, levando a erros de uma maior escala durante esse tempo (Sathyamoorthy *et al.*, 2016).

Erros orbitais

Estes tipos de erros são principalmente causados devido a forças que perturbam a órbita do satélite, número de estações de monitorização e a sua distribuição no terreno, o número e precisão de parâmetros orbitais, e as propriedades das efemérides não serem em tempo real (Khojasteh *et al.*, 2016). Utilizando posicionamento diferencial é possível eliminar este tipo de problema, uma vez que são utilizadas estações fixas de alta precisão, permitindo assim calcular correções de posição (Kornhauser, 2006).

Erros de propagação do sinal através da atmosfera

Este tipo de problemas ocorre durante a passagem do sinal pelas camadas da atmosfera, nomeadamente a ionosfera e a troposfera. Na ionosfera, os sinais estão sujeitos a efeitos de refração causados pela variação de densidade de eletrões na camada, que por sua vez, varia consoante a intensidade da radiação proveniente do sol e outros corpos celestes, a estação do ano, posição, etc (Khojasteh *et al.*, 2016). Uma vez que a intensidade da radiação solar varia de forma imprevisível, não é possível criar um modelo matemático preciso que ajude a mitigar estes problemas. No entanto, são adotados métodos que permitem diminuir o seu impacto, como a utilização de recetores de dupla frequência e a utilização do coeficiente ionosférico transmitida na mensagem de navegação (Kornhauser, 2006). Já na troposfera, as diferentes camadas provocam um atraso na propagação dos sinais levando a erros nos cálculos das distâncias. Ao contrário da ionosfera, para a troposfera é possível criar um modelo matemático que permite corrigir os sinais através dos valores da temperatura, pressão e humidade relativa (Kornhauser, 2006).

Erros de relógio do recetor

O recetor, utilizado para receber os sinais dos satélites, é também uma fonte de erros devido ao uso de relógios de baixa precisão, como por exemplo de quartzo, inferiores em comparação com os que estão a bordo dos satélites. Isto leva a um desvio em relação ao tempo de referência GNSS que aumenta com o passar do tempo. Através do posicionamento diferencial é possível corrigir este tipo de problema (Kornhauser, 2006).

Erros de multicaminho

Os erros de multicaminho são provocados por reflexão, dispersão e difração dos sinais antes de chegar ao recetor, em edifícios, carros, árvores, etc. Isto leva a que o recetor receba múltiplos sinais adicionais com diferentes fases e atrasos, contribuindo assim para o erro do cálculo da posição (Ferreira and Nunes, 2007). De forma a mitigar este tipo de problemas, podem ser aplicadas técnicas ao nível da antena, do processamento do sinal e na navegação. Quanto ao nível da antena, é possível rejeitar parte dos sinais refletidos tirando partido do facto de os sinais GNSS serem polarizados circularmente à direita. Estas antenas terrestres normalmente apresentam uma forma hemisférica que permite rejeitar sinais que possam ser refletidos no plano terrestre. Já no nível do processamento do sinal e navegação, são aplicadas técnicas de processamento que permitem reduzir este tipo de erros (Ferreira and Nunes, 2007).

2.2 Visão Computacional

A visão computacional não é uma área de estudo recente, é aceite que o pai da visão computacional seja Lawrence Roberts, que na sua tese de doutoramento discutiu a possibilidade de extrair informação geométrica 3D a partir de uma perspetiva 2D de blocos (Huang, 1996). A visão computacional pode ser definida como o campo científico que se encarrega da extração de informação sobre imagens digitais. É uma área que agrega um grande conjunto de disciplinas como a neurociência, arquitetura de sistemas, processamento de imagem, robótica, inteligência artificial, entre outras (Krishna, 2017).

De forma geral, quer seja um sistema computacional ou animal, a visão divide-se em dois componentes. Um primeiro, encarregue de captar o máximo de detalhes possível de uma imagem. No caso do sistema animal os nossos olhos captam a luz que atravessa a íris e é projetada na retina, onde células específicas irão transmitir a informação para o cérebro através dos neurónios. Já no sistema computacional, a camara apresenta uma arquitetura semelhante, em que recebe a luz e envia os pixels com a informação para o computador. A segunda componente representa um dispositivo capaz de processar a informação. No caso animal, o cérebro está encarregue de resolver este problema que, até à data, continua a ter uma melhor performance do que a visão computacional (Krishna, 2017).

A informação extraída das imagens digitais pode ser dividida em duas categorias: medições e informação semântica. Quanto às medições, a visão computacional permite medir distâncias através de um sistema estereoscópico, isto é, com a utilização de duas ou mais câmaras. Este tipo de sistema permite obter informação sobre a profundidade, tal como os nossos dois olhos o fazem, através da triangulação. Se obtivermos imagens em torno de um objeto, é possível criar um modelo 3D da sua superfície (Krishna, 2017). Já na informação semântica, estes sistemas permitem-nos rotular diferentes objetos nas imagens, reconhecer pessoas, ações e gestos, entre outros. Na área da medicina, é também possível diagnosticar certos tipos de doença tendo por base imagens, como por exemplo cancro da pele (Krishna, 2017).

2.3 Odometria Visual

Odometria visual refere-se ao processo de determinação da posição e orientação de uma câmara através da análise de uma sequência de imagens, seja apenas com recurso a uma câmara (monocular) ou múltiplas (estéreo) (Scaramuzza et al., 2011). O nome foi escolhido devido à semelhança com a odometria a partir de rodas, que estima o movimento incremental de um veículo através de um codificador que conta o número de voltas das rodas com o passar do tempo (Scaramuzza et al., 2011). Grande parte da pesquisa inicial sobre este tema, tinha como objetivo a exploração de planetas através de rovers e foi motivada pelo programa da NASA para a exploração de Marte (Scaramuzza et al., 2011). A grande mais-valia desta técnica, está no equilíbrio entre o custo, complexidade e precisão, uma vez que são utilizadas câmaras comuns ao contrário de sensores com um custo elevado (Bai et al., 2023). No entanto, existem alguns desafios associados à utilização desta técnica em ambientes ao ar livre, como por exemplo zonas pouco iluminadas, superfícies muito lisas e sem textura e objetos em movimento (Aqel et al., 2016).

2.3.1 Técnicas de Odometria Visual Monocular

Diferentes abordagens têm sido desenvolvidas ao longo do tempo para implementar odometria visual monocular, cada uma com as suas vantagens e desafios:

Método baseado em características

Este método envolve a extração de características como cantos, linhas retas e linhas curvas entre frames consecutivos de uma sequência, a correspondência e o rastreamento das características mais distintas entre as extraídas e, por fim, a estimativa da trajetória do movimento (Aqel et al., 2016). Para a deteção de características podem ser utilizados algoritmos como SIFT (Scale-Invariant Feature Transform), SURF (Speeded Up Robust Features), ORB (Oriented FAST and Rotated BRIEF) ou FAST (Features from Accelerated Segment Test), na correspondência são utilizados métodos como FLANN (Fast Library for Approximate Nearest Neighbors), e por fim, na estimação de movimento podem ser aplicados algoritmos como RANSAC (Random Sample Consensus) para estimar a matriz essencial e recuperar o movimento da câmara (Shashi et al, 2018).

Método direto

Esta abordagem opera diretamente sobre as intensidades dos pixels, evitando a etapa de extração de características. O alinhamento de imagens que minimiza as diferenças de intensidade entre os pixels correspondentes em frames consecutivos, já para estimar a transformação, são aplicadas otimizações que utilizam métodos como Gauss-Newton (Shashi et al, 2018).

Método semi-direto

Combina elementos dos dois métodos anteriores, em características e direto, de forma a aproveitar vantagens de ambos. Para selecionar pixels, identifica os que apresentam um maior gradiente para serem rastreados, aplica otimização direta em secções ao redor dos pixels selecionados (Shashi et al, 2018).

A odometria visual monocular apresenta vantagens como o custo do hardware ser reduzido e a complexidade do sistema ser baixa em relação à precisão que produz. Já quanto aos desafios, esta técnica tem dificuldades em ambientes com pouca textura uma vez que dificulta a deteção de características distintas, movimentos rápidos da câmara podem causar distorções nas imagens, grandes variações de iluminação podem comprometer a correspondência das características entre os frames, e por fim, encontrar um balanço entre precisão e velocidade de processamento, especialmente para aplicações em tempo real (Shashi et al, 2018).

2.4 OpenCV

OpenCV (Open Source Computer Vision Library) é uma poderosa biblioteca em C++ que contém centenas de algoritmos dedicados à visão computacional e ao processamento de imagem. Posto isto, esta biblioteca tem um papel bastante importante na implementação de sistemas de odometria visual monocular, que permitem auxiliar sistemas GNSS a estimar o posicionamento com uma maior precisão.

3. Metodologia

3.1 Visão Geral do Sistema

O sistema proposto pretende implementar técnicas de odometria visual monocular de forma a permitir a reconstrução do trajeto efetuado pelo observador. Este pode ser dividido em 5 componentes, um primeiro que se encarrega de carregar e gerir o dataset importado, seja este um conjunto de imagens ou um ficheiro de vídeo, bem como um pré-processamento sobre os frames. Um segundo responsável pela deteção das principais caraterísticas nas imagens. Um terceiro que rastreia estas caraterísticas, permitindo identificar o local em que se encontra cada uma nos diferentes frames ao longo do tempo. Um quarto que possibilita estimar a pose da câmara bem como o trajeto efetuado durante a sequencia. Pode também ser considerado um quinto componente, responsável pela renderização de toda a interface bem como a trajetória estimada.

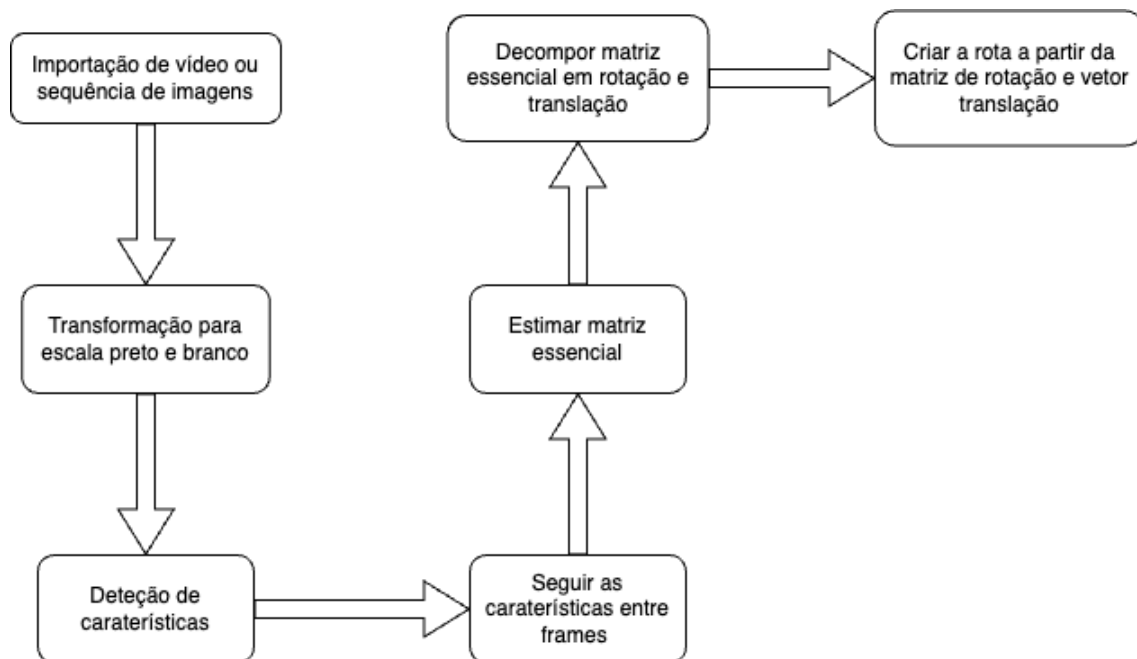


Figura 1 Arquitetura do sistema de odometria visual.

3.2 Implementação do Sistema de Odometria Visual Monocular

3.2.1 Importação de dados e pré-processamento

A primeira etapa do processamento, foca-se no carregamento de todos os dados necessários, sendo neste caso, um dataset de uma sequência de imagens ou um ficheiro mp4. Isto é realizado através de uma classe da biblioteca OpenCV chamada *VideoCapture*, que nos fornece uma interface capaz de ler ficheiros de diferentes formatos, ou até mesmo uma transmissão de vídeo em tempo real através de uma câmara conectada ao computador. Esta abstração permite-nos focar no algoritmo de odometria visual monocular em vez dos detalhes da decodificação e processamento de baixo nível dos vídeos. Os dados ficam armazenados num objeto em memória que nos permite ler frame a frame através da função *read*. Sempre que o sistema acaba de processar um frame, é pedido o próximo até que o dataset ou vídeo cheguem ao fim.

Sempre que uma imagem nova é carregada, esta passa por um pré-processamento que a transforma numa escala a preto e branco. Este passo serve propósitos importantes como:

- Eficiência computacional, uma vez que imagens a preto e branco são representadas apenas por uma matriz em vez de 3 como no caso RGB. Deste modo, é possível reduzir a carga computacional necessária para processar cada frame.
- Menor variação de iluminação, dado que as imagens a preto e branco são mais resistentes a mudanças na iluminação e na cor do ambiente, criando uma imagem mais homogénea.
- Melhor extração de características, uma vez que grande parte dos algoritmos desenvolvidos para extrair características das imagens funcionam melhor com este tipo de escala, sendo que trabalham com gradientes de intensidade que são mais perceptíveis em imagens a preto e branco.
- Eficiência de memória, já que apenas é necessário armazenar os dados referentes a uma matriz de valores, ao contrário das imagens a cores que são representadas por 3 matrizes.

Na biblioteca OpenCV, é a função *cvtColor* que está encarregue desta conversão de forma eficiente, sendo que tenta manter a melhor qualidade de imagem possível quando reduz a informação para apenas um canal. No sistema desenvolvido, esta função é sempre chamada quando um novo frame é carregado.

3.2.2 Detecção e seguimento de caraterísticas

Um dos pontos importantes num sistema de odometria visual, foca-se em detetar caraterísticas distintas e segui-las entre os diferentes frames. Esta etapa é fundamental para posteriormente estimar o movimento da câmara. Para tal, utilizaram-se dois algoritmos para proceder à detecção das caraterísticas, o ORB e o FAST, para os quais será feita uma comparação entre estes dois no capítulo dos resultados.

O FAST é um algoritmo que deteta caraterísticas em imagens de forma rápida e eficiente. Veio solucionar a falta de algoritmos que fossem capazes de serem aplicados a ambientes em tempo real (Rosten et al., 2006). O algoritmo funciona da seguinte forma, primeiro seleciona um pixel na imagem para testar se é um canto, depois é considerado um círculo de 16 pixels em volta deste, sendo que, o pixel é considerado um canto se houver n pixels contíguos dentro do círculo com uma diferença de intensidade elevada. Depois existem uma série de otimizações aplicadas para tornar o processo mais rápido e eficiente.

O ORB, baseado no algoritmo BRIEF (Binary Robust Independent Elementary Features) e no FAST, foi desenvolvido pela OpenCV Labs. Esta combinação apresenta diversas vantagens, entre elas:

- Velocidade, o ORB é bastante eficiente, o que o torna numa boa escolha para aplicações em tempo real ou sistema com poucos recursos.
- Invariância quanto à rotação, permitindo ser mais resistente a rotações da câmara.
- Invariância quanto à escala, dado que utiliza uma pirâmide de escalas para detetar caraterísticas em múltiplas escalas.
- Ser livre de licenças, ao contrário de outros algoritmos como o SIFT e SURF, está livre de licenças e pode ser utilizado para qualquer tipo de projeto de forma gratuita (Rublee et al., 2011).



Figura 2 Imagem de um dos frames da sequência de vídeo processada pelo Sistema, mostrando os pontos-chave detetados pelo algoritmo ORB.

Como é possível verificar na Figura 2, o algoritmo deteta pontos que se realcem de forma a facilitar o seu seguimento nas imagens seguintes da sequência.

Para cada ciclo de processamento, isto é, sempre que um frame é processado, o algoritmo, ORB ou FAST, é chamado para detetar novas caraterísticas, uma vez que muitas destas acabam por ficar fora das imagens ou até mesmo obstruídas por outros objetos no decorrer da sequência. Depois de identificadas as caraterísticas, o sistema utiliza um método chamado *Lucas-Kanade optical flow* que nos permite seguir os pontos entre os diferentes frames (Figura 3). O fluxo ótico corresponde ao padrão do movimento aparente de um objeto numa sequência de imagens, devido ao movimento da câmara ou do próprio objeto.

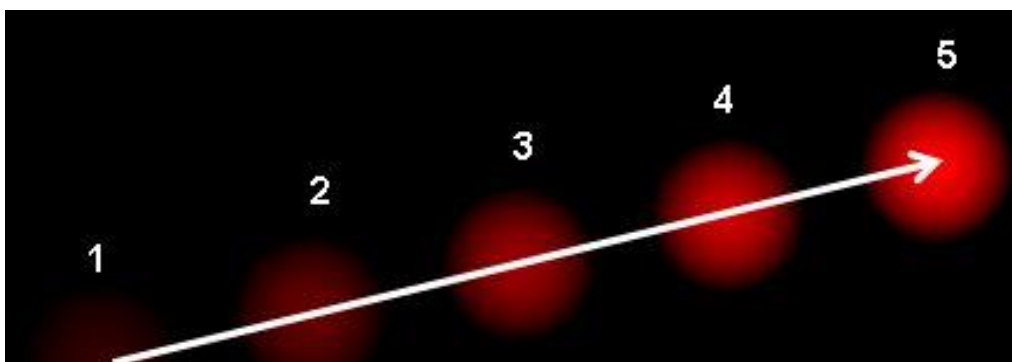


Figura 3 Representação de vetores de fluxo ótico calculados entre dois frames consecutivos. Os vetores indicam o movimento aparente dos pontos-chave detetados pelo algoritmo, estimado para calcular a trajetória da câmara.

Pode ser representado por um vetor 2D (Figura 3) que representa o deslocamento do ponto de um frame para o outro. O método Lucas-Kanade assume que o fluxo ótico é constante dentro de uma janela 3x3 em redor do pixel a seguir, permitindo resolver a equação de fluxo ótico para todos os pixels nessa vizinhança, usando o critério dos mínimos quadrados. Na biblioteca OpenCV este método é dado pela função `calcOpticalFlowPyrLK`. Os principais parâmetros de entrada desta função são:

- Tamanho da janela, define o tamanho da região vizinha ao ponto a seguir, sendo que janelas maiores permitem seguir movimentos mais rápidos.
- Níveis da pirâmide, otimização que permite obter melhores resultados com movimentos de maior dimensão.
- Critério de término, define quando o processo iterativo deve terminar baseado no número máximo de iterações e um threshold mínimo de melhoria.

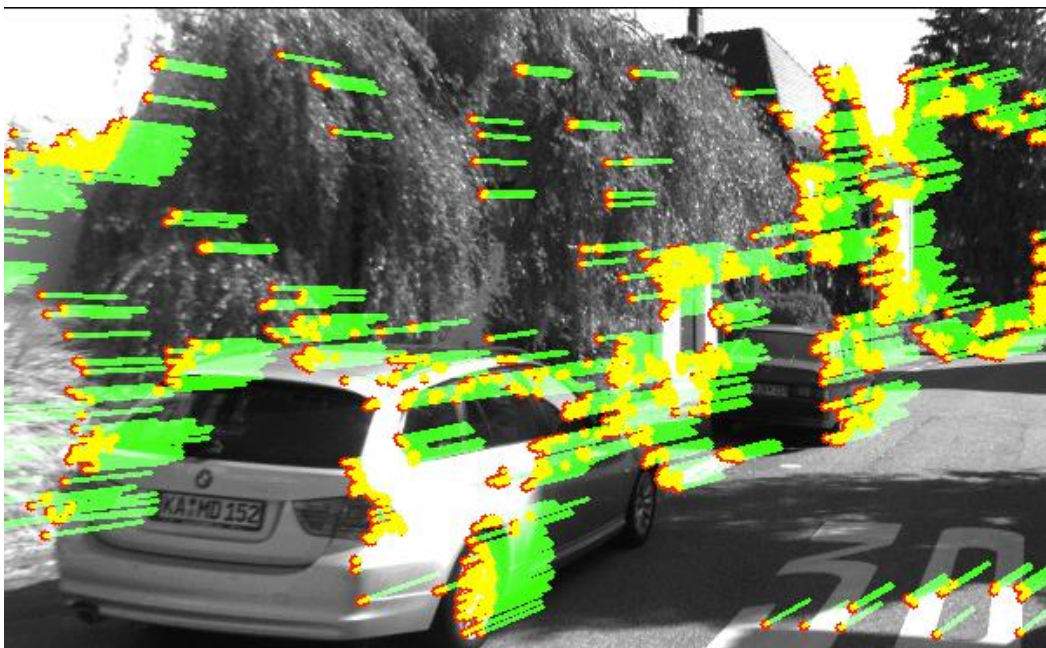


Figura 4 Representação gráfica dos vetores de fluxo ótico entre dois frames consecutivos, calculados pelo método Lucas-Kanade. Cada vetor indica o deslocamento do ponto-chave detetado.

Na Figura 4, é possível visualizar o vetor de fluxo ótico para as características detetadas pelo algoritmo ORB. Na interface foi adicionada uma opção para que seja possível ativar e desativar esta visualização. Este último algoritmo retorna as novas posições para estes pontos, bem como um vetor que indica se o novo ponto foi calculado com sucesso ou não. Isto permite rejeitar os pontos maus e usar os bons para estimar as novas posições do frame seguinte.

3.2.3 Estimar movimento

Já com um conjunto de características correspondentes entre dois frames, o atual e o anterior, podemos começar a estimar o movimento da câmara. Para tal, será necessário em primeiro lugar calcular a matriz essencial, e de seguida decompô-la para retirar a rotação relativa e a translação. A matriz essencial é um conceito fundamental na visão computacional. É uma matrix 3x3 que codifica a geometria epipolar entre duas imagens calibradas. Dado um ponto de uma imagem, multiplicando-o pela matriz essencial vamos obter a linha epipolar na segunda imagem. Posto isto, ela é capaz de englobar a rotação e translação relativa entre dois frames. Para tal, a biblioteca OpenCV contém uma função chamada `findEssentialMat`, que nos permite calcular esta matriz. Alguns dos parâmetros mais importantes para esta função são:

- Pontos correspondentes nas duas imagens, características que aparecem nas duas imagens.
- Parâmetros internos da câmara, sendo eles a distância focal e o ponto principal que definem a sua geometria interna.
- Método para calcular a matriz, para este parâmetro foi escolhido o algoritmo RANSAC (Random Sample Consensus).
- Nível de confiança e distância máxima de um ponto até à linha epipolar, aqui foram escolhidos os valores 0.999 e 1.0 respetivamente.

RANSAC é um algoritmo iterativo usado para estimar parâmetros de um modelo matemático a partir de um conjunto de pontos observados que contém valores fora do normal. Este método funciona selecionando repetidamente pequenos subconjuntos aleatórios dos dados, depois o algoritmo ajusta um modelo a este subconjunto e testa se o modelo se ajusta bem ao resto dos dados. (Derpanis, 2010)

Depois de termos a matriz essencial definida, é possível fazer a sua decomposição para obtermos a rotação relativa e translação entre dois frames, ou seja, entre duas posições da câmara. Para este passo, foi utilizada a função `recoverPose` da biblioteca OpenCV que nos retorna uma matriz de rotação e um vetor de translação.

4. Resultados com o dataset KITTI

4.1 Conjunto de dados KITTI

O dataset KITTI foi introduzido por Geiger *et al.* (2013), e é utilizado como referência no campo da condução autónoma e da visão computacional. Para o desenvolvimento do sistema de odometria visual monocular desta dissertação, este conjunto de dados forneceu um ponto de partida muito bom para testar e validar a solução desenvolvida, para que posteriormente seja possível testar com dados recolhidos no terreno.

Os dados KITTI foram gravados a partir de uma carrinha equipada com sensores de alta precisão, em ruas da cidade Karlsruhe na Alemanha. A configuração de gravação era composta por dois pares de câmaras, um deles a cores e o outro a preto e branco, um laser scanner 3D e um sistema de navegação inercial GPS/IMU. Este dataset contém diversos tipos de dados, no entanto, apenas foram utilizadas as imagens em formato PNG da câmara central a preto e branco e a posição relativa de cada imagem. As imagens têm uma dimensão de 1226×370 .

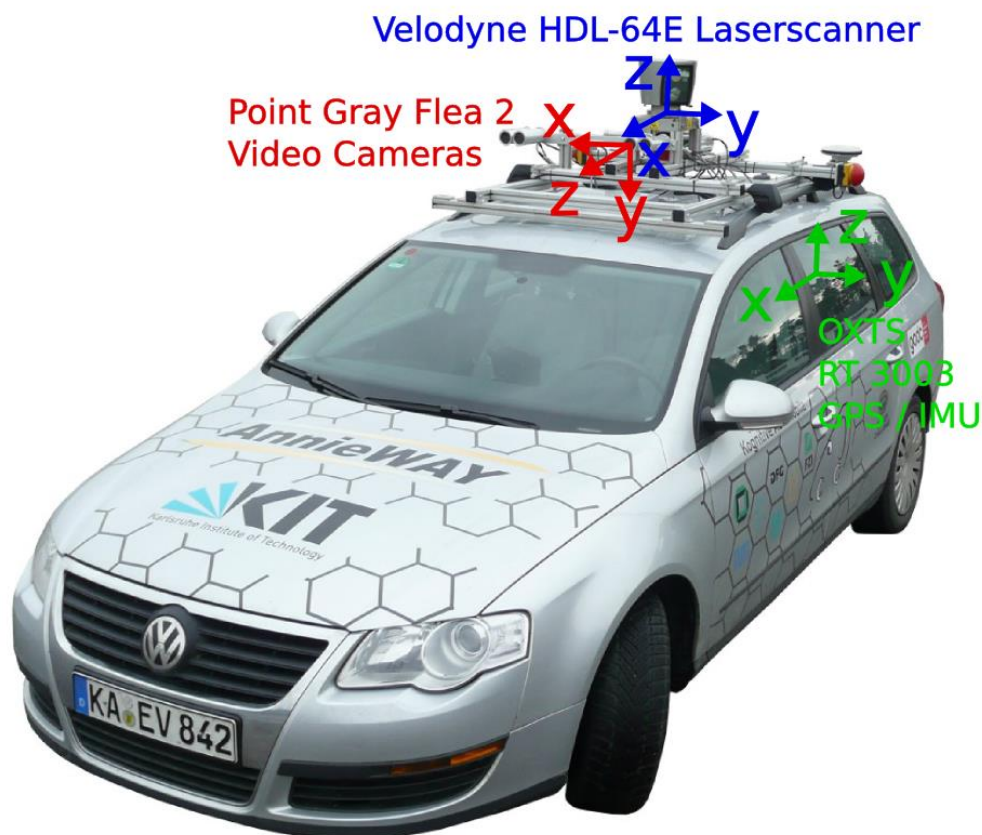


Figura 5 Configuração da carrinha utilizada para obter o dataset KITTI (<https://www.cvlibs.net/datasets/kitti/setup.php>).

4.2 Resultados obtidos com o dataset KITTI

Através do dataset KITTI, começou-se por fazer uma comparação dos trajetos estimados utilizando o algoritmo FAST e o algoritmo ORB. Para tal, foram selecionadas duas sequências em zonas rurais do dataset. Depois de todos os dados serem processados, foram obtidos os seguintes trajetos compostos por coordenadas no sistema de referência da câmara, isto é, um sistema de coordenadas cartesiano tridimensional (x,y,z), onde a origem corresponde à posição inicial da câmara, e os eixos são definidos em relação à orientação inicial.

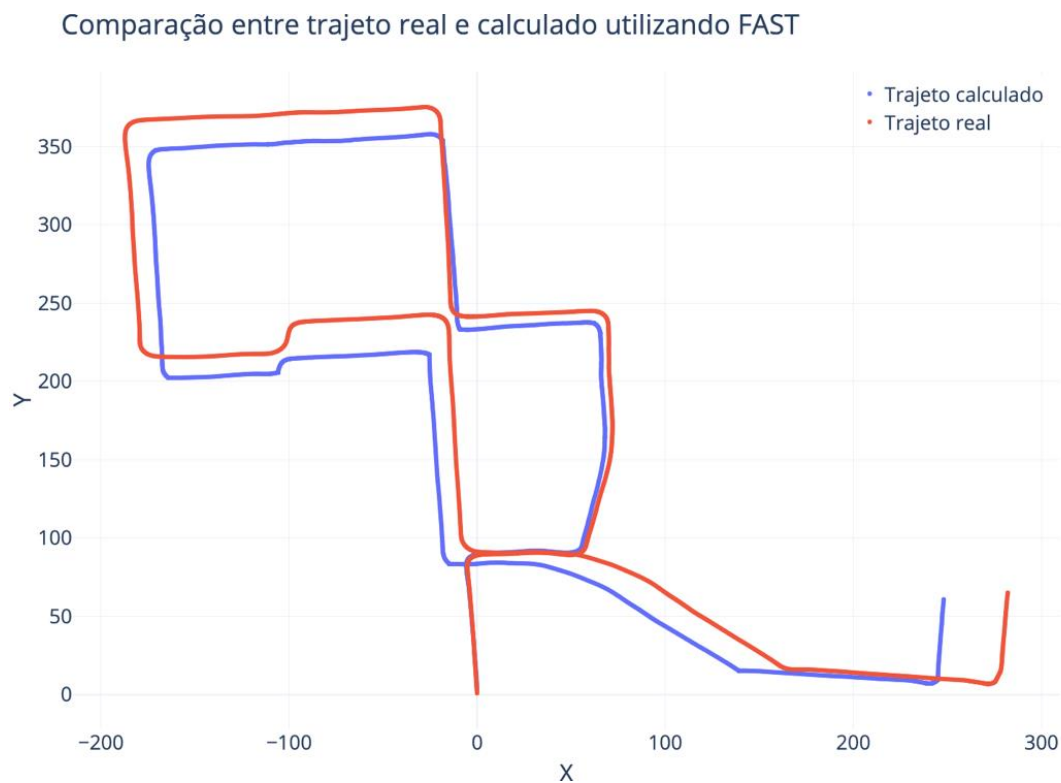


Gráfico 1 Trajeto estimado da sequência 1 utilizando o algoritmo FAST.

Comparação entre trajeto real e calculado utilizando ORB

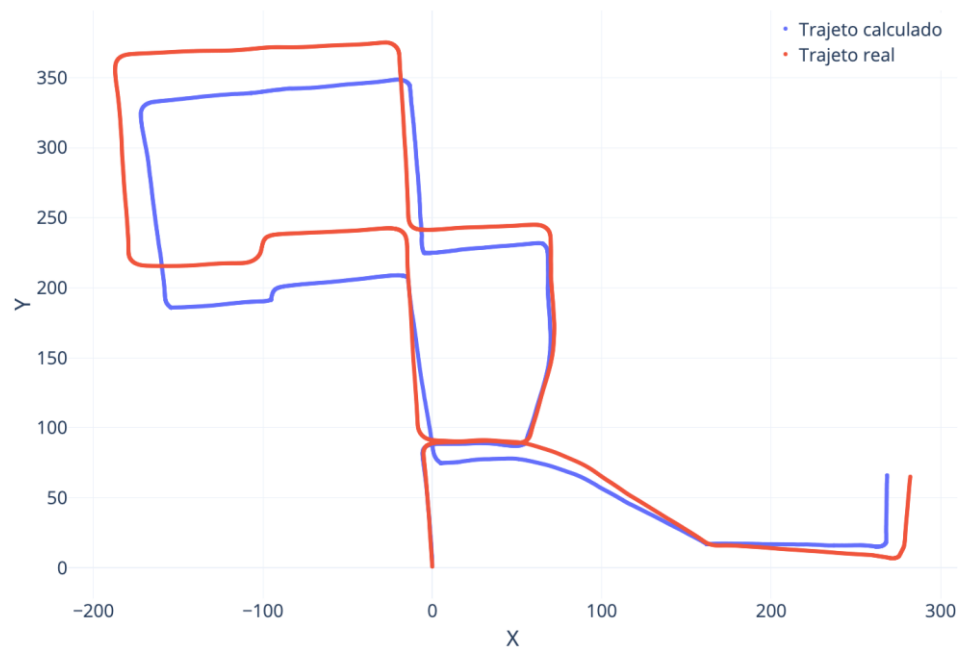


Gráfico 3 Trajeto estimado da sequência 1 utilizando o algoritmo ORB.

Comparação entre trajeto real e calculado utilizando FAST

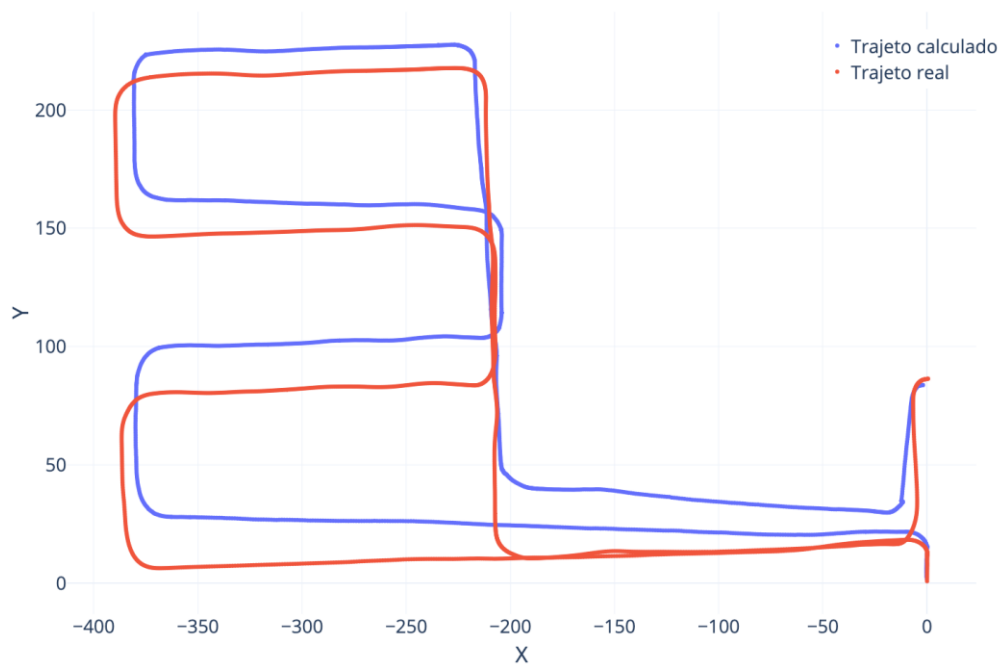


Gráfico 2 Trajeto estimado da sequência 2 utilizando o algoritmo FAST.

Comparação entre trajeto real e calculado utilizando ORB

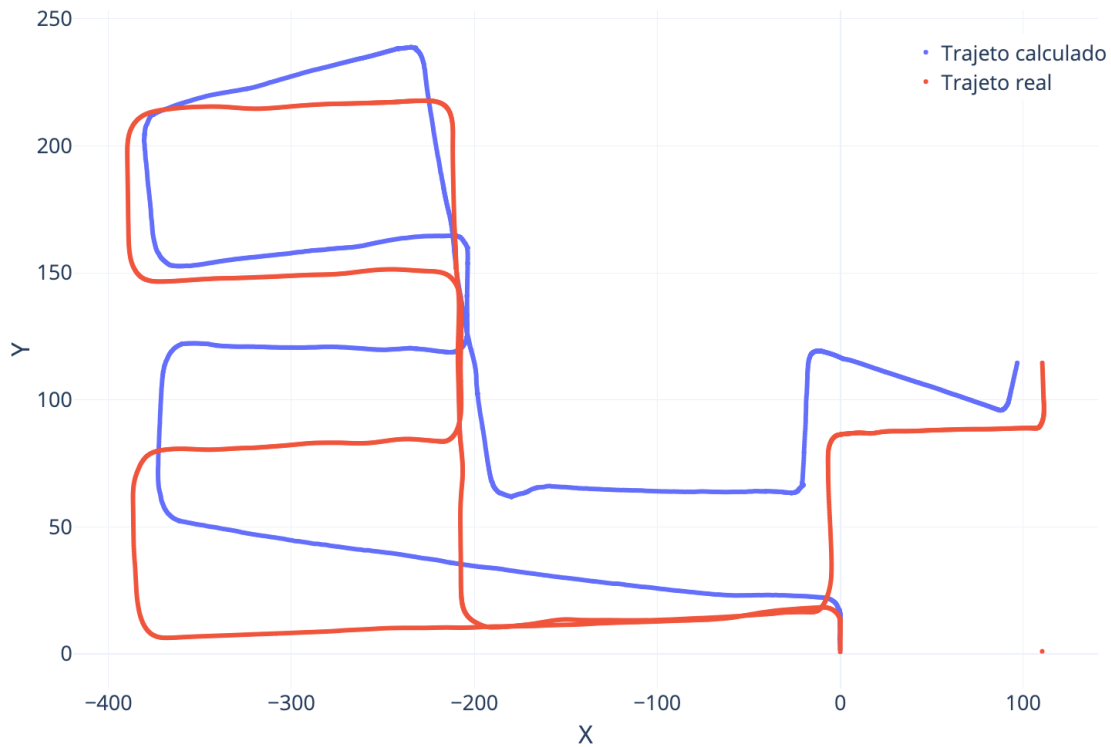


Gráfico 4 Trajeto estimado da sequência 2 utilizando o algoritmo ORB.

Ao observarmos os gráficos 1, 2, 3 e 4 das trajetórias estimadas, podemos concluir que os trajetos estimados com o algoritmo FAST são mais próximos do trajeto real do que os estimados com o algoritmo ORB. Para a primeira sequência com o algoritmo FAST (Gráfico 1) verifica-se que o valor do desvio máximo é aproximadamente 20 metros, enquanto que com o algoritmo ORB (Gráfico 2) este valor é de aproximadamente 40 metros. Podemos ainda observar que o algoritmo ORB teve dificuldade acrescida na sequência 2 (Gráfico 4), podendo estar relacionado com existência de um maior número de carros e ciclistas em movimento, que levam a erros ao calcular as posições da câmara. Posto isto, o algoritmo FAST foi o escolhido para ficar como definitivo no sistema.

5. Resultados obtidos com GoPro

Para avaliar o desempenho do sistema de odometria visual monocular com dados obtidos no terreno, utilizou-se uma câmera GoPro fixada num veículo durante um trajeto na região de Matosinhos, com extensão aproximada de 1132 metros. Antes de iniciar as gravações, a GoPro foi configurada no modo linear, uma funcionalidade que corrige automaticamente as distorções causadas pela lente grande-angular característica desse tipo de câmara. Essa configuração desempenha um papel fundamental, pois a distorção geométrica, caso não corrigida, introduz inconsistências nas imagens que poderiam comprometer a precisão dos cálculos realizados pelo algoritmo de odometria visual (Figura 7).

A eliminação da distorção é especialmente crítica para o sucesso da estimativa de trajetória, uma vez que a odometria visual depende de correspondências precisas entre os pontos detectados em frames consecutivos. Qualquer deformação na geometria da imagem poderia resultar em erros significativos na determinação da posição e orientação da câmara ao longo do trajeto. Assim, garantir a integridade geométrica das imagens recolhidas não é apenas uma etapa preparatória, mas sim um requisito indispensável para a robustez e confiabilidade do sistema.



Figura 6 Trajeto efetuado com a GoPro na região de Matosinhos



Figura 7 Imagem num dos pontos do trajeto obtida com a GoPro.

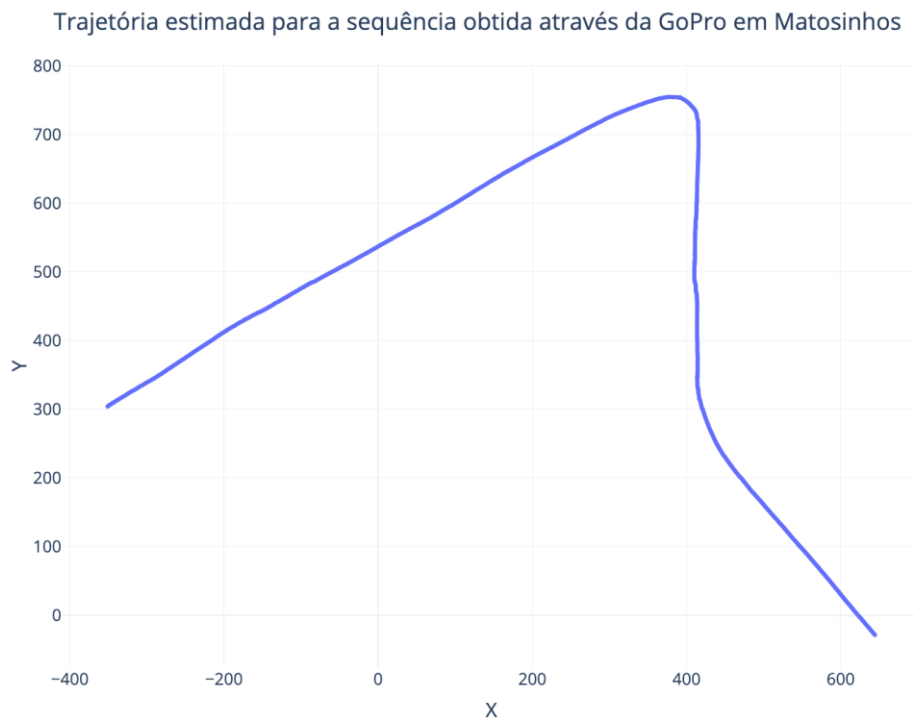


Figura 8 Trajeto estimado da sequência gravado com a GoPro em Matosinhos.

Ao analisarmos a Figura 6 e 8 percebemos que o sistema conseguiu estimar uma trajetória semelhante à real. No entanto, nesta região existe muito movimento de carros e ciclista que provocam erros ao estimar as posições da câmara, dificultando todo o processo.

Embora o sistema de odometria visual tenha demonstrado boa capacidade para estimar trajetórias em condições desafiadoras, é inevitável que ocorram derivas ao longo do tempo, especialmente em trajetos mais extensos. Como proposta para trabalhos futuros, sugere-se integrar o sistema de odometria visual com o posicionamento GNSS de forma a corrigir essas derivas sempre que o sinal GNSS apresente qualidade suficiente. Nesse cenário, a trajetória reconstruída pela odometria visual seria ajustada entre dois pontos GNSS confiáveis, funcionando de forma análoga a uma poligonal topográfica. Essa abordagem permitiria que trajetos extensos ou em ambientes de sinal intermitente fossem corrigidos de forma eficaz, utilizando apenas algumas posições GNSS válidas para eliminar as falhas acumuladas ao longo do percurso. A implementação dessa técnica poderia aumentar significativamente a precisão e robustez do sistema, tornando-o mais adequado para aplicações práticas em ambientes urbanos e naturais.

6. Interface gráfica

Este capítulo descreve o desenvolvimento da interface gráfica para análise de vídeos através de um sistema de odometria visual monocular. Esta interface foi desenvolvida em C++ utilizando a biblioteca ImGui, que permite criar janelas do estilo popup de forma simples e eficaz. O principal objetivo era criar uma pequena aplicação gráfica para facilitar a visualização, análise do trajeto estimado e controlo sobre o vídeo.

O sistema foi desenvolvido tendo por base duas classes, a *MainView* que é responsável por criar e gerir todas as janelas e definições da interface, e a classe *VideoPlayback* que trata de todo o processamento dos frames. É nesta última classe que está definido o sistema de odometria visual monocular.

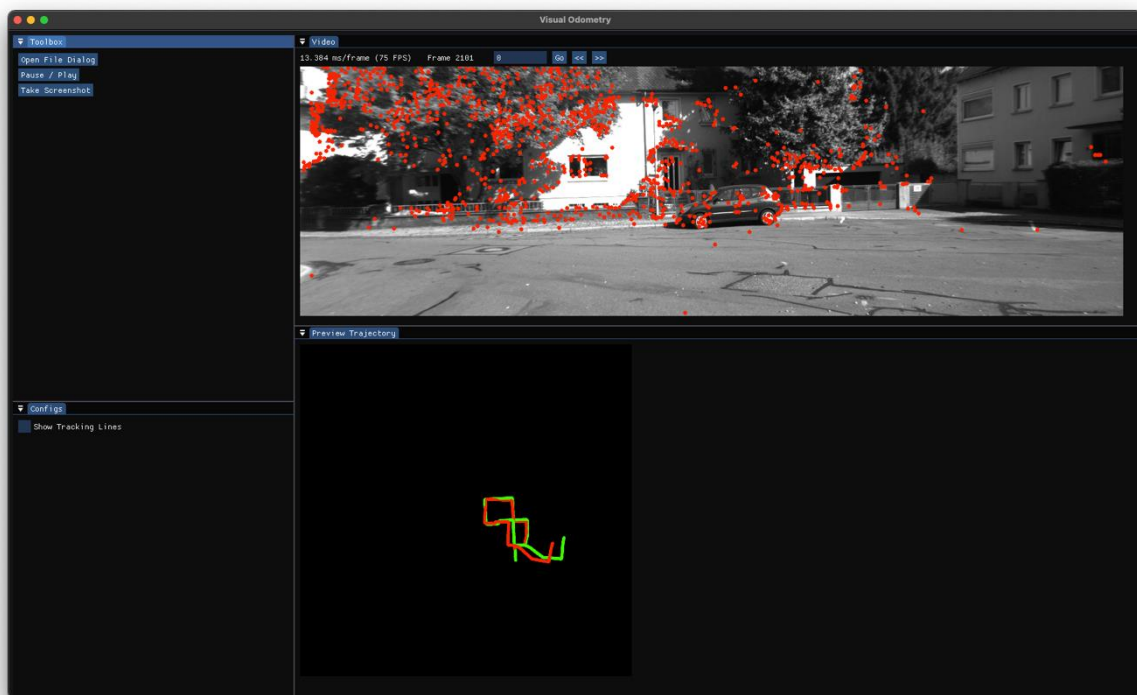


Figura 9 Interface gráfica

Como é possível visualizar na Figura 8, na janela *Toolbox* no canto superior do lado esquerdo temos alguns botões que permitem abrir novos vídeos/sequências de imagens, um botão de pausa e um botão que permite guardar uma imagem do frame atual com as características representadas. No canto inferior esquerdo temos uma opção para ativar os vetores de fluxo que permitem visualizar onde o ponto estava na imagem anterior e o seu deslocamento. Na parte superior do lado direito, temos a reprodução do vídeo com informações sobre o tempo de cada frame, o número de frames por segundo e o número do frame atual. Também é possível escolher um frame em específico ou

navegar entre os frames manualmente. Na região inferior encontramos a vermelho a rota estimada pelo sistema e a verde a rota real da sequência de imagens.

A interface desenvolvida revelou ser uma ferramenta indispensável durante o processo de análise e avaliação do sistema de odometria visual. A sua simplicidade de utilização e funcionalidades interativas permitiram uma visualização clara das trajetórias estimadas, facilitando a comparação com o percurso real e a identificação de pontos de melhoria no sistema. Além disso, a possibilidade de acompanhar dinamicamente os pontos-chave detetados e os vetores de fluxo ótico entre os frames, contribuiu de forma significativa para a análise detalhada do comportamento do algoritmo em diferentes condições.

Para trabalhos futuros, esta interface possui um elevado potencial de evolução, de forma a incorporar novas funcionalidades que alarguem o seu âmbito de aplicação. Uma proposta seria integrá-la com módulos capazes de realizar cálculos de coordenadas georreferenciadas através de técnicas fotogramétricas, tirando partido de informações de calibração e georreferenciação, caso estas estejam disponíveis. Com esta adição, a interface poderia ser utilizada não só para análises de trajetórias, mas também como uma ferramenta valiosa para projetos que envolvam mapeamento detalhado e navegação em ambientes complexos.

Adicionalmente, funcionalidades como a visualização em tempo real de trajetórias georreferenciadas, a integração com dados GNSS e até a exportação de resultados em formatos compatíveis com softwares de mapeamento, permitiam ampliar a sua aplicabilidade em contextos práticos. Desta forma, a interface deixaria de ser apenas um suporte para testes e validações, tornando-se um componente central em sistemas integrados de navegação e mapeamento, respondendo a necessidades ainda mais abrangentes no domínio das geotecnologias.

7 Conclusão

Esta dissertação de mestrado apresentou os conceitos, o desenvolvimento e avaliação de um sistema de odometria visual monocular, com o objetivo de representar o trajeto percorrido por uma câmara, permitindo assim caso seja necessário, corrigir posições obtidas por um recetor GNSS onde o sinal é fraco ou instável.

O sistema desenvolvido demonstrou ser capaz de reconstruir trajetórias com uma precisão suficiente, tanto utilizando o dataset KITTI quanto com dados obtidos no terreno através de uma câmara GoPro. Pelo que, ficou demonstrado que a odometria visual monocular é uma técnica viável para complementar sistema GNSS tradicionais.

No decorrer do projeto, foi também efetuada uma comparação entre os algoritmos FAST e ORB para identificar características nas imagens, que revelou melhores resultados utilizando o algoritmo FAST na reconstrução das trajetórias.

Os testes realizados em ambientes urbanos com veículos e pedestres em movimento, evidenciaram os desafios adicionais que objetos em movimento apresentam para a odometria visual. Este ponto seria algo a melhorar no futuro, para que o sistema possa ser mais robusto nos grandes centros urbanos.

Foram também observadas algumas limitações como a acumulação de erros ao longo de trajetórias mais longas, desafios em cenas com pouca textura ou iluminação variável e objetos em movimento no ambiente. Futuramente poderiam ser aplicadas técnicas de otimização global ou até mesmo a fusão com outros sensores, de forma a criar um sistema mais robusto que aguarde em grande parte dos cenários.

Em suma, esta dissertação de mestrado estabelece uma base para futuros desenvolvimentos em sistemas de navegação que utilizem a odometria visual monocular como técnica suplementar de baixo custo e complexidade. Os resultados obtidos são promissores e abrem caminho para aplicações práticas em diversos campos, desde condução autónoma, mapeamento urbano e até mesmo operações de resgate em ambientes complexos.

Referências Bibliográficas

Alain L. Kornhauser (2006) 'Global Navigation Satellite System (GNSS)'.

Ali N. Khojasteh *et al.* (2016) *Introduction to Global Navigation Satellite Systems and Its Errors, International Academic Journal of Science and Engineering Vol. 3, No. 3.* Available at: https://www.researchgate.net/publication/303494970_Introduction_to_Global_Navigation_Satellite_Systems_and_Its_Errors (Accessed: 21 July 2024).

Aqel, Mohammad O A *et al.* (2016) 'Review of visual odometry: types, approaches, challenges, and applications', *SpringerPlus*, 5, pp. 1–26.

Aqel, Mohammad O. A. *et al.* (2016) 'Review of visual odometry: types, approaches, challenges, and applications', *SpringerPlus*, 5(1), p. 1897. Available at: <https://doi.org/10.1186/s40064-016-3573-7>.

Bai, Y. *et al.* (2023) 'Vision-based navigation and guidance for agricultural autonomous vehicles and robots: A review', *Computers and Electronics in Agriculture*, 205, p. 107584. Available at: <https://doi.org/10.1016/J.COMPAG.2022.107584>.

Blewitt, G. (1997) 'Basics of the GPS Technique: Observation Equations §'.

Derpanis, K. (2010) *Overview of the RANSAC Algorithm*.

Ferreira, T. and Nunes, F. (2007) 'Advanced multipath mitigation techniques for GNSS receivers'.

Geiger, A. *et al.* (2013) 'Vision meets robotics: The kitti dataset', *The International Journal of Robotics Research*, 32(11), pp. 1231–1237.

Huang, T.S. (1996) 'Computer vision: Evolution and promise', pp. 21–26.

Krishna, R. (2017) *Computer vision: Foundations and applications*. Stanford University.

Rosten, E. and Drummond, T. (2006) 'Machine Learning for High-Speed Corner Detection', in, pp. 430–443. Available at: https://doi.org/10.1007/11744023_34.

Rublee, E. *et al.* (2011) 'ORB: An efficient alternative to SIFT or SURF', in *2011 International Conference on Computer Vision*. IEEE, pp. 2564–2571. Available at: <https://doi.org/10.1109/ICCV.2011.6126544>.

Sathyamoorthy, D. *et al.* (2016) 'Evaluating the effect of Global Positioning System (GPS) satellite clock error via GPS simulation', *IOP Conference Series: Earth and*

Environmental Science, 37(1). Available at: <https://doi.org/10.1088/1755-1315/37/1/012013>.

Scaramuzza, D. and Fraundorfer, F. (2011) 'Tutorial: Visual odometry', *IEEE Robotics and Automation Magazine*, 18(4), pp. 80–92. Available at: <https://doi.org/10.1109/MRA.2011.943233>.

Shashi, P., Rahul, K. and Vinod, K. (2018) *Evolution of Visual Odometry Techniques*.

Tran, M.T. et al. (no date) 'Global Navigation Satellite Systems (GNSS): An Overview'.

UN (2024) *Dia Mundial da População*, <https://unric.org/pt/mensagem-do-dia-mundial-da-populacao/>.

Anexo 1 – Processamento de frames

```
#include "VideoPlayback.h"

namespace App {
    VideoPlayback::VideoPlayback()
    {
        cap = new cv::VideoCapture;
        Texture2d tex;
        texture = std::make_unique<Texture2d>(tex);
    }

    VideoPlayback::~VideoPlayback()
    {
        delete cap;
    }

    void VideoPlayback::LoadVideo(std::string filename)
    {
        frame_count = 0;
        m_filename = filename;
        cap->open(filename);
        if (!cap->isOpened())
        {
            std::cout << filename << std::endl;
            std::cout << "Cannot open the video.\n" << std::endl;
        }
        pause = false;
        points_text_file1.open ("test.csv");
        points_text_file2.open ("ground_truth.csv");
        ground_truth_points = getGreyCamGroundPoses();
    }

    GLuint VideoPlayback::GetFrame()
    {
        if (!pause)
        {
            ProcessFrame();
        }
        return texture->GetTexture1();
    }

    GLuint VideoPlayback::GetFrameTraj()
    {
        return texture->GetTexture2();
    }

    double VideoPlayback::getAbsoluteScale(int frame_id, int sequence_id,
double z_cal) {
```

```

std::string line;
int i = 0;
std::ifstream myfile ("/Users/joao/Downloads/dataset_4/poses/00.txt");
double x =0, y=0, z = 0;
double x_prev, y_prev, z_prev;
if (myfile.is_open())
{
    while (( getline (myfile,line) ) && (i<=frame_id))
    {
        z_prev = z;
        x_prev = x;
        y_prev = y;
        std::istringstream in(line);
        for (int j=0; j<12; j++) {
            in >> z ;
            if (j==7) y=z;
            if (j==3) x=z;
        }
        i++;
    }
    myfile.close();
}
else {
    std::cout << "Unable to open file";
    return 0;
}

return sqrt((x-x_prev)*(x-x_prev) + (y-y_prev)*(y-y_prev) + (z-z_prev)*(z-
z_prev)) ;

}

std::vector<cv::Point2f> VideoPlayback::getGreyCamGroundPoses() {
    std::string line;
    int i = 0;
    std::ifstream myfile ("/Users/joao/Downloads/dataset_4/poses/00.txt");
    double value = 0;
    std::vector<cv::Point2f> poses;
    if (myfile.is_open())
    {
        while ( getline (myfile,line) )
        {
            cv::Point2f pose;
            std::istringstream in(line);
            for (int j=0; j<12; j++) {
                in >> value;
                if (j==11) pose.y=value;
                if (j==3) pose.x=value;
            }

            poses.push_back(pose);

```

```

        i++;
    }
    myfile.close();
}

return poses;

}

void VideoPlayback::ProcessFrame()
{
    if (first_frame)
    {
        std::cout << "First frame" << std::endl;
        cap->read(frame);

        cv::cvtColor(frame, last_frame, cv::COLOR_BGR2GRAY);

        std::vector<cv::KeyPoint> keypoints_fast;
        int fast_threshold = 20;
        bool nonmaxSuppression = true;
        cv::FAST(last_frame, keypoints_fast, fast_threshold,
nonmaxSuppression);
        cv::KeyPoint::convert(keypoints_fast, last_keypoints,
std::vector<int>());

        mask = cv::Mat::zeros(frame.size(), frame.type());
        draw_traj = cv::Mat::zeros(2000, 2000, frame.type());
        std::cout << frame.size() << std::endl;
    }
    cap->read(frame);
    std::cout << frame_count << std::endl;
    if (frame_count > 537 && frame_count < 561) {
        frame_count++;
        return;
    }
    cv::cvtColor(frame, frame_gray, cv::COLOR_BGR2GRAY);
    cv::Mat draw_frame;

    std::vector<uchar> status;
    std::vector<float> err;
    cv::TermCriteria criteria = cv::TermCriteria(cv::TermCriteria::COUNT +
cv::TermCriteria::EPS, 30, 0.01);
    cv::calcOpticalFlowPyrLK(last_frame, frame_gray, last_keypoints,
keypoints, status, err, cv::Size(21,21), 3, criteria, 0, 0.001);

    std::vector<cv::Point2f> good_new;
    std::vector<cv::Point2f> good_old;
    mask = cv::Mat::zeros(frame.size(), frame.type());

```

```

for(uint i = 0; i < last_keypoints.size(); i++)
{
    if(status[i] == 1)
    {
        good_new.push_back(keypoints[i]);
        good_old.push_back(last_keypoints[i]);
        if (show_track_lines)
            cv::line(mask, keypoints[i], last_keypoints[i],
cv::Scalar(0, 255, 0), 2);
        cv::circle(frame, keypoints[i], 3, cv::Scalar(0, 0, 255), -1);
        if (save_points)
            points_text_file1 << frame_count << "," << keypoints[i].x
<< "," << keypoints[i].y << "\n";
    }
}

double focal = 718.8560;
cv::Point2d pp(607.1928, 185.2157);
cv::Mat Essential, R, t, mask2;
std::cout << good_old.size() << std::endl;
Essential = cv::findEssentialMat(good_new, good_old, focal, pp,
cv::RANSAC, 0.999, 1.0, mask2);
cv::recoverPose(Essential, good_old, good_new, R, t, focal, pp,
mask2);

if (first_frame)
{
    R_final = R;
    T_final = t;
}
std::cout << frame.type() << std::endl;

double scale = getAbsoluteScale(frame_count, 0, t.at<double>(2));
std::cout << "scale" << std::endl;
std::cout << scale << std::endl;

if (scale>0.1 && t.at<double>(2) < t.at<double>(0) && t.at<double>(2)
< t.at<double>(1))
{
    T_final = T_final + scale*(R_final * t);
}
R_final = R * R_final;

std::cout << "position" << T_final << std::endl;
std::cout << "Angles" << rotationMatrixToEulerAngles(R_final) <<
std::endl;
std::cout << "Angles 2 " << ExtractRollPitchYaw(R_final) << std::endl;

double x = -double(T_final.at<double>(0));

```

```

double y = double(T_final.at<double>(2));

cv::cvtColor(draw_traj, draw_traj, cv::COLOR_RGBA2BGR);
cv::circle(draw_traj, cv::Point(x+1300, y+1300), 4, cv::Scalar(0, 0,
255, 0), 10);

cv::circle(draw_traj,
cv::Point(ground_truth_points[frame_count].x+1300, -
ground_truth_points[frame_count].y+1300), 4, cv::Scalar(0, 255, 0, 0), 10);
points_text_file1 << x << "," << y << "\n";
cv::cvtColor(draw_traj, draw_traj, cv::COLOR_BGR2RGBA);
points_text_file2 << ground_truth_points[frame_count].x << "," <<
ground_truth_points[frame_count].y << "\n";
texture->TexImage2(draw_traj);

cv::add(frame, mask, draw_frame);
cv::cvtColor(draw_frame, draw_frame, cv::COLOR_BGR2RGBA);

texture->TexImage1(draw_frame);
last_keypoints = good_new;
last_frame = frame_gray.clone();
first_frame = false;

if (last_keypoints.size() < 1000)
{
    DetecFeatures();
}

frame_count++;
}

void VideoPlayback::SetPause()
{
    if (pause) pause = false;
    else pause = true;
}

void VideoPlayback::TakeScreenShot()
{
    cv::imwrite("screenshot.jpg", frame);
}

int VideoPlayback::GetFrameCount()
{
    return frame_count;
}

void VideoPlayback::GetSingleFrame(int frameId)
{
    cap->set(1, frameId);
}

```

```

        frame_count = frameId-1;
        first_frame = true;
        ProcessFrame();
    }

    void VideoPlayback::NextFrame()
    {
        ProcessFrame();
    }

    void VideoPlayback::PreviousFrame()
    {
        frame_count -= 1;
        GetSingleFrame(frame_count);
    }

    void VideoPlayback::DetecFeatures()
    {
        std::vector<cv::Point2f> new_points;

        std::vector<cv::KeyPoint> keypoints_fast;
        int fast_threshold = 20;
        bool nonmaxSuppression = true;
        cv::FAST(last_frame, keypoints_fast, fast_threshold,
nonmaxSuppression);
        cv::KeyPoint::convert(keypoints_fast, new_points, std::vector<int>());

        mask = cv::Mat::zeros(frame.size(), frame.type());
        last_keypoints = new_points;
    }

    cv::Vec3f VideoPlayback::rotationMatrixToEulerAngles(cv::Mat &R)
    {
        float sy = sqrt(R.at<double>(0,0) * R.at<double>(0,0) +
R.at<double>(1,0) * R.at<double>(1,0) );
        bool singular = sy < 1e-6; // If
        float x, y, z;

        if (!singular)
        {
            x = atan2(R.at<double>(2,1) , R.at<double>(2,2));
            y = atan2(-R.at<double>(2,0), sy);
            z = atan2(R.at<double>(1,0), R.at<double>(0,0));
        }

        else
        {
            x = atan2(-R.at<double>(1,2), R.at<double>(1,1));
            y = atan2(-R.at<double>(2,0), sy);
            z = 0;
        }
    }

```

```

    }
    x = x*(180/M_PI);
    y = y*(180/M_PI);
    z = z*(180/M_PI);
    return cv::Vec3f(x, y, z);
}

cv::Vec3d VideoPlayback::ExtractRollPitchYaw(const cv::Mat1d&
in_rotationMat)
{
    cv::Vec3d eulerAngles;
    // Roll angle
    eulerAngles[0] = std::atan2(in_rotationMat(1, 2), in_rotationMat(2, 2));
    const double c2 = std::sqrt(std::pow(in_rotationMat(0, 0), 2) +
std::pow(in_rotationMat(0, 1), 2));
    // Pitch angle
    eulerAngles[1] = std::atan2(-in_rotationMat(0, 2), c2);
    const double s1 = std::sin(eulerAngles[0]);
    const double c1 = std::cos(eulerAngles[0]);
    // Yaw angle
    eulerAngles[2] = std::atan2(s1 * in_rotationMat(2, 0) - c1 *
in_rotationMat(1, 0),
        c1 * in_rotationMat(1, 1) - s1 * in_rotationMat(2, 1));

    return eulerAngles;
}

void VideoPlayback::ToggleTrackLine()
{
    if (show_track_lines)
    {
        show_track_lines = false;
    }
    else
    {
        show_track_lines = true;
    }
}
}

```

Anexo 2 – Interface gráfica

```
#include "MainView.h"
#include "../../include/imgui/imgui.h"
#include "../../include/ImGuiFileDialog/ImGuiFileDialog.h"
#include "../../include/linmath.h"

namespace App
{
    MainView::MainView()
    {
        video = new VideoPlayback;
        video->LoadVideo("../../dataset/sequences/00/image_1/%06d.png");
        std::cout << "loading complete\n" << std::endl;
    }

    MainView::~MainView()
    {
        delete video;
    }

    void MainView::Toolbox()
    {
        bool show_demo_window;
        bool show_another_window;
        ImVec4 clear_color;
        static float f = 1.0f;

        ImGui::Begin("DockSpace Demo", 0, ImGuiWindowFlags_NoTitleBar |
ImGuiWindowFlags_NoCollapse | ImGuiWindowFlags_NoResize |
ImGuiWindowFlags_NoMove | ImGuiWindowFlags_NoBringToFrontOnFocus |
ImGuiWindowFlags_NoNavFocus);
        ImGuiID dockspace_id = ImGui::GetID("MyDockspace");
        ImGui::DockSpace(dockspace_id);
        ImGui::End();

        ImGui::Begin("Configs", 0, ImGuiWindowFlags_NoBringToFrontOnFocus |
ImGuiWindowFlags_NoNavFocus);

        if (ImGui::Checkbox("Show Tracking Lines", &show_tracking_lines))
            video->ToggleTrackLine();

        ImGui::End();

        ImGui::Begin("Preview Trajectory", 0,
ImGuiWindowFlags_NoBringToFrontOnFocus | ImGuiWindowFlags_NoNavFocus);
        ImGui::Image( reinterpret_cast<void*>( static_cast<intptr_t>( video-
>GetFrameTraj() ) ), ImVec2(500, 500) );
        ImGui::End();
    }
}
```

```

ImGui::Begin("Toolbox", 0, ImGuiWindowFlags_NoBringToFrontOnFocus |
ImGuiWindowFlags_NoNavFocus);
    if (ImGui::Button("Open File Dialog"))
        ImGuiFileDialog::Instance()->OpenDialog("ChooseFileDlgKey", "Choose File",
".mp4, .cpp, .h, .hpp, .mov, .MP4", ".");

    if (ImGuiFileDialog::Instance()->Display("ChooseFileDlgKey"))
    {
        if (ImGuiFileDialog::Instance()->IsOk())
        {
            std::string filePathName = ImGuiFileDialog::Instance()-
>GetFilePathName();
            std::string filePath = ImGuiFileDialog::Instance()->GetCurrentPath();
            video->LoadVideo(filePathName);
        }
        ImGuiFileDialog::Instance()->Close();
    }

    if (ImGui::Button("Pause / Play"))
    {
        video->SetPause();
    }

    if (ImGui::Button("Take Screenshot"))
    {
        video->TakeScreenShot();
    }
    ImGui::End();
}

void MainView::VideoPlaybackWindow()
{
    ImGui::Begin("Video");
    ImGui::Text("%.3f ms/frame (%.0f FPS)", 1000.0f / ImGui::GetIO().Framerate,
ImGui::GetIO().Framerate);
    ImGui::SameLine(200);
    ImGui::Text("Frame %i", video->GetFrameCount());
    ImGui::SameLine(300);
    ImGui::PushItemWidth(80);
    ImGui::SetCursorPosY(24.0);
    ImGui::InputInt("##", &orb_counter_input, 0);
    ImGui::SameLine();
    ImGui::SetCursorPosY(24.0);
    if (ImGui::Button("Go"))
    {
        video->GetSingleFrame(orb_counter_input);
    }
    ImGui::SameLine();
    ImGui::SetCursorPosY(24.0);
    if (ImGui::Button("<<"))

```

```
{
    video->PreviousFrame();
}
ImGui::SameLine();
ImGui::SetCursorPosY(24.0);
if (ImGui::Button(">>"))
{
    video->NextFrame();
}
ImGui::Image(reinterpret_cast<void*>(static_cast<intptr_t>(video-
>GetFrame())), ImVec2(1241, 376));
ImGui::End();
}
}
```

Anexo 3 – Ciclo principal da aplicação

```
#include <iostream>
#include "Application.h"
#include <stdlib.h>
#include <stdio.h>
#include <string>
#include "../include/stb_image.h"
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtc/type_ptr.hpp>

namespace App
{
    Application::Application(const char* title)
    {
        m_title = title;
        m_window = new Window;
        m_window->CreateWindow();
    }

    Application::~Application()
    {
        delete m_window;
    }

    void Application::run()
    {
        ImGuiHandler hand;
        m_imgui_handler = std::make_unique<ImGuiHandler>(hand);
        m_imgui_handler->Init(m_window->GetWindow());

        ImVec4 clear_color = ImVec4(0.40f, 0.44f, 0.52f, 1.00f);
        bool app_running = true;
        int display_w, display_h;
        glfwSetKeyCallback(m_window->GetWindow(), key_callback);

        const unsigned int width = 800;
        const unsigned int height = 800;

        glEnable(GL_DEPTH_TEST);

        while (!glfwWindowShouldClose(m_window->GetWindow()))
        {
            glfwGetFramebufferSize(m_window->GetWindow(), &display_w,
&display_h);
            glViewport(0, 0, display_w, display_h);
            glClearColor(clear_color.x, clear_color.y, clear_color.z,
clear_color.w);
            glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

```

        m_imgui_handler->Render();

        glfwSwapBuffers(m_window->GetWindow());
        glfwPollEvents();
    }
    delete m_window;
}

void Application::key_callback(GLFWwindow* window, int key, int scancode,
int action, int mods)
{
    if (key == GLFW_KEY_E && action == GLFW_PRESS)
        std::cout << "A key was pressed" << std::endl;
}
}

```