Universidade de Brasília
Faculdade de Tecnologia

Estimação em tempo real de posição e orientação de câmeras inteligentes

Júlio Eduardo França Dumont
Matheus Ribeiro de Brito Vieira

TRABALHO DE GRADUAÇÃO
ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Brasília
2022

Universidade de Brasília
Faculdade de Tecnologia

Estimação em tempo real de posição e orientação de câmeras inteligentes

Júlio Eduardo França Dumont
Matheus Ribeiro de Brito Vieira

Trabalho de Graduação submetido como requisito parcial para obtenção do grau de Engenheiro de Controle e Automação.

Orientador: Prof. Jones Yudi Mori Alves da Silva

Brasília
2022

Dumont, Júlio Eduardo França.
DD893e Estimação em tempo real de posição e orientação de câmeras inteligentes / Júlio Eduardo França Dumont; Matheus Ribeiro de Brito Vieira; orientador Jones Yudi Mori Alves da Silva. –– Brasília, 2022.
214 p.

Trabalho de Graduação em Engenharia de Controle e Automação –– Universidade de Brasília, 2022.

1. Odometria Visual. 2. Odometria Inercial. 3. Estimação de movimento. I. Vieira, Matheus Ribeiro de Brito. II. da Silva, Jones Yudi Mori Alves, orient. III. Título

Universidade de Brasília
Faculdade de Tecnologia

Estimação em tempo real de posição e orientação de câmeras inteligentes

Júlio Eduardo França Dumont
Matheus Ribeiro de Brito Vieira

Trabalho de Graduação submetido como requisito parcial para obtenção do grau de Engenheiro de Controle e Automação.

Trabalho aprovado. Brasília, 02 de maio de 2022:

Prof. Jones Yudi Mori Alves da Silva,
UnB/FT/ENM
Orientador

Prof. Andre Carmona Hernandes,
DEE/UFSCar
Examinador externo

Prof. Carlos Humberto Llanos Quintero,
UnB/FT/ENM
Examinador interno

Brasília
2022

Dedico este trabalho a mim.

Júlio Eduardo França Dumont

Dedico este trabalho à minha família do Piauí e do Distrito Federal.

Matheus Ribeiro de Brito Vieira

Agradecimentos

Agradeço primeiramente a Deus, que me deu absolutamente tudo que tenho. Ao meu orientador, Jones, que permitiu que esse projeto fosse realizado e me auxiliou durante todo o processo com seus valiosos conselhos. Aos meus pais, Leonardo e Adriana, que me apoiaram durante todas as lutas, à minha família, especialmente a minha tia, Maria, que me deu todo o apoio necessário para realizar o curso. Agradeço também à minha dupla e grande amigo, Matheus Ribeiro, que me acompanhou durante toda a graduação escutando minhas reclamações diárias, e novamente durante esse projeto. Por fim, ao meu professor, Heitor Hardy, que se tornou uma inspiração para que eu me torne alguém melhor a cada dia, e toda Associação Hardy de Karate, que se tornou uma segunda casa para mim, e foi fundamental para que eu não desistisse do curso

Júlio Eduardo França Dumont

Agradeço a Deus, que anima minha alma, dia após dia, enquanto posso viver o hoje, e à minha Mãe, Maria Santíssima, que me guarda e protege. Ao meu orientador, Jones, que me concedeu essa oportunidade e me auxiliou com bastante solicitude. Aos meu pais, Júnior e Mauricélia, à minha família, especialmente a de Brasília, que me deu moradia e todo o apoio necessário para realizar o curso, à minha namorada, Vanessa, aos meus amigos do Piauí e Brasília e, em especial, à minha dupla, Júlio Eduardo, com quem dividi o sofrimento do curso e que me acompanha desde o início da graduação.

Matheus Ribeiro de Brito Vieira

Resumo

Com o advento de novas tecnologias, surge a necessidade de estimar a posição e orientação no espaço de um agente, em condições e locais adversos, como a exploração em solo marciano por *rovers* e o uso de VANTs bélicos em áreas de conflito. Neste cenário, umas das técnicas mais utilizadas é o uso de câmeras e sensores inerciais.

As câmeras fazem parte do sistema de localização baseado em visão computacional, conhecido como odometria visual. Através desse sistema é possível estimar a *pose* do agente através das mudanças visuais decorridas do movimento, como a intensidade dos *pixels* e o rastreamento de *features*. A odometria visual fornece a estimação da trajetória de forma acurada e com baixíssimos erros de posição, além de possibilitar o seu uso em terrenos irregulares e onde o serviço de GPS não é acessível. Todavia, há limitações como necessidade de iluminação no ambiente, cenas com textura e taxa de amostragem limitada.

Por outro lado, sensores inerciais como, giroscópios, acelerômetros, e magnetômetros, geralmente integrantes de uma IMU, possibilitam a estimação da posição e orientação do agente com precisão, através da medição da velocidade angular, aceleração, e intensidade do campo magnético terrestre. Tal processo é conhecido como odometria inercial. Diferentemente das câmeras, a IMU não depende das cenas do ambiente e demais características, além de possuir alta taxa de amostragem. Apesar do processo de estimação através dos sensores inerciais ser bastante preciso, existe um acúmulo de erro bastante significativo, não sendo tão confiável para trajetórias longas. Como ambos os processos possuem características complementares, é possível integrá-los em uma única solução, para obter um processo robusto e preciso de estimação.

Nesse contexto, este projeto consiste em implementar ambos os algoritmos de odometria visual e odometria inercial em uma plataforma híbrida para a estimação da posição e orientação do agente em tempo real. É realizado um amplo estudo dos principais algoritmos, implementação e teste dos algoritmos mais adequados e análise da influência das condições de teste e fatores internos e externos nos resultados. Além disso, é proposto um estudo de fusão de dados, que mescla ambas as vantagens da câmera e da IMU, para estimar a posição e orientação de um agente com acurácia e erros reduzidos, processo conhecido como odometria visual e inercial.

Palavras-chave: Odometria Visual. Odometria Inercial. Estimação de movimento. Visão Computacional. IMU.

Abstract

With the advent of new technologies, arises the necessity of estimating position and orientation of an agent in adverse environmental conditions, such as the exploration of Martian soil by rovers, and the usage of combat drones in warzones. In this case, one of the most common techniques to attain such estimates in the aforementioned environments, is by employing cameras and inertial sensors to derive a robust solution.

Cameras are an integral part of the computer vision-based localization system, known as visual odometry system. This system enables to estimate the agent's pose through the perceived variations in between different image frames captured by the cameras, such as pixel intensity and features. The visual odometry yields an accurate low-error position estimation, in addition to enabling estimation on uneven terrain and GPS denied environments. However, there are limitations, such as good ambient lighting, scenes with texture, and limited sample rate.

On the other hand, inertial sensors such as, gyroscopes, accelerometers, and magnetometers, usually integral parts of an inertial measurement unit, IMU, enable an accurate estimate of the agent's position and orientation, through angular velocity, acceleration, and magnetic field intensity measurements. Such estimation process is known as inertial odometry. Unlike cameras, the IMU operation is not affected by the scene's features and further environmental characteristics, also, it has a relatively high sample rate. Even though the estimation process through inertial sensors is fairly accurate, there's a significant error accumulation in the process, therefore, it is not so reliable for lengthy trajectories. As both systems have complementary features, it's possible to integrate them into a unique solution, to yield a robust and accurate estimation process.

In such context, this project aims to implement both the visual, and inertial odometry processes in a hybrid platform to allow real-time estimation of an agent's orientation and position. A broad study on the most relevant algorithms is carried out, also, the algorithms that were considered the most suitable for the application are implemented and tested, lastly, an analysis on the effects of test conditions, external and internal factors on the results is carried out. Furthermore, a study on data fusion is proposed to enhance the process by merging the visual and inertial odometry algorithms to yield a low-error high-accuracy estimation process known and visual inertial odometry.

Keywords: Visual Odometry. Inertial Odometry. Motion Estimation. Computer Vision. IMU.

Lista de ilustrações

Lista de tabelas

Sumário

1 Introdução

1.1 Contextualização

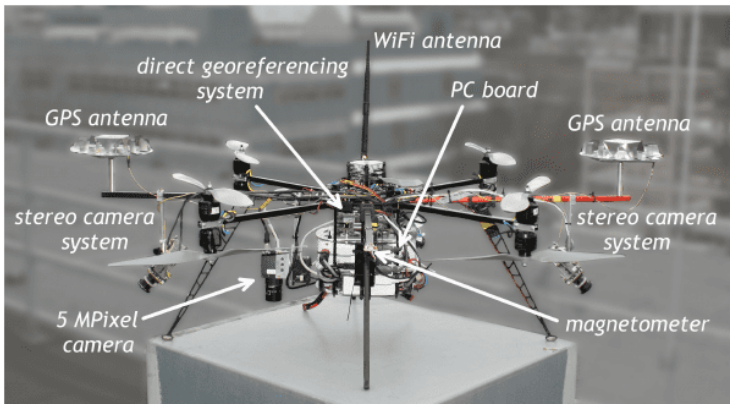
Localizar-se sempre foi de suma importância para a humanidade. Nos primórdios das primeiras civilizações, saber se situar no espaço geográfico era questão de sobrevivência tanto para seres humanos, cujo estilo de vida era nômade, quanto para os animais. O êxodo para determinadas regiões, seja para fugir do frio intenso nas eras glaciais ou em busca de recursos naturais para sobrevivência culminou na necessidade de se localizar.

Uma das primeiras técnicas de localização conhecidas eram as marcações em superfícies rochosas ou por meio de pontos de referência, como grandes rochas ou árvores altas. Outras técnicas primitivas de localização também eram utilizadas como movimento aparente dos astros e constelações referências, como a Estrela Polar, no hemisfério norte, e o Cruzeiro do Sul, no hemisfério Sul. Impulsionados pelo estabelecimento de habitações fixas e o advento das navegações marítimas, instrumentos e técnicas de localização, como a rosa dos ventos, bússolas e mapas, e medição de distâncias, como o astrolábio, foram surgindo.

Atualmente, o instrumento de localização mais utilizado é o Sistema de Posicionamento Global (do inglês, *Global Position System* - GPS), possível graças ao lançamento de satélites ao espaço, que utilizam o princípio da Trilateração para estimar a localização de um ponto na terra em tempo real (UNICAMP, 2020). É válido citar outros sistemas semelhantes ao GPS para navegação, como o Beidou, Galileo e GLONASS. Assim, a necessidade de se orientar e estimar a localização foi desenvolvida de acordo com a necessidade de cada setor, como o automobilístico, de defesa, industrial, etc. Por exemplo, na agroindústria, máquinas agrícolas autônomas são utilizadas para colheita de plantações sob monitoramento de um operador via GPS. Já na robótica, cita-se os *rovers* utilizados na exploração de Marte, que utilizam sensores de orientação e dados de posicionamento via satélite para se deslocar na superfície.

Uma das aplicações de localização bastante estudadas na última década é o voo de Veículo Aéreo Não Tripulado (VANT). Segundo a Agência Nacional de Aviação Comercial (ANAC) (ANAC, 2014), VANT é uma aeronave projetada para operar sem piloto a bordo e controláveis nos três eixos. Um VANT, para obter uma navegação segura e precisa, precisa enfrentar diversas adversidades, como desvio de obstáculos em voos dentro de construções, voos em alta velocidade e baixa altitude, pertubações, como rajadas de vento, identificação de alvos em movimento etc. Para contornar esses problemas, a aeronave precisa estar munida de complexos componentes de orientação e localização, conforme ilustra a Figura 1.

Figura 1 – VANT equipado com diferentes sensores e sistemas



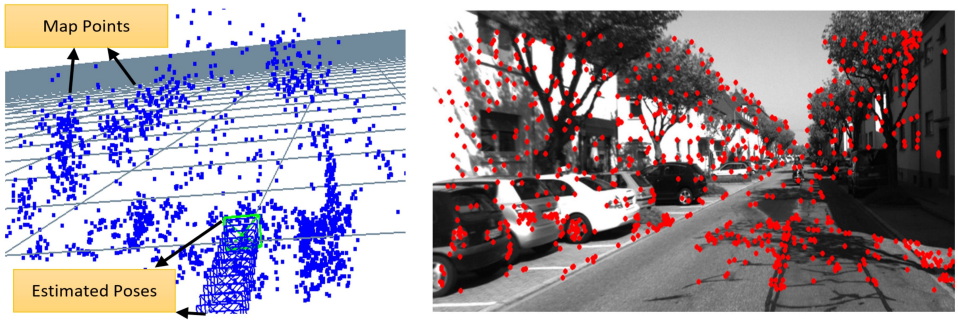
Fonte: Eling, Klingbeil e Kuhlmann (2015)

Uma das únicas tecnologias de localização existentes à época do desenvolvimento dos primeiros VANTs era o radar. Este sensor externo é amplamente utilizado em VANTs autônomos de grande porte. Entretanto, com o surgimento de VANTs reduzidos, as restrições de peso e dimensão se tornaram fatores cruciais, já que os radares são sensores grandes e pesados. Esse cenário culminou no desenvolvimento de sistemas microeletromecânicos (do inglês, *Microelectromechanical systems* - MEMS), assim como sensores de dimensão reduzida. Citam-se os GPSs para navegação baseada em GPS (do inglês, *Global Navigation Satellite System* - GNSS) e a *Inertial Measurement Unit* (IMU) para sistemas de navegação inercial (do inglês, *Inertial Navigation System* - INS).

IMUs são dispositivos eletrônicos que, possuem giroscópio, acelerômetro e, às vezes, magnetômetros em cada um dos seus três eixos principais (WIKIPEDIA, 2021). Através dessa combinação, a IMU é capaz de medir a velocidade angular e a aceleração externa que age sobre um corpo. A IMU não depende das cenas do ambiente em que o agente está localizado, logo, não é afetado por dificuldades relacionadas à visão computacional, a serem citadas posteriormente. É valido citar também que a alta taxa de amostragem da IMU (aproximadamente 1000Hz) é outra vantagem com relação à aquisição de dados. A diversidade de IMUs atende às inúmeras aplicações e requisitos necessários, como formato, tamanho e custo. Apesar do amplo uso, as IMUs não são capazes de atender a todas as modalidade dos VANTs, devido à desvantagem do acúmulo de erro na aquisição de dados, já que a aceleração é integrada em relação ao tempo para calcular a velocidade e a posição. Dessa forma, qualquer erro, por menor que seja, é acumulado ao longo do tempo (WIKIPEDIA, 2021). Dito isso, a escolha do componente e tipo de orientação depende da modalidade do VANT. Por exemplo, para VANT's táticos e estratégicos, o uso GNSS e IMU são os mais adequados, mas para voos perto do chão e com desvio de obstáculos, eles são ineficazes (FÁTIMA BENTO, 2008). Por outro lado, câmeras *Complementary Metal-Oxide Semiconductors* (CMOS) resultam numa estimação de posição precisa relativa a objetos próximos por meio de visão computacional, sendo os mais adequados para esta modalidade.

As câmeras fazem parte do sistema de localização baseados em visão computacional, conhecido como Odometria Visual (do inglês, *Visual Odometry* - VO). VO é o processo de estimar o movimento de um agente, como um VANT, usando somente dados de câmeras acopladas a ele (SCARAMUZZA; FRAUNDORFER, 2011). Basicamente, na odometria visual, *frames* consecutivos são adquiridos por meio das câmeras embarcadas para estimar a *pose* através das mudanças visuais decorridas do movimento do agente. Estas mudanças podem ser processadas por três métodos: métodos baseados nas características (*Feature-based methods*), métodos baseados na aparência (*Appearance-based methods*) e métodos híbridos. No primeiro, utilizam-se características repetidas das imagens, como cantos, que são rastreadas ao longo do movimento, conforme retratado na Figura 2. Já o método baseado na aparência, usufrui da intensidade dos *pixels* em regiões localizadas na imagem. Por sua vez, o método híbrido mescla as vantagens de ambos os métodos anteriores.

Figura 2 – Algoritmo de correspondência entre pontos e features: (a) mapa 3D local e (b) features da imagem 2D detectadas



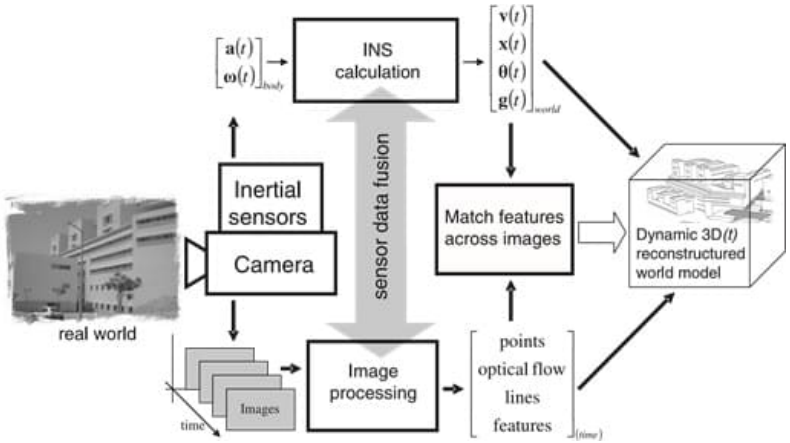
Fonte: Aladem e Rawashdeh (2018)

A estimação do movimento baseada em visão computacional fornece estimações de trajetórias acuradas com erros de posição baixíssimos. Comparada a outros métodos, como *Wheel Odometry*, que utiliza o número de voltas das rodas de um veículo para estimar o movimento, uma das principais vantagens da Odometria Visual é o uso em terrenos irregulares. Além disso, em locais onde o serviço de GPS não é acessível, a VO é crucial. Entretanto, limitações devem ser levadas em conta, como problemas de iluminação no ambiente e cenas estáticas com textura insuficientes.

Ainda, câmeras comuns possuem taxa de amostragem limitada (aproximadamente 100Hz), escalas ambíguas em câmeras de modelo monoculares e não são robustas em movimentos de alta velocidade. Ademais, no contexto da navegação aérea, embora sistemas baseados em visão fornecem dados acurados para estimar a posição relativa de um VANT, tarefas como evitar colisões, controle de atitude, decolagem e pouso são melhores executadas por IMUs. Assim, a necessidade de mesclar as vantagens da IMU e da odometria visual acarretou na Odometria Visual e Inercial (*Visual and Inertial Odometry* - VIO), que é a

fusão de ambos para diminuir os erros de estimação e estimar o movimento do agente com acurácia, conforme mostrado na Figura 3.

Figura 3 – Combinação de sensores visuais e inerciais



Fonte: Corke, Lobo e Dias (2007)

Odometria visual e inercial é o processo de estimar o movimento de um agente, por meio do acoplamento de uma ou mais câmeras com uma IMU, combinando as principais vantagens dos dados inerciais e visuais. A IMU, por não depender da cena no ambiente, é um complemento ideal para as câmeras conseguirem robustez em cenários com baixa textura e movimentos de alta velocidade (SCARAMUZZA; ZHANG, 2019). Por outro lado, o acúmulo de erro na estimação de movimento pela IMU sozinha, ocasionado pelo *sensor bias* em casos de pequenas variações angulares é corrigido pela visão computacional.

Nesse contexto, como o processamento de ambos os dispositivos é realizado simultaneamente, é importante analisar os problemas de tempo real. Primeiramente, como citado anteriormente, a taxa de amostragem da câmera e da IMU são naturalmente diferentes. Os métodos de filtragem, a serem abordados posteriormente, precisam lidar com essa diferença de taxa de amostragem, o que gera obstáculos para operações em tempo real, já que é inviável a definição de um estado para cada aquisição de dados da IMU, cuja taxa de amostragem é de aproximadamente 1000 Hz. Desse modo, uma alternativa é a integração das medidas entre *frames* adjacentes para estimação do movimento. Ainda, o processamento simultâneo de dispositivos com diferenças significativas de taxas de amostragem suscitam na necessidade de mecanismos de sincronia entre tarefas.

Além disso, após cada aquisição de dados dos dispositivos, o processo de análise dos dados e sua otimização devem ser realizados para a estimação do movimento. Desse modo, é importante que o *deadline* relativo entre duas coletas de dados adjacentes em cada dispositivo seja cumprido, isto é, o tempo de resposta dos processos deve estar de acordo com a janela de tempo em questão. Caso o *deadline* não seja cumprido, consequências catastróficas podem suceder, dependendo da aplicação. Por exemplo, em VANTs estratégicos de missão relacionada à intercepção de mísseis, se a velocidade do tempo de resposta não

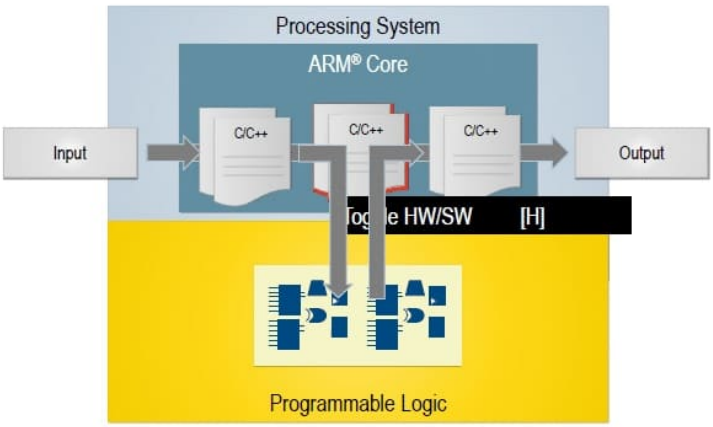
for adequado para a realização de manobras, pode ocorrer do alvo não ser interceptado. Já em um cenário aéreo de desvios de obstáculos, um não cumprimento de um deadline da tarefa pode resultar em um choque entre o VANT e o objeto.

Por conseguinte, a inviabilidade de operações em tempo real em aplicações de alta complexidade é outro obstáculo. Conforme ressaltam Scaramuzza e Zhang (2019), devido à elevada complexidade do algoritmo de otimização de visão computacional, a operação em tempo real rapidamente se torna inviável, conforme a trajetória e o mapa cresce com o decorrer do tempo. Com isso, são práticas usuais a seleção de apenas *keyframes* ou executar o algoritmo de otimização em uma arquitetura de mapeamento paralela.

Nota-se que estimar o movimento de um agente por meio da odometria visual e inercial é um processo não apenas de alta complexidade, mas que exige um rápido processamento. Este processamento de sinais rápido é possível através de circuitos integrados (CI). Um circuito de *hardware* amplamente utilizado por empresas para desenvolvimento de *hardware* é a *Field-Programmable Gate Array* (FPGA), devido à dinamicidade na reprogramação. No contexto da odometria visual e inercial, através da FPGA é possível a aquisição de *frames* das câmeras e de medições da IMU simultaneamente e na taxa de amostragem necessária. Contudo, o complexo processamento necessário para a realização de técnicas de visão computacional, algoritmos de otimização e análise de dados da IMU requerem FPGAs de alta performance, que, consequentemente, possui um alto custo de aquisição. É válido destacar que há FPGAs de diversos tamanhos e para diferentes aplicações, como aeroespacial, automotiva, industrial, médica etc (XILINX, 2017).

Por outro lado, um processador com arquitetura moderna, como a ARM, é uma alternativa para suprir a demanda de desempenho necessário para processamento de algoritmos. A desvantagem dos processadores é o lento processamento de sinais. Nesse cenário, uma abordagem híbrida pode ser utilizada combinando as vantagens de ambos circuitos de *hardware* e processadores, conhecida como plataformas híbridas. Nestas plataformas, um processador é incorporado ao chip do circuito de *hardware*, o que diminui o espaço ocupado pela placa e possibilita a interconexão entre a lógica programável e de processamento, conforme ilustra a Figura 4. O intuito destas plataformas é a distinção do fluxo de projetos tanto do *hardware* quanto do *software*, mantendo-se as ferramentas necessárias para o desenvolvimento de ambos (EMBARCADOS, 2014). Por exemplo, em uma plataforma híbrida ARM-FPGA, é possível a execução de um método de estimação de movimento baseado na aparência, isto é, análise da intensidade de *pixels* em imagens, na área do chip do processador ARM, ao passo que a FPGA é utilizada para estabelecer a comunicação com a IMU via I2C e a câmera, além da aquisição de dados na taxa de amostragem necessária.

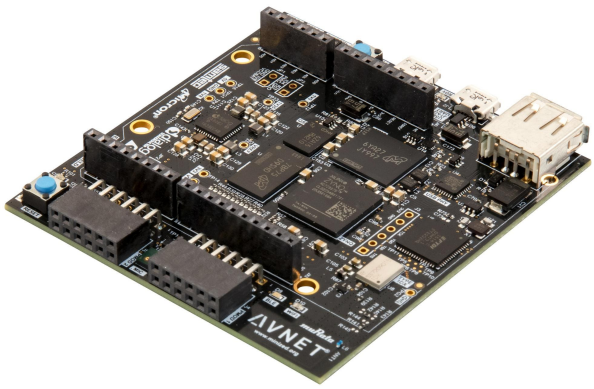
Figura 4 – Operação híbrida com Processador e Lógica Programável



Fonte: Embarcados (2018)

Em contrapartida, implementações de projetos em plataformas híbridas é de alta complexidade, exigindo tempo e esforço para estabelecer um ótimo processamento de *hardware* e *software*. Apesar disso, plataformas híbridas, no caso ARM-FPGA, surge como uma alternativa ideal para o processo de estimação de movimento através da visão computacional e dados inerciais, possibilitando a implementação de um sistema mais dinâmico, com menos subutilização de componentes e menos consumo de energia. Um dos modelos da plataforma híbrida ARM-FPGA disponíveis no mercado é a placa MiniZed (ver Figura 5), desenvolvida pela Avnet com a Xilinx, e que será utilizada neste projeto.

Figura 5 – Placa ARM-FPGA Minized



Fonte: Avnet e Xilinx (2017b)

1.2 Definição do problema

A princípio, este projeto trata da estimação em tempo real de posição e orientação de câmeras inteligentes. Em diversas aplicações de processamento de imagens e visão computacional, há a necessidade de identificar a localização e a posição de uma câmera em relação aos objetos de uma cena, como por exemplo, durante a realização de voo por um

VANT estratégico, conforme descrito na seção 1.1. Uma das formas de fazer isso é utilizando informações coletadas pela própria câmera e/ou sensores a ela acoplados. A proposta deste projeto é a implementação de técnicas para estimação de posição e orientação de uma câmera a partir de sua movimentação em um sistema embarcado. A partir de imagens capturadas e de informações de sensores inerciais, um estimador deverá ser capaz de identificar a pose da câmera em relação a sua posição original. A implementação e testes dos sistemas desenvolvidos devem levar em conta características da plataforma híbrida ARM+FPGA. A parte de visão computacional com a câmera deverá ser implementada em linguagem de alto nível para uso posterior da biblioteca de visão computacional da *Xilinx*. Ressalta-se ainda que, neste projeto, deve-se lidar com os problemas em tempo real, como a sincronia de tarefas distintas e o cumprimento dos *deadlines* de tarefas recorrentes. Ainda, deve-se garantir, através de técnicas de sistemas de medição e algoritmos de otimização, que a estimação do movimento possua certa precisão e acurácia, para que produto possa ser aplicado em um ambiente real com confiabilidade.

1.3 Objetivos do projeto

1.3.1 Objetivo geral

Diante da definição do problema, o projeto tem como objetivo geral a implementação de um sistema embarcado que estime a posição e orientação de câmeras inteligentes, através de imagens capturadas pela própria câmera e pelos dados providos por uma IMU. Ambos os sensores devem ser acoplados ou pelo menos levar em conta características da plataforma híbrida ARM+FPGA, Minized, desenvolvida pela *Avnet* com a *Xilinx*.

1.3.2 Objetivos específicos

- Na etapa de visão computacional, será utilizada uma câmera perspectiva e feita uma análise sobre o método clássico mais adequado, denso ou esparso, para estimação do movimento em tempo real;
- Na etapa de aquisição de dados da IMU, serão utilizados os sensores magnetômetro, acelerômetro e giroscópio para estimar a localização e deslocamento do agente;
- O sistema de odometria visual deve utilizar uma sequência própria de *frames*

1.4 Apresentação do manuscrito

O manuscrito é divido em 5 seções: introdução, fundamentos, implementação do algoritmo de estimação de posição e orientação de câmeras inteligentes, resultados e conclusões.

2 Fundamentos

2.1 Introdução

Este capítulo descreve a fundamentação teórica necessária para a realização do projeto e conceitos relacionados.

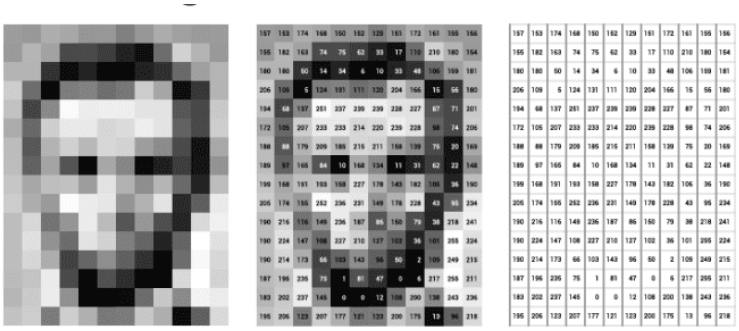
2.2 Visão computacional

Na literatura científica, é possível encontrar várias definições sobre o conceito de visão computacional, já que o campo de aplicação é bastante amplo. Segundo Trucco e Verri (1998), visão computacional pode ser definida como um conjunto de técnicas computacionais que têm como objetivo estimar ou explicitar características geométricas e dinâmicas do mundo físico a partir de imagens digitais obtidas. Já a IBM (2021) afirma que visão computacional é o campo da Inteligência Artificial (IA) que permite computadores e sistemas extraírem informações de interesse de imagens digitais, vídeos e outras entradas visuais, e tomar decisões ou fazer recomendações baseada nesta informação. De uma forma geral, visão computacional está relacionada ao processamento de informações do mundo 3D ou mundo físico a partir de uma ou várias imagens digitais obtidas, ou seja, retirar informações consideradas relevantes para uma determinada aplicação através de imagens obtidas. Existem diversas propriedades que podem ser de interesse para uma aplicação, por exemplo, informações como forma ou posição de um objeto, características dinâmicas associadas como velocidade e uma miríade de outras possibilidades.

2.2.1 Conceitos básicos de óptica

Uma imagem é representada por um vetor 2-D de números inteiros, como mostrado na Figura 6. Tais números, a depender do tipo e da natureza da imagem, podem representar diferentes informações, como por exemplo, intensidades, distâncias e outras características físicas (SHARMA, 2019). Com efeito, a relação da imagem digital com o mundo físico é determinada pelo método de aquisição e quaisquer informações extraídas das imagens é realizada por meio do *array* 2D no qual a imagem é codificada.

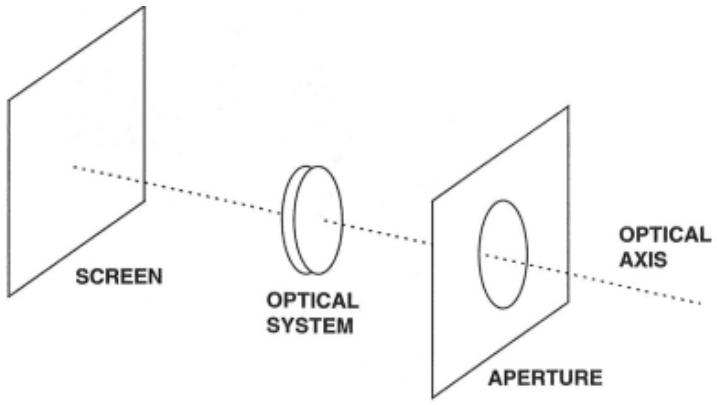
Figura 6 – Representação da intensidade dos pixels em vetor 2D de uma imagem digital



Fonte: Sharma (2019)

Assim como grande parte de sistemas de visão presentes na natureza, o processo de formação de imagens digitais também ocorre por meio de raios de luz que atravessam certa abertura (do inglês, *aperture*), passam por um sistema ótico e atingem um plano de formação de imagem. A Figura 7 apresenta o sistema descrito de forma genérica.

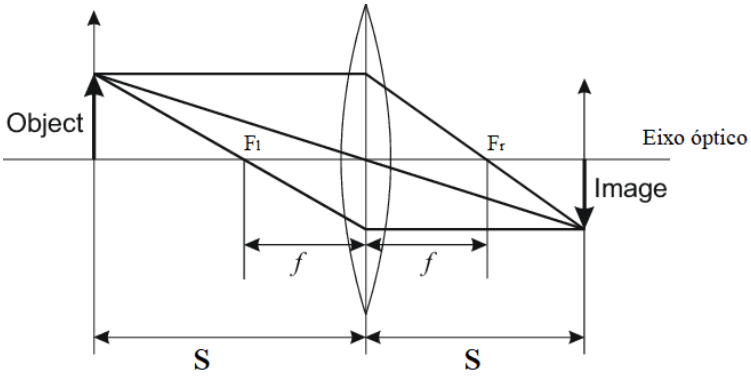
Figura 7 – Elementos básicos da formação de uma imagem



Fonte: Trucco e Verri (1998)

Para compreender ideias básicas da ótica, adota-se o cenário de um sistema óptico simplificado: as lentes finas (do inglês, *thin lenses*), como mostrado na Figura 8. F_l e F_r são os pontos focais, equidistantes do centro da lente fina em um tamanho f. Esta distância, f, é chamada de distância focal, que é a distância do centro da lente ao ponto onde os raios de luz convergem para formar uma imagem nítida.

Figura 8 – Formação da imagem em uma lente fina



Fonte: Qu et al. (2016)

Já o campo de visão (do inglês, *field of view*) da lente fina é uma medida que mensura o ângulo máximo de uma cena, cuja imagem pode ser formada pela câmera. Dado o diâmetro da lente fina, d, o campo de visão, ω é dado por (QU et al., 2016):

$$\omega = \frac{d}{2f} \tag{2.1}$$

Estes raios de luz são provenientes da reflexão de fontes de luz vindas de várias direções por cada ponto da cena, de tal forma que vários raios de luz refletidos pelo mesmo ponto pode entrar pela câmera, desfocando a imagem, caso estes raios não convirjam em um ponto no plano de formação da imagem. Caso eles convirjam, diz-se que a imagem está em foco. Ressalta-se que uma das maneiras de garantir esta convergência é diminuindo a abertura a um ponto, chamado de *pinhole*. Esta mínima abertura é conhecida como *pinhole aperture*. O problema do *pinhole aperture* é que ele permite a passagem de pouquíssimos raios de luz, aumentando o tempo de exposição necessário para a formação da imagem, chamado de *exposure time*. No entanto, o *exposure time* pode ser controlado por por meio de modos de exposição, também chamado de *shutter modes* (TRUCCO; VERRI, 1998).

2.2.1.1 Modos de exposição: *Global Shutter* e *Rolling Shutter*

Há dois modos de exposição presentes nas câmeras: o *Global Shutter* e o *Rolling Shutter*. Basicamente, os modos de exposição descrevem formas distintas de se ler a imagem de um sensor CMOS. No *Rolling Shutter*, o sensor é exposto em um modo progressivo, em que diferentes linhas do *array* são expostas em diferentes tempos do topo para a base. Em uma cena de movimento, como o voo de um helicóptero, o *Rolling Shutter*, causa uma espécie de deformação na imagem, pois os pontos do topo e da base de qualquer *frame* não são capturados simultaneamente. Por outro lado, no *Global Shutter*, o sensor é exposto de uma vez (PAUL, 2016), em que cada *pixel* no sensor começa e termina a exposição simultaneamente, tal qual um sensor CCD, evitando distorções na imagem. Tais diferenças são ilustradas na

Figura 9, em que pode-se observar a deformação das hélices no sentido do movimento na imagem inferior devido ao *Rolling Shutter*.

Figura 9 – Aquisição da imagem com sensores:
(a) *Global Shutter* na imagem de cima e (b) *Rolling Shutter* na imagem de baixo

Fonte: Paul (2016)

No entanto, o *trade-off* é alto custo aquisitivo de um sensor *Global Shutter*, além de um melhor desempenho em ambientes de baixa iluminação e um *range* dinâmico pelo *Rolling Shutter*. Dessa forma, devido aos requisitos do projeto, na implementação utilizou-se uma câmera com sensor *Rolling Shutter*. Para contornar as distorções causadas por este sensor, são utilizadas técnicas como o uso de equipamentos estabilizadores que minimizam *micro-jitters*, mudança de ângulo de captura da imagem etc (KROLL, 2015).

2.2.2 Fluxo óptico

Nos conceitos da subseção 2.2.1, abordou-se exclusivamente sobre imagens únicas. Nesta seção, será analisado o processamento de imagens ao longo do tempo, especificamente, nas mudanças temporais e espaciais que ocorrem em uma sequência de imagens. A princípio, assumindo que a iluminação do ambiente no qual a câmera está localizada não varia, mudanças em imagens subsequentes são causadas por um movimento relativo entre a câmera e a cena. Um das etapas para a análise deste movimento é a correspondência de elementos entre *frames* subsequentes. Como a diferença espacial e temporal entre *frames* é relativamente pequena, a correspondência pode ser tratada como uma estimação do movimento aparente do padrão de brilho da imagem (TRUCCO; VERRI, 1998), isto é, o fluxo óptico, conforme ilustrado na Figura 10.

Figura 10 – Fluxo óptico subsequente de dois *frames*

Fonte: Fan (2015)

O fluxo óptico é uma técnica para detectar movimentos numa sequência de imagens, por meio dos *pixels*. Conceitualmente, ele é definido como o movimento aparente do padrão de brilho de uma imagem. Trata-se de um campo vetorial que indica o sentido do movimento e é calculado através da variação da intensidade do brilho dos *pixels*, de forma que a posição inicial e final do *pixel* que forma o vetor que auxilia na estimação do movimento relativo. O fluxo óptico pode ser realizado por métodos densos ou métodos esparsos. No primeiro caso, o fluxo ótico da imagem em sua totalidade é calculado, de forma que cada pixel da imagem é computado. O método denso utiliza derivada no tempo, logo, uma sequência de imagens relativamente próximas são necessárias. Por outro lado, no método esparso, o fluxo óptico de apenas certas regiões da imagem são computados.

Apesar da simplicidade do algoritmo de fluxo ótico denso, o custo computacional é alto, visto que o fluxo ótico da imagem toda é calculada. Assim, dependendo dos requisitos do *hardware* utilizado, um fluxo óptico denso pode comprometer o processamento em tempo real.

2.2.2.1 Equação de constância do brilho da imagem

Visto que o fluxo óptico trata-se de se estimar o *motion field* de uma sequência de imagens, é necessário obter uma relação entre a variação de brilho e o *motion field*. Para isso, deve-se gerar a equação de constância do brilho da imagem (do inglês, *image brightness constancy equation*). A princípio, assume-se que o brilho da imagem é contínuo e diferenciável no domínio do espaço e do tempo. Dado o plano de uma imagem nas coordernadas x e y ao longo do tempo t. Sabendo que a irradiância da imagem é proporcional à radiância da cena na direção do eixo óptico da câmera (TRUCCO; VERRI, 1998) e considerando que tal fator de proporcionalidade perdura ao longo do plano da imagem, a constância do brilho aparente pode ser representada pelo estacionário do brilho E ao longo do tempo:

$$\frac{dE(x(t),y(t),t)}{dt} = 0. \tag{2.2}$$

A Equação 2.2 pode ser reescrita com derivadas parciais como:

$$\frac{dE(x(t),y(t),t)}{dt} = \frac{\partial E}{\partial x}\frac{dx}{dt} + \frac{\partial E}{\partial y}\frac{dy}{dt} + \frac{\partial E}{\partial t}. \tag{2.3}$$

Sabendo que as derivadas parciais no domínio do espaço são o gradiente espacial da imagem, ∇E, e as derivadas parciais no domínio do tempo são os componentes do *motion field*, $\nu = (u,v)^T$, então, chega-se na equação de constância do brilho da imagem:

$$(\nabla E)^T \nu + E_t = 0. \tag{2.4}$$

A aplicabilidade da Equação 2.4 é visível na estimação do *motion field*, a ser explorado nos tipos de métodos densos.

2.2.3 Sistemas de calibração

Câmeras do tipo *pinhole* introduzem distorções significativas nas imagens. A mais conhecida é a Distorção de Barril (vide Figura 11a), que ocasiona deformação radial e tangencial na imagem (CATTANEO; MAINETTI; SALA, 2015). Na componente radial, linhas retas são distorcidas com característica curvilínea. Quanto mais distante estiver localizado um ponto em relação ao centro da imagem, maior a distorção radial. Pode-se modelá-la da seguinte forma (OPENCV, 2013):

$$\begin{aligned} x_{distorted} &= x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \\ y_{distorted} &= y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \end{aligned} \tag{2.5}$$

em que k_1, k_2 e k_3 são coeficientes de distorção. Já a componente tangencial ocorre devido ao desalinhamento da lente com o plano da imagem, assim, algumas áreas da imagem podem parecer mais próximas do que o esperado. A distorção tangencial pode ser descrito matematicamente como (OPENCV, 2013):

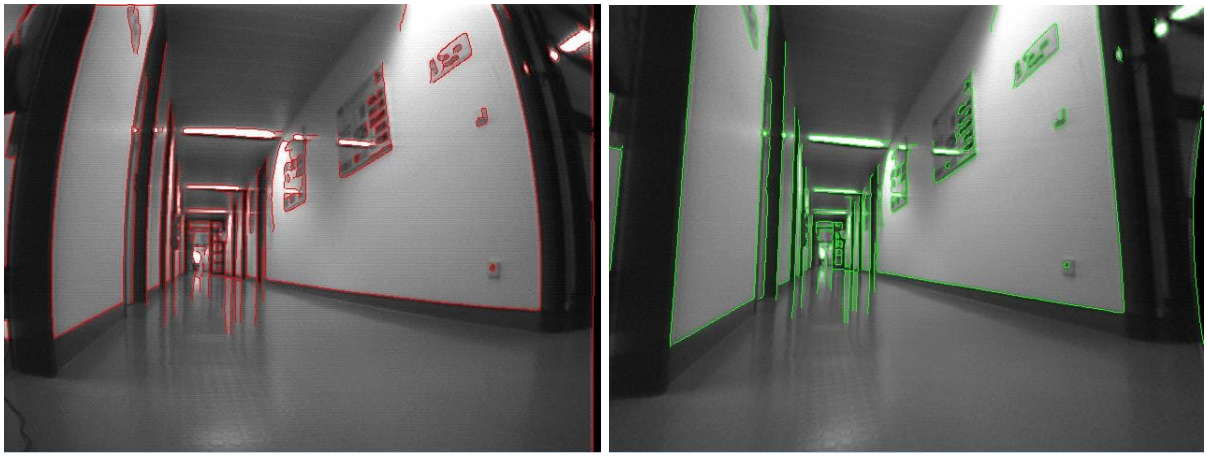
$$\begin{aligned} x_{distorted} &= x + [2p_1 xy + p_2(r^2 + 2x^2)] \\ y_{distorted} &= y + [p_1(r^2 + 2y^2) + 2p_2 xy] \end{aligned} \tag{2.6}$$

em que p_1 e p_2 são, também, coeficientes de distorção, conforme listados na matriz abaixo (OPENCV, 2013):

$$coeficientes\_distorcao = \begin{bmatrix} k_1 & k_2 & p_1 & p_2 & k_3 \end{bmatrix} \tag{2.7}$$

Através da extração destes parâmetros, pode-se corrigir a distorção das imagens (vide Figura 11b) por meio da calibração da câmera.

Figura 11 – Imagem com distorção e sua correção



(a) Imagem com distorção tangencial (b) Mesma imagem corrigida

Fonte: Cattaneo, Mainetti e Sala (2015)

Além dos parâmetros de distorção das lentes, outros parâmetros devem ser extraídos por meio da calibração. A princípio, para obter as medidas de posição e tamanho de objetos no ambiente e obter a equivalência de cada *pixel* na imagem ao tamanho real, são necessárias equações que façam a ligação entre as coordenadas do mundo físico com as coordenadas da imagem digital. Para isso, é preciso conhecer os parâmetros que são intrínsecos e extrínsecos a uma câmera. Conceitualmente, os parâmetros intrínsecos são os parâmetros internos de uma câmera, como a distorção das lentes, ponto principal (c_x e c_y) e distância focal (f_x e f_y) (JO, 2015), que podem ser modelados de forma matricial, formando a matriz de calibração:

$$\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.8}$$

O ponto principal é a coordenada do centro da imagem, enquanto a distância focal fornece o ângulo de visão. Já os parâmetros extrínsecos são aqueles que definem a posição da câmera com relação às coordenadas do objeto. No caso do sistema estéreo, eles também descrevem a relação entre câmeras.

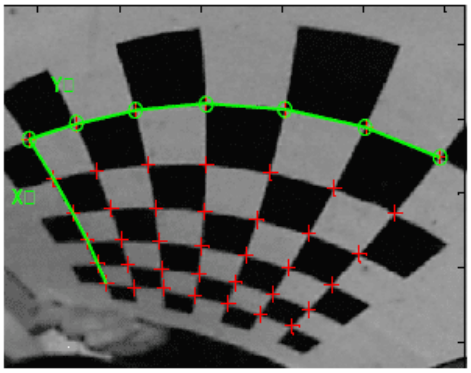
Basicamente, a calibração consiste em obter as equações que descrevem a ligação entre coordenadas conhecidas de um conjunto dos pontos no mundo físico e as suas projeções na imagem digital, e resolver essas equações para os parâmetros da câmera citados. Tais pontos com coordenadas conhecidas podem ser obtidos por meio de padrões de calibração (TRUCCO; VERRI, 1998).

Padrões de calibração são objetos cuja geometria e posição no espaço são conhecidas no sistema de coordenadas. O padrão de calibração mais utilizado é o padrão de Tsai (CABRAL, 2016), que se trata de um plano com desenho de tabuleiro de xadrez (vide Figura 12). Nele, a câmera é posicionada em frente ao tabuleiro com posições e orientações diferentes,

ao passo que ocorre a aquisição de imagens. Assim, é feita a correspondência entre os cantos das imagens e os cantos do padrão em 3D para se obter os valores da Equação 2.8. A solução das equações fornece os parâmetros intrínsecos e extrínsecos da câmera.

Outros métodos clássicos são os de Heikkila e Zhang (JO, 2015). No primeiro, Heikkila e Silven (1997) propõe um processo de calibração baseado em 4 passos: transformação linear das coordenadas do objeto para as coordenadas da câmera; estimação não-linear dos parâmetros da câmera; compensação da distorção causada por *features* circulares e correção das coordenadas da imagem distorcida. Já no segundo, Zhang (2000) propõe um método de calibração que necessita apenas da câmera para observar uma padrão plano mostrado em pelo menos duas diferentes orientações, cujo procedimento proposto consiste em uma solução de forma fechada, seguida de uma otimização não linear baseada no critério da probabilidade máxima.

Figura 12 – Padrão de Tsai ou padrão "tabuleiro de xadrez"



Fonte: Scaramuzza, Martinelli e Siegwart (2006)

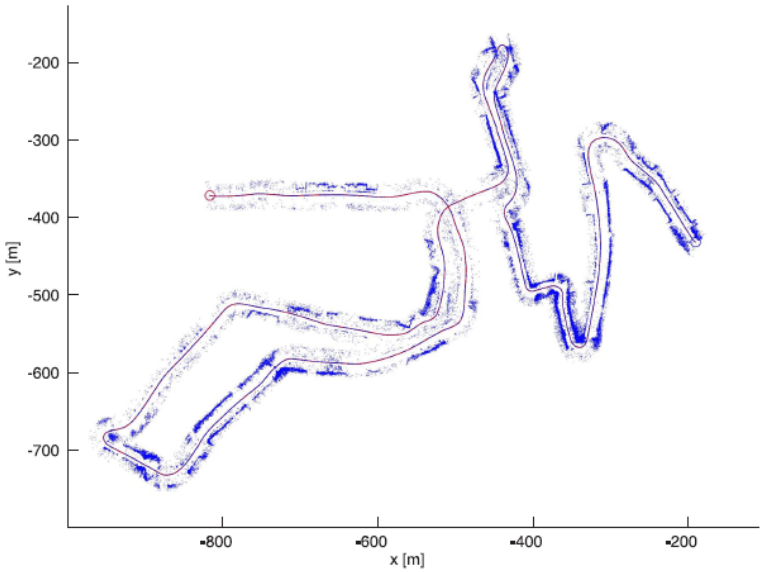
A acurácia da calibração depende das medições realizadas no padrão de calibração, mais especificamente, das tolerâncias na estrutura física. Já a precisão da calibração está relacionada à acurácia da localização dos pontos de referência tanto no mundo físico quanto na imagem digital. O quão acurados devem ser estes pontos depende dos requisitos da aplicação. Projetos de visão computacional de alta precisão, como por exemplo um *scanner* 3D utilizando aquisição de imagens por microscópio, não podem estar suscetíveis a uma calibração com precisão grosseira, pois os erros na estimação dos parâmetros se propagam para os resultados da aplicação. Dessa forma, são os requisitos da aplicação que irá mensurar a precisão da calibração.

2.3 Odometria Visual

A Odometria Visual (do inglês, *Visual Odometry* - VO), é um processo de estimação de informações relativas à posição e orientação de um objeto ou elemento no espaço através do uso de câmeras. Ela é amplamente empregada em veículos, como carros e aeronaves, robôs,

e até mesmo em seres humanos. Por exemplo, a Figura 13 mostra o percurso estimado da trajetória de um carro e sua trajetória real por meio de uma câmera omnidirecional acoplada ao veículo (SCARAMUZZA; FRAUNDORFER; SIEGWART, 2009). As câmeras possibilitam precisão em *slow motion* e informações sobre reconhecimento de lugares (SCARAMUZZA; ZHANG, 2019). Com efeito, uma das principais vantagens da VO em comparação com outros métodos de estimação de movimento, como a *Wheel Odometry*, é que ela pode operar em condições adversas, como terrenos irregulares e sem acesso à GPS, gerando estimativas mais precisas, cujos erros se situam no intervalo de 0,1 a 2% (SCARAMUZZA; FRAUNDORFER, 2011). Por outro lado, dentre as desvantagens da odometria visual, pode-se citar problemas de iluminação no ambiente, cenas estáticas com textura insuficiente, baixa taxa de amostragem da câmera, escalas ambíguas em sistemas de visão monocular e falta de robustez em movimentos de alta velocidade (SCARAMUZZA; ZHANG, 2019).

Figura 13 – Vista superior do mapa 3D gerado e trajetória do veículo por meio da odometria visual



Fonte: Scaramuzza, Fraundorfer e Siegwart (2009)

As etapas de um algoritmo de odometria visual dependem da configuração da câmera, da técnica para estimação de fluxo óptico etc. Em geral, as principais etapas são (SHAN et al., 2016):

1. Obtenção de uma sequência de imagens correspondentes a uma trajetória;
2. Alinhamento de *pixels*/*features*;
3. Estimação do movimento;
4. Ajuste da *pose*;

2.4 Sistemas de visão monocular e stereo

No processo de odometria visual, todos os cenários capturados pela câmera são tridimensionais. Objetos com profundidades diferentes em um cenário real podem se encontrar adjacentes na captura de imagem realizada pela câmera. Na Figura 14, por exemplo, os pontos 1 e 2 são adjacentes, embora correspondam a elementos com profundidades diferentes.

Figura 14 – A motocicleta no primeiro plano está mais perto da câmera do que a caixa vermelha na estante

Fonte: adaptado de Instruments (2020)

A visão humana consegue facilmente detectar uma diferença de profundidade entre objetos em uma cena ou mesmo a distância do olho a determinados objetos. Para uma câmera, no entanto, a análise de perspectiva não é trivial. Neste cenário, destacam-se dois tipos de sistemas de visão: monocular, que utiliza uma única câmera para processamento visual e o stereo, que utiliza duas câmeras separadas uma da outra.

2.4.1 Sistema monocular

O sistema monocular utiliza apenas uma câmera, capturando um *frame* por vez. A medida da distância da câmera a um objeto através de dados 2D pode ser feito pela relação proximidade-tamanho do sistema ou pelo *Structure-from-motion* (SFM).

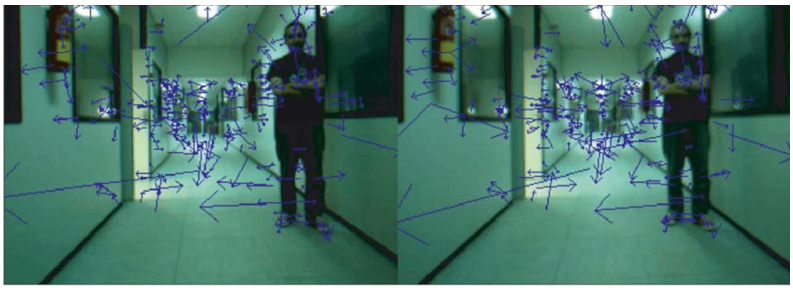
O SFM é um conceito geral e engloba a odometria visual. Nele, *frames* consecutivos são capturados e comparados um com os outros para relacionar *features* ou *pixels* em comum. Com o uso da geometria epipolar, é possível definir parâmetros restritivos para analisar o movimento de um ponto no espaço 3D em *frames* consecutivos (INSTRUMENTS, 2020).

Por fim, suposições de ruídos na imagem, como a constância do brilho, são mais suscetíveis de serem violados em um sistema monocular que no stereo, o que gera uma necessidade de recálculos e mais restrições para compensar os ruídos.

2.4.2 Sistema stereo

No sistema de visão stereo, a reconstrução da informação tridimensional de objetos é realizada a partir de duas câmeras que capturam imagens simultaneamente, mas com um pequeno deslocamento lateral, ilustrado na Figura 15. Assim como na visão humana, um par de imagens deslocadas lateralmente possui diferenças quase que imperceptíveis quando observadas separadamente, porém, elas que resultam em uma percepção tridimensional do mundo externo.

Figura 15 – Imagens esquerda e direita de um sistema stereo



Fonte: adaptado de Núñez, Vázquez-Martín e Bandera (2011)

Conceitualmente, as duas câmeras são posicionadas com direção e distância conhecidas uma em relação a outra. Dessa forma, é possível determinar a posição de um ponto no espaço, caso ele esteja presenta em ambas as imagens capturadas simultaneamente, através da disparidade (OTUYAMA, 1998). De forma simplificada, a disparidade é definida como o número de *pixels* que se moveram na imagem direita em comparação com a esquerda, dado um ponto referência (INSTRUMENTS, 2020). A disparidade é inversamente proporcional à distância de um objeto, de modo que objetos que estão mais distantes ao local de captura das imagens possuem uma disparidade maior e, por outro lado, objetos que estão mais próximos do local de captura das imagens possuem uma disparidade menor (OTUYAMA, 1998).

2.5 Estimação do *motion field*

A entrada para análise de uma cena dinâmica é uma sequência de *frames* capturados do mundo real. Assumindo que a iluminação da cena não muda, as mudanças na imagem são devidas ao movimento relativo entre os objetos da cena e a câmera. Nesse contexto, o *motion field* é uma representação 2D no plano da imagem de um movimento tridimensional de pontos na cena (TRUCCO; VERRI, 1998). Para cada ponto é designado um vetor velocidade correspondente à direção do movimento e a sua magnitude. A estimação do *motion field*

pode ser agrupada em duas conhecidas áreas: métodos densos e métodos esparsos, embora haja outras, como técnicas baseadas na fase e baseadas na energia (BARRON; FLEET; BEAUCHEMIN, 1994). Ainda, segundo Scaramuzza e Fraundorfer (2011), os métodos densos não são tão acurados e rápidos quanto os métodos esparsos e requerem um custo computacional mais alto, no entanto, são mais fáceis de se implementar.

2.5.1 Métodos densos

Conforme descrito na subseção 2.2.2.1, a suposição mais básica feita no fluxo óptico é a constância do brilho da imagem, que é matematicamente formulada pela Equação 2.4, chamada de equação de constância do brilho da imagem. Nesta equação, o problema converge para a solução de ν, que será a estimação do fluxo óptico (DELIGIANNIDIS; ARABNIA, 2014). Como há duas variáveis desconhecidas para uma única equação linear, restrições são necessárias para resolver ambos os componentes de ν. Dentre os vários algoritmos possíveis, destacam-se os seguintes para a solução de ν (TRUCCO; VERRI, 1998):

- Método Horn and Schunck: resolução de um sistema de equações de derivadas parciais por iterações.
- Método Nagel: cálculo da derivada de segunda ou maior ordem do brilho da imagem.
- Método Lucas-Kanade: estimativa dos quadrados mínimos dos parâmetros que caracterizam o fluxo óptico;

Dentre os métodos, o mais utilizado na literatura científica é o Lucas-Kanade, e, portanto, é válido descrevê-lo de forma mais aprofundada.

2.5.1.1 Método Lucas-Kanade

Um dos métodos mais populares para cálculo do *motion field* é o Lucas-Kanade, apresentado originalmente por Lucas e Kanade (1981). Ele envolve computar o *motion field* assumindo que o vetor será similar a uma pequena vizinhança do *pixel* (DELIGIANNIDIS; ARABNIA, 2014). Para isso, é utilizado um método de quadrados mínimos ponderados para aproximar o fluxo óptico em cada pequena vizinhança Ω do *pixel* p = (x,y), minimizando o erro (BARRON; FLEET; BEAUCHEMIN, 1994):

$$\varepsilon = \sum_{p \in \Omega} W^2(p)[\nabla E(p)\nu + E_t(p)]^2 \tag{2.9}$$

em que W(p) é o peso associado à vizinhança do *pixel*. Os pesos são utilizados para diminuir a importância de vizinhanças distantes (LUCAS; KANADE, 1981). Quanto mais distante o *pixel* vizinho está do *pixel* referência, menor será o peso associado. Dessa forma,

a influência dos *pixels* mais distantes é reduzida. A solução da Equação 2.9 é dada por (BARRON; FLEET; BEAUCHEMIN, 1994):

$$A^T W^2 A v = A^T W^2 b \tag{2.10}$$

em que, para n *pixels* $p_i \in \Omega$ no tempo t, A é um vetor dos gradientes espaciais de todos os n vizinhos da vizinhança Ω, W são os pesos para cada vizinho e b é um vetor dos gradientes temporais, dados por (BARRON; FLEET; BEAUCHEMIN, 1994):

$$\begin{aligned} A &= [\nabla E(p_1), ..., \nabla E(p_n)]^T, \\ W &= diag[\nabla W(p_1), ..., \nabla W(p_n)], \\ b &= -[\nabla E_t(p_1), ..., \nabla E_t(p_n)]^T \end{aligned} \tag{2.11}$$

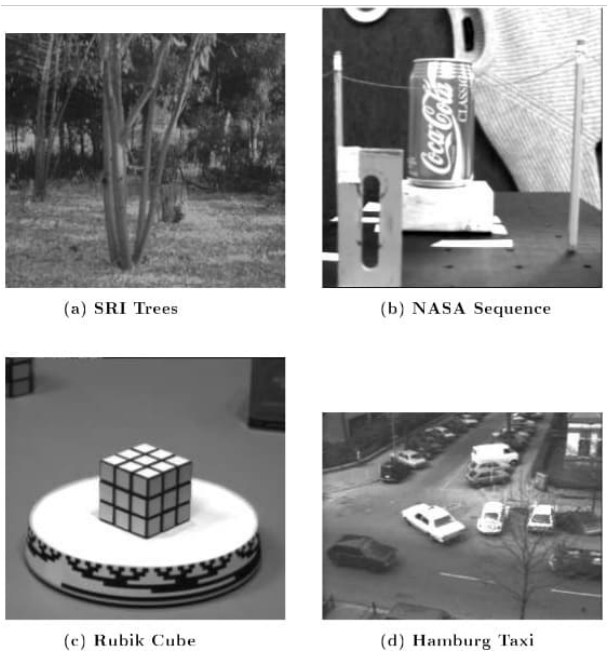
A solução da Equação 2.11 é dada por (BARRON; FLEET; BEAUCHEMIN, 1994):

$$v = [A^T W^2 A]^{-1} A^T W^2 b \tag{2.12}$$

que é resolvida de forma fechada quando $A^T W^2 A$ é não-singular.

Para comparar resultados entre os métodos densos, Barron, Fleet e Beauchemin (1994) utilizam quatro sequências de imagens reais, conforme ilustra a Figura 16. Na sequência SRI, a câmera translada paralelamente ao chão plano, perpendicular a sua linha de visão. Na sequência NASA, a câmera se move junto com sua linha de visão em direção à lata de Coca-Cola. Na sequência Rubik Cube, o cubo está girando no sentido anti-horário na mesa giratória. Por fim, na sequência Hamburg Taxi, há quatro carros em movimento.

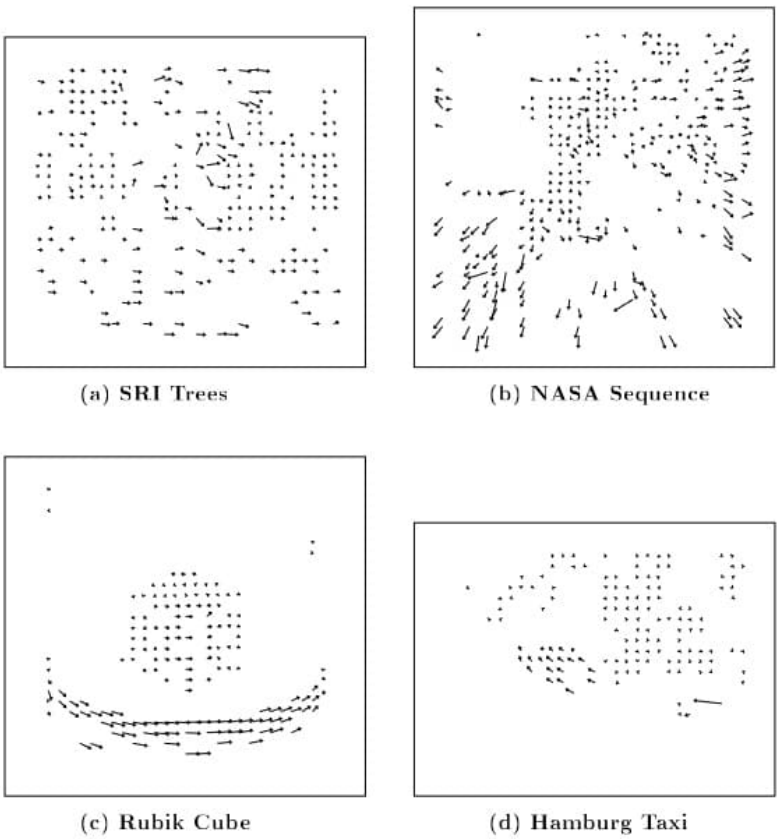
Figura 16 – Quatro sequências de imagens reais



Fonte: Barron, Fleet e Beauchemin (1994)

A Figura 17 ilustra o resultado das sequências de imagens com o método Lucas-Kanade.

Figura 17 – *Motion field* das sequências com o uso da técnica Lucas-Kanade



Fonte: Barron, Fleet e Beauchemin (1994)

Uma das vantagens do método Lucas-Kanade é o fato de não envolver cálculos de derivadas de segunda ordem ou maior, logo, são menos sensíveis a ruídos que métodos com derivadas maior que primeira ordem, como o método Nagel (TRUCCO; VERRI, 1998).

2.5.2 Métodos esparsos

Métodos esparsos são método de estimação do *motion field* baseado em salientes e repetidos *features* que são localizados ao longo dos *frames* (SCARAMUZZA; FRAUNDORFER, 2011). Esta abordagem é uma alternativa aos métodos densos, pois são mais adequadas em casos que a diferenciação e métodos numéricos não são eficazes devido ao ruído ou pequena quantidade de *frames* (BARRON; FLEET; BEAUCHEMIN, 1994). O resultado, diferentemente dos métodos densos, é um *motion field* esparso (TRUCCO; VERRI, 1998). Os passos principais para realização da Odometria Visual baseada em *features* pode ser dada por (SCARAMUZZA; FRAUNDORFER, 2011):

1. Sequência de imagens

2. Detecção de *features*
3. *Matching* (ou *tracking*) de *features*
4. Estimação do movimento (*2D-to-2D*, *3D-to-3D* ou *3D-to-2D*)

Pode-se perceber que após a aquisição de *frames*, isto é, obtida a sequência de imagens, é necessário detectar as *features* da sequência. Há duas maneiras de encontrar *features* e suas correspondências: *feature tracking* e *feature matching* (SCARAMUZZA; FRAUNDORFER, 2011). A primeira consiste em encontrar *features* em uma imagem e rastreá-las nas imagens seguintes, utilizando técnicas de buscas local, como a correlação (SCARAMUZZA; FRAUNDORFER, 2012). Esta abordagem é mais adequada quando as imagens são capturadas de *viewpoints* próximos. A segunda consiste em detectar *features* em todas as imagens de uma vez e fazer a correspondência (*match*) em seguida, baseada na similaridade, que é mais adequada quando é esperado muito movimento ou mudança de *viewpoint*.

2.5.3 Comparação entre métodos de estimação do fluxo óptico

Conforme descrito na subseção 2.2.2, o fluxo óptico pode ser estimado através de métodos densos ou esparsos. Nos métodos densos, todo o fluxo óptico da imagem é calculado, conforme mostrado na Figura 18a. Já nos métodos esparsos, é calculado o fluxo óptico para apenas certas regiões da imagem, como ilustrado na Figura 18b.

Figura 18 – Exemplos de fluxo óptico denso e esparso

(a) Fluxo óptico denso utilizando o algoritmo Gunnar-Farnerback

(b) Fluxo óptico esparso utilizando o detector de cantos Shi-Tomasi e rastreador de features iterativo Lucas-Kanade

Fonte: Produzido pelos autores

Pode-se visualizar pelas figuras que nos métodos densos, o fluxo óptico de toda a imagem é calculada. Dessa forma, embora simples de implementá-los, possuem custo computacional alto. Já nos métodos esparsos, percebe-se que o fluxo óptico de apenas certas regiões da imagem são computados, tornando baixo o custo computacional. Nesse cenário, Zhang, Han et al. (2019) realizaram um experimento no qual foi calculado o tempo médio de execução para algoritmos de métodos densos e esparsos em uma mesma CPU. As Figuras 19a e 19b ilustram o resultado do experimento.

Figura 19 – Tempo médio para algoritmos densos e esparsos

Method	Noise-Free Images	Noisy Images
BRISK	0.2847	0.2832
KAZE	0.1348	0.1304
SURF	0.0431	0.0437
RANRESAC	–	6.2648
Proposed (Demp)	0.3934	0.3933
Proposed (PCR6)	0.3938	0.3989

(a) Comparação de tempo médio de execução para algoritmos esparsos

Method	Noise-Free Images	Noisy Images
MI	16.7622	16.4789
PSNR	12.8583	13.4214
NCC	14.7187	15.1050
Proposed (Demp)	17.0734	16.7945.
Proposed (PCR6)	17.0812	16.8729

(b) Comparação de tempo médio de execução para algoritmos densos

Fonte: Zhang, Han et al. (2019)

Nota-se que o custo computacional para algoritmos densos é bastante alto, comparado aos algoritmos esparsos. Dessa forma, para uma aplicação em tempo real, o uso de algoritmos esparsos são mais adequados, visto que o algoritmo denso pode comprometer os requisitos do sistema.

Já no estudo realizado por Barron, Fleet e Beauchemin (1994), foi conduzido um experimento de performance de algoritmos de estimação de fluxo óptico por métodos densos e esparsos para várias sequências. As Figuras 20a e 20b destacam os resultados da estatística dos erros para cada algoritmo.

Figura 20 – Erros estatísticos para componente velocidade de sequências

Technique	Average Error	Standard Deviation	Density
Horn and Schunck (original)	12.02°	11.72°	100%
Horn and Schunck (original) $\|\|\nabla I\|\| \geq 5.0$	8.93°	7.79°	59.8%
Horn and Schunck (modified)	2.55°	3.67°	100%
Horn and Schunck (modified) $\|\|\nabla I\|\| \geq 5.0$	2.50°	3.89°	32.9%
Lucas and Kanade ($\lambda_2 \geq 1.0$)	1.94°	2.06°	48.2%
Lucas and Kanade ($\lambda_2 \geq 5.0$)	1.65°	1.48°	24.3%
Uras et al. (unthresholded)	4.64°	3.48°	100%
Uras et al. ($det(H) \geq 1.0$)	3.83°	2.19°	60.2%
Nagel	2.94°	3.23°	100.0%
Nagel $\|\|\nabla I\|\|_2 \geq 5.0$	3.21°	3.43°	53.5%

(a) Erros estatísticos da componente velocidade da sequência *Diverging tree*

Technique	Average Error	Standard Deviation	Density
Horn and Schunck (original)	32.43°	30.28°	100%
Horn and Schunck (original) $\|\|\nabla I\|\| \geq 5.0$	25.41°	28.14°	59.6%
Horn and Schunck (modified)	11.26°	16.41°	100%
Horn and Schunck (modified) $\|\|\nabla I\|\| \geq 5.0$	5.48°	10.41°	32.9%
Lucas and Kanade ($\lambda_2 \geq 1.0$)	4.10°	9.58°	35.1%
Lucas and Kanade ($\lambda_2 \geq 5.0$)	3.05°	7.31°	8.7%
Uras et al. (unthresholded)	10.44°	15.00°	100%
Uras et al. ($det(H) \geq 1.0$)	6.73°	16.01°	14.7%
Nagel	11.71°	10.59°	100%
Nagel $\|\|\nabla I\|\|_2 \geq 5.0$	6.03°	11.04°	32.9%

(b) Erros estatísticos da componente velocidade da sequência *Yosemite*

Fonte: adaptado de Zhang, Han et al. (2019)

Nota-se que o algoritmo Lucas-Kanade foi o que apresentou o menor erro médio e menor desvio padrão para ambas as sequências.

2.5.4 Detecção de *features*

Na etapa de detecção de *features*, é realizado um mapeamento na imagem, buscando por pontos salientes prováveis de haver correspondência com pontos de outras imagens. Basicamente, *feature* é a parte de uma imagem, que é local e detectável, com algumas propriedades especiais, como um círculo, uma linha ou uma região texturizada (TRUCCO; VERRI, 1998). Um detector de *features*, para ser aplicado em um sistema de odometria

visual, deve ser capaz de localizar *features*, mesmo em ambiente de desordem e oclusão; deve possuir repetibilidade, já que a mesma *feature* deve ser localizada em inúmeras imagens e eficiência de computação (LAZEBNIK et al., 2012). Para a odometria visual, *features* como cantos (ponto de intersecção entre duas ou mais bordas) e bolhas (padrão que difere de sua vizinhança imediata em termos de intensidade, cor e textura) são importantes, visto que suas posições na imagem podem ser medidas com acurácia (SCARAMUZZA; FRAUNDORFER, 2012). Comparados aos detectores de canto, os detectores de bolha são mais distintivos, porém, são mais lentos na detecção (SCARAMUZZA; FRAUNDORFER, 2012), isto é, possui um custo computacional maior, logo, não é interessante para uma aplicação em tempo real.

2.5.5 Detector de cantos

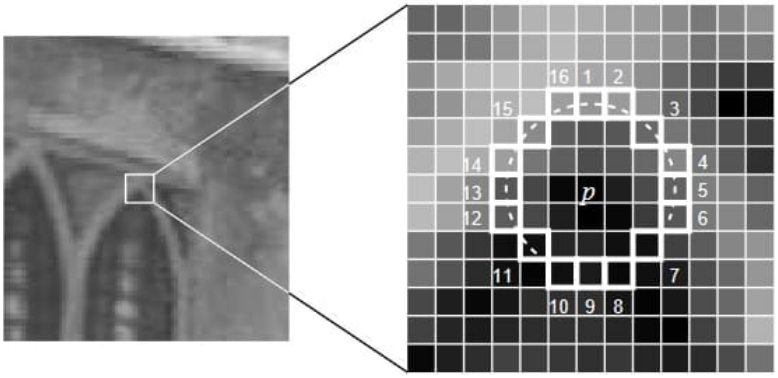
Um dos primeiros detectores de cantos foi implementado por Moravec, que utilizou a *Sum of Squared Differences* (SSD) como uma medida de similaridade entre dois locais, em que se o número é localmente máximo, então um canto é detectado (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011). Já o detector de cantos Harris é uma melhoria do detector de Moravec, pois considera as derivadas parciais da SSD em vez de usar janelas deslocadas (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011). Outros detectores são detector de cantos Shi-Tomasi, detector de cantos SUSAN e detector de cantos FAST (SCARAMUZZA; FRAUNDORFER, 2012). Conforme descrito por Siegwart, Nourbakhsh e Scaramuzza (2011), o detector de Harris é uma ferramenta conveniente para extrair uma grande quantidade de cantos e é considerado o detector mais estável. Da mesma forma, são estáveis os detectores SUSAN e FAST, sendo mais eficientes, no entanto, são mais sensíveis a ruídos. Dentre os detectores, merece destaque e maior detalhamento o detector FAST, devido ao seu altíssimo desempenho computacional, que será melhor elucidado na subseção 2.5.6.

2.5.5.1 *Features from Accelerated Segment Test* (FAST)

O detector de cantos FAST foi originalmente proposto por Edward Rosten e Tom Drummond (2006) e é conhecido pela performance em alta velocidade. Basicamente, busca-se, em toda a imagem, por *pixels* que representem cantos, conhecidos como pontos de interesse, *p*. Conforme descrevem Edward Rosten e Tom Drummond (2006), considera-se um círculo de 16 *pixels* ao redor do candidato a canto, *p*. *p* é considerado um canto se existe um conjunto de *n pixels* contíguos no círculo que são todos mais brilhantes que a intensidade de *p*, I_p, mais um limite *t*; ou todos mais escuros que I_p - *t*, conforme ilustrado na Figura 21. Originalmente, *n* foi fixado em 12. A fim de obter um teste em alta velocidade, verifica-se primeiro os *pixels* dos pontos cardeais, 1, 5, 9 e 13. Se pelo menos 3 destes *pixels* forem mais brilhantes que $I_p + t$ ou mais escuros que I_p - *t*, *p* pode ser um canto. Então, checa-se todos os 16 *pixels* e se 12 contíguos se enquadrarem no critério, *p* é um canto. No entanto, se nenhum dos *pixels* dos pontos cardeais satisfazerem estas equações, *p* não pode ser um canto.

Repete-se o procedimento para todos os *pixels* da imagem.

Figura 21 – Teste de detecção de canto com segmento de 12 pontos em uma imagem



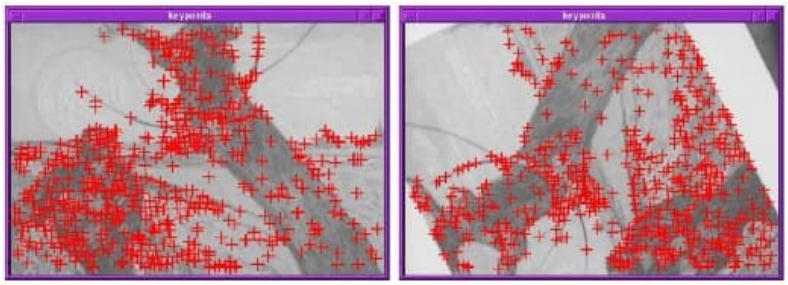
Fonte: Edward Rosten e Tom Drummond (2006)

Os autores acrescentam que o algoritmo apresenta algumas limitações (ROSTEN, Edward; DRUMMOND, Tom, 2006), como a queda de desempenho de velocidade para $n < 12$ e a ordem em que os 16 *pixels* são analisados determina a velocidade do algoritmo. Para lidar com estes problemas, faz-se uso de técnicas de *machine learning*. Além disso, outro problema é a detecção de múltiplos pontos de interesse em *pixels* adjacentes. Para lidar com isso, utiliza-se supressão não-máxima. Basicamente, uma função de pontuação (do inglês, *score function*), V, é computada para cada canto detectado, em que V é a soma da diferença absoluta entre p e os 16 valores de *pixels* ao redor. Dado dois pontos de interesses adjacentes, o que possuir V menor é removido.

O detector FAST apresenta algumas desvantagens como a não-robustez para níveis altos de ruído e a dependência de um limite. Porém, sua performance em alta velocidade destaca-o entre os detectores de canto, sendo um candidato excelente para aplicações com odometria visual ou SLAM.

A Figura 22 ilustra a atuação de um detector de cantos em uma imagem.

Figura 22 – Detector de cantos em uma imagem



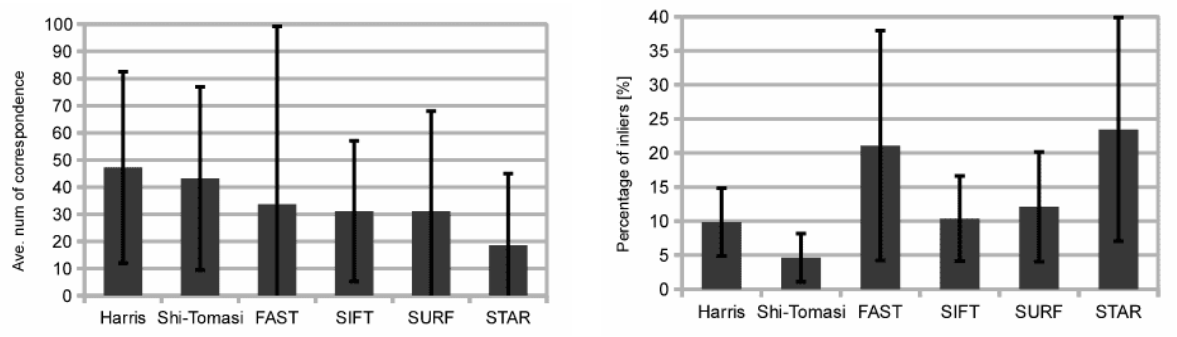
Fonte: Lazebnik et al. (2012)

2.5.6 Comparação entre detectores de *features*

A escolha do detector de *features* apropriado deve levar em conta fatores como restrições computacionais, requisitos de tempo real, tipo de ambiente etc (SCARAMUZZA; FRAUNDORFER, 2012). Um detector de bolhas é um detector robusto, usados principalmente em aplicações com mudança de escala. No entanto, seu alto custo computacional é um fator limitante para projetos com *hardwares* de baixa capacidade computacional. Já os detectores de canto possuem alta velocidade e acurácia de localização de *features* e, portanto, geralmente são os mais utilizados na odometria visual, como Harris e FAST (OTSU et al., 2013). Porém, em ambientes com baixa textura, cantos são difíceis de serem localizados.

Um teste de performance para estes detectores foi conduzido por Otsu et al. (2013), usando um *dataset* com mais de 900 pares de imagens estéreas em uma área vulcânica no Japão. Os resultados estatísticos são mostrados nas Figuras 23a e 23b.

Figura 23 – Média de correspondência e porcentagem de *matches* de features



(a) Média de correspondência entre *features*

(b) Porcentagem de *matches* corretos entre todos os *features* extraídos

Fonte: Otsu et al. (2013)

Geralmente, para uma implementação de odometria visual, mais de 20-30 *matches* corretos são suficientes para estimar a trajetória de uma câmera (OTSU et al., 2013). Pode-se perceber na Figura 23a que o detector Harris apresentou a melhor performance na correspondência entre *features*. No entanto, Harris, Shi-Tomasi e detectores similares não apresentam uma alta taxa de *matching* entre *features*, conforme ilustrado na Figura 23b, comprometendo a acurácia e eficiência do resultado final.

Quando se compara os algoritmos em relação ao tempo de execução, como ilustrado na Figura 24, os detectores de cantos são significativamente mais rápidos que os de bolha, sendo o FAST o detector mais eficiente de todos. Embora o FAST não seja robusto para todo tipo de terreno, sua detecção é rápida, com repetibilidade e distinção altas (OTSU et al., 2013).

Figura 24 – Tempo médio de execução por *frame* (Imagens 320 × 240 em grayscale em um Intel Core 2 Quad 2667MHz CPU)

Detectors	Harris	Shi-Tomasi	FAST	SIFT	SURF	STAR
Ave. runtime [ms]	11.87	14.36	1.32	54.98	27.90	9.86

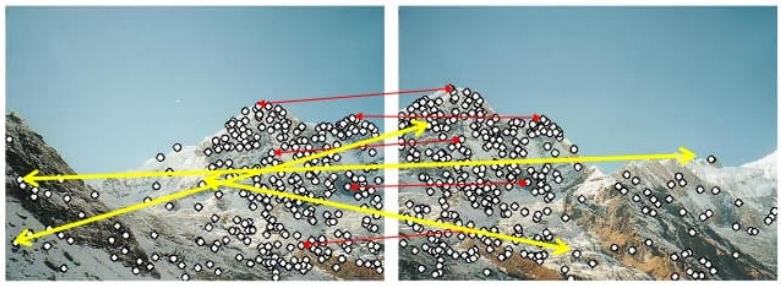
Fonte: Otsu et al. (2013)

Detectadas as *features*, é realizada a etapa da descrição das *features*, em que a região ao redor de cada *feature* detectada é convertida em um descritor compacto, para ser feita a posterior correspondência com outros descritores (SCARAMUZZA; FRAUNDORFER, 2012).

2.5.7 *Feature matching (tracking)*

Após as etapas de detecção e descrição das *features* da sequência de imagens, é realiza a correspondência entre elas, por meio da *feature tracking* ou *feature matching*. A Figura 25 ilustra a correspondência entre *features* de imagens distintas.

Figura 25 – Correspondência entre *features* de imagens distintas



Fonte: Lazebnik et al. (2012)

A correspondência entre *features* pode ser feita por meio do *feature matching* e do *feature tracking*. No primeiro caso, procura-se por *features* em todas as imagens de uma vez e então, o processo da correspondência é realizado, baseado na similaridade. Um dos algoritmos mais conhecidos é o *Scale Invariant Feature Transform* (SIFT). Basicamente, os descritores são comparados entre si, utilizando uma medida de similaridade (SCARAMUZZA; FRAUNDORFER, 2012), de forma que, dado um *feature* de uma imagem A, a melhor correspondência deste *feature* com os *features* da imagem B é escolhido como o descritor mais similar. No entanto, pode acontecer de haver mais de uma correspondência de um *feature* na imagem B com vários *features* na imagem A. Neste caso, para decidir qual correspondência aceitar, deve-se parear cada *feature* na imagem B com *features* na imagem A, de tal maneira que apenas pares de *features* correspondentes que mutuamente tem um ao outro são aceitos (SCARAMUZZA; FRAUNDORFER, 2012).

Já no caso do *feature tracking*, *features* de um único *frame* de cada vez são detectados para buscar por pares correspondentes nos demais *frames*. Conforme descrevem Scaramuzza

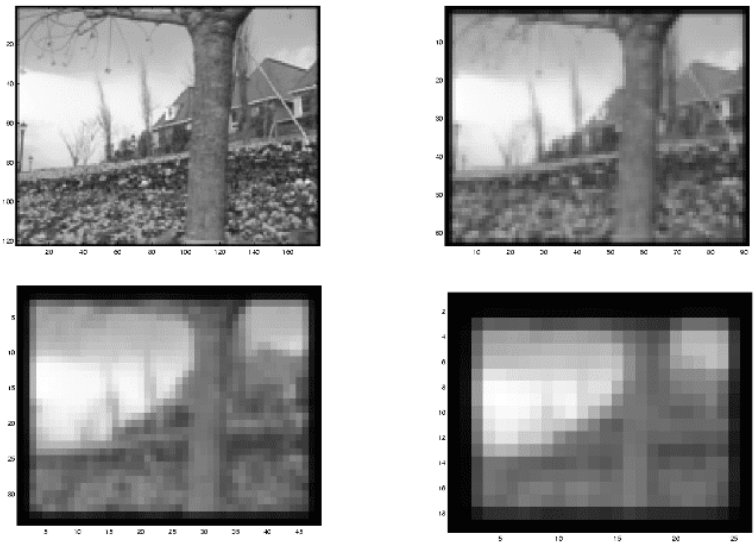
e Fraundorfer (2012), esta abordagem de detectar-e-rastrear é adequada para aplicações de odometria visual em que as imagens são capturadas em localizações próximas uma das outras, isto é, onde a quantidade de movimento é pequeno. Para casos em que há uma sequência longa de imagens e, consequentemente, espera-se que a aparência das *features* sofram significativas mudanças, o *Kanade-Lucas-Tomasi tracker* (KLT) é o mais apropriado. O KLT é um dos rastreadores mais utilizados em visão computacional, visto sua fácil implementação, rapidez e acurácia. Ele combina o detector de *features*, *Good Features To Track*, utilizando o detector de cantos Shi-Tomasi (SHI; TOMASI, 1994) com o método iterativo Lucas-Kanade.

2.5.7.1 Método Lucas-Kanade

O método Lucas-Kanade foi apresentado na subseção 2.5.1.1 para cálculo do fluxo óptico denso. Aqui serão abordados conceitos complementares do algoritmo para fluxo óptico esparso, visto que os fundamentos são os mesmos.

Conforme descrito na subseção 2.5.1.1, o método Lucas-Kanade de estimação de fluxo óptico é baseado em algumas suposições. Primeiramente, para um dado *pixel*, o fluxo é essencialmente constante para uma vizinhança local (suposição da constância do brilho), ou seja, o nível de brilho em um *pixel* não vai mudar significativamente entre *frames*. Assim, as equações básicas de fluxo óptico são resolvidas para todos os *pixels* naquela vizinhança, utilizando o critério dos quadrados mínimos. Supõe-se, também que o movimento da cena entre *frames* consecutivos é pequeno. Se o movimento for muito grande e o *pixel* se deslocar para fora da vizinhança local, o algoritmo não será capaz de detectá-lo. Para contornar este problema, visto que na prática os movimentos são grandes, surge a representação de pirâmide de imagem, em que, basicamente, a resolução do *frame* é reduzida, conforme ilustrado na Figura 26.

Figura 26 – Mesma imagem com múltiplas resoluções



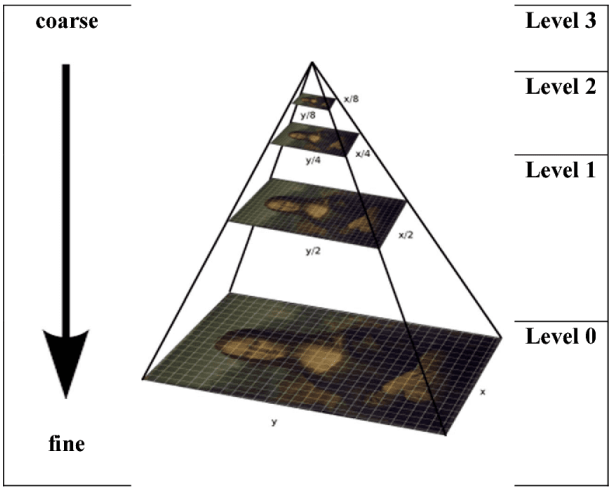
Fonte: Shafique (2003)

Seja uma representação piramidal de uma imagem genérica I de tamanho n_x x n_y. Seja $I^0 = I$ o nível zero da imagem, que é a mais alta resolução da imagem, isto é, a imagem original (BOUGUET, 2000). A representação piramidal é implementada de uma maneira recursiva: Calcula-se I^1 de I^0, então I^2 de I^1 e assim suscetivamente. Seja L = 1, 2 ... um nível genérico piramidal e I^{L-1} a imagem no nível L-1. Considerando n_x^{L-1} e n_y^{L-1} a largura e altura da imagem I^{L-1}, respectivamente, a imagem I^{L-1} é definida como (BOUGUET, 2000):

$$I^L(x,y) = \frac{I^{L-1}(2x,2y)}{4} + \frac{I^{L-1}(2x-1,2y) + I^{L-1}(2x+1,2y) + I^{L-1}(2x,2y-1) + I^{L-1}(2x,2y+1)}{8} + \frac{I^{L-1}(2x-1,2y-1) + I^{L-1}(2x+1,2y-1) + I^{L-1}(2x-1,2y+1) + I^{L-1}(2x+1,2y+1)}{16} \tag{2.13}$$

A representação piramidal é construída recursivamente. Conforme descreve Bouguet (2000), valores práticos para a altura de uma pirâmide são 2,3 e 4. Para uma imagem I de tamanho 1280 x 720, por exemplo, as imagens I^1, I^2, I^3 e I^4 são, respectivamente, 640 x 360, 320 x 180, 160 x 90 e 80 x 45. Nota-se que não faz sentido ir além do nível 4. Reitera-se que a motivação central pela representação piramidal é possibilitar o algoritmo de rastrear *features* lidar com movimentos grandes de *pixels*, isto é, maior que a vizinhança local. Dessa forma, a altura da pirâmide deve ser escolhida apropriadamente de acordo com fluxo óptico máximo esperado na imagem (BOUGUET, 2000). A Figura 27 ilustra o funcionamento do fluxo óptico piramidal. Note que a cada nível, a resolução da imagem é diminuída para lidar com grande quantidade de movimento.

Figura 27 – Imagem piramidal com 4 níveis. A cada nível, o tamanho da imagem é diminuído

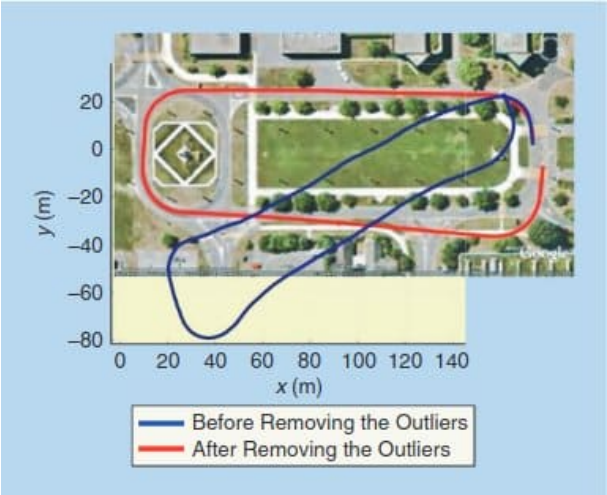


Fonte: Husseini (2017)

2.6 Rejeição de valores atípicos

É importante ressaltar que nas etapas anteriores, mormente a correspondência entre *features*, pressupõe-se que a luminosidade ao longo da trajetória é constante, que a quantidade de movimento é baixa e outras modelagens matemáticas. No entanto, nas aplicações reais, diversos fatores, como ruídos nas imagens, oclusões, borramentos, mudanças de luminosidade resultam em valores atípicos, isto é, correspondências entre *features* erradas (SCARAMUZZA; FRAUNDORFER, 2012). Tais associações equivocadas afetam significativamente a acurácia dos resultados, conforme mostrado na Figura 28.

Figura 28 – Comparação entre trajetórias estimadas de odometria visual antes e depois da remoção de valores atípicos



Fonte: Scaramuzza e Fraundorfer (2012)

Assim, é de suma importância a rejeição de valores atípicos. Uma das técnicas mais empregadas na odometria visual é o *Random Sample Consensus* (RANSAC).

2.6.1 RANSAC

O RANSAC (FISCHLER; BOLLES, 1981) é considerada a técnica-padrão para remoção de valores atípicos. Fundamentalmente, consiste em levantar modelos de hipóteses a partir de conjuntos de dados amostrados aleatoriamente e então verificar a veracidade de tais hipóteses com outros dados (SCARAMUZZA; FRAUNDORFER, 2012) de forma iterativa. A hipótese que mostrar o maior consenso entre dados é considerada uma solução. No caso da odometria visual, o modelo em questão é o movimento relativo - rotação e translação - entre *frames*, e a correspondência entre *features* destas imagens são os dados.

Os pontos dentro da curva para uma dada hipótese são encontrados ao se calcular a distância máxima entre um ponto e a linha epipolar, conhecida como Distância de Sampson (HARTLEY; ZISSERMAN, 2004). Dessa forma, se um ponto está além desta distância, ele é considerado um ponto fora da curva e, consequentemente, não é utilizado para calcular a

matriz final. Como se trata de um algoritmo iterativo, o número de iterações N necessárias para garantir uma solução correta é dada por (SCARAMUZZA; FRAUNDORFER, 2012):

$$N = \frac{log(1-P)}{log(1-(1-\epsilon)^s)} \tag{2.14}$$

em que s é o número de correspondência entre *features*, ϵ é a porcentagem de valores atípicos e P é a probabilidade desejada de sucesso. Conforme afirmam Scaramuzza e Fraundorfer (2012), geralmente, N é multiplicado por um fator 10, a fim de trazer mais robustez ao algoritmo. A solução tende a ser estável quando o número de iterações cresce. Um maior detalhamento do algoritmo pode ser encontrado em (FISCHLER; BOLLES, 1981).

2.7 Transformações geométricas

A obtenção das características e parâmetros associados as transformações são o objeto de interesse final da odometria visual, pois é através dessas transformações que o movimento relativo de uma entidade é definido, seja ele translacional, rotacional, ou uma combinação complexa dos dois tipos. Para evitar problemas de indeterminismo na movimentação relativa entre objetos, geralmente assume-se que os corpos envolvidos são corpos rígidos e condições ambientais são constantes, como por exemplo, luminosidade.

2.7.1 Matrizes de translação

A matriz de translação define os parâmetros associados a movimentação translacional de um corpo, ou seja, de ponto a ponto. Para uma transformação 2-D, a operação de translação pode ser definida pela Equação 2.15.

$$\begin{aligned} x_r &= x + t_x \\ y_r &= y + t_y \end{aligned} \tag{2.15}$$

De forma mais geral, a operação pode ser representada pela Equação 2.16, no qual a matriz P_r define os pontos resultantes, P os pontos de referência, e a matriz de translação propriamente dita é definida por T(t_x,t_y), no qual os fatores t_x e t_y representam as translações efetuadas nos respectivos eixos coordenados. A matriz de transformação pode ser estendida facilmente para mais dimensões a depender da situação analisada, conforme mostrado na Equação 2.17.

$$P_r = \begin{bmatrix} x_r \\ y_r \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = T(t_x, t_y)P \tag{2.16}$$

$$P_r = \begin{bmatrix} x_r \\ y_r \\ z_r \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = T(t_x, t_y, t_z)P \tag{2.17}$$

2.7.2 Matrizes de rotação

A matriz de rotação define os parâmetros associados ao movimento rotacional da entidade (MAJUMDER; GOPI, 2018). Para uma rotação 2D a referência de rotação é o próprio plano ao qual pertence o ponto, e é definida pela Equação 2.18, no qual θ define a rotação especificada. Para definir completamente o movimento no espaço, porém, a rotação deve ocorrer em 3D, de forma que a rotação não só deve ocorrer no plano, mas também em relação a um eixo específico, dessa forma, para a rotação completa nos três eixos é necessário definir três matrizes de rotação distintas, considerando como referência os eixos **x**, **y** e **z**. Tal conjunto de matrizes é definido na Equação 2.19. Naturalmente, a rotação sobre um eixo arbitrário pode ser definida a partir da concatenação das matrizes definidas pela Equação 2.19.

$$P_r = \begin{bmatrix} x_r \\ y_r \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = R(\theta)P \tag{2.18}$$

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix}, R_y(\beta) = \begin{bmatrix} \cos\beta & 0 & -\sin\beta \\ 0 & 1 & 0 \\ \sin\beta & 0 & \cos\beta \end{bmatrix}, R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.19}$$

2.7.3 Matrizes de transformação de corpo rígido

A matriz de transformação de corpo rígido (MAJUMDER; GOPI, 2018) é um dos objeto mais importantes da odometria visual, pois representa completamente o movimento espacial de um corpo rígido. Associando os fatores de translação aos de rotação, é possível descrever a trajetória espacial de um corpo em sua completude, logo, a tarefa principal de um sistema de odometria visual é fazer a computação dessa matriz para imagens capturadas em diferentes instantes de tempo em um conjunto de quadros subsequentes, e, através da estimação dos parâmetros dessa matriz com auxílio dos dados coletados, estimar a posição atual de uma entidade ou a trajetória descrita.

A matriz pode ser descrita pela Equação 2.20, no qual é possível observar que a mesma é um combinação das matrizes de translação e de rotação, $T_{k,k-1}$ e $R_{k,k-1}$ respectivamente,

em que o fator k e k-1 indica que a respectiva matriz está relacionada a dois instantes de tempo diferentes e subsequentes.

$$T_r = \begin{bmatrix} R_{k,k-1} & T_{k,k-1} \\ 0 & 1 \end{bmatrix} \tag{2.20}$$

2.7.4 Matriz essencial e fundamental

As relações geométricas entre um par de imagens são conhecidas como geometria epipolar e podem ser descritas por meio de uma matriz essencial ou matriz fundamental. A matriz é dita fundamental, F, se o mapeamento entre pontos obtidos de pontos correspondentes é feito sem informação prévia da câmera (TRUCCO; VERRI, 1998), isto é, com uma câmera não-calibrada. Já a matriz é dita essencial, E, se o mapeamento é feito com auxílio dos parâmetros extrínsecos do sistema, isto é, através de uma câmera calibrada. Em outras palavras, a matriz essencial é a matriz fundamental para câmeras calibradas. Ela é uma representação fiel do movimento (rotação e translação, até uma escala) (LI; HARTLEY, 2006). Assim, posteriormente, ela pode ser decomposta para recuperar o movimento relativo entre imagens.

Para definir a matriz fundamental e a matriz essencial, segundo Nister (2004), seja P_1 e P_2 vetores homogêneos tridimensionais que representam os pontos na primeira e segunda visão, respectivamente. Os pontos universais são representados por Q, um vetor homogêneo quadridimensional. Uma visão perspectiva é representada por uma matriz 3×4, P, em que $P_1 \sim PQ$, em que $\sim$ denota igualdade até uma escala. Conforme prossegue Nister (2004), uma visão com um centro de projeção finita pode ser fatorado em $\mathrm{P} = \mathrm{K}[\mathrm{R}|\mathrm{t}]$, em que K é uma matriz de calibração 3×3 triangular superior contendo os parâmetros intrínsecos da câmera, R e t são as matrizes de rotação e translação do corpo, definidas nas sub-seções 2.7.2 e 2.7.1, respectivamente. Seja as matrizes das câmeras para as duas visões definidas como $K_1[\mathrm{I}|0]$ e $\mathrm{P} = K_2[\mathrm{R}|\mathrm{t}]$, respectivamente. Ainda, seja $[t]_\times$ a seguinte matriz antissimétrica (NISTER, 2004):

$$[t]_\times = \begin{bmatrix} 0 & -t_3 & t_2 \\ t_3 & 0 & -t_1 \\ -t_2 & t_1 & 0 \end{bmatrix} \tag{2.21}$$

de tal forma que $[t]_\times \mathrm{x} = \mathrm{t} \times \mathrm{x}$ para todo x. Assim, a matriz fundamental é definida como (NISTER, 2004):

$$F \equiv K_2^{-\mathrm{T}}[t]_\times R K_1^{-1} \tag{2.22}$$

A matriz fundamental é regida pela restrição epipolar (NISTER, 2004):

$$P_1^{\mathrm{T}} F P_2 = 0 \tag{2.23}$$

Conforme dito anteriormente, a matriz essencial é a matriz fundamental para câmeras calibradas. Desse modo, se a câmera está calibrada, isto é, K_1 e K_2 são conhecidos, então, a Equação 2.23 pode ser simplificada como (NISTER, 2004):

$$P_1^{\intercal} E P_2 = 0 \tag{2.24}$$

em que $\mathrm{E} \equiv [\mathrm{t}]_{\times}\mathrm{R}$ é a chamada matriz essencial

Uma matriz essencial válida E deve satisfazer a seguinte condição de singularidade cúbica (LI; HARTLEY, 2006):

$$det(E) = 0 \tag{2.25}$$

A matriz essencial é uma matriz real e homogênea 3 x 3 e pode ser calculada utilizando restrição epipolar a partir de uma correspondência de *features* 2D-2D. A solução mínima envolve 5 correspondências 2D-2D. Devido a estes únicos 5 Graus de Liberdade (GDL), para E ser uma matriz essencial válida, ela deve satisfazer duas restrições, caracterizadas pelo seguinte resultado (LI; HARTLEY, 2006):

$$2EE^{\intercal}E - tr(EE^{\intercal})E = 0 \tag{2.26}$$

A expressão acima é necessária para recuperar a matriz essencial. Ela lida com as 9 equações nos elementos de E, mas somente duas delas são algebricamente independentes. Dados 5 pontos correspondentes, há 5 equações epipolares Equação 2.24, mais 9 equações resultantes da Equação 2.26 e a condição de singularidade da Equação 2.25. Dessa forma, há equações suficientes para estimar a matriz essencial e, assim, recuperar R e t de E.

Dentre uma gama de métodos para estimação, o algoritmo de 8 pontos (LONGUET-HIGGINS, 1981) é considerado o algoritmo clássico, devido a sua linearidade e simplicidade, além de resultados razoavelmente bons (LI; HARTLEY, 2006). No entanto, uma abordagem mais recente que apresenta resultados mais acurados e uma melhor eficiência computacional (LONGUET-HIGGINS, 1981) é o algoritmo de 5 pontos de Nister (NISTER, 2004).

2.7.4.1 Algoritmo de 5 pontos de Nister

Este algoritmo de 5 pontos foi desenvolvido por Nister (2004) e é utilizado para calcular a orientação relativa em câmeras calibradas. É considerado a solução-padrão na obtenção da solução direta da matriz essencial. O algoritmo envolve resolver um polinômio de grau 10, obtido através das equações epipolares da Equação 2.24 para os 5 pontos e as 9 equações. Resolvendo-o, chegam-se em 10 soluções possíveis. As raízes deste polinômio de décimo grau são extraídas. Por fim, recupera-se a rotação - R - e a translação - t - correspondente

para cada raiz real e triangulação dos pontos para desambiguidade (NISTER, 2004). Um maior detalhamento pode ser encontrado em (NISTER, 2004) e (LI; HARTLEY, 2006).

O algoritmo de Nister é frequentemente utilizado com o RANSAC, apresentado na subseção 2.6.1, e esta combinação, inclusive, é recomendada por Nister para aumentar a efetividade do algoritmo (NISTER, 2004). Nesta combinação, o RANSAC é um algoritmo iterativo: a cada iteração, aleatoriamente são amostrados 5 pontos do conjunto de pontos-candidatos de correspondência entre *features*, estima-se a matriz essencial, e, então, verifica-se se os outros pontos estão dentro da curva quando usada esta matriz essencial (FISCHLER; BOLLES, 1981). A condição de parada do algoritmo é um número fixo de iterações.

2.7.5 Extração de R e t da matriz essencial

Conforme descrito na subseção 2.7.4.1, no algoritmo de 5 pontos de Nister, R e t são recuperados através da matriz essencial estimada.

Para isso, seja (NISTER, 2004):

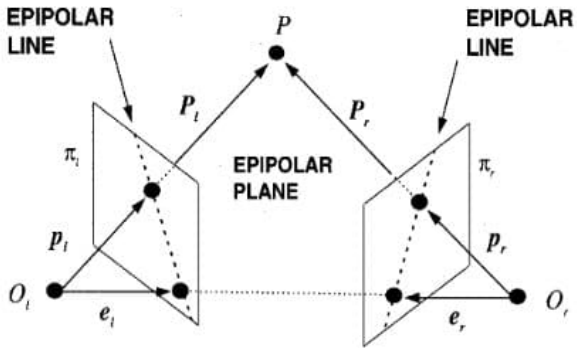
$$D = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.27}$$

Ainda, seja a Decomposição de Valor Singular (do inglês, *Singular Value Decomposition* - SVD) da matriz essencial $E \sim \mathrm{U}diag(1\ 1\ 0)V^T$, em que U e V são escolhidos de tal forma que det(U) > 0 e det(V) > 0 (NISTER, 2004). Então $t \sim t_u \equiv [\ u_{13}\ u_{23}\ u_{13}]^T$ e R é igual a $R_a \equiv \mathrm{U}DV^T$ ou $R_b \equiv \mathrm{U}D^TV^T$ (NISTER, 2004).

Conforme explica Nister (2004), qualquer combinação de R e t, de acordo com o detalhamento acima, satisfaz a restrição epipolar descrita na Equação 2.24. Para solucionar ambiguidades inerentes, é assumido que a matriz da primeira visão é [I|0] e que t é uma unidade de distância (NISTER, 2004). Dessa forma, há 4 soluções possíveis para a matriz da segunda visão: $P_A \equiv [R_a|t_u]$, $P_B \equiv [R_a|-t_u]$, $P_C \equiv [R_b|t_u]$ e $P_D \equiv [R_b|-t_u]$, sendo que apenas uma delas é a configuração verdadeira, outra é obtida ao se rotacionar em 180º a base, e as duas restantes são reflexões destas. Nister (2004) descreve que, para identificar a configuração que corresponde à solução correta, é necessário utilizar a *cheirality constraint*.

Conceitualmente, pontos correspondentes entre duas imagens satisfazem a restrição epipolar. No entanto, nem todos os conjuntos de pontos satisfazem a restrição epipolar correspondente a qualquer geometria real, pois podem existir pontos nas cenas de tal forma que todos os pontos na imagem possuam profundidade positiva (HARTLEY; ZISSERMAN, 2004). Assim, a *cheirality constraint* identifica quais pontos devem estar na frente das duas visões, conforme ilustra a Figura 29. Um maior detalhamento pode ser encontrado em (HARTLEY, 1998) e (HARTLEY; ZISSERMAN, 2004).

Figura 29 – Geometria epipolar



Fonte: Trucco e Verri (1998)

Assim, com a *cheirality constraint*, identifica-se qual das 4 combinações corresponde à solução verdadeira, extraindo-se, então, R e t. É válido salientar que a escala absoluta de uma translação não pode ser calculada com apenas informações das imagens (SCARAMUZZA; FRAUNDORFER, 2011). São extraídas, apenas, a direção da translação em uma unidade de distância, em que se obtém a escala relativa entre poses.

2.8 Reconstrução da trajetória

Para recuperar a trajetória de uma sequência de imagens, as transformações de rotação e translação, representadas pela matriz de transformação homogênea, descrita pela Equação 2.20, devem ser concatenadas ao longo do tempo.

Seja o conjunto de *poses* da câmera, $C_{0:n} = \{C_0,...,C_n\}$, que contém as transformações da câmera com respeito às coordenadas iniciais do *frame* em r = 0 (SCARAMUZZA; FRAUNDORFER, 2011). A posição atual, C_n, pode ser calculada ao se concatenar todas as transformações T_r (vide Equação 2.20), em que r = {1 ... n}. Dessa forma, a posição é dada por (SCARAMUZZA; FRAUNDORFER, 2011):

$$C_n = C_{n-1}T_n \tag{2.28}$$

Reescrevendo a Equação 2.28, a posição e orientação da câmera no instante n é dada por (SICILIANO; KHATIB, 2007):

$$C_n = R_{n,n-1}C_{n-1} + T_{n,n-1} \tag{2.29}$$

em que $R_{n,n-1}$ é a matriz de rotação descrita na subseção 2.7.2 e $T_{n,n-1}$ é a matriz de translação descrita na subseção 2.7.1.

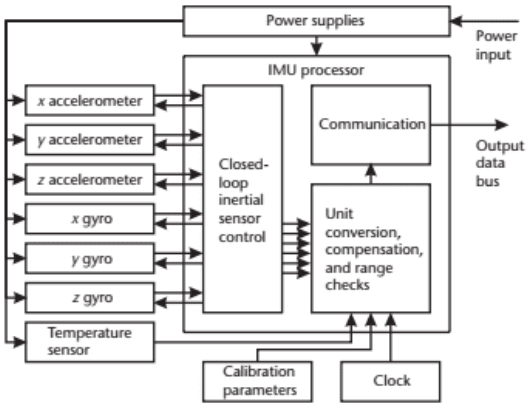
2.9 Elementos sensores

Os elementos sensores são parte fundamental na obtenção de uma melhor estimativa do movimento de um objeto de interesse. Na odometria, o uso de diversos tipos de sensores em conjunto com a obtenção e processamento de imagens por câmeras para estimação da posição de um objeto, a chamada *visual inertial odometry* ou odometria visual inercial, não somente promove robustez ao processo de medição e estimação, mas também possibilita uma maior precisão e acurácia nos dados estimados através de técnicas de fusão de dados. Entre os sensores mais comuns para esse tipo de aplicação, destacam-se os acelerômetros, giroscópios e magnetômetros, geralmente integrantes de uma IMU.

2.9.1 *Inertial Measurement Unit* - IMU

A *Inertial Measurement Unit* (IMU) é uma unidade que permite estimar a posição, velocidade e atitude de um sistema ou corpo através de medidas obtidas por giroscópios, acelerômetros e, em alguns casos, magnetômetros (GROVES, 2008). Na Figura 30 é possível ver a composição típica de uma IMU, composta por acelerômetros e giroscópios associados a cada um dos eixos, x, y e z, um sensor de temperatura, e uma unidade de processamento que pode realizar tarefas como conversão de unidades, compensação de erros e verificação de integridade da unidade, além disso, essa unidade pode ser responsável também por integrar os dados obtidos de forma a obter novas informações a partir das medições realizadas pelos sensores.

Figura 30 – IMU - Construção típica



Fonte: Groves (2008)

Apesar do extenso uso em aplicações de navegação para os mais diversos tipos de sistemas, os sensores que compõe a unidade de medida inercial apresentam problemas de desvios que levam a erros de estimação que se intensificam ao longo do tempo (LEIBSON, 2019). Com o intuito de reduzir ou minimizar os erros de estimação é possível utilizar técnicas de fusão de dados como o filtro de Kalman (AGUIRRE, 2015) (DORF, 2011). Através da fusão

dos dados obtidos pelos sensores que compõe a unidade é possível reduzir significativamente os erros observados e produzir resultados mais apropriados.

Além de erros devido a ruídos aleatórios, as IMUs também possuem erros relativamente constantes que podem ser tratados de forma a mitigar a influência dos mesmos e promover resultados mais precisos. Entre esses erros é possível citar os *biases*, fatores de escala, erros de alinhamento entre os eixos dos sensores e o eixo de referência do corpo do dispositivo, o chamado erro de *cross-coupling*, entre outros (GROVES, 2008) (LEIBSON, 2019).

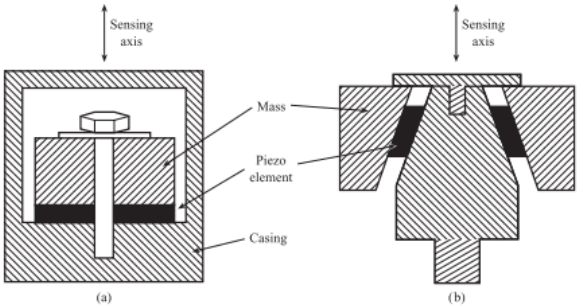
2.9.2 Acelerômetro

O acelerômetro é um tipo de sensor que possibilita a obtenção de dados a respeito das forças de aceleração que atuam um corpo e, através da integração dos dados obtidos, possibilita obter informações acerca da posição e movimento do corpo especifico (GROVES, 2008) (WILSON, 2005). Os modelos típicos de construção desse tipo de sensor são o piezoelétrico, piezoresistivo, e capacitivo. Para outras aplicações é possível ainda o uso de sensores mecânicos.

2.9.2.1 Acelerômetro piezoelétrico

Acelerômetros piezoelétricos tem como base de funcionamento a utilização de cristais piezoelétricos em conjunto com um elemento de massa teste e um elemento de retenção. Se o acelerômetro sofre uma aceleração, uma força resultante agirá sobre a massa e o cristal fazendo com que o cristal adquira uma carga proporcional a aceleração que pode ser convertido em um sinal de tensão correspondente (GROVES, 2008) (BENTLEY, 2005) (HANLY, 2016). A Figura 31 apresenta duas construções possíveis para esse tipo de acelerômetro.

Figura 31 – Acelerômetros piezoelétricos - a) modo de compressão - b) modo de cisalhamento



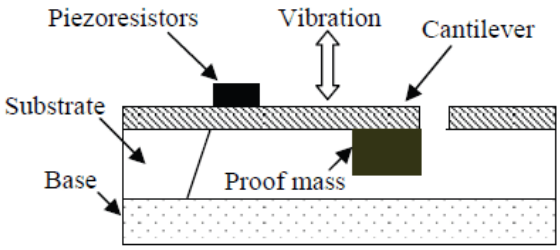
Fonte: Bentley (2005)

2.9.2.2 Acelerômetro piezoresistivo

Acelerômetros piezoresistivos trabalham com base no efeito piezoresistivo, em que variações na resistividade de um elemento são provocadas por tensões cisalhantes que atuam

sobre o mesmo, essa alteração na resistência pode ser associada a uma ponte de deflexão apropriada promovendo uma saída proporcional a aceleração provocada no elemento de massa interno pelas forças atuantes no elemento sensor (BENTLEY, 2005) (ALBARBAR et al., 2008). Esse tipo de acelerômetro possui baixa sensitividade mas pode ser utilizado para obter medidas de velocidade e deslocamento de forma precisa. A Figura 32 apresenta uma construção típica para esse tipo de acelerômetro.

Figura 32 – Acelerômetro piezoresistivo modelo tipo cantiléver

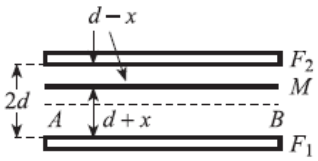


Fonte: Albarbar et al. (2008)

2.9.2.3 Acelerômetro capacitivo

Acelerômetros capacitivos tem como princípio básico de funcionamento a medição do deslocamento diferencial ocorrido devido a uma aceleração sofrida pelo corpo. Sensores de deslocamento diferencial capacitivos trabalham com a variação da capacitância entre placas paralelas, entre essas placas atua um elemento que se desloca alterando a capacitância entre essas placas e o elemento de acordo com o deslocamento sofrido (BENTLEY, 2005) (ALBARBAR et al., 2008) (HANLY, 2020). Em acelerômetros o elemento que sofre o deslocamento é tipicamente uma massa teste. Na Figura 33, F_1 e F_2 representam as placas paralelas ao passo que M representa o elemento que permite alterar a capacitância.

Figura 33 – Sensor de deslocamento diferencial genérico



Fonte: Bentley (2005)

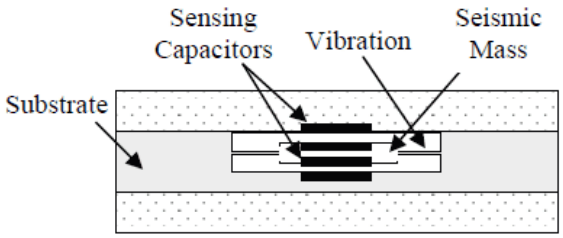
Entre M e F_1 e M e F_2 é possível definir capacitâncias C_1 e C_2 e, a medida que o deslocamento definido por **x** varia, C_1 e C_2 variam proporcionalmente como definido pela Equação 2.30 e Equação 2.31. Para ambas equações **A** define a área da placa, ϵ define a constante dielétrica do material do elemento entre essas placas, ϵ_0 a permissividade do vácuo e, por fim, **d** define a distância entre cada placa e a linha de centro que separa as mesmas. Naturalmente faz-se necessária ainda etapa posterior de condicionamento de sinal

para obtenção de voltagem proporcional a aceleração medida. A Figura 34 apresenta uma construção típica para esse tipo de acelerômetro.

$$C_1 = \frac{\epsilon\epsilon_0 A}{d + x} \tag{2.30}$$

$$C_2 = \frac{\epsilon\epsilon_0 A}{d - x} \tag{2.31}$$

Figura 34 – Acelerômetro capacitivo baseado em modelo tipo membrana



Fonte: Albarbar et al. (2008)

2.9.2.4 Características

Conforme o *datasheet* (INVENSENSE, 2012), o acelerômetro utilizado, parte integrante da MPU-6050, é um acelerômetro *Micro Electro-Mechanical Systems* (MEMS) do tipo capacitivo. A IMU possui uma massa teste em cada eixo de medição, x, y e z, que, quando induzidas a se deslocar, sensores capacitivos captam o deslocamento de forma diferencial permitindo que a medição seja realizada, conforme apresentado na subseção 2.9.2.3.

O sensor utilizado possui saída digital com faixa de indicação programável, podendo ser atribuídos valores de ±2g, ±4g, ±8g e ±16g, de forma similar, o sensor conta com sensibilidade também programável a depender do modo de operação. A sensibilidade pode variar discretamente de 16,384 LSB/g a 2,048 LSB/g, tomando valores de 16,384 LSB/g, 8,192 LSB/g, 4,096 LSB/g e 2,048 LSB/g. Por fim, para realizar a digitalização o sensor conta com um *Analog-to-Digital Converter* (ADC) integrado com codificação em 16 *bits*. Como a aceleração observada para a aplicação em questão será bastante baixa, será possível contar com uma alta sensibilidade para realização das medições.

Considerando também erros relacionados ao *zero-bias*, o acelerômetro apresenta saída de ±50mg para os eixos x e y e ±80mg para o eixo z, assim como uma variação com a temperatura de ±35mg e ±60mg para os eixos x, y e z respectivamente, esses erros podem ser mitigados através da compensação da própria IMU assim como através de uma calibração apropriada do dispositivo, garantindo que os erros não tenha impacto apreciável no processo.

É importante salientar também que, devido ao fato das medições ocorrerem em tempo discreto, além dos erros dos erros advindos do processo de medição, as integrações efetuadas para obtenção dos dados relacionados a velocidade e posição do corpo também introduzirão erros que dependerão das taxas de amostragem utilizadas, sendo que, quanto maior a taxa de amostragem, menor o erro observado, o contrário também é verdade.

O dispositivo utilizado possui taxa de amostragem programável de 4Hz a 1KHz, é possível observar ainda que, apesar de uma taxa de amostragem maior apresentar um menor erro associado, como a aplicação apresentada trabalha com recursos escassos no que diz respeito a memória e poder de processamento, uma maior taxa de amostragem poderá gerar dados que podem não ser processados em tempo hábil, levando a um uso ineficiente dos recursos disponíveis, comprometendo inclusive possíveis requisitos de tempo real estabelecidos.

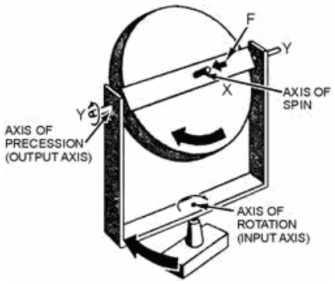
2.9.3 Giroscópio

Também parte integrante da IMU, o giroscópio é um sensor que permite medir a velocidade angular, movimentação e orientação de um objeto ou corpo. A medida dos ângulos é efetuada tomando como referência os três eixos principais, nos quais os ângulos relativos a esses eixos são também conhecidos por ângulos de atitude, *yaw, pitch* e *roll*, esses ângulos permitem definir rotações verticais, transversais e longitudinais respectivamente, assumindo que o objeto esteja alinhado aos eixos principais. Os modelos mais comuns são de funcionamento mecânico ou óptico, sendo que o modelo *Coriolis Vibratory Gyroscope* (CVG) é um dos mais utilizados.

2.9.3.1 Giroscópio mecânico

Giroscópios mecânicos clássicos (GROVES, 2008) (NAVY, 2016) se baseiam no princípio da conservação do momento angular. Eles consistem em uma massa que realiza rotação em torno de um eixo fixo e um corpo de apoio que permita que a mesma gire livremente em torno de eixos perpendiculares ao eixo de rotação. Utilizando o princípio de conservação do momento angular, uma massa isolada, como é o caso da construção apresentada, tenderá a manter sua posição angular em relação a um plano de referência inercial, quando um torque externo atua nessa massa o eixo de rotação passa a sofrer um movimento de precessão em direção normal ao torque aplicado, dessa forma surgirá um torque devido ao movimento de precessão induzido que tenderá a manter a massa alinhada ao seu plano de referência. Esse torque devido ao movimento de precessão pode então ser utilizado para obtenção da velocidade angular medida devido a sua relação com a velocidade de precessão observada e o torque original aplicado. A Figura 35 ilustra a composição de um dispositivo simples sem nenhum aparato que permita obtenção de medidas com indicação dos respectivos eixos de entrada e saída.

Figura 35 – Giroscópio mecânico simples



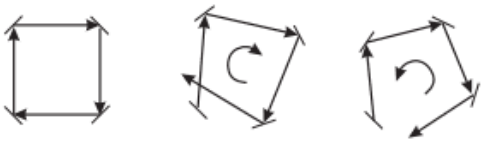
Fonte: Navy (2016)

2.9.3.2 Giroscópio óptico

Giroscópios de efeito óptico (GROVES, 2008) funcionam com base no fato de que a luz sempre se propaga a uma velocidade constante em um meio de referência. O funcionamento básico se dá através do envio de feixes luminosos em direções opostas em uma malha fechada onde são guiados por espelhos ou fibra ótica. Caso não ocorra rotação da referência, ambos feixes percorrerão distância iguais, porém, caso ocorram rotações em direção perpendicular ao plano da referência, os dispositivos de reflexão se moveram de forma a se aproximar ou se afastar dos feixes a depender de sua direção, de forma que, rotacionar na mesma direção do caminho do feixe luminoso provoca um aumento no caminho que a luz percorre, e rotacionar em direção contrária provoca diminuição desse caminho, tal efeito é conhecido como Sagnac (GROVES, 2008).

A Figura 36 demonstra a ideia do fenômeno, no primeiro caso o dispositivo não sofre rotação, no segundo o dispositivo gira na direção da luz e por fim, no último caso o dispositivo gira em direção contraria a luz.

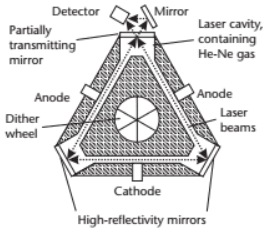
Figura 36 – Efeito Sagnac



Fonte: Groves (2008)

Observando tal efeito, uma velocidade angular aplicada a um dispositivo desse tipo induz uma diferença de tempo, mesmo que bastante baixa, entre os tempos que ambos os raios levam para percorrer o caminho e, ao medir a variação ocorrida é possível determinar a velocidade angular observada. Por fim, a Figura 37 mostra um modelo típico, o tipo anel de laser (do inglês, *Ring Laser*).

Figura 37 – Giroscópio tipo anel de laser



Fonte: Groves (2008)

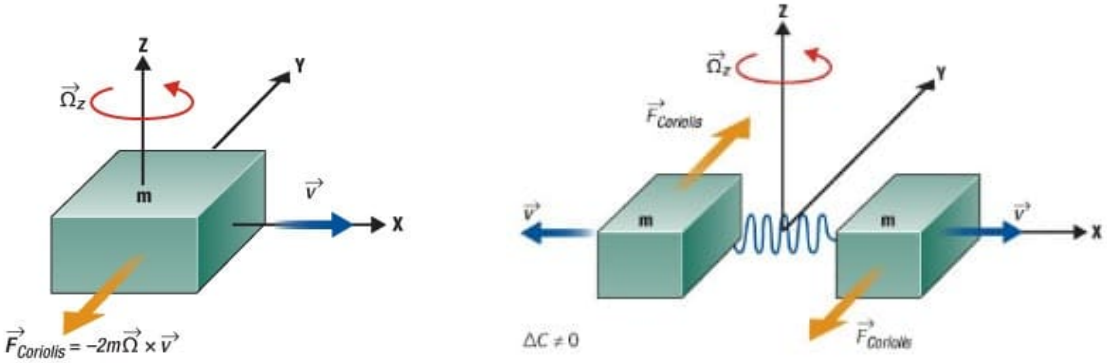
2.9.3.3 Giroscópio de efeito Coriolis - CVG

Giroscópios de efeito Coriolis (GROVES, 2008) (SCHWEBER, 2018) tem como princípio básico de funcionamento detectar a aceleração de Coriolis de algum elemento vibratório que pode ser observada quando o giroscópio sofre alguma ação que o induza a rotacionar .

Esses giroscópios são construídos utilizando como base algum elemento que possa ser induzido a descrever um movimento harmônico simples e uma estrutura de corpo que comporte esse elemento. Caso o elemento que esteja vibrando ao longo de um dos eixos seja submetido a uma velocidade angular em um eixo perpendicular a direção de oscilação, uma força de Coriolis, que por sua vez da origem a aceleração de Coriolis, surgiria em direção perpendicular a direção de oscilação e da projeção da velocidade angular, levando a um movimento harmônico na mesma direção com amplitude proporcional a velocidade angular a qual o dispositivo foi submetido.

A Figura 38a descreve o fenômeno, em que $\vec{v}$ denota uma velocidade associada ao modo de vibração, **m** o elemento submetido a vibração, $\vec{\Omega}_z$ a velocidade angular a qual o elemento é submetido e $F_{Coriolis}$ a força de Coriolis que surge a partir da relação citada. Na prática, a implementação é feita utilizando composição similar a Figura 38b de forma que os efeitos de aceleração e aceleração centrífuga a qual o corpo possa estar submetido possam ser desprezados e apenas o efeito desejado seja observado.

Figura 38 – Efeito Coriolis e construção típica

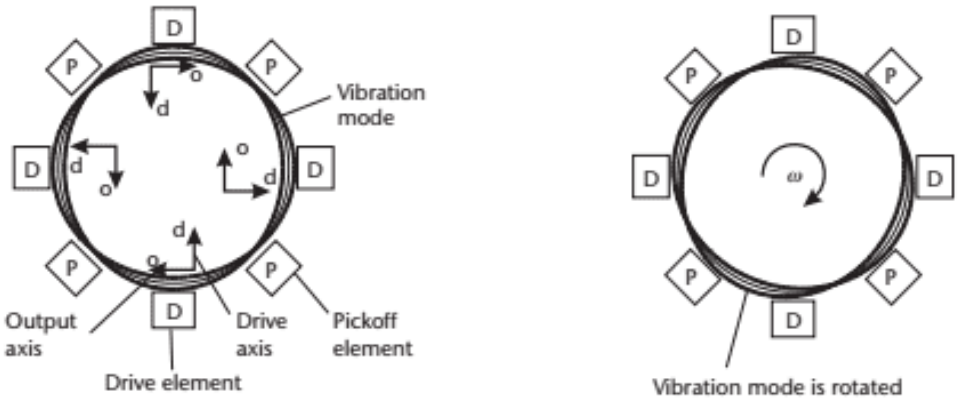


(a) Efeito Coriolis (b) Construção típica - Coriolis

Fonte: Schweber (2018)

Para que se possa avaliar o efeito, a vibração induzida pelo efeito Coriolis deve então ser detectada por algum elemento para que se possa efetuar a medição da velocidade angular. As Figuras 39a e 39b mostram de forma genérica como é o funcionamento do dispositivo, em que **D** representa os elementos responsáveis por induzir a vibração no elemento que descreve o movimento harmônico, **P** os elementos responsáveis pela medição da vibração e as linhas internas a esses representam os elementos vibratórios. Quando não existe nenhuma rotação no dispositivo, os nós da oscilação se alinham com os detectores de forma que nenhuma vibração é captada, como mostrado na Figura 39a. Por outro lado, quando uma rotação ocorre, Figura 39b, o modo de vibração rotaciona de forma que os detectores passam a captar uma amplitude não nula.

Figura 39 – Captação da vibração pelos elementos de medição



(a) Elemento de medição - Sem vibração captada

(b) Elemento de medição - Após rotação - Vibração captada

Fonte: Groves (2008)

2.9.3.4 Características

Conforme o *datasheet* (INVENSENSE, 2012), o giroscópio utilizado, parte integrante da MPU-6050, é um giroscópio MEMS do tipo CVG. A IMU conta com um dispositivo em cada eixo, x, y e z, dessa forma, uma vez que o dispositivo é rotacionado em algumas das direções, o efeito Coriolis induzirá uma vibração a ser captada, sendo que, para essa construção, os elementos responsáveis por captar as vibrações induzidas são capacitores. Captada tal vibração, o sinal obtido será amplificado e filtrado para que se possa produzir uma voltagem proporcional a velocidade angular observada.

O giroscópio conta com *range* programável de ±250, ±500, ±1000 e ±2000 °/s, de forma similar, o sistema conta com sensibilidade a depender do modo de operação, 131, 65,5, 32,8 e 16,4 LSB/(°/s) , além disso, conta com um ADC de 16 *bits*. Como na aplicação em questão espera-se que a velocidade angular média observada seja relativamente baixa, muito provavelmente será possível utilizar modos de operação com um *range* menor, o que garantirá uma maior sensibilidade ao processo. Ainda em relação a sensibilidade, o dispositivo possui

um fator de tolerância de ±3% em relação sensibilidade nominal operando a 25°C e uma variação da sensibilidade com a temperatura de ±2% e, por fim, uma não-linearidade de 0.2%. Como a aplicação se dará em condições normais de temperatura e pressão, características relativas a temperatura não afetaram de forma apreciável a sensibilidade observada, de forma que inclusive é possível contar com um fator de tolerância relativamente satisfatório.

Considerando erros relacionados ao *zero-bias*, o dispositivo trabalha com tolerância de ±20 °/s em relação ao zero real e uma variação ±20 °/s com a temperatura, respeitando o intervalo de -40°C a 85°C. Como a IMU possui capacidade de compensação de erros, boa parte desses fatores se tornam bem menos significativos, garantido uma menor necessidade de calibração do dispositivo assim como menores erros associados.

Por fim, de forma similar ao que foi apresentado na subseção 2.9.2.4, os giroscópios também sofrem com problema similar no que diz respeito a taxa de amostragem, porém, contam com uma taxa de amostragem programável diferente, de 4Hz a 8Khz. Note que as taxas de amostragem apresentadas são relativas ao *clock* interno do dispositivo, ao fornecer um clock externo é possível aumentar essas taxas, mas não de maneira apreciável.

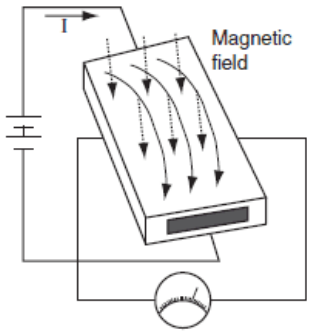
2.9.4 Magnetômetro

Magnetômetros são sensores capazes de detectar campos magnéticos que podem ser aplicados para medir tanto a direção como a força desses campos a depender da sua construção (WILSON, 2005). Esses sensores possuem diversas aplicações a depender do dispositivo utilizado, entre as aplicações mais comuns e importantes se destaca a medição do campo magnético terrestre. Para a aplicação de odometria inercial a medição do campo magnético terrestre permite que um dispositivo conheça sua orientação em relação ao norte magnético de forma bem similar a operação de uma bússola clássica e, com a utilização das informações obtidas, é possível obter estimações mais precisas da posição devido a fusão de não somente os dados dos acelerômetros e giroscópios, mas também dos magnetômetros da IMU e, naturalmente, da câmera utilizada.

Esses sensores podem funcionar com base em diversos princípios diferentes, mas de forma geral, a maioria se baseia em um dispositivo magnético sensível a campos magnéticos. Nas IMUs é bem comum o uso de sensores magnéticos de efeito Hall em cada um dos eixos de medição do dispositivo. O funcionamento de sensores com base no efeito Hall se dá através do estabelecimento de uma tensão, a chamada tensão Hall, relacionada com as características do campo magnético a qual o dispositivo foi exposto.

A Figura 40 ilustra a base do funcionamento, é circulada uma corrente no dispositivo em direção estabelecida, caso o dispositivo interaja com um campo magnético externo, os elétrons sofrem deflexão de forma que as extremidades do dispositivo sejam carregadas com polaridades opostas, dessa forma se estabelece uma tensão no dispositivo.

Figura 40 – Funcionamento - Magnetômetro de efeito Hall



Fonte: Wilson (2005)

2.10 Odometria Visual e Inercial

Apesar das vantagens da odometria visual, como a estimação de trajetórias acuradas com erros de posição baixíssimos, o sistema ainda sofre com uma taxa de saída limitada, ambiguidade na escala em um sistema monocular e falta de robustez em movimentos rápidos e ambientes com pouca textura (SCARAMUZZA; ZHANG, 2019), conforme descrito na seção 2.3. Por outro lado, IMUs não são afetadas por tais adversidades, visto que elas não dependem das cenas do ambiente do sistema, possuem uma alta taxa de amostragem, e não são tão afetadas por movimentos rápidos quanto uma câmera. Além disso, a desvantagem da IMU relacionada ao acúmulo de erro na aquisição de dados poderia ser mitigada ao se comparar os dados do sistema inercial com os da odometria visual.

Percebe-se que as desvantagens de cada sistema são contornadas com as vantagens do outro. Dessa forma, a combinação da odometria visual com um sistema inercial, conhecida como odometria visual e inercial (do inglês, *Visual and Inertial Odometry* - VIO), é uma alternativa viável para a aplicação em questão, já que mescla as vantagens de ambos os dispositivos. Ela é atualmente utilizada em veículos autônomos, navegação robótica, realidade virtual (do inglês, *Virtual Reality* - VR) e realidade aumentada (do inglês, *Augmented Reality* - AR) (ARBABMIR; EBRAHIMI, 2019). A Figura 41 mostra o resultado da estimação da trajetória de um veículo como uma VIO monocular.

Figura 41 – Estimação da trajetória de um veículo com VIO e *Adjusted VIO - AVIO*

Fonte: Arbabmir e Ebrahimi (2019)

A VIO possibilita maior robustez e confiabilidade ao sistema, assim como promove ganhos significativos na precisão dos resultados, através da fusão dos dados dos elementos envolvidos no processo (SCARAMUZZA; ZHANG, 2019). Na aplicação proposta, será utilizada uma VIO monocular, que inclui uma câmera monocular e um sensor IMU, sendo uma escolha adequada, visto o peso e tamanho reduzido, com um ótimo desempenho computacional.

2.11 Filtro de Kalman

O filtro de Kalman é um algoritmo utilizado para estimar estados de um sistema linear invariante com o tempo sujeito a erros ou perturbações de natureza estocástica (DORF, 2011).

Apesar do filtro de Kalman só poder ser aplicado a sistemas lineares, existem variações ou extensões do algoritmo original que permitem trabalhar com sistemas não-lineares como, por exemplo, o *Extended Kalman Filter* (EKF) e o *Unscented Kalman Filter* (UKF).

O EKF trabalha com a linearização analítica das equações que descrevem a dinâmica do sistema, o UKF por outro lado trabalha com linearização estatística, fornecendo um desempenho mais eficiente no que diz respeito a propagação temporal da média, e da matriz de covariância de estado do sistema, a serem comentados posteriormente (AGUIRRE, 2015).

Esse algoritmo consiste em um processo iterativo composto por um conjunto de equações que utiliza uma entrada ou um conjunto de entradas consecutivas para estimar o valor verdadeiro de algum tipo de informação como, por exemplo, velocidade, posição, e

aceleração.

De forma resumida, ele funciona tomando uma estimativa inicial para o valor verdadeiro e um erro inicial associado a esse valor e, a medida que novos dados de entrada passam a chegar, essas duas estimativas inicias são corrigidas para refletir a contribuição dos novos dados e são propagadas para o instante subsequente.

A medida que mais dados são incorporados, espera-se que a variação se torne cada vez menor de forma que o valor estimado esteja cada vez mais próximo do valor real. Esse processo de estimação possui uma convergência muito mais rápida quando se comparado com métodos mais simples como, por exemplo, através do uso de uma média e, portanto, é ideal para promover um melhor desempenho ao sistema.

Para a aplicação de odometria visual proposta, conforme especificado em oportunidade anterior, a obtenção dos dados se dará de forma discreta e, portanto, será comentado aqui apenas tal caso de aplicação.

O caso discreto pode ser subdividido em dois casos específicos, caso estático e caso dinâmico. A diferença básica entre esses dois casos está no sistema em análise, no caso dinâmico o sistema apresenta leis que ditam a dinâmica do sistema como, por exemplo, uma lei de movimento a qual está sujeito, no caso estático isso não se aplica.

No que diz respeito ao algoritmo, essas diferenças se traduzem em diferentes estruturas ou etapas, no caso estático as correções de estimativa e estado podem ser propagadas diretamente para realização da nova estimativa no instante de tempo subsequente, no caso dinâmico, porém, é necessário incorporar uma etapa intermediária, a etapa de predição, em que, a partir da lei que descreve a dinâmica do sistema e suas incertezas associadas, é realizada uma previsão do novo valor provável das variáveis de estado, e só depois é incorporada a nova medição de forma a realizar a correção da estimativa. Em resumo, tem-se uma estimativa a priori para o estado do sistema, utiliza-se a lei que descreve a dinâmica do sistema para prever o novo estado e através da nova medição se corrige o resultado.

2.11.1 Caso estático

Conforme apresentado na subseção 2.9.1, uma unidade de medida inercial está sempre sujeita a diversas fontes de erro, seja devido aos elementos que a compõe, seja por fatores externos (GROVES, 2008) (LEIBSON, 2019). Dentre as diversas fontes de erro existentes, é possível citar, por exemplo, mudanças na condição de operação, desvios nos sensores, a construção dos sensores, e assim por diante.

Dessa forma, é importante garantir que as estimativas obtidas serão as mais próximas possíveis dos valores que se tem como verdadeiros, portanto, é necessário minimizar os erros associados a todo o processo de medição e estimação de forma a garantir um resultado mais preciso. Uma maneira de reduzir esses erros é através do uso de técnicas de fusão de

dados, técnicas essas que permitem combinar medidas obtidas por diversas fontes e obter um resultado mais satisfatório, e possivelmente mais próximo ao valor real esperado, caso os dados sejam adequados. O filtro de Kalman é uma das possíveis ferramentas que permite realizar tal procedimento e será abordado mais a fundo nesta subseção, subseção 2.11.1.

Para entender o funcionamento da ferramenta é possível introduzir um caso de análise qualquer, por exemplo, a obtenção da posição de um objeto ou elemento. Quando se realiza tal medição, independente de quão preciso for o sensor, espera-se observar erros advindos das mais diversas fontes, sejam erros externos de natureza estocástica ou não, sejam erros associados aos instrumentos e sistema utilizado. Devido a essas característica é possível tomar a variável a ser medida como uma variável aleatória e defini-la por meio de funções de probabilidade.

Conforme ideia exposta por Aguirre (2015), considerando uma variável de posição **p** e uma única medida realizada por um instrumento, p_1 é possível definir essa variável por meio de um função de densidade de probabilidade condicional $f(p|p_1)$ que definiria qual o valor mais provável para a variável **p** dado a medição p_1, ou seja, qual a melhor estimativa para a posição real dada a medição p_1 realizada.

Como o objetivo da fusão de dados é considerar dados de diversas fontes e combiná-los de forma a obter uma melhor estimativa, é possível estender a ideia para N medidas realizadas, por exemplo, para os sensores da unidade MPU 6050 (INVENSENSE, 2012) e qualquer outro dado que se deseje incorporar, dessa forma define-se a função de densidade de probabilidade condicional $f(p|p_1, p_2, ..., p_n)$.

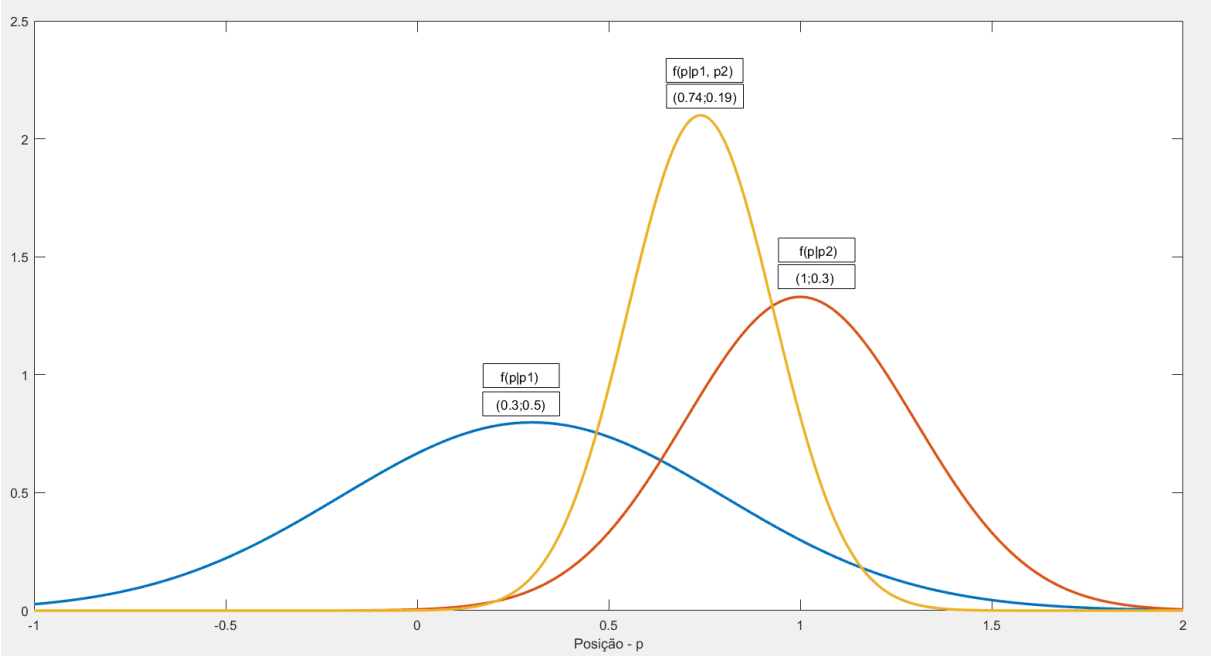
Apesar de ser difícil trabalhar com funções do tipo $f(p|p_1, p_2, ..., p_n)$ diretamente, é possível aproximá-la por uma Gaussiana e lidar somente com a média e desvio padrão para defini-la. De forma similar, devido a natureza aleatória imprevisível dos erros associados ao processo, eles também são tratados de forma estatística e aproximados por uma Gaussiana. Naturalmente, os sinais podem seguir diversos tipos de distribuição de probabilidade, porém, considera-se o uso de Gaussianas por simplicidade.

A Figura 42 mostra um exemplo de como é feita a combinação de dados e as vantagens de se utilizar tal ferramenta. São apresentadas duas medições, p_1 tal que $f(p|p_1)$ é definida por média igual 0.3 e desvio padrão igual a 0,5, e p_2 tal que $f(p|p_2)$ é definida por média igual 0.3 e desvio padrão igual a 0,5, ao se utilizar a função combinada, $f(p|p_1, p_2)$, obtêm-se um distribuição com um desvio menor que qualquer um dos desvios que compõe a combinação e, portanto, mais preciso que qualquer uma das estimativas obtidas, representando uma nova estimativa. Essa função tem os dois momentos, média e desvio padrão definidos pela Equação 2.32 e Equação 2.33, em que μ apresenta a nova estimativa e σ o novo desvio padrão associado.

$$\mu = \frac{\sigma_2^2}{\sigma_1^2 + \sigma_2^2} p_1 + \frac{\sigma_1^2}{\sigma_1^2 + \sigma_2^2} p_2 \tag{2.32}$$

$$\sigma^2 = \left(\frac{1}{\sigma_1^2} + \frac{1}{\sigma_2^2} \right)^{-1} \tag{2.33}$$

Figura 42 – Combinação de distribuições



Fonte: adaptada de Aguirre (2015).

É importante salientar ainda que esse resultado pode ser estendido para N funções associadas a dispositivos ou fontes diferentes de forma a produzir $f(p|p_1, p_2, ..., p_n)$, conforme apresentado anteriormente e, portanto, pode ser utilizado para realizar a fusão de dados de todas as estimativas obtidas pelos sensores da IMU e também da estimativa obtida através da captura das imagens da câmera.

2.11.2 Caso dinâmico

Introduzido brevemente na subseção 2.11.1 e apresentado em seu caso estático para realização do procedimento fusão de dados, nesta subseção o filtro de Kalman será considerado em seu caso dinâmico para realização da odometria inercial, ou seja, estimação de posição e orientação através do uso de sensores inerciais.

Desde seu descobrimento, ou mais precisamente, seu desenvolvimento, o filtro de Kalman tem sido uma ferramenta essencial para diversos campos de aplicação. Sua evolução tem sido contínua ao longo dos anos, seja devido a problemas matemáticos e indeterminismos eventuais que podem surgir em determinados sistemas, seja pela necessidade de resultados cada vez mais robustos e que utilizem menos recursos computacionais.

Quase todos os sistemas físicos são dinâmicos a algum nível, portanto, para retirar informações ou estimativas precisas desses sistemas é preciso levar a dinâmica em consideração. Porém, nem sempre essa dinâmica é conhecida ou muito precisa, levando a resultados insatisfatórios. Uma das formas de lidar com essas incertezas é a utilização de métodos estatísticos, ou seja, sabendo que somos ignorantes em relação às características dos sistemas, fazemos o melhor para modela-las através da probabilidade.

Considerando um sistema dinâmico, nem sempre em sistemas de controle é possível medir uma ou mais variáveis de interesse diretamente por diversos motivos, seja devido a limitações físicas e tecnológicas, seja devido aos elevados custos associados a operação. O filtro de Kalman fornece um método prático capaz de inferir essas informações que de outra forma não estariam disponíveis através de medidas indiretas e ruidosas linearmente relacionadas ao estado do sistema. Assim sendo, o filtro de Kalman apresenta um meio prático para separar o sinal e o ruído. Neste projeto, serão considerados apenas sistemas lineares perturbados por ruídos brancos.

De forma bastante genérica, o filtro de Kalman é um estimador de estados. Considerando que um estado geralmente é composto de uma ou mais informações de interesse para o sistema, essencialmente o que se têm é uma ferramenta capaz de aproximar um conjunto de informações dos seus valores reais. Informações redundantes também podem ser incorporadas na estimação.

O filtro de Kalman é um estimador considerado ótimo, sendo que o critério de otimalidade é a minimização do erro quadrático médio de uma variável aleatória X utilizando toda a informação disponível (BROWN; HWANG, 2012). Um ganho deve ser definido de forma que a estimava atualizada seja ótima, ou seja, possua uma variância mínima.

Conforme apresentado, o filtro se baseia no sistema linear de equações, sendo definidas na Equação 2.34 e Equação 2.35. Essas equações são essencialmente a descrição do sistema de controle em análise no espaço de estados. Apesar de existirem diversas técnicas e formulações diferentes para derivar a representação em espaço de estados de um sistema, nesse projeto essa representação será obtida diretamente das equações que regem a dinâmica do sistema.

$$x[k+1] = A.x[k] + B.u[k] + w[k] \tag{2.34}$$

$$y[k] = H.x[k] + z[k] \tag{2.35}$$

Nas Equações 2.34 e 2.35, A, B e H definem respectivamente a matriz de estado, a matriz de entrada, e a matriz de saída, também chamada de matriz de medições. Essas matrizes definem toda a dinâmica do sistema, incluindo como o sistema responde a entradas e como a saída se relaciona com os estados definidos.

Ainda em relação às referidas equações, x, u e y representam respectivamente o vetor de estado, o vetor de entrada, e a saída medida. Por fim, w e z representam as contribuições do ruído na dinâmica do sistema, sendo respectivamente o ruído do processo e o ruído da medida. Define-se ainda $w[k] \sim N(0, Q_k)$ e $z[k] \sim N(0, R_k)$, conforme Grewal e Andrews (2008).

Do ponto de vista de informação, x contém o estado atual do sistema que pode ser estimado a partir das medidas do vetor y e da entrada u, além disso, y não pode ser a única fonte de informação para obter y devido ao ruído associado.

O filtro de Kalman pode ser divido essencialmente em dois estágios, previsão, também chamado de *time update*, e correção, também chamado de *measurement update*. As etapas aqui apresentadas foram adaptadas das exposições relativamente simples apresentadas por Brown e Hwang (2012) e Grewal e Andrews (2008).

Na primeira etapa calcula-se uma previsão para o novo estado do sistema e para a matriz de covariância associada a partir das informações disponíveis conforme as Equações 2.36 e 2.37.

$$x_{k+1}^{p} = A.x_{k}^{u} + B.u_k \tag{2.36}$$

$$P_{k+1}^{p} = A.P_{k}^{u}.A^{T} + Q \tag{2.37}$$

Para as equações definidas em 2.36 e 2.37, x_{k+1}^{p} e P_{k+1}^{p} representam a estimativa do estado associado ao sistema e a estimativa da matriz de covariância para o instante atual respectivamente. De forma similar, x_{k}^{u} e P_{k}^{u} também representam o estado e a matriz de covariância, porém, advindos da etapa de atualização do instante de amostragem anterior, t = k. Os sobrescritos u e p indicam associação da informação com a etapa de atualização (do inglês, *update*) e com a etapa de predição ou estimação (do inglês, *predict*) respectivamente.

Durante a etapa de atualização as informações obtidas são incorporadas de forma a obter uma estimativa atualizada para o estado do sistema. Nessa etapa têm-se essencialmente uma estimativa a priori obtida na etapa de predição e as medições observadas. Essas informações devem ser combinadas por meio de um ganho apropriado, chamado de ganho de Kalman, para gerar um estimativa a posteriori ou atualizada do estado do sistema e também da matriz de covariância associada ao sistema, conforme as Equações 2.38, 2.39 e 2.40.

$$K_k = P_{k}^{p}.H^{T}.(H.P_{k}^{p}.H^{T} + R)^{-1} \tag{2.38}$$

$$X_{k}^{u} = x_{k}^{p} + K_k.(Y_k - H.x_{k}^{p}) \tag{2.39}$$

$$P_k^u = P_k^p - K_k.H.P_k^p \tag{2.40}$$

Nas equações acima, 2.38, 2.39 e 2.40, K_k é o ganho de Kalman para o instante atual, X_k^u é a estimativa do estado atualizada para o instante atual, Y_k é a informação da medida a ser incorporada no instante atual, e P_k^u é a matriz de covariância estimada atualizada para o instante de tempo atual

Para definir as matrizes Q e R nas Equações 2.37 e 2.38 assume-se que o ruído associado ao processo e o ruído associado às medições não apresentam correlação entre si e possuem média zero (ROMANIUK; GOSIEWSKI, 2014). Para satisfazer esses requisitos a matriz de covariância do ruído associado ao processo e do ruído associado as medições são definidas de acordo com as Equações 2.41 e 2.42 adaptadas de Brown e Hwang (2012), a correlação entre as duas medidas é definida pela Equação 2.43.

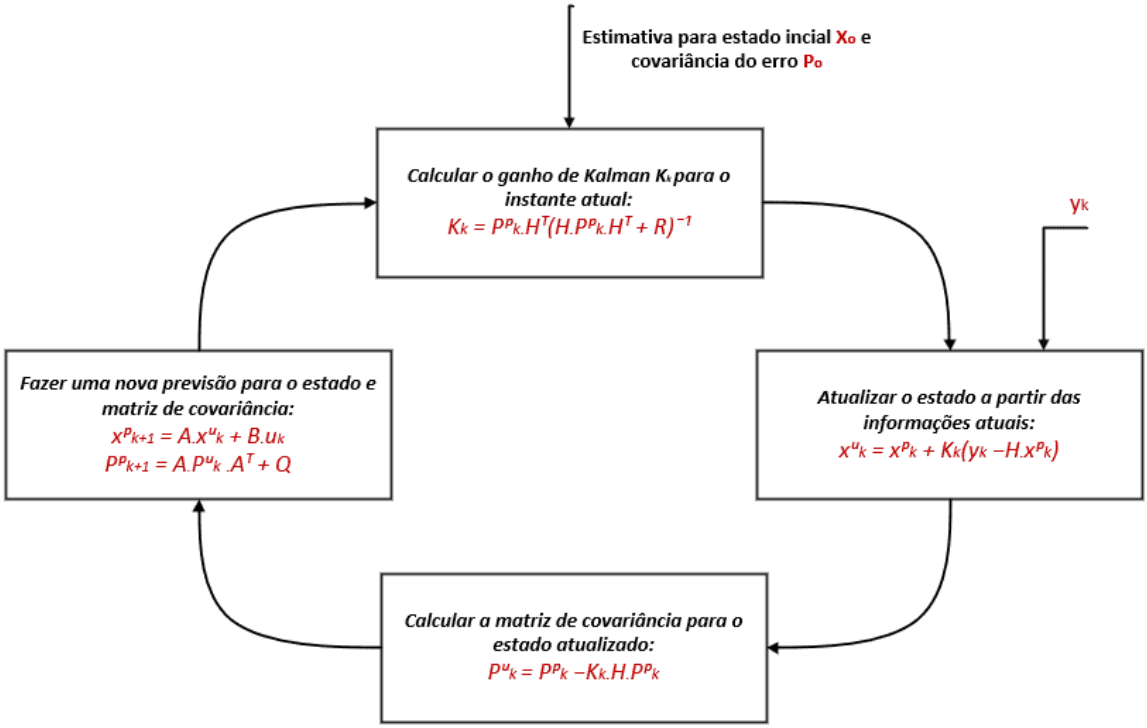
$$Q = \begin{cases} \mathrm{E}[w_k.w_j^T], & k = j \\ 0 & \mathrm{k} \neq \mathrm{j} \end{cases} \tag{2.41}$$

$$R = \begin{cases} [E[z_k.z_j^T], & k = j \\ 0 & \mathrm{k} \neq \mathrm{j} \end{cases} \tag{2.42}$$

$$E(w_k.z_j^T) = 0 \; \forall \; i, j \tag{2.43}$$

A Figura 43 apresenta o ciclo básico do algoritmo e as etapas envolvidas. Note que no instante inicial o ciclo se inicia na etapa de atualização pois nesse instante X_o e P_o já são as melhores estimativas disponíveis a priori para o estado inicial e para a covariância do erro respectivamente.

Figura 43 – Ciclo de operação do algoritmo - Filtro de Kalman

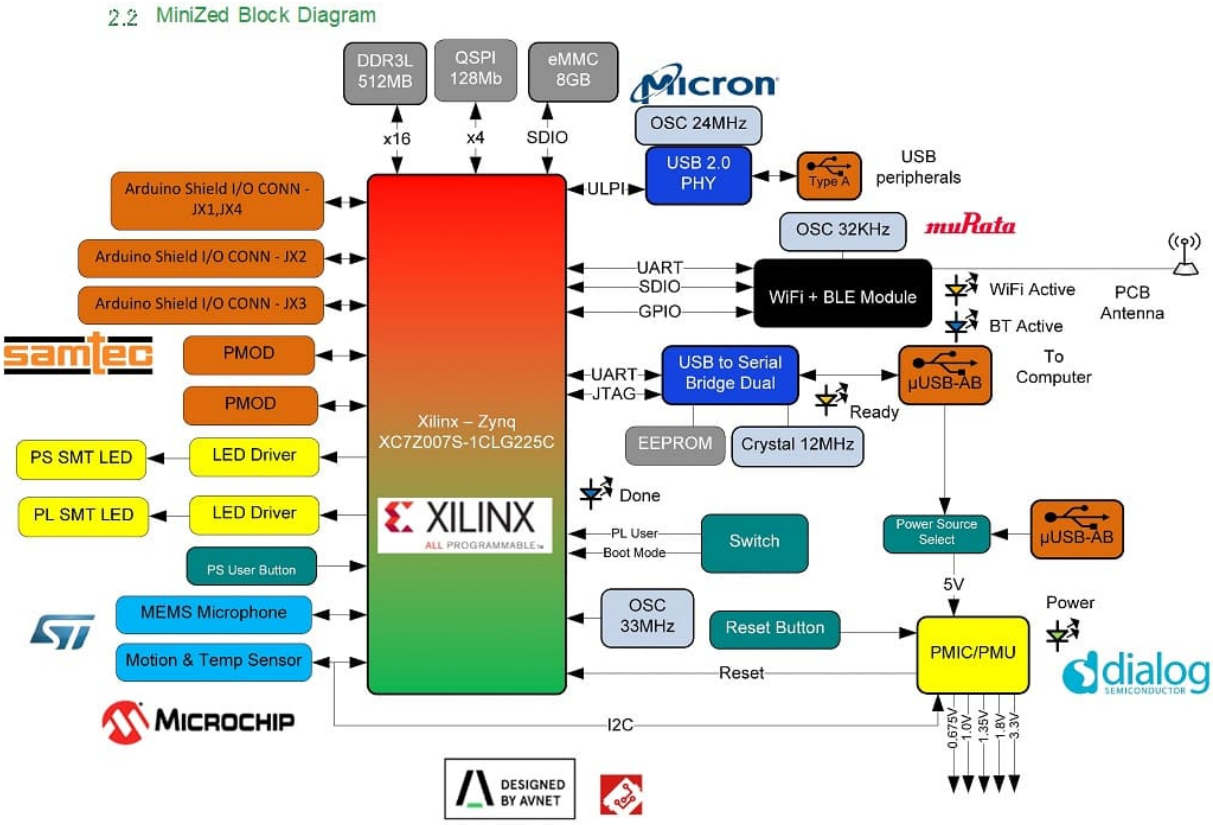


Fonte: adaptada de Brown e Hwang (2012)

2.12 Placa de desenvolvimento - Minized

A estimação do movimento por meio do uso de câmera e IMU será realizada através da placa Minized, desenvolvida pela Avnet com a Xilinx. Trata-se de uma placa de desenvolvimento de baixo custo com uma eficiente capacidade de processamento. Ela é constituída de um processador *Single-core* ARM Cortex-A9, que se comunica com diversas interfaces, conforme sumarizado na Figura 44.

Figura 44 – Diagrama de blocos da Minized



Fonte: Avnet e Xilinx (2020)

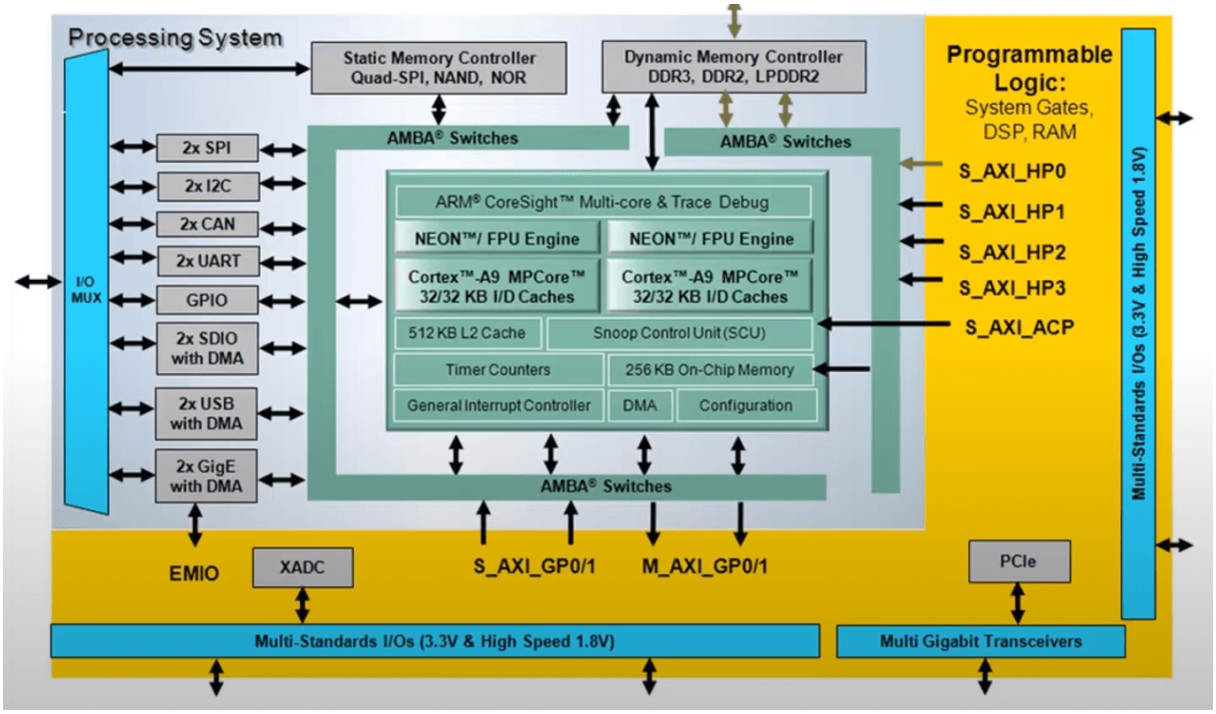
Especificamente, a Minized é caracterizada por (AVNET; XILINX, 2020):

- Processador *Single-core* ARM Cortex-A9 com frequência máxima de 667 MHz, 256 KBytes de on-chip RAM e 84 pinos I/O no sistema processador;
- Suporte para 3 tipos de memória externa: Micron 512 MB DDR3L, Micron 128 MB QSPI flash e Micron 8GB eMMC mass storage;
- Arquitetura lógica-programável equivalente a um Artix -7 FPGA com 54 pinos I/O High Range;
- Configuração e debug: On-board USB para JTAG e circuito debug UART
- Comunicação Wi-Fi, Bluetooth e USB
- Conector expansor para compatibilidade com Arduino *shield* e *Peripheral Module Interface* - Pmod;
- Sensores acelerômetro, de temperatura ST Micro LIS2DS12 e microfone MEMS digital;
- Interface com o usuário, por meio de LEDs, botões e switches;
- Osciladores de precisão CMOS de baixo consumo;

2.12.1 Arquitetura

Arquitetura básica do sistema (AVNET; XILINX, 2017a) é definida pelo sistema de processamento e por um lógica programável, tem como principal componente o processador Single-core ARM® Cortex™-A9 que é parte integrante do sistema de processamento, o sistema de processamento inclui também memórias e periféricos integrados. A lógica programável é composta por uma FPGA serie 7 baseada na família Artix-7 e permite um projeto de *hardware* flexível e customizado, permite também o projeto de aceleradores de *hardware* tanto proprietários do fabricante, Xilinx, quanto de outros fornecedores. A Figura 45 apresenta a arquitetura dos dispositivos Zynq-7000.

Figura 45 – Arquitetura dos dispositivos Zynq-7000



Fonte: Avnet e Xilinx (2020)

O sistema tem acesso a I/Os flexíveis permitindo uma melhor conexão da placa a dispositivos externos. Múltiplos periféricos podem ser configurados na plataforma do sistema de processamento através das diversas entradas, SPI, I2C, CAN, UART, GPIO, entre outras.

Definindo de forma bastante resumida a composição , no centro da arquitetura temos o processador *single-core* com uma cache de 512KB (L2 cache), assim como uma memória compartilhada integrada de 256KB. Existe também um bloco de configuração que permite a configuração da lógica programável com o sistema de processamento por meio de *bitstreams*. Para lidar com os mais diversos tipos de interrupções o sistema conta com um controlador de interrupções genérico que permite que periféricos externos chamem a atenção do processador, esse controlador utiliza mecanismos de *spin-lock* para controle de acesso, além disso, tem acesso a semáforos. Em relação as memórias, o sistema de processamento

tem um controlador de memória tipo DDR, que suporta até 1GB de RAM, no hardware esse controlador está conectado a uma memória RAM SD DDR3. Por fim, o sistema possui também um controlador de memória estática, responsável por armazenar *bitstreams*, e executáveis, por exemplo.

2.12.1.1 Sistema de processamento

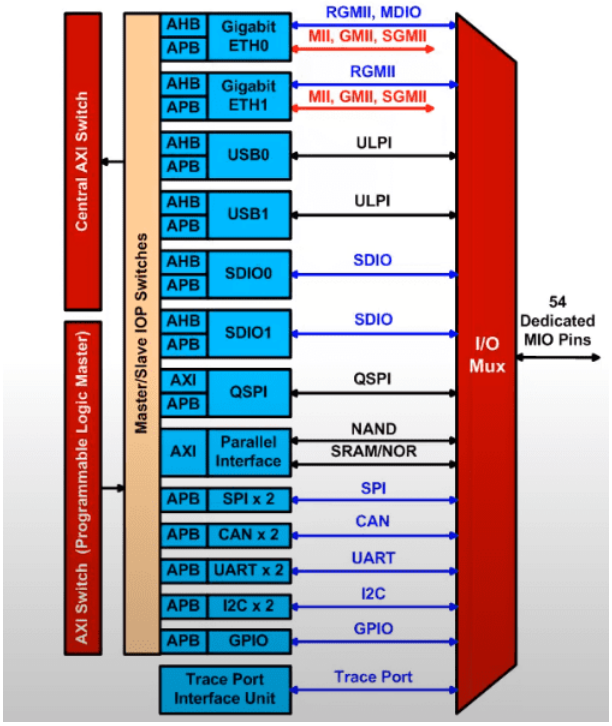
Em relação ao sistema de processamento, conforme definido em Avnet e Xilinx (2017a) e Avnet e Xilinx (2020), a MiniZed Zynq 7Z007S SoC conta um um processador *single-core* ARM® Cortex™-A9 com uma frequência máxima de 667MHz. Em relação a arquitetura, é utilizada a arquitetura Harvard com 64 *bits* para instruções e 64 *bits* para dados. O sistema conta também com memória cache L1 e L2. A cache L1 é de 32KB, 32KB para instruções, 32KB para dados, que utiliza associação por conjunto do tipo *4-way*, tipo *write-back* em que não há escrita imediata na memória principal. A cache L2 é de 512KB com associação por conjunto *8-way* suportando tanto *write-through*, em que a escrita é feita tanto na cache como no local de origem, como *write-back*.

Para o controle de memória o sistema conta com um controlador de alta largura de banda e suporta uma variedade de tipos de memória como, por exemplo, DDR3, DDR2 e outros. Em relação ao armazenamento, o sistema possui memória RAM integrada de 256 KBytes e possui capacidade de armazenamento externo de 512 MBytes de DDR3L tanto para uso de RAM como para armazenamento de programas, 128 Mbits de QSPI NOR Flash e até 8 Gbytes de eMMC ou SD via interface PMOD externa.

O processador também conta com coprocessadores chamados NEON que estendem o conjunto de instruções ARM para atender a aplicações de áudio, vídeo, gráficos 3-D, processamento de imagens e áudio.

Em relação aos periféricos, o sistema tem disponível interfaces integradas capazes de atender a periféricos de diversos tipos como, por exemplo, USB, SD, UART, I2C, SPI e GPIO que são mapeadas diretamente na interface *Multiplexed I/O* (MIO) no sistema de processamento ou através da interface *Extended Multiplexed I/O* (EMIO) na lógica programável, a ideia é apresentada na Figura 46, como o mapeamento é feito na mesma interface, por vezes dispositivos são mapeados no mesmo endereço e, portanto, é necessário a realização de um *trade-off* entre certas funções. A placa garante suporte a diversos fabricantes e, portanto, permite o uso de uma ampla gama de dispositivos.

Figura 46 – Periféricos disponíveis e mapeamento



Fonte: Avnet e Xilinx (2017a)

Por fim, para fazer a integração de todas as funções no sistema de processamento, o *Snoop Control Unit* (SCU) conecta o processador, as memórias e periféricos do sistema através das interfaces AXI, garantindo a coerência das memórias caches e provendo todos os serviços necessários.

2.12.1.2 Lógica programável

Conforme especificado em Avnet e Xilinx (2020), além do sistema de processamento, a MiniZed Zynq 7Z007S SoC conta também com uma *Field-Programmable Gate Array* (FPGA) que permite a implementação de circuitos em *hardware* de forma a complementar a funções do sistema de processamento, garantindo um melhor desempenho e flexibilidade.

Uma FPGA essencialmente é um chip que permite a implementação de circuitos lógicos definidos por uma forma de descrição de *hardware*, ou seja, uma *Hardware Description Language* (HDL). Esses circuitos podem ser construídos abrangendo um amplo espectro de complexidade, desde aplicações bastante simples à extremamente complexas. Em relação ao tamanho, esses circuitos estão limitados a construção do dispositivo utilizado.

A FPGA consiste em um arranjo de células lógicas ou blocos lógicos configuráveis, blocos de entrada e saída e também chaves de interconexão, todos contidos em um único circuito integrado. Cada uma das células lógicas ou blocos lógicos permite a implementação de funções lógicas e estão dispostas em formato de matriz com duas dimensões, um *array*

2-D. As chaves de interconexão são responsáveis por realizar a conexão e comunicação entre cada uma dessas células lógicas.

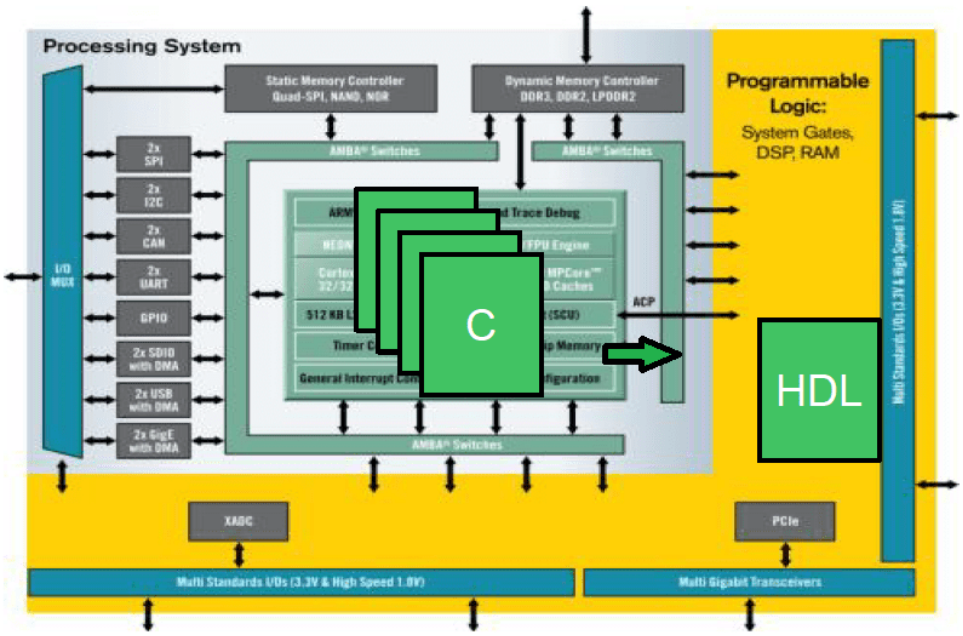
Para entender como o uso da FPGA pode impactar no sistema é preciso compreender o comportamento do sistema de processamento. Se o processador está sobrecarregado, ou seja, há várias tarefas a serem realizadas e/ou que se repetem com frequência, um aumento no tempo de resposta das tarefas pode ser ocasionado, comprometendo o desempenho geral do sistema. Especificamente, pode ocorrer um aumento na latência e no tempo de resposta do tratador de interrupções e uma diminuição na vazão de dados. Além disso, a natureza sequencial dos processadores é outro agravante para a queda de desempenho geral do sistema, visto que ela limita as operações e a velocidade das tarefas.

Para resolver os problemas apresentados tanto quando se lida com sistemas críticos como quando se deseja obter um melhor desempenho, é possível realizar processamento paralelo com o uso da FPGA.

Através do uso da FPGA é possível particionar os requisitos de projeto do sistema através da transferência de funções que poderiam estar rodando no processador para o *hardware* implementado, conforme apresentado na Figura 47. Essa aceleração de *hardware* permite escalar o desempenho do *software* significativamente para lidar com diversas aplicações, principalmente pelo fato de proporcionar paralelismo, mas também por permitir que operações de alto custo computacional possam ser delegadas para o *hardware*.

Como a lógica implementada é reprogramável, uma atualização em tempo real pode ser realizada, o que permite que o sistema lide com problemas complexos. Cita-se, como exemplo, algoritmos adaptativos que utilizam tal abordagem, como SRD, aplicações de video, criptografia, e processamento de dados.

Figura 47 – Delegação para lógica programável



Fonte: adaptado de Avnet e Xilinx (2017a)

Além das questões já citadas, essa integração permite reduzir também o consumo e a potência. Isso é bastante importante para as portas de entrada e saída pois reduz ruídos nas interfaces de conexão, o *electromagnetic noise* (EMI).

Em relação as características da lógica programável contida na MiniZed Zynq 7Z007S SoC, conforme Avnet e Xilinx (2020) e Avnet e Xilinx (2017a). A placa contém uma arquitetura lógico programável equivalente a uma FPGA serie 7 baseada na família Artix-7. Além disso, possui um total de $23x10^3$ células lógicas e, portanto, é de pequena capacidade quando se comparado com modelos mais robustos. Por fim, ela conta com uma memória RAM de 1,8Mb composta por 50 blocos de 36Kb (50x36Kb), e também com 54 pinos I/O de 1.2V a 3.3V.

Em relação as conexões, a conexão entre a lógica programável e o sistema de processamento é feita através de $3x10^3$ conexões de alto desempenho, além disso, o sistema utilizado permite uma baixa latência entre as interfaces de forma a permitir o co-processamento assim como uma alta vazão de dados para transferência, garantindo também que se mantenha uma alta taxa de vazão de dados sempre que possível.

Por fim, a conexão entre a lógica programável e o sistema de processamento é feito através de uma interface AMBA AXI garantindo as características já apresentadas. Existem um total de 9 portas, 4 portas dedicadas, 4x 32 *bits*, tipo mestre-escravo, com 2 mestres e 2 escravos, 4 portas de alto desempenho do tipo mestre, 4x64 *bits*, e 1 porta ACP, que permite conectar a lógica programável diretamente as caches L1 e L2, servindo como um meio de troca de dados de baixa latência entre o processador e qualquer acelerador implementado em *hardware*.

3 Desenvolvimento do sistema de estimação de posição e orientação de câmeras inteligentes

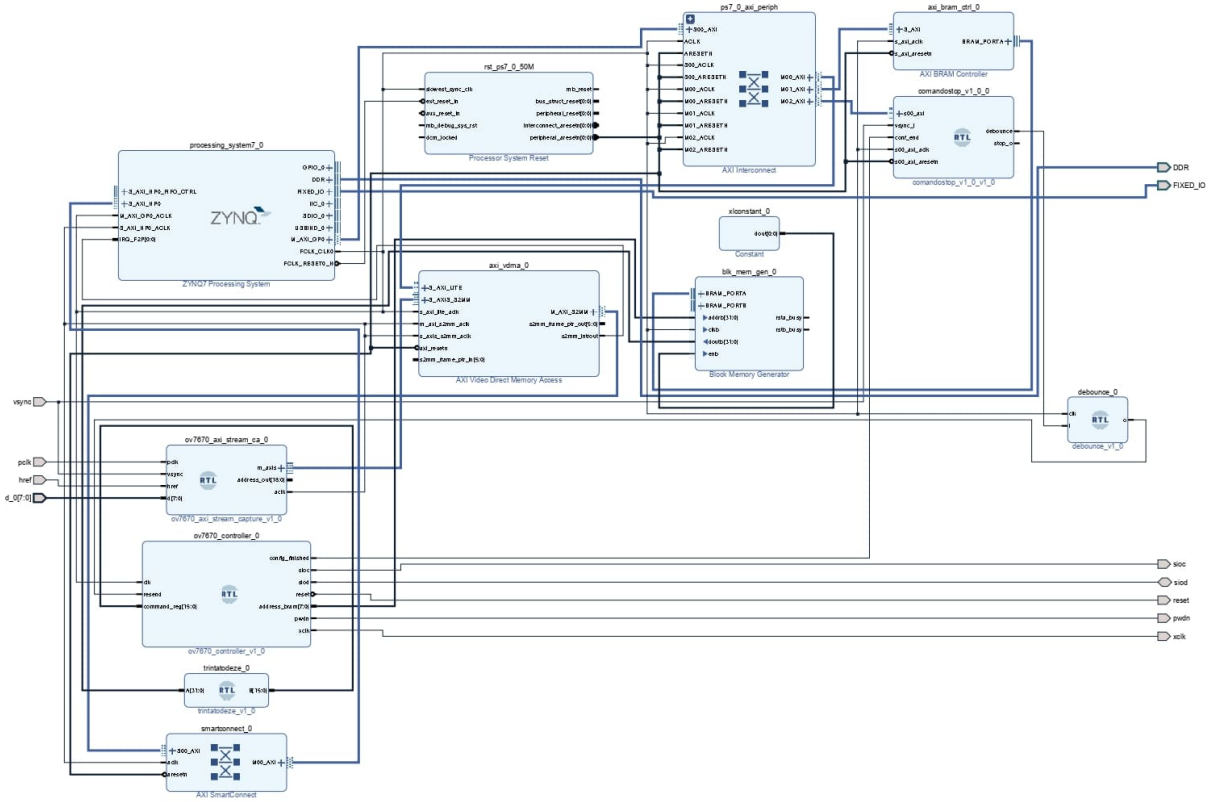
Conforme descrito na seção 1.3, objetivo geral do sistema é a implementação de um sistema embarcado que estime a posição e orientação de câmeras inteligentes, através de imagens capturadas pela própria câmera e pelos dados providos por uma IMU em uma plataforma híbrida. Dessa forma, este capítulo tratará da integração da câmera e dos sensores inerciais à Minized, e da seleção e implementação de algoritmos de odometria visual e odometria inercial.

3.1 Integração da câmera à plataforma minized

A câmera utilizada para integrar na Minized é a OV7670 da *Omnivision*. É um sensor de imagem CMOS de baixo consumo de potência que pode operar em até 30 quadros por segundo (do inglês, *Frames Per Second - FPS*) em resolução VGA (640x480 pixels). A câmera também possui funcionalidades de processamento de imagem, como controle de exposição, saturação de cor e outros, programáveis através da interface, e também possui tecnologia de melhoramento de qualidade de imagem ao reduzir ou eliminar ruídos (OMNIVISION, 2006).

A integração da câmera OV7670 na Minized foi feita baseado no desenvolvimento realizado por Silva et al. (2021), que utilizaram o *kit* ZCU104 da *Xilinx* e o sensor OV7670. Nesse desenvolvimento, a ZCU104 recebe o *stream* de *pixels* e armazena a imagem de entrada no *buffer* de aquisição de *frames* para serem processados. Após o processamento de imagem, a imagem de saída é armazenada no *buffer* de visualização de *frames*, que é lido através de acesso direto à memória (do inglês, *Direct Memory Access*- DMA) pelo processador (SILVA et al., 2021). O sensor OV7670 foi configurado na resolução QVGA, 320×240 *pixels*, alcançando até 60 FPS. Os *pixels* são convertidos para a saída com codificação RGB444, que é convertida para escala de cinza. A Figura 48 ilustra o esquemático do *hardware* para o processo de captura de *frames*. Dentre os blocos, vale destacar *AXI Video Direct Memory Access*, que possibilita a leitura dos *frames* através do acesso direto à memória e *comandostop*, que permite o processo de interrupção.

Figura 48 – *Block design* no *software* Vivado para captura de *frames*



Fonte: Produzido pelos autores.

Já para integrar a câmera à Minized, o algoritmo foi adaptado de forma a funcionar o mecanismo de *buffer* triplo circular. Basicamente, são capturados três *frames* a cada ciclo por meio do processo de interrupção. Terminado um ciclo, os próximos *frames* capturados são reescritos no mesmo buffer, que pode armazenar 3 *frames* simultaneamente. Dessa forma, a cada ciclo é possível realizar o processamento dos *frames* enquanto eles são capturados. As Figuras 49a, 49b e 49c ilustram a sequência de *frames* capturados em um único ciclo.

Figura 49 – *Frames* capturados pela câmera em um único ciclo

(a) Primeiro *frame* capturado no ciclo

(b) Segundo *frame* capturado no ciclo

(c) Terceiro *frame* capturado no ciclo

Fonte: Lima (2021)

Embora a captura de imagens esteja funcionando corretamente, o sistema possui algumas limitações. Primeiramente, a implementação de um algoritmo de odometria visual é um processo longo, constituído de várias etapas, como detecção e rastreamento de *features*, estimação de rotação e translação etc. Para a implementação do algoritmo na plataforma híbrida, é necessário validá-lo e fazer *debug* em uma CPU. Porém, a transferência de dados entre a Minized e uma CPU é realizada através da UART, que possui uma taxa de transferência de 115200 bps. Assim, transferir um único *frame* com resolução VGA pode demorar unidades de minutos. Dessa forma, validar o funcionamento do algoritmo seria um processo custoso, impossibilitando a implementação do algoritmo na plataforma híbrida nas condições descritas. O problema pode ser contornado ao se utilizar transferência de dados por meio de conexão Wi-Fi, que é uma das funcionalidades da Minized e possui taxa de transferência superior. Porém, estabelecer conexão Wi-Fi na plataforma é um processo complexo e demorado, não sendo possível realizar em tempo hábil no prazo disponível. Portanto, optou-se por não implementá-lo neste projeto. Assim, neste projeto o algoritmo de odometria visual foi implementado em uma CPU utilizando a biblioteca OpenCV (BRADSKI, 2000).

É importante ressaltar que a *Xilinx* possui uma biblioteca baseada em OpenCV, denominada *Vitis Vision Library* (XILINX, 2021b), que é otimizada para dispositivos *Xilinx*. Ela possui um conjunto de mais de 90 *kernels*, que inclui funções necessárias para odometria visual. Dessa forma, a fim de que este projeto seja aproveitado em projetos futuros de implementação de odometria visual na plataforma Minized, as funções da biblioteca OpenCV utilizadas neste projeto foram escolhidas não apenas com base na performance, mas também na garantia de que elas estejam inclusas na *Vitis Vision Library*. Um maior detalhamento sobre a adaptação do algoritmo de odometria visual na Minized pode ser encontrado no Apêndice A.

3.2 Seleção de algoritmos de odometria visual e implementação

A formulação do problema para estimar posição e orientação de uma trajetória se resume em calcular a matriz de rotação e o vetor de translação, que descrevem o movimento da câmera, para cada par de *frames* subsequentes (SCARAMUZZA; FRAUNDORFER, 2011). Para calculá-los, é necessário, dentre outras etapas, estimar o fluxo óptico.

3.2.1 Escolha do método de estimação do fluxo óptico

A escolha do método de estimação do fluxo óptico é crucial para o projeto devido à relação entre custo computacional e aplicações em tempo real. Conforme descrito na subseção 2.5.3, o custo computacional é alto para estimação de fluxo óptico por métodos densos comparado aos métodos esparsos. No experimento de Zhang, Han et al. (2019) ilustrado nas Figuras 19a e 19b, nota-se que, em média, o custo computacional de um método denso pode ser até 200 vezes maior que o custo computacional de um método esparso. Assim, para aplicações em tempo real, que é o caso deste projeto, a estimação de fluxo óptico por métodos esparsos é mais adequado para que os requisitos temporais do projeto sejam cumpridos. Assim, foi adotado o método esparso para estimar o fluxo óptico.

3.2.2 Escolha dos algoritmos de odometria visual

Visto que será adotado método esparso para estimação do fluxo óptico, a odometria visual com abordagem esparsa apresenta as etapas (SCARAMUZZA; FRAUNDORFER, 2012) (ZHANG; HAN et al., 2019):

1. Calibração da câmera.
2. Captura e remoção de distorção dos *frames*.
3. Detecção de *features*.
4. Rastreamento de *features*.
5. Cálculo da matriz essencial.
6. Estimação da rotação e translação.
7. Coleta de dados da IMU para computar a escala absoluta (odometria visual e inercial).

Para cada etapa, há uma gama de algoritmos disponíveis para implementação. Por exemplo, um estado da arte para detecção e rastreamento de *features* é o algoritmo Kanade-Lucas-Tomasi (KLT), conforme ressalta Scaramuzza e Fraundorfer (2012). Este algoritmo é

implementado em OpenCV através da função *GoodFeaturesToTrack* (SHI; TOMASI, 1994), que utiliza o detector de cantos *Shi-Tomasi*, com o método iterativo *Lucas-Kanade* para rastreamento de *features*. No entanto, até a data de desenvolvimento deste projeto, a função *GoodFeaturesToTrack* não está disponível na biblioteca *Vitus Vision Library*, que é uma condição necessária para escolha dos algoritmos de odometria visual, conforme descrito anteriormente. Dessa forma, foram analisados algoritmos que apresentem as melhores performances, que estejam presentes na literatura e que façam parte da biblioteca.

Para a etapa de detecção de *features*, é desejável um algoritmo que apresente a maior média de correspondências detectadas, a maior porcentagem de *matching* corretos e o menor tempo de execução possível. Conforme descrito na subseção 2.5.6, ao se analisar o teste de performance de detectores de *features* amplamente utilizados conduzido por Otsu et al. (2013), e ilustrado nas Figuras 23a e 23b, nota-se que o FAST é um dos detectores que apresentam a maior eficiência no respectivo estudo, já que possui uma relação otimizada entre média de correspondência detectadas e porcentagem de *matching* corretos. Além disso, ao se analisar o resultado do tempo médio de execução por *frame*, mostrado na Figura 24, nota-se que o FAST é o algoritmo mais rápido no teste realizado, sendo o mais indicado para uma aplicação em tempo real. Assim, o detector FAST foi o escolhido para este projeto.

Já para a escolha do algoritmo de estimação do fluxo óptico por método esparso, usou-se como referência o experimento de Barron, Fleet e Beauchemin (1994), apresentado na subseção 2.5.3. Conforme descrito, o algoritmo Lucas-Kanade apresentou a maior acurácia dentre os algoritmo de fluxo óptico e, portanto, foi o escolhido.

A combinação do detector de cantos FAST com o algoritmo de fluxo óptico Lucas-Kanade não é novidade na literatura. Zhang, Xiao e Xu (2020) propuseram um método de rastreamento baseado em um detector de cantos FAST melhorado com o algoritmo Lucas-Kanade piramidal. Já Pratama (2021) propôs um algoritmo de realidade aumentada usando o detector de cantos FAST e Lucas-Kanade. No contexto da odometria visual, Gebre (2018) implementou um algoritmo em OpenCV utilizando o detector FAST com o método Lucas-Kanade em uma configuração monocular, baseado em estudos realizados por Singh (2015). Devido aos resultados satisfatórios, este projeto foi baseado nos estudos de Singh e na implementação de Gebre, disponível publicamente no Github em Gebre (2018). O algoritmo de odometria visual deste projeto foi escrito em C++.

3.2.3 Contribuições do algoritmo de odometria visual

À princípio, a fim de validar algoritmos de odometria visual, é bastante frequente na literatura o uso de *datasets*, que são sequências de *frames* que correspondem a uma filmagem em movimento, complementado de dados como parâmetros de calibração da câmera utilizada e o *ground truth*, que indica a trajetória real da câmera, geralmente obtida através de um GPS. Atualmente, um dos *datasets* mais conhecidos e utilizado para testes

de odometria visual é o *KITTI Vision Benchmark* (GEIGER; LENZ; URTASUN, 2012), que fornece uma coletânea de 22 sequências stereo de *frames* específicas para a odometria visual. Dentre estas 22 sequências, 11 possuem *ground truth*. Uma análise dos *dataset* disponíveis para odometria visual pode ser encontrado em (REBERT et al., 2019).

Embora útil para validação de algoritmos de odometria visual, o uso de *dataset* dificulta a validação de algoritmos de odometria visual e inercial, pois o sensor inercial utilizado deve extrair dados do local em que a câmera estiver posicionada ao longo de toda a trajetória. Como o algoritmo de odometria visual de Mez, que é o algoritmo-base para este projeto, foi implementado para uso de *datasets*, uma adaptação foi realizada para que possa ser utilizada uma sequência particular de *frames*. Assim, devem ser levadas em conta as etapas de captura de *frames* e calibração da câmera.

Além disso, é válido ressaltar que o algoritmo de Mez utiliza os dados do *ground truth* fornecido pelo KITTI *dataset* para corrigir a posição e orientação estimada, o que não reflete as condições de um sistema de odometria visual e inercial real, já que não é possível ter acesso ao valor verdadeiro da posição e orientação. Uma das sugestões para trabalhos futuros citado por Mez (GEBRE, 2018) é a adaptação do algoritmo para que sejam utilizados outras fontes para correção da posição e orientação, que não seja o *ground truth* fornecido pelo *dataset*. Assim, outra contribuição foi a adaptação do algoritmo de forma que os *frames* capturados pela câmera fossem os únicos dados de entrada do sistema. Ainda, é apresentado um estudo sobre o uso de dados inerciais provenientes de uma IMU para a implementação de um sistema de odometria visual e inercial na seção 3.5.

Por outro lado, visto que foram utilizadas sequências próprias de *frames*, os testes realizados no algoritmo representam, de forma fidedigna, as condições reais aos quais um sistema de odometria visual será exposto, como o tipo de câmera utilizada e a influência de suas ferramentas automáticas, o aparato para acoplagem da câmera ao veículo, ausência de textura do cenário, o comportamento do algoritmo diante de objetos dinâmicos nos cenários e diante de terrenos irregulares etc, que são detalhados nos resultados.

3.2.4 Configuração da câmera e captura de *frames*

Em *datasets* amplamente conhecidos na odometria visual, são utilizados diversos equipamentos de aquisição de imagens e diversos meios para percorrer a trajetória planejada que deve ser filmada. Por exemplo, no RGB-D SLAM *dataset* (STURM et al., 2012), um *Kinect* da *Microsoft* foi acoplado a um robô móvel. No KITTI *dataset* (GEIGER; LENZ; URTASUN, 2012), uma câmera profissional foi acoplada a um carro. Já no *Monocular Visual Odometry Dataset* (MUNICH, 2016) da Universidade Técnica de Munique, a trajetória foi percorrida a pé e a câmera presa à mão.

Neste projeto, há dois grupos de testes: ambientes controlados e não controlados. Os

testes em ambiente controlado se caracterizam por pequenas distâncias (dezenas de metros) e não há influência de agentes móveis na cena, como pedestres, veículos etc. Já no ambiente não controlado, foram percorridas longas distâncias (unidades de quilômetros) e há presença de movimentos de pedestres, veículos, terrenos irregulares etc. As opções disponíveis para percorrer a trajetória no ambiente controlado eram a pé ou em uma bicicleta. A primeira opção foi descartada devido aos movimentos indesejáveis que causavam instabilidade na filmagem. Já no segundo caso, testes foram realizados com a câmera acoplada diretamente no guidão da bicicleta e com a câmera presa à mão. Como a trajetória foi percorrida em um terreno irregular, no caso da câmera acoplada no guidão, os movimentos indesejáveis eram transmitidos à câmera, prejudicando os resultados. No entanto, estes movimentos eram amortecidos quando a câmera era segurada à mão. Assim, a configuração escolhida para o grupo do ambiente controlado foi percorrer a trajetória em uma bicicleta, sendo a câmera presa à mão do ciclista. Já no grupo do ambiente não controlado, visto que deve ser percorridas longas distâncias, foi utilizada uma motocicleta, em que o piloto percorria a trajetória planejada previamente e o passageiro filmava a cena com a câmera à mão.

Para o processo de captura de *frames*, optou-se por utilizar câmera de *smartphones*, devido ao alto custo de aquisição de câmeras profissionais. Dentre os modelos disponíveis, foram testadas as câmeras do Iphone 7 *Plus* da *Apple* e do Redmi 9 da *Xiaomi*. A câmera do Iphone 7 *Plus* apresentou os melhores resultados devido à funcionalidade de estabilidade cinemática para vídeos, que reduz movimentos indesejáveis durante a filmagem, e, portanto, foi a selecionada. É válido destacar que é essencial que o sistema de estabilidade se torne caso de estudo futuro, a fim de que o sistema de odometria visual embarcado possa ser implementado de forma íntegra. Segue abaixo as especificações da câmera (APPLE, 2021) necessárias para este projeto:

1. Câmera tipo *rolling shutter*
2. Dual 12MP wide-angle
3. Wide-angle: f/1.8 aperture

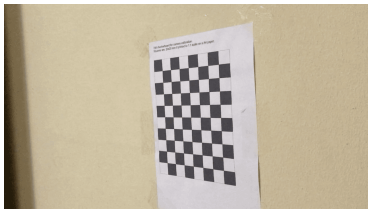
3.2.5 Calibração da câmera

Antes de capturar os *frames*, é necessário realizar a calibração da câmera para extrair parâmetros, como a matriz de calibração, que será utilizada nos algoritmos de odometria visual; e os coeficientes de distorção, necessários para corrigir as distorções das imagens a serem capturadas, conforme descrito na subseção 2.2.3. Na etapa de calibração, foi utilizado o padrão de calibração Tsai, conhecido como padrão "tabuleiro de xadrez"(vide subseção 2.2.3). Basicamente, foram feitas 15 capturas de imagens de um tabuleiro de xadrez 7x9 (7 cantos internos na horizontal e 9 cantos internos na vertical) em ângulos e posições diferentes, com uma resolução de 720p, conforme exemplificado nas Figuras 50a, 50b e 50c. Posteriormente,

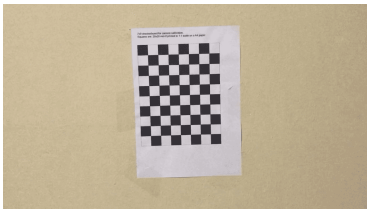
as imagens foram processadas em um algoritmo de calibração de câmera em *OpenCV*, disponível publicamente no site oficial da biblioteca (GÁBOR, 2013).

Figura 50 – Capturas para calibração da câmera

(a) Captura 1 para calibração da câmera



(b) Captura 2 para calibração da câmera



(c) Captura 3 para calibração da câmera

Fonte: Produzido pelos autores.

A matriz de calibração e os coeficientes de distorção gerados pelo execução do algoritmo de calibração são apresentados na Equação 3.1 e Equação 3.2, respectivamente.

$$\begin{bmatrix} 1220{,}0899121457219 & 0 & 639{,}5 \\ 0 & 1220{,}0899121457219 & 595{,}5 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.1}$$

$$C_d = \begin{bmatrix} 0{,}25878587994055857 & -1{,}4822072855187385 & 0 & 0 & 0{,}79401453905466135 \end{bmatrix} \tag{3.2}$$

Nota-se, pela matriz de calibração, que $f_x = f_y$, visto que a distância focal é a mesma para ambos os eixos, e que o ponto central da imagem encontra-se na coordenada (639,5 , 359,5). Estes parâmetros são necessários para calcular a matriz essencial e estimar a rotação e translação, que serão detalhados nas subseções 3.2.9 e 3.2.10. Já os coeficientes de distorção da Equação 3.2 são necessários na etapa de remoção de distorções dos *frames*, apresentada na subseção 3.2.6.

3.2.6 Captura dos *frames* e correção de distorção

A primeira parte na etapa de captura dos *frames* com o Iphone 7 *Plus* é a gravação em vídeo do trajeto planejado. Utilizou-se resolução de 720p. Finalizado o percurso, o vídeo é convertido em uma sequência de *frames* à taxa de 10 FPS utilizando o *software open-source* FFmpeg.

Após isso, é necessário corrigir as distorções intrínsecas dos *frames*. Com os parâmetros extraídos do processo de calibração, é possível reduzir as distorções, conforme descrito na subseção 2.2.3. Para isso, utilizou-se a função *undistort* do OpenCV para cada *frame*.

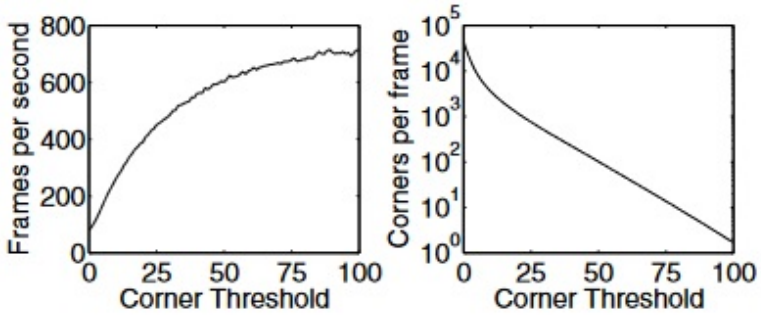
Por fim, os *frames* foram convertidos para escala de cinza (*grayscale*), a fim de reduzir o custo computacional na execução dos algoritmos de odometria visual.

3.2.7 Detecção de *features*

Após a conversão de cada *frame* para *grayscale* e sua correção de distorção, é preciso detectar as *features*. Conforme descrito na subseção 3.2.2, o detector de cantos FAST foi escolhido, pois apresenta a melhor performance de alta velocidade entre os detectores de cantos, sendo um dos mais indicados para implementações de odometria visual e SLAM. De forma simplificada, para detectar se um pixel é um canto ou não, utiliza-se uma circunferência com comprimento de 16 *pixels* centralizada no *pixel*-referência e verifica se cada *pixel* no interior da circunferência ultrapassa ou é menor que a intensidade do *pixel*-referência em um fator *t*. Uma descrição detalhada do algoritmo pode ser encontrada na subseção 2.5.5.1. O FAST está disponível na biblioteca OpenCV através da função *FAST()*. Vale destacar os seguintes parâmetros:

- *Threshold*. Este parâmetro define o fator *t* descrito na subseção 2.5.5.1. Para atribuí-lo determinado valor, é necessário analisar o comportamento da velocidade do detector de acordo com *t*, ilustrado na Figura 51, extraída do artigo referente ao detector FAST (ROSTEN, E.; DRUMMOND, T., 2005). Percebe-se que quanto maior o valor de *t*, menor a quantidade de cantos detectados por *frame*, e, consequentemente, mais rápido o algoritmo. Assim, a velocidade está relacionada ao número de cantos detectados. O limite utilizado pelos autores na versão original (ROSTEN, E.; DRUMMOND, T., 2005) e melhorada (ROSTEN, Edward; DRUMMOND, Tom, 2006) do algoritmo foi 25 e 35, respectivamente. Neste projeto, *t* foi fixado em 20, pois, para $t > 20$, o número mínimo estabelecido de cantos necessários por *frame* para rastreamento não é atingido (vide subseção 3.2.7.1).

Figura 51 – Gráficos mostrando como a velocidade de detecção de cantos varia com o limite *t*

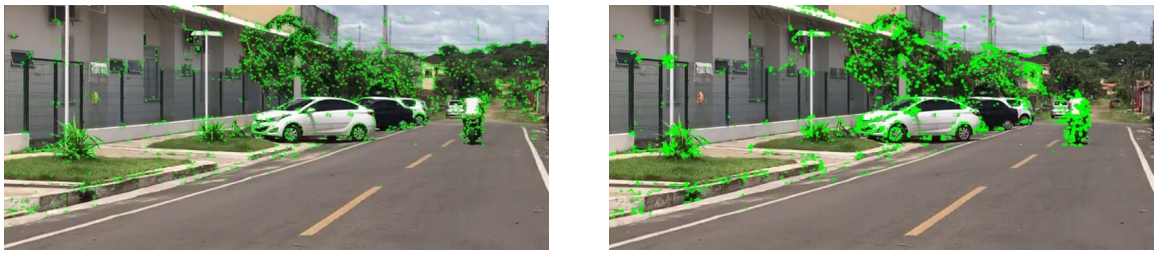


Fonte: E. Rosten e T. Drummond (2005)

- *NonmaxSuppression*. Este parâmetro permite ativar a supressão não-máxima para remover cantos em *pixels* adjacentes, que são indesejáveis. Assim, são filtrados apenas os pontos de interesse relevantes, evitando redundâncias. A Figura 52a mostra a detecção de *frames* com o recurso ativado, e a Figura 52b, sem o recurso. Percebe-se que a densidade de cantos é significativamente superior sem a supressão não máxima,

resultando em quase 5x mais cantos detectados, conforme mostra a Tabela 1. Como o desempenho de velocidade do algoritmo decresce com o aumento de cantos detectados por *frame*, o parâmetro foi ativado, já que é priorizada a alta velocidade para detecção de *features*. A ativação da supressão não ocasiona erro, pois o número mínimo de *features* detectados necessários para a respectiva resolução da imagem foi atingido, conforme será descrito na subseção 3.2.7.1.

Figura 52 – Detecção de cantos com e sem supressão não máxima



(a) Detecção de cantos com supressão não máxima ativada para uma imagem 800x360

(b) Detecção de cantos com supressão não máxima desativada para uma imagem 800x360

Fonte: Produzido pelos autores.

Tabela 1 – Relação de quantidade de *features* detectados com supressão não máxima

Supressão não máxima	*Features* detectados
ativada	2242
desativada	10182

Fonte: Produzido pelos autores.

3.2.7.1 Re-detecção de *features*

É válido ressaltar que, em grande quantidade de movimento, ao longo da trajetória o cenário pode variar significativamente e, consequentemente, a textura, ao ponto de não serem detectados *features* suficientes para estimar a rotação e translação. Nestes casos, é necessária a re-detecção. Trata-se de uma prática recorrente na odometria visual, sendo tema de pesquisas na literatura, como no estudo de Li e Wan (2018). Conceitualmente, a re-detecção consiste em executar novamente o processo de detecção de *features*, caso um limiar de pontos detectados não for atingido. Conforme ressaltam Scaramuzza e Fraundorfer (2012), uma relação adequada para se estimar o número de *features* necessários em um algoritmo de odometria visual consiste em detectar 1000 *features* para uma imagem de 640 $\times$ 480 *pixels*. Como o projeto em questão utiliza *frames* de 1280 $\times$ 720 *pixels*, os cálculos resultam em um número mínimo de 3000 *features*. No entanto, testes foram realizados com esse valor e verificou-se que o algoritmo não consegue detectar um número mínimo de 3000

features em cada *frame*, nem mesmo na re-detecção, o que acaba resultando em falha de segmentação no algoritmo. Já com um número mínimo de 2000 *features*, todos os testes conseguiram ser realizados com êxito e, portanto, foi o valor escolhido. Dessa forma, a cada *frame* capturado, se o detector FAST não detectar pelo menos 2000 cantos, a re-detecção ocorre.

Por fim, o conjunto de *features* detectados em um *frame* é armazenado em um vetor 2D de inteiros com suas localizações dada em *pixels*.

3.2.8 Rastreamento de *features*

Finalizada a detecção de cantos em dois *frames* consecutivos, F_1 e F_2, é preciso fazer a correspondência dos cantos detectados no *frame* F_1 com o *frame* F_2. Uma técnica bastante empregada é o método iterativo Lucas-Kanade, apresentado de forma detalhada nas subseções 2.5.1.1 e 2.5.7.1. Basicamente, o rastreador verifica cada canto a ser rastreado e usa esta informação local para fazer a correspondência do canto com a próxima imagem. Neste projeto foi utilizado o método Lucas-Kanade piramidal, através da função *calcOpticalFlowPyrLK()* da biblioteca OpenCV. A escolha da versão piramidal do método iterativo decorre da necessidade de trazer robustez ao algoritmo de odometria visual, visto que ele deve ser capaz estimar a trajetória da câmera mesmo em condições de grande quantidade de movimento, conforme descrito na subseção 2.5.7.1.

Dentre os parâmetros do algoritmo, é valido analisar:

- Altura máxima da pirâmide. É o parâmetro responsável por definir a quantidade máxima de nível de pirâmides a ser utilizado. A cada nível, reduz-se à metade a resolução da imagem. Visto que a câmera utilizada apresenta uma resolução de 1280 x 720, o nível 1 apresenta tamanho 640 x 360 e o nível 4, 80 x 45. O nível da pirâmide deve ser escolhido de acordo com a movimentação do fluxo óptico esperado na trajetória percorrida. Durante os testes de validação do algoritmo, evitou-se realizar movimentos bruscos na câmera, pois causariam borrões nas imagens, já que a câmera utilizada apresenta configuração *rolling shutter*. Desse modo, foi escolhida a altura piramidal 3, que é um valor prático, segundo o estudo de Boughet sobre o rastreador piramidal Lucas-Kanade (BOUGUET, 2000).

- Tamanho da janela de *pixel*. É o parâmetro que define o tamanho da vizinhança local do *pixel* a ser analisado. O detalhamento deste parâmetro pode ser visto na subseção 2.5.7.1. Para lidar com movimentos grandes, é preferível selecionar um tamanho grande de janela. Por outro lado, há um *trade-off* natural entre a robustez e a acurácia local (BOUGUET, 2000). Para rastreamento, valores típicos de tamanho de janela são 15 x 15 a 31 x 31 (SHAFIQUE, 2003). Para este projeto, foi escolhido uma janela de 21 x 21, valor sugerido na descrição da função implementada pela biblioteca OpenCV.

- *Termination criteria*: parâmetro responsável pela condições de parada do método de busca iterativa. A parada pode ocorre quando o algoritmo atinge um número máximo de iterações ou quando a janela de procura se move menos que determinado valor de janela de procura mínima, ϵ, o que ocorrer primeiro. Nos testes conduzidos por Boughet com o rastreador piramidal (BOUGUET, 2000), são utilizados os valores 20 e 0,03 para o número máximo de iterações e ϵ, respectivamente. Ainda, os valores sugeridos pela própria função implementada na biblioteca OpenCV são, respectivamente, 30 e 0,01, sendo portanto os valores utilizados neste projeto.
- Limite-mínimo de autovalor. A função *calcOpticalFlowPyrLK()* calcula o autovalor mínimo da matriz de gradiente espacial (SHAFIQUE, 2003) dividido pelo número de *pixels* em uma janela. Se o valor for menor que o limite-mínimo estabelecido, então o *feature* correspondente é rejeitado. Dessa forma, aumenta-se a acurácia do algoritmo, pois além da rejeição dos pontos que ultrapassam as bordas da imagem, as correspondências ruins são rejeitadas. Foi utilizado o valor 0.0001, sugerido pela função implementada na biblioteca OpenCV.

A Figura 53 demonstra o funcionamento do método Lucas-Kanade para o rastreamento de *features*. Note que as linhas coloridas representam o deslocamento de determinados *features* ao longo da trajetória.

Figura 53 – Funcionamento do método Lucas-Kanade para rastreamento de *features*

Fonte: Produzido pelos autores.

3.2.9 Cálculo da matriz essencial, E

Após a etapa de correspondência de *features* entre *frames*, calcula-se a matriz essencial. Neste projeto, utilizou-se o algoritmo 5 pontos de Nister, que calcula E a partir de equações não lineares e requer o número mínimo de pontos possíveis. Um maior detalhamento do algoritmo pode ser encontrado na subseção 2.7.4.1. O algoritmo está disponível na biblioteca OpenCV através da função *findEssentialMat()*, que calcula a matriz essencial através de pontos correspondentes entre duas imagens. Visto que a matriz essencial é a matriz

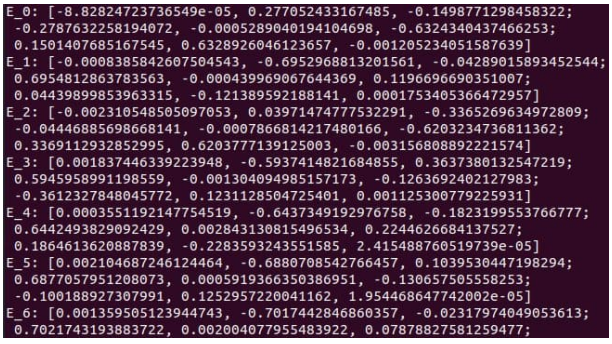
relacionada a câmeras calibradas, um dos parâmetros da função é a matriz de calibração da câmera utilizada, que já foi calculada na etapa de calibração da câmera (vide subseção 3.2.5) e cujo resultado é apresentado na Equação 3.1.

A função permite utilizar o algoritmo de rejeição de pontos fora da curva: RANSAC, que deve ser passado como parâmetro, conforme descrito na subseção 2.7.4.1. Como Nister (2004) recomenda que o algoritmo de 5 pontos seja combinado com o algoritmo de rejeição de pontos fora da curva, RANSAC (FISCHLER; BOLLES, 1981), tal combinação foi utilizada. Os seguintes parâmetros referentes ao RANSAC devem ser repassados à função:

- Nível desejado de confiança. Trata-se da probabilidade desejada de que a matriz essencial estimada esteja correta, descrita na subseção 2.6.1 e apresentada na Equação 2.14. Baseado em estudos de Hartley e Zisserman (2004), escolheu-se uma probabilidade de 99%.
- Distância máxima de um ponto a uma linha epipolar, dada em *pixels*. Também conhecida como a *Distância de Sampson*, descrita na subseção 2.6.1. É o parâmetro que definirá se o ponto é considerado fora da curva, caso esteja localizado além dessa distância. Conforme sugere a biblioteca OpenCV, podem ser utilizados valores de 1 a 3 *pixels*, dependendo da acurácia de localização do ponto, resolução da imagem e ruído referente. Baseado nos experimentos realizados por Hartley e Zisserman (2004), a distância de 1 *pixel* foi selecionada.

A Figura 54 mostra o resultado da matriz essencial calculada ao longo do tempo para uma dada trajetória. Nota-se que E é uma matriz 3×3, cujos valores dos elementos variam ao longo do tempo, já que estão relacionados aos movimentos de rotação e translação da câmera.

Figura 54 – Matriz essencial calculada ao longo do tempo para uma dada trajetória



Fonte: Produzido pelos autores.

3.2.10 Estimação da rotação e translação

Calculada a matriz essencial, deve-se extrair, finalmente, o movimento relativo da câmera - rotação e translação. Conforme descrito na subseção 2.7.5, qualquer uma das 4

combinações de R e t, $[R_a|t_u]$, $[R_a|-t_u]$, $[R_b|t_u]$ e $[R_b|-t_u]$, satisfaz a restrição epipolar descrita na Equação 2.24 (NISTER, 2004). Porém, apenas um desses conjuntos satisfaz a *cheirality constraint*, que identifica a configuração correta. Dessa forma, foi utilizada a função *recoverPose()* da biblioteca OpenCV, que decompõe a matriz essencial, através da SVD (vide subseção 2.7.5), para identificar essas 4 configurações, e verifica possíveis hipóteses de poses, ao checar os pontos que satisfazem a *cheirality constraint*. Dessa forma, são estimados R e t.

É valido salientar que, ao se decompor E, pode-se obter somente a direção da translação, t, que é retornada como unidade de distância, pois um sistema de odometria visual com configuração monocular só consegue estimar a escala absoluta por meio de informações de fontes extras, como a altura da câmera, por exemplo, ou conhecimentos prévios, o que o faz suscetível a grande *drift* e mais desafiador que configuração stereo (WANG et al., 2017). Como o projeto utiliza configuração monocular, não é possível obter informação de escala apenas com dados provenientes da câmera (SCARAMUZZA; FRAUNDORFER, 2012). É possível utilizar dados de um sensor inercial, como uma IMU, que será explora na seção 3.5. Essa combinação do uso de câmeras com sensores inerciais é classificada odometria visual e inercial.

São fornecidos como parâmetros de entrada a matriz essencial E, calculada na subseção 3.2.9; a matriz de calibração, calculada na subseção 3.2.5; e as *features* correspondentes entre os dois *frames* subsequentes. Os parâmetros de saída são R e t para cada matriz essencial decomposta.

A Figura 55 mostra o resultado da recuperação de R e t a partir da matriz essencial ao longo do tempo para uma dada trajetória. Nota-se que as matrizes de rotação e translação são tridimensionais, portanto, pode-se obter a rotação e translação nos três eixos, se houver.

Figura 55 – R e t recuperados ao longo de uma dada trajetória

```
R_0: [-0.9418902608192695, 0.02180707818115222, -0.3352121535908397;
 0.0202769898453524, -0.9923801570832241, -0.1215338122099634;
 -0.3353081869811995, -0.1212686075152029, 0.93427102308415]
t_0: [-0.1716055537328893;
 -0.06296980272317886;
 0.9831512283840327]
R_1: [0.5393737648673224, -0.8358242990174517, -0.1023410130145096;
 -0.8352978393787978, -0.5464507699820421, 0.06057289422791202;
 -0.1065526222200238, 0.05281379704223823, -0.9929034401894368]
t_1: [0.8776866815599653;
 -0.4758581472134751;
 -0.05679007609446055]
R_2: [-0.9377563733294608, 0.1796207202625166, 0.2972362379191865;
 0.1821206730975184, -0.4744062796614716, 0.8612611347601075;
 0.2957110831715365, 0.8617859819410711, 0.412164865823593]
t_2: [0.1738380409107899;
 0.5108219052169438;
 0.8419271445219176]
R_3: [0.9999741720263112, 0.002452733298980719, -0.006755692389216686;
 -0.002490253669043959, 0.999981491854409, -0.005551088658479258;
 0.006741952013880235, 0.00556776867286678, 0.999961772286845]
t_3: [-0.3235953863737966;
 -0.262895963549858;
 0.9089398980498021]
```

Fonte: Produzido pelos autores.

3.2.11 Reconstrução da trajetória

Com os dados de rotação e translação em mãos, pode-se reconstruir a trajetória percorrida pela câmera, através da concatenação das matrizes de transformação, descritas na seção 2.8. Assim, a posição e orientação da câmera é atualizada ao longo da trajetória de acordo com a Equação 2.29.

Para visualizar a trajetória da câmera, a cada *frame* processado, a localização das respectivas posição e orientação é desenhada em uma matriz bidimensional, conforme ilustrado nas Figuras 56a, 56b, 56c e 56d. Note que não há informação sobre a posição absoluta, conforme ressaltado na subseção 3.2.10.

Figura 56 – Trajetória correspondente a uma sequência de imagens

(a) Trajetória no instante de tempo t_1

(b) Trajetória no instante de tempo t_2

(c) Trajetória no instante de tempo t_3

(d) Trajetória no instante de tempo t_4

Fonte: Produzido pelos autores.

3.3 Integração dos sensores inerciais à plataforma Minized

No que tange a odometria inercial, a integração dos sensores inerciais à plataforma Minized é a etapa inicial no desenvolvimento do projeto proposto, naturalmente a integração da câmera à plataforma apresentada pode ter seu desenvolvimento realizado de forma paralela, porém, não é foco desta seção.

Apesar de ser um pré-requisito e proporcionar apenas uma interface entre os sensores disponíveis e a plataforma para que se possa ter acesso a todas as funcionalidades desejadas, essa etapa é etapa integrante e essencial do projeto e, portanto, deve ser abordada com a devida atenção no que diz respeito a sua implementação e as funcionalidades a serem incorporadas.

3.3.1 Considerações sobre o desenvolvimento

Para realizar essa integração, inicialmente foi necessário realizar o planejamento das funcionalidades a serem incorporadas e decidir qual a abordagem a ser utilizada durante a implementação no que diz respeito ao uso dos recursos disponíveis e também da acessibilidade do código.

Para realizar essas decisões adotou-se a contribuição do projeto como filosofia máxima, ou seja, garantir que o projeto servisse de referência ou base para futuros projetos relacionados a plataforma proposta, a Minized. Naturalmente, também foram observados aspectos relacionados a utilização dos recursos, ou seja, buscou-se otimizar o uso de memória e minimizar o processamento necessário sempre que possível.

Conforme apresentado em seção anterior, seção 2.9, os sensores utilizados são parte integrante da IMU MPU-6050, dispositivo esse que fornece diversas funcionalidades relacionadas a captura e processamento de movimento, além disso, a unidade conta com funções que permitem reduzir o consumo de energia do dispositivo, e até mesmo realizar estimação com auxílio de processador integrado, todas essas características contribuem bastante para um bom uso de recursos por parte da unidade e podem ser exploradas.

O dispositivo incorpora giroscópios e acelerômetros em cada um dos três eixos para realizar a medição de velocidade angular e aceleração linear respectivamente, além disso, ele conta com um sensor de temperatura que pode, por exemplo, ser utilizado para realizar a compensação de desvio ou calibração em *runtime* dos dispositivos em relação a variação de temperatura, levando a medições mais precisas.

Dessas forma, todas as configurações relacionadas a captura e processamento de dados, além da configuração de cada um dos sensores, devem ser habilitadas por meio da implementação adequada de cada uma das funcionalidades respectivas.

3.3.2 Considerações sobre a comunicação

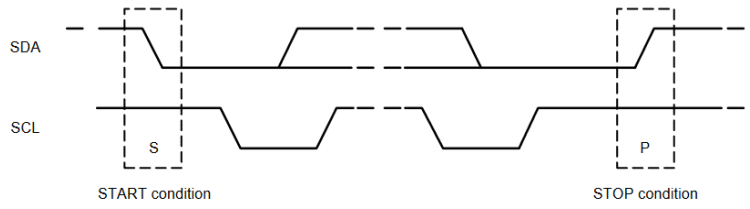
Antes de comentar cada característica mais a fundo e apresentar sua implementação, faz-se necessário primeiro promover a conexão, no que diz respeito a comunicação, entre a unidade e a plataforma Minized. A unidade conta essencialmente com dois protocolos distintos de comunicação, *Serial Peripheral Interface* (SPI) e *Inter-Integrated Circuit* (I2C), cada protocolo conta com suas características próprias no que diz respeito a velocidade, forma de conexão e transmissão, e funcionalidade, para essa implementação o protocolo I2C foi utilizado.

3.3.2.1 Introdução ao protocolo I2C

Em relação a estrutura, a interface I2C é composta essencialmente por dois sinais, *serial clock* (SCL), responsável pela sincronização de *clocks* entre dispositivos comunicantes, e *serial data* (SDA), responsável pela transferência de dados e com comunicação bidirecional. Esse protocolo trabalha com separação de dispositivos em duas classificações, mestres e escravos, mestres realizam solicitações a um escravo específico por meio de um endereço não conflitante, e os escravos as atendem. No caso em questão, o dispositivo MPU6050 sempre trabalha em modo escravo, respondendo as solicitações do sistema de processamento ou do *hardware* implementado, o mestre.

Em relação a comunicação, ela começa quando algum mestre coloca no barramento um sinal de condição de início chamada START, indicada por (S), a transmissão continua até que um condição de parada, STOP, indicada por (P), seja colocada, conforme indicado pela Figura 57.

Figura 57 – Início e parada da comunicação



Fonte: InvenSense (2013)

Para realizar a comunicação completa, após fornecer o sinal de início o mestre deve enviar o endereço do escravo que deve atender a solicitação seguida por um *bit* indicativo do tipo de operação a ser realizada, leitura ou escrita, que indica se o mestre está escrevendo ou lendo do dispositivo escravo.

Cada mensagem contendo dados é composta por um único *byte*, após cada *byte* enviado deve ocorrer o envio de um *bit* do tipo ACK, *acknowledge bit*, que indica que o conteúdo foi transferido, após feita a comunicação, o mestre pode tanto finalizar através do

sinal de parada ou também continuar a esse comunicar através do barramento por meio de um sinal do tipo *REPEATED START* (SR).

As Figuras 58 e 59 apresentam exemplos de 4 formas de se comunicar, escrita de um único *byte* e escrita em rajada, Figura 58, leitura de um único *byte* e leitura em rajada, Figura 59.

Figura 58 – Exemplos de operação de escrita I2C

Single-Byte Write Sequence

Master	S	AD+W		RA		DATA		P
Slave			ACK		ACK		ACK	

Burst Write Sequence

Master	S	AD+W		RA		DATA		DATA		P
Slave			ACK		ACK		ACK		ACK	

Fonte: InvenSense (2013)

Figura 59 – Exemplos de operação de leitura I2C

Single-Byte Read Sequence

Master	S	AD+W		RA		S	AD+R			NACK	P
Slave			ACK		ACK			ACK	DATA		

Burst Read Sequence

Master	S	AD+W		RA		S	AD+R			ACK		NACK	P
Slave			ACK		ACK			ACK	DATA		DATA		

Fonte: InvenSense (2013)

3.3.3 Considerações sobre o método de implementação

Feita essa breve apresentação do protocolo, para que seja possível configurar o dispositivo é necessário implementar essa comunicação.

Na Minized existem essencialmente três possibilidades para implementar tal comunicação, em *firmware* através do sistema de processamento, *hardware* através do uso da lógica programável e da interface AMBA AXI, ou através da criação de uma imagem de um sistema PetaLinux, apesar de permitir implementar facilmente tal comunicação, o PetaLinux não foi utilizado em nenhuma etapa do projeto devido a restrições de tempo e, para não sobrecarregar o sistema de processamento com atividades de pouca relevância, a forma de implementação escolhida foi por meio do *hardware*.

Para iniciar o desenvolvimento é necessário entender inicialmente como se dará cada etapa do projeto e como cada *software* disponível contribui para o projeto em sua totalidade.

Na plataforma Minized são utilizados essencialmente 2(dois) *softwares*, **Xilinx Vivado** e **Xilinx Vitis**, em versões anteriores a unificação da plataforma de desenvolvimento **Xilinx SDK** tomará o lugar do **Xilinx Vitis**.

O projeto consistirá em duas etapas, o desenvolvimento do *hardware*, e do *software*. Xilinx Vivado é o programa responsável pela parte do desenvolvimento do *hardware*, é nele onde será feita configuração do sistema de processamento *Processing System* (PS) e serão incorporadas todas as funcionalidades necessárias para o projeto através do uso da *Programmable Logic* (PL).

Xilinx Vitis ou Xilinx SDK serão responsáveis pela parte de desenvolvimento do *software*, é neles onde é possível fazer a toda implementação da aplicação e extrair todas as funcionalidades necessárias dos dispositivos incorporados na etapa de desenvolvimento de *hardware*.

Além dos *softwares* citados, **Xilinx Vitis HLS - Vitis High-Level Synthesis** também é um programa que pode ser utilizado em aplicações mais complexas ou em que seja necessário um maior desempenho como, por exemplo, processamento de imagens ou *streaming*.

O Vitis HLS é um programa que permite realizar o desenvolvimento e implementação de aplicações em linguagens de alto nível como C e C++, essas aplicações serão convertidas em blocos de *hardware* ou IPs em linguagens de descrição de *hardware* através de síntese de alto nível, permitindo um desenvolvimento e controle mais eficiente dessas aplicações, IPs desenvolvidas por tal método permitem em muitos casos alcançar desempenho muito superior a um programa que utilize apenas o sistema de processamento, cerca de 40 a 100 vezes mais rápido, nesse documento serão utilizados apenas Xilinx Vivado e Xilinx Vitis.

3.3.4 Desenvolvimento - *Hardware*

Para realizar a integração de dispositivos à plataforma é essencial garantir que exista um *hardware* capaz atender a todos os requisitos necessários para a aplicação específica. Por vezes, esses requisitos podem se traduzir em configurações mais simples envolvendo apenas o sistema de processamento. Em outros casos, em configurações extremamente robustas e complexas em que um série de dispositivos operam em sinergia para garantir todas as funcionalidades necessárias.

Nesse caso, é de interesse a incorporação de um dispositivo I2C na plataforma para que seja possível promover a comunicação entre o sistema de processamento e a unidade responsável pelos sensores inercias, a MPU6050. Dessa forma é necessário garantir não somente as funcionalidades básicas mas também as funcionalidades específicas para dispositivos que utilizam o protocolo I2C.

As nuances relacionadas a todo o processo de desenvolvimento serão omitidas, tendo em vista que é um processo complexo e que envolve diversas etapas até que se obtenha ao

menos um *hardware* simples que atenda as especificações.

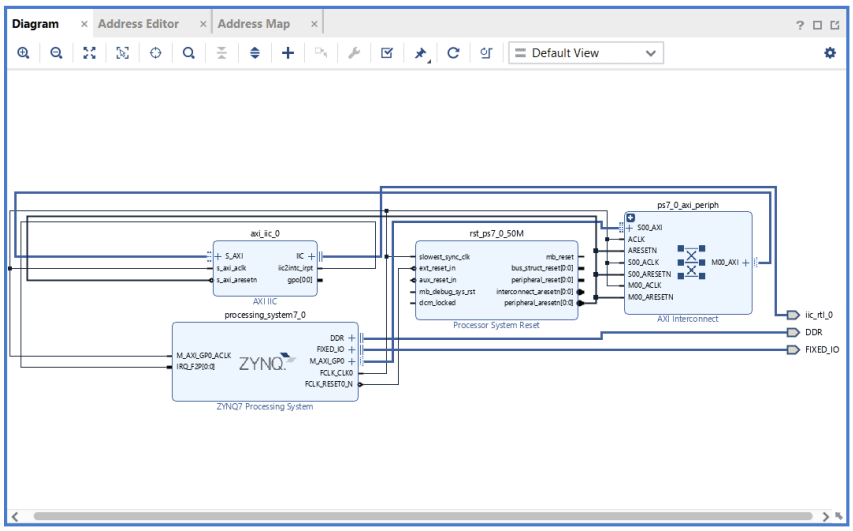
Para realizar o desenvolvimento, buscou-se garantir primeiramente as funcionalidades essenciais para a utilização do sistema de processamento. As configurações necessárias para para garantir pleno funcionamento do sistema pode variar de aplicação para aplicação. Como nesse caso o projeto é relativamente simples, essas configurações envolvem essencialmente garantir que o sistema de processamento possua dispositivos de memória para realizar o armazenamento, como memórias DDR, interrupções, que são fundamentais para atender a requisições de dispositivos, *clocks* para realizar a sincronização do sistema e, por fim, garantir que periféricos como UART e GPIO estivessem disponíveis para realizar a coleta de dados e comunicação com a plataforma.

Naturalmente, existem diversas configurações intermediárias necessárias e parâmetros que devem ser definidos para se obter um sistema de processamento completamente operante, porém, não são de interesse imediato para o projeto.

Com um sistema de processamento configurado é possível então se comunicar com outros dispositivos no sistema. Existem diversas IPs podem ser adicionadas ou criadas e todas elas podem contribuir para enriquecer as funcionalidades da plataforma do projeto, nesse caso, porém, foi utilizada apenas a IP AXI IIC. Essa IP incorpora ao projeto a funcionalidade necessária para que a comunicação I2C possa ser efetivada, sendo necessário ainda incorporar uma interface AXI apropriada para promover a conexão entre a PS e a PL e um sistema de reinicialização, como as IPs *Processor System Reset* e AXI *Interconnect*.

Ao final do processo obtém-se o sistema apresentado na Figura 60. Notavelmente, o sistema é relativamente simples, mas é ideal para que a integração dos sensores inerciais possa ser efetivada de forma adequada na plataforma.

Figura 60 – Sistema completo com I2C incorporado



Fonte: Produzido pelos autores.

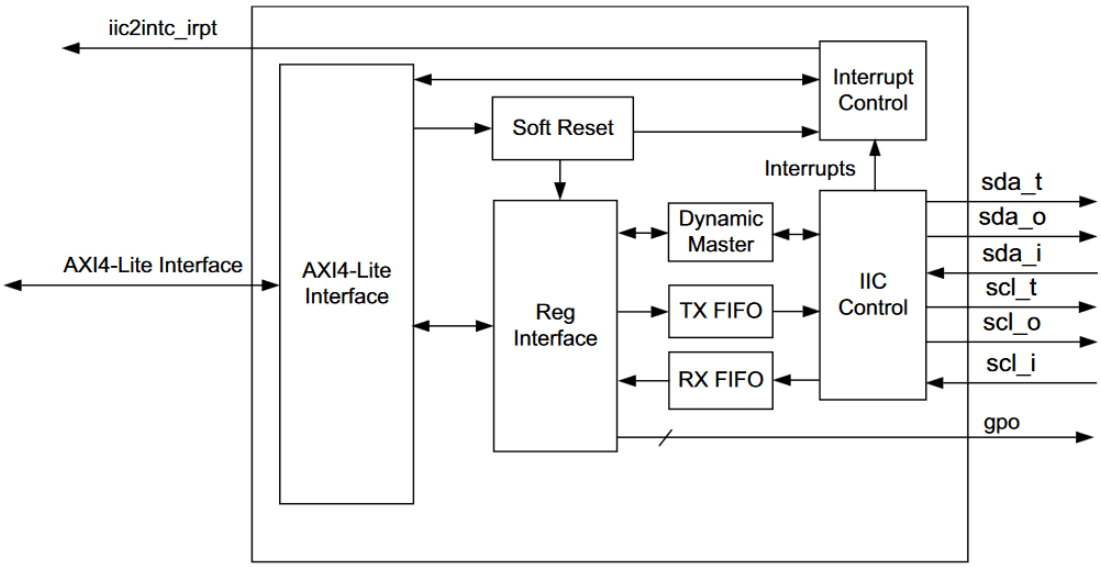
3.3.5 Desenvolvimento - *Software*

Feita a introdução do *hardware*, nesta subseção será apresentado o desenvolvimento do *software* em si, sendo dividida em duas partes: comunicação I2C e funcionalidades da MPU6050.

3.3.5.1 Comunicação I2C

Conforme apresentado, do ponto de vista de *hardware* a comunicação I2C entre o sistema e o dispositivo MPU6050 é feita através do uso da IP AXI IIC ou AXI IIC Bus Interface, essa IP implementa todas as funcionalidades necessárias para que a comunicação possa ser realizada por meio de linguagens de descrição de *hardware* como Verilog e VHDL. A Figura 61 apresenta o diagrama dos constituintes do bloco, chamado *Top-Level Block Diagram*.

Figura 61 – Composição - AXI IIC Bus Interface v2.1



Fonte: Xilinx (2021a)

Entre esses constituintes estão, por exemplo: a interface de controle de comunicação I2C que lida com os sinais de clock e dados (SCL e SDA), o controlador de interrupções, filas necessárias para realizar a transmissão e recepção de dados, a interface de registradores que armazenam configurações, sinais de interrupção, entre outros.

Naturalmente, o sistema conta também com a interface AXI para que esse dispositivo possa responder a algum mestre, a AXI4-Lite Interface. A Figura 62 apresenta os registradores disponíveis e suas respectivas descrições.

Figura 62 – Registradores - AXI IIC Bus Interface v2.1

Table 2-4: **AXI IIC Core Register Map**

Address Space Offset[1]	Register Name	Description
01Ch	GIE	Global Interrupt Enable Register
020h	ISR	Interrupt Status Register
028h	IER	Interrupt Enable Register
040h	SOFTR	Soft Reset Register
100h	CR	Control Register
104h	SR	Status Register
108h	TX_FIFO	Transmit FIFO Register
10Ch	RX_FIFO	Receive FIFO Register
110h	ADR	Slave Address Register
114h	TX_FIFO_OCY	Transmit FIFO Occupancy Register
118h	RX_FIFO_OCY	Receive FIFO Occupancy Register
11Ch	TEN_ADR	Slave Ten Bit Address Register
120h	RX_FIFO_PIRQ	Receive FIFO Programmable Depth Interrupt Register
124h	GPO	General Purpose Output Register
128h	TSUSTA	Timing Parameter Register
12Ch	TSUSTO	Timing Parameter Register
130h	THDSTA	Timing Parameter Register
134h	TSUDAT	Timing Parameter Register
138h	TBUF	Timing Parameter Register
13Ch	THIGH	Timing Parameter Register
140h	TLOW	Timing Parameter Register
144h	THDDAT	Timing Parameter Register

1. Address Space Offset is relative to C_BASEADDR assignment.

Fonte: Xilinx (2021a)

Do ponto de vista de *software*, não somente esse dispositivo apresentado mas também diversos outros contam com *drivers* desenvolvidos pela Xilinx. Esses *drivers* tem como objetivo permitir que os dispositivos possam ser configurados e suas funcionalidades possam ser extraídas e utilizadas de forma adequada.

Em diversos casos é possível encontrar *drivers* específicos para cada tipo de implementação, por exemplo, por meio do *hardware* através das IPs disponíveis, ou por meio dos periféricos integrados do sistema de processamento. Além disso, também é possível encontrar *drivers* de diferentes níveis de abstração para um mesmo tipo de implementação.

No nível mais baixo de abstração, a comunicação é feita diretamente com os registradores dos dispositivos e todas as operações necessárias para o funcionamento correto devem ser garantidas pelo usuário, por exemplo, o controle de interrupções, a inserção de dados nas filas, e assim por diante.

No segundo nível, estão as implementações que contam com essas operações já bem definidas e o usuário fica responsável apenas por garantir que as funcionalidades sejam utilizadas de forma correta para realizar uma determinada função e que as mesmas possuam consistência.

Em mais alto nível, é criada uma instância de um objeto relacionado ao dispositivo específico e todas as operações são realizadas sobre o mesmo.

A priori, implementações de mais alto nível foram utilizadas, porém, essas não realizaram a configuração adequada do dispositivo. Foi possível perceber que quanto mais simples

os dispositivos utilizados mais inadequadas e inconsistentes eram essas implementações de alto nível.

Portanto, para esse projeto foi escolhido o nível intermediário para evitar as complexidades e minúcias relacionadas ao controle das operações em baixo nível, mas também ainda garantir certo controle sobre as operações e evitar problemas eventuais que ocorrem com as implementações de alto nível. Isso garante que a implementação seja consistente em qualquer situação, sendo possível incorporar qualquer dispositivo que utilize esse tipo de protocolo.

O *driver* utilizado é o iic_v3_8, que conta com a implementação para qualquer tipo de dispositivo I2C, escravo ou mestre, sendo implementado de forma particionada o que reduz a utilização efetiva de memória. Além disso, ela conta tanto com *Application Programming Interface* (API) de alto nível quanto APIs de baixo nível, sendo que essas últimas serão as utilizadas, mais especificamente XIic_Send() e XIic_Recv().

A comunicação com o dispositivo pode se dar por meio de interrupções ou por *polling*. As interrupções fazem um uso bem mais eficiente dos recursos pois o dispositivo somente solicitará que seja atendido caso realmente precise e, durante o período de ociosidade do dispositivo, os recursos podem ser utilizados de maneira mais eficiente para atender outras tarefas, promovendo um ganho significativo de desempenho caso essas solicitações ou interrupções não sejam constantes.

Por outro lado, no modo de *polling* o processador deve ativamente escanear o dispositivo para atender as solicitações dele e, portanto, é um método que necessita de muitos ciclos de processamento e, caso o dispositivo não faça requisições constantes, apresenta um desempenho inferior.

Nesse projeto, o método de *polling* foi utilizado devido a problemas durante a implementação das interrupções e a limitação de tempo. É previsto que o desempenho da plataforma apresentará certa deficiência e, portanto, pode ser alvo de aprimoramento no futuro. Apesar dos problemas em relação ao uso de recursos e consumo, essa implementação ainda é aceitável devido à natureza recorrente e com frequência razoável das operações de coleta de dados para realização das estimações da posição e orientação.

A implementação consiste em 5 (cinco) funções genéricas que podem ser utilizadas para qualquer dispositivo I2C na plataforma desde que configurados por meio do *hardware*. Mais especificamente foram implementadas as funções: *I2C_readByte*, *I2C_writeByte*, *I2C_readAddrRange*, *I2C_writeAddrRange* e *I2C_readRepeatedStart*.

As funções I2C_readByte e I2C_writeByte são responsáveis por realizar a leitura e escrita de um único *byte* em um dos registradores do dispositivos escravo. Ambas tomam como parâmetros o endereço do dispositivo escravo e o registrador destino da operação, além disso, devem ser informados o local de armazenamento da informação lida ou o conteúdo a

ser escrito, a depender da operação. As Figuras 63 e 64 apresentam os protótipos das duas funções.

Figura 63 – Função - Leitura de *byte* único através de I2C

```
int I2C_readByte(u8 slaveAddr, u8 registerAddr, u8* bufferPtr)
```

Fonte: Produzido pelos autores.

Figura 64 – Função - Escrita de *byte* único através de I2C

```
int I2C_writeByte(u8 slaveAddr, u8 registerAddr, u8 msgData)
```

Fonte: Produzido pelos autores.

Para realizar a leitura e escrita em uma extensão definida e válida dos registradores dos dispositivos, são utilizadas as funções I2C_readAddrRange e I2C_writeAddrRange. Tais funções se utilizam da característica de deslocamento de ponteiro que esse tipo de dispositivo costuma utilizar para fazer o acesso a um conjunto de registradores subsequentes.

Assim como as funções apresentadas anteriormente, ambas tomam como argumento o endereço do dispositivo que irá responder à solicitação, o endereço base, ou seja, a partir de qual registrador deve começar a leitura ou escrita, o número de *bytes* a ser lido ou escrito e, por fim, o *buffer* de escrita ou leitura para que os dados sejam transmitidos ou recebidos.

As Figuras 65 e 66 apresentam o protótipo de cada uma das funções apresentadas.

Figura 65 – Função - Leitura em extensão definida da memória através de I2C

```
int I2C_readAddrRange(u8 slaveAddr, u8 registerAddr, u8* bufferPtr, u16 bytesToRead)
```

Fonte: Produzido pelos autores.

Figura 66 – Função - Escrita em extensão definida de memória através de I2C

```
int I2C_writeAddrRange(u8 slaveAddr, u8 registerAddr, u8* bufferPtr, u16 bytesToWrite)
```

Fonte: Produzido pelos autores.

Por fim, define-se a função I2C_readRepeatedStart, assim como a função I2C_readAddrRange, essa função toma como argumento o endereço do dispositivo escravo, o registrador base alvo da leitura, o *buffer* de armazenamento de dados e o número de *bytes* que deve ser lido.

Apesar das similaridades apresentadas, essa função conta com uma característica diferente, ao invés de utilizar uma *flag* do tipo *STOP*, essa função utiliza uma *flag* do tipo *REPEATED START* para realizar a operação, com isso é possível realizar um leitura contínua de um endereço específico do dispositivo para leitura de um número específico de *bytes* de dados.

Essa função pode ser utilizada, por exemplo, para extrair informações de bancos de memória interna de dispositivos mais robustos ou, como é caso desse projeto, fazer a leitura de estruturas tipo FIFO presentes em alguns dispositivos.

Figura 67 – Função - Leitura com método de *repeated start*

```
int I2C_readRepeatedStart(u8 slaveAddr, u8 registerAddr, u8* bufferPtr, u16 bytesToRead)
```

Fonte: Produzido pelos autores.

3.3.5.2 Funcionalidades - MPU6050

Conforme abordado, para o desenvolvimento do *software* responsável por extrair as funcionalidades desejadas da unidade MPU6050.

Na implementação o foco foi proporcionar ao usuário todas as funcionalidades necessárias para o uso adequado do dispositivo. Muitas das características implementadas não foram utilizadas nesse projeto, e visam apenas garantir que a plataforma possua um *software* abrangente que garanta a possibilidade de implementação de diversas aplicações.

A implementação utilizou-se de um tipo definido e bastante simples para armazenar configurações e informações relativas ao dispositivo, apesar de utilizar memória para armazenar informações que não são estritamente necessárias, essa estrutura permite reduzir o número de operações necessárias para extrair dados do dispositivo e consequentemente aumentar o desempenho dos algoritmos de estimação incorporados na plataforma.

Apesar do fato do ganho de desempenho não ser tão significativo ele ainda existe, além disso, também é possível garantir a consistência das configurações habilitadas no dispositivo por meio do monitoramento dos dados armazenados nessa estrutura. A Figura 68 apresenta a construção apresentada.

Figura 68 – Estrutura do dispositivo MPU6050

```
typedef struct{
    struct {
        u8  gyroResolution;
        u8  accelResolution;
        u8  digitalLowPassFilter;
        u16 sampleRate;
    }config;

    struct {
        u8  gyroscopeStatus;
        u8  accelerometerStatus;
        u8  temperatureSensorStatus;
        u8  activeSensors;
        u8  fifoEnabled;
        u8  intrEnabled;
        float   accelSensitivity;
        float   gyroSensitivity;
    }status;
}deviceMPU6050;
```

Fonte: Produzido pelos autores.

Em relação as funções a serem incorporadas, o código foi planejado de forma a incorporar essencialmente 7 tipos de funções, funções relacionadas a inicialização do dispositivo, associadas as configurações gerais, as configurações dos sensores, funções relacionadas a FIFO e, por fim, funções de utilidade.

As funções relacionadas a inicialização do dispositivo apresentadas na Figura 69 definem um procedimento de inicialização completo para que o dispositivo possa ser utilizado corretamente.

Apesar do método utilizado abranger diversos aspectos da configuração, ele não é único, e visa apenas apresentar um procedimento simples e configurável para o usuário de forma que o mesmo possa ativar apenas as características que deseja utilizar e as configure da maneira desejada, naturalmente limitado as características do dispositivo.

Apesar de configurável, o procedimento de inicialização é pré-definido, ou seja, uma vez configurado pelo usuário o procedimento realizará sempre uma função de inicialização padrão e, portanto, a maioria das funções desse tipo devem ser utilizadas somente durante a primeira inicialização do dispositivo para garantir a consistência das configurações.

Figura 69 – Funções - Inicialização do dispositivo

```
int mpu6050_init();
int mpu6050_setInitialConfig();
int mpu6050_turnDeviceOn();
int mpu6050_initFIFO();
int mpu6050_initFinish();
```

Fonte: Produzido pelos autores.

A função mostrada na Figura 70 é a função padrão de inicialização, ela serve para realizar a inicialização da instância da estrutura associada ao dispositivo e chamar as outras funções de inicialização.

Apenas essa função é essencial para a inicialização do dispositivo e todas as outras funções de inicialização são auxiliares e servem apenas para garantir que o código seja facilmente compreendido.

Essa função realiza diversos procedimentos, por exemplo, inicializa a estrutura, realiza um teste rápido para garantir que o dispositivo está respondendo, e adequadamente, define configurações gerais, entre outras coisas.

Figura 70 – Função - Inicialização padrão

```
int mpu6050_init();
```

Fonte: Produzido pelos autores.

A função apresentada na Figura 71 é a função responsável pela etapa de configurações gerais e específicas do dispositivo como, por exemplo, configuração da taxa de amostragem,

configuração do filtro passa baixa, inicialização a FIFO, entre outros.

Figura 71 – Função - Inicialização das configurações do dispositivo

```
int mpu6050_setInitialConfig()
```

Fonte: Produzido pelos autores.

Ainda continuando com os procedimentos de inicialização, a função definida na Figura 72 é responsável por configurar a FIFO de acordo com o que foi definido pelo usuário, por exemplo, ativando a inserção de dados capturados pelo giroscópio, acelerômetro, sensor de temperatura e sensores externos.

Figura 72 – Função - Inicialização da FIFO

```
int mpu6050_initFIFO();
```

Fonte: Produzido pelos autores.

Por fim, para finalizar as questões relacionadas a inicialização do dispositivo, a função apresentada na Figura 73 é definida, ela é responsável por procedimentos garantem que uma vez que o dispositivo volte a ser ativado, e a coleta de dados comece, seja garantida a consistência dos dados obtidos.

Essa consistência é essencial para que algoritmos de estimação não convirjam para um estado inválido devido a dados que não refletem a realidade do instante de tempo atual, e também para que a convergência futura não seja prejudicada, levando a resultados imprecisos.

Figura 73 – Função - Finalização do procedimento de inicialização

```
int mpu6050_initFinish();
```

Fonte: Produzido pelos autores.

O segundo módulo de funções a ser apresentado é o de configurações gerais, a Figura 74 apresenta as funções que compõe esse módulo. Essas funções realizam, entre outras coisas, a configuração do filtro passa-baixas, taxa de amostragem, *clock source* e das interrupções.

Figura 74 – Funções - Configurações gerais

```
void mpu6050_getDLPF(u8* LPF);
int mpu6050_setDLPF(u8 LPF, u8 checkConsistency);
int mpu6050_getSampleRate(u16* sampleRate);
int  mpu6050_setSampleRate(u8 sampleRate, u8 adjustLPF, u8 checkConsistency);
int mpu6050_setClockSource(u8 clockSource);
int mpu6050_setIntr(u8 enaDataRdy);
int mpu6050_controlStandby(u8 gyroControl, u8 accelControl, u8 checkConfigConsistency);
```

Fonte: Produzido pelos autores.

As funções definidas na Figura 75 são responsáveis por retornar a configuração atual do filtro passa-baixas e realizar a configuração do mesmo.

DLPF é defino do inglês, *Digital Low Pass Filter* ou filtro passa- baixas digital, a MPU6050 conta com um filtro passa baixas configurável que pode ser utilizado para filtrar os dados tanto do giroscópio como do acelerômetro.

Essa configuração é essencial pois permite realizar uma filtragem mais fina ou mais livre dos dados a depender da necessidade do usuário, é importante notar que quanto menor a banda configurada maior o atraso entre a captura e processamento dos dados para que possam ser disponibilizados para o usuário, por exemplo, considerando uma taxa de amostragem de 200Hz a coleta de dados será realizada a cada 5ms, com uma filtragem mais fina seria introduzido um atraso quase 4 vezes maior que o período de amostragem e, portanto, pode ser um fator a se considerar.

Além de configurar o filtro digital, essas definições realizam ainda uma função bastante importante, permitem realizar a sincronização da captura dos dados do giroscópio e do acelerômetro, dessa forma, mesmo que o filtro não seja necessário é sempre interessante configurá-lo para permitir tal sincronização.

Figura 75 – Funções - Filtro passa baixas digital

```
void mpu6050_getDLPF(u8* LPF);
int mpu6050_setDLPF(u8 LPF, u8 checkConsistency);
```

Fonte: Produzido pelos autores.

Dando continuidade ao módulo, serão apresentadas as funções definidas na Figura 76. De forma bastante similar ao apresentado anteriormente, essas funções são responsáveis por retornar o estado atual da configuração da taxa de amostragem do dispositivo e realizar a configuração da mesma.

O giroscópio conta com uma taxa de amostragem de 4Hz a 8Khz, o acelerômetro por outro lado conta com uma taxa de 4Hz a 1Khz, ambas taxas são configuradas em conjunto através de um divisor de 8 *bits* e podem ser sincronizadas caso o filtro passa-baixas seja configurado adequadamente conforme apresentado.

Figura 76 – Funções - Taxa de amostragem

```
int mpu6050_getSampleRate(u16* sampleRate);
int  mpu6050_setSampleRate(u8 sampleRate, u8 adjustLPF, u8 checkConsistency);
```

Fonte: Produzido pelos autores.

A função apresentada na Figura 77, é responsável por configurar o *clock source* do dispositivo, *clock source* é essencialmente a origem do *clock* do circuito interno de sincronização do dispositivo.

A MPU6050 é bastante flexível em relação a essa característica, permitindo que várias fontes externas e internas possam ser utilizadas, entre as fontes disponíveis estão: osciladores internos, osciladores associados ao giroscópio, e *clocks* externos definidos.

Cada uma dessas fontes podem apresentar vantagens a depender do modo de operação utilizado, porém, algo importante a ser citado é a questão acurácia, cada uma dessas fontes apresenta uma determinada acurácia, essa acurácia se traduz em erros temporais que implicam diretamente na qualidade dos dados e consequentemente nas estimativas obtidas. Uma boa fonte a ser selecionada para esse projeto são os osciladores atrelados aos giroscópios que estão ativados a todo tempo e, portanto, podem ser acessados.

Esses *clocks* apresentam uma excelente acurácia e serão utilizados durante o projeto.

Figura 77 – Função - Configuração da origem do *clock* do circuito de sincronização

```
int mpu6050_setClockSource(u8 clockSource);
```

Fonte: Produzido pelos autores.

Para finalizar as funções expressivas desse módulo, será apresentada a função definida na Figura 78, essa função é responsável por ativar as interrupções disponíveis no dispositivo, por exemplo, devido a *overflow* da FIFO e devido a disponibilidade de novos dados.

Figura 78 – Função - Configuração das interrupções

```
int mpu6050_setIntr(u8 enaDataRdy);
```

Fonte: Produzido pelos autores.

Por fim, a última função desse módulo é apresentada na Figura 79, essa função realiza uma única função, ativar ou desativar o estado de ociosidade do dispositivo.

Figura 79 – Função - Controle da ativação do dispositivo

```
int mpu6050_controlStandby(u8 gyroControl, u8 accelControl, u8 checkConfigConsistency);
```

Fonte: Produzido pelos autores.

Conforme apresentado, para trabalhar com os sensores também é necessário definir as funções associadas a configuração de cada dispositivo e a coleta de dados de cada um deles.

As funções que realizam os procedimentos citados são apresentadas na Figura 80. Como os dispositivos operam de forma bem similar e os requisitos de implementação são praticamente idênticos, serão apresentados em conjunto. Algumas das funções definidas na Figura 80 foram omitidas por não possuírem características relevantes a serem comentadas.

Figura 80 – Funções - Configuração e coleta de dados dos sensores

```
//Gyroscope - Functions associated with gyroscope configurations and data
int mpu6050_getGyroscopeRaw(u16* receivedDataBuffer, u32* timestamp);
int mpu6050_getGyroscopeData(float* receivedDataBuffer, u8 genOption);
int mpu6050_sampleGyroscopeData(float* receivedDataBuffer, int samples, u8 genOption);
int mpu6050_setGyroFSR(u8 gyroFSR, u8 checkConsistency);
void mpu6050_getGyroFSR(u16* gyroFSR);
void mpu6050_getGyroSens(float* gyroSens);
float mpu6050_convertGyroData(u16 encodedData2sComplement);
//Accelerometer - Functions associated with accelerometer configurations and data
int mpu6050_getAccelerometerRaw(u16* receivedDataBuffer, u32* timestamp);
int mpu6050_getAccelerometerData(float* receivedDataBuffer, u8 genOption);
int mpu6050_sampleAccelerometerData(float* receivedDataBuffer, int samples, u8 genOption);
int mpu6050_setAccelFSR(u8 accelFSR, u8 checkConsistency);
void mpu6050_getAccelFSR(u16* accelFSR);
void mpu6050_getAccelSens(float* accelSens);
float mpu6050_convertAccelData(u16 encodedData2sComplement);
//Temperature sensor - Functions associated with temperature sensor configurations and data
int mpu6050_getTemperatureRaw(float* receivedDataBuffer, u32* timestamp);
int mpu6050_getTemperatureData(float* receivedDataBuffer, u8 genOption);
int mpu6050_sampleTemperatureData(float* receivedDataBuffer, int samples, u8 genOption);
float mpu6050_convertTempData(u16 encodedData2sComplement);
int mpu6050_controlTemperatureSensor(u8 temperatureSensorOption);
```

Fonte: Produzido pelos autores.

Para coleta de dados são definidas duas variações para cada sensor disponível e uma função de coleta de dados de forma contínua.

As funções definidas na Figura 81 coletam dados dos registradores associados a cada um dos sensores definidos e os retorna ao usuário, esse valores são os valores de 16 *bits* em complemento de 2 dos valores obtidos pelos sensores.

Não somente para essas funções apresentadas mas para todas as funções que leem diretamente dos registradores associados aos dados dos sensores, é necessário garantir a consistência dos dados através de leitura adequada dos *bits* do registrador de interrupções para monitorar o valor do *bit* que indica a disponibilidade de novos dados para coleta.

Do ponto de vista de desempenho, utilizar esses métodos de leitura com modo de *polling* não é eficiente e, caso o controle não seja realizado de forma adequada, pode ser que os resultados não sejam satisfatórios, dessa forma utilizar essas funções não é recomendado e estão aqui definidas apenas por motivo de completude da implementação apresentada.

Para esse projeto somente a leitura através da FIFO foi utilizada pois essa garante um melhor uso dos recursos e também que os dados lidos de forma subsequente são temporalmente consistentes e representam instantes de amostragem adequados.

Figura 81 – Funções - Coleta de dados sem conversão

```
int mpu6050_getGyroscopeRaw(u16* receivedDataBuffer, u32* timestamp);
int mpu6050_getAccelerometerRaw(u16* receivedDataBuffer, u32* timestamp);
int mpu6050_getTemperatureRaw(float* receivedDataBuffer, u32* timestamp);
```

Fonte: Produzido pelos autores.

Outra variação das funções de coleta de dados são as funções definidas na Figura 82,

ao contrário das funções apresentadas anteriormente, essas funções realizam a conversão de cada um dos dados para sua respectiva unidade, ou seja, g para o acelerômetro, $°/s$ para o giroscópio e $°C$ para o sensor de temperatura.

Figura 82 – Funções - Coleta de dados com conversão

```
int mpu6050_getGyroscopeData(float* receivedDataBuffer, u8 genOption);
int mpu6050_getAccelerometerData(float* receivedDataBuffer, u8 genOption);
int mpu6050_getTemperatureData(float* receivedDataBuffer, u8 genOption);
```

Fonte: Produzido pelos autores.

Para finalizar as funções de coleta de dados, são definidas as funções na Figura 83, essas funções são essencialmente as mesmas funções apresentadas anteriormente, com a diferença de que essas funções tomam como argumento um número definido de amostras a ser coletado.

Figura 83 – Funções - Coleta de dados contínua com conversão

```
int mpu6050_sampleGyroscopeData(float* receivedDataBuffer, int samples, u8 genOption);
int mpu6050_sampleAccelerometerData(float* receivedDataBuffer, int samples, u8 genOption);
int mpu6050_sampleTemperatureData(float* receivedDataBuffer, int samples, u8 genOption);
```

Fonte: Produzido pelos autores.

Para realizar a configuração dos valores de fundo de escala é possível fazer uso das funções na Figura 84, o sensor de temperatura não tem acesso a tal programação pois tem uma escala fixa.

Um fator importante a ser considerado é que cada uma das configurações possui um valor de sensibilidade associado e, quanto maior o fundo de escala definido, menor a sensibilidade, por exemplo, o acelerômetro possui uma sensibilidade de 16384 LSB/g para $\pm 2g$ e de somente 2048 LSB/g para $\pm 16g$.

A questão do ajuste de sensibilidade através da configuração do fundo de escala pode ser uma característica essencial para um bom desempenho dos algoritmos de estimação tendo em vista que uma menor sensibilidade estabelece um limiar maior a ser superado para que movimentos sejam detectados podendo gerar portanto sinais com menor oscilação.

De um ponto de vista um pouco mais técnico, ao se realizar a conversão A/D dos sinais obtidos pelos sensores, a resolução irá controlar qual a faixa de valores que o dispositivo consegue efetivamente medir. Uma resolução muito alta permite realizar a detecção de pequenas variações, por outro lado, uma resolução mais baixa permite que o sinal de saída oscile menos, portanto, uma característica passa-baixas com frequência de corte dada pelo menor valor detectável pelo dispositivo é estabelecida.

Figura 84 – Funções - Configuração de fundo de escala

```
int mpu6050_setGyroFSR(u8 gyroFSR, u8 checkConsistency);
int mpu6050_setAccelFSR(u8 accelFSR, u8 checkConsistency);
```

Fonte: Produzido pelos autores.

Por fim, para finalizar as funções associadas ao sensores, a função definida na Figura 85 é responsável por desativar ou ativar o sensor de temperatura.

Apesar de não ter sido apreciado durante esse projeto devido a restrições de tempo, sensores de temperatura embutidos nesses tipos de dispositivos são essenciais para realizar a calibração ou compensação em *runtime* devido a variação de temperatura observada, levando a um ganho de precisão e consequentemente melhores resultados, principalmente por esse tipo de compensação representar um custo computacional muito baixo.

Figura 85 – Função - Controle do sensor de temperatura

```
int mpu6050_controlTemperatureSensor(u8 temperatureSensorOption);
```

Fonte: Produzido pelos autores.

As funções relacionadas a FIFO são apresentadas na Figura 86, essas funções são responsáveis por fazer a coleta de dados da FIFO e também a configuração associada a mesma. Algumas das funções presentes na Figura 86 foram omitidas por não apresentarem características relevantes a serem comentadas.

Essas funções foram separadas em um módulo próprio por serem bastante significantes para o projeto e para uma utilização mais adequada do dispositivo pois permitem um uso mais eficientes dos recursos de processamento disponíveis.

Além disso, essas funções também permitem garantir que os dados são temporalmente consistentes, desde que os algoritmos de estimação ou outros quaisquer que façam uso do dispositivo MPU6050 apresentem um desempenho aceitável.

Figura 86 – Funções - Configuração e coleta de dados da FIFO

```
int mpu6050_readFifoRaw(u16* gyroRawData, u16* accelRawData, u8* sensorsMask, u16* remainingAmount);
int  mpu6050_readFifoData(float* receivedDataBuffer, u8 genOption,u16* remainingAmount);
int mpu6050_readFifoParse(u16 bytesToRead, u8* fifoData, u16* remainingAmount);
int mpu6050_loopRead();
int mpu6050_configureFifoEna(u8 fifoEnaConfig, u8 checkConfigConsistency);
void mpu6050_getFIFOEnaStatus(u8* fifoEnaConfig);
int mpu6050_controlFifo(u8 setFifo);
int mpu6050_resetFifo();
u16  mpu6050_samplesInFIFO();
```

Fonte: Produzido pelos autores.

Assim como no caso das funções que coletam dados diretamente dos registradores dos dispositivo, também são definidas variações para as funções de coleta de dados na FIFO, nesse caso serão apresentadas 3(três) variações, essencialmente essas funções apresentam

funcionalidade e características bastante similares e se distinguem apenas em relação ao formato do dado retornado. .

A função apresentada na Figura 87 é responsável por coletar o dado da FIFO e devolve-lo ao usuário. Essa função sempre lê um número bem definido em *bytes*, um total de 12, referentes aos aos 3 eixos do giroscópio, e aos 3 eixos do acelerômetro.

Os pares de dados referentes a cada eixo são concatenados e devolvidos diretamente ao usuário em formato de 16 *bits* em complemento de 2 sem nenhum outro tipo de conversão, devido ao fato dessa função ser definida apenas para a coleta específica desses 2 sensores, caso sensores externos ou de temperatura estejam ativados na FIFO utilizar essa função levará a erros grosseiros nos valores coletados.

Figura 87 – Função - Leitura de dados da FIFO sem conversão

```
int mpu6050_readFifoRaw(u16* gyroRawData, u16* accelRawData, u8* sensorsMask, u16* remainingAmount);
```

Fonte: Produzido pelos autores.

A função definida na Figura 88 apresenta o mesmo funcionamento e problemas apresentados na função anterior, a diferença está no fato de ocorrer a conversão dos dados de 16 *bits* em complemento de 2 para as unidades dos sensores correspondentes.

Figura 88 – Função - Leitura de dados da FIFO com conversão

```
int  mpu6050_readFifoData(float* receivedDataBuffer, u8 genOption,u16* remainingAmount);
```

Fonte: Produzido pelos autores.

A ultima variação das funções de coleta de dados é apresentada na Figura 89, ela é responsável por uma coleta bem mais genérica, essa função recebe um número arbitrário de *bytes* a ser lido da FIFO e os retorna diretamente a um *buffer* especificado pelo usuário sem realizar nenhum tipo de operação sobre os mesmos, portanto, qualquer análise pode e deve ser feita de maneira externa.

É importante salientar que todas essas três funções apresentadas garantem não somente a consistência temporal dos dados mas também que os dados lidos são válidos através do monitoramento da condição de *overflow* da fila, e também do número de dados disponíveis para leitura.

Naturalmente, caso ocorra um *overflow*, apesar de dados subsequentes serem consistentes temporalmente, não há absolutamente nenhuma garantia de que são dados válidos para o instante atual, de forma similar, ler um número de *bytes* não disponível levará ao mesmo problema.

Devido aos problemas apresentados, é importante garantir que os algoritmos incorporados na plataforma tem um desempenho adequado, por exemplo, caso ocorra um *overflow* durante o período de operação, haverá a necessidade de realizar a reinicialização da fila,

essa reinicialização em muitos casos pode corresponder a vários instantes de amostragem a depender da taxa utilizada e o algoritmo ficará sem nenhuma nova informação durante todo esse período.

Figura 89 – Função - Leitura de dados da FIFO para análise externa

```
int mpu6050_readFifoParse(u16 bytesToRead, u8* fifoData, u16* remainingAmount);
```

Fonte: Produzido pelos autores.

Para que a FIFO disponível possa ser configurada é definida a função apresentada na Figura 90. No dispositivo MPU6050 diversos dispositivos podem ser configurados para que seus dados possam ser escritos na FIFO e, portanto, acessados diretamente através dela. Entre esses dispositivos estão os giroscópios, acelerômetros, sensor de temperatura e dispositivos externos.

Figura 90 – Função - Configuração de habilitação da FIFO

```
int mpu6050_configureFifoEna(u8 fifoEnaConfig, u8 checkConfigConsistency);
```

Fonte: Produzido pelos autores.

Dada a necessidade de reinicialização eventual da FIFO, a função apresentada na Figura 91 é responsável por realizar tal ação. Idealmente, caso o código implementado possua desempenho adequado, essa função só deveria ser utilizada uma única vez, durante o procedimento de inicialização.

Como durante a configuração inicial do dispositivo diversas funcionalidades são ativadas e, caso habilitada pelo usuário, ao finalizar o processo de configuração a fila já conterá um número arbitrário de dados, e em muitos caso já estará em condição de *overflow* antes mesmo de iniciar qualquer processo de estimação, com isso, após finalizar o procedimento de inicialização do dispositivo é recomendável reiniciar a estrutura.

Figura 91 – Função - Reinicialização da FIFO

```
int mpu6050_resetFifo();
```

Fonte: Produzido pelos autores.

A ultima função relacionada a FIFO é apresentada na Figura 92, essa função basicamente retorna o número atual de amostras presentes na FIFO e é utilizada como auxiliar para outras funções para que se possa evitar possível *overflow* na fila e também garantir que os dados que o usuário está tentando ler são válidos.

Além disso, essa função pode ser utilizada diretamente para que o usuário possa construir alguma estrutura de controle para leitura dos dados na fila.

Figura 92 – Função - Retorno no número de amostras na FIFO

```
u16   mpu6050_samplesInFIFO();
```

Fonte: Produzido pelos autores.

Para finalizar o procedimento de integração dos sensores inerciais à plataforma Minized será apresentado o ultimo módulo de funções, as funções de utilidade apresentadas na Figura 93.

Esse módulo foi pensado inicialmente para incluir funções bastante simples e que pudessem auxiliar durante o desenvolvimento do código, porém, durante o desenvolvimento do projeto observou-se a necessidade de adicionar mais funções para auxiliar no processo de estimação, para realização de *debug*, entre outros.

Muitas dessas funções poderiam facilmente ser agrupadas junto a outros tipos apresentados, porém, optou-se por mantê-las agrupadas nesse módulo por não representarem características essenciais para plataforma em si.

Figura 93 – Funções - Utilidades

```
int mpu6050_getData(float* receivedDataBuffer);
int mpu6050_sanityTest();
int mpu6050_deviceReset(u8 resetOption);
int mpu6050_controlSleepMode(u8 sleepModeOption);
u8 getWhoAmI();
u16 getU16Data(u8 dataMSB, u8 dataLSB);
void delay(u32 delay_us);
void mpu6050_getSensorsStatus(u8* sensorsON);
int mpu6050_getIntrStatus(u16* intrStatus);
int mpu6050_readDebug(u8 registerAddr, u8 expectedData, u8 option);
int mpu6050_regDump();
int mpu6050_readReg(u8 registerAddr, u8* dataReceived);
int mpu6050_setBypass(u8 bypass);
int mpu6050_configMasterRead(u8 i2c_slave_addr, u8 read_addr, u8 byte_count);
int mpu6050_calibrate(float* gyroBias, float* accelBias);
```

Fonte: Produzido pelos autores.

As funções a serem apresentadas tem relação ao desenvolvimento do processo de estimação e foram incorporadas para promover algumas funcionalidades que poderiam permitir aprimorar os resultados da estimação. Algumas das funções foram omitidas por não apresentarem característica de interesse paro o projeto.

A função apresentada na Figura 94 permite ativar o modo de *bypass* do dispositivo, esse modo permite que as portas I2C auxiliares da MPU6050 possam ter lógica controlada diretamente pelo mestre principal, ou seja, o mestre responsável por controlar a MPU6050. Esse modo permite uma configuração muita mais rápida e muito mais simples de dispositivos auxiliares.

Figura 94 – Função - Ativação do modo de *bypass*

```
int mpu6050_setBypass(u8 bypass);
```

Fonte: Produzido pelos autores.

As funções apresentadas na Figura 95 são responsáveis por configurar a escrita e leitura para um escravo auxiliar definido caso o modo de mestre seja necessário durante operação com sensores auxiliares.

Elas podem ser utilizadas por exemplo, para fazer a leitura de dados dos registradores de dados de um dispositivo auxiliar de acordo com a taxa de amostragem configurada, ou enviar um comando de coleta única de dados.

Figura 95 – Função - Configuração de leitura e escrita em escravo auxiliar

```
int mpu6050_configMasterRead(u8 i2c_slave_addr, u8 read_addr, u8 byte_count);
int mpu6050_configMasterWrite(u8 i2c_slave_addr, u8 write_addr, u8 byte_count, u8 data);
```

Fonte: Produzido pelos autores.

Por fim, a função definida na Figura 96 é responsável por fazer uma calibração bastante simples dos sensores giroscópios e acelerômetros do dispositivo, para tanto, o dispositivo deve estar em uma orientação adequada e permanecer inerte por um breve momento para que possa ser feita a coleta de dados e possa se estabelecer um valor médio observado para que possa ser atribuído aos vieses associados a cada eixo dos sensores testados.

A janela de dados para análise é definida pelo usuário sendo que os primeiros dados coletados são rejeitados para evitar coleta de dados durante o período de estabilização dos sensores.

Figura 96 – Função - Calibração dos sensores giroscópios e acelerômetros

```
int mpu6050_calibrate(float* gyroBias, float* accelBias);
```

Fonte: Produzido pelos autores.

3.3.5.3 Funcionalidades - QMC5883L

Por fim, essa subseção abordará brevemente o código implementado para garantir que as funcionalidades do dispositivo QMC5883L pudessem ser extraídas.

Esse dispositivo é o magnetômetro utilizado no projeto e foi incorporado após a realização de testes e verificação da necessidade de aprimoramento do processo de estimação da orientação, mais especificamente no que tange a estimação do ângulo de guinada ou *yaw*.

Como o código implementado essencialmente possui características bastante similares a alguns dos trechos abordados na subseção 3.3.5.2 e também não apresenta tantas características a serem destacadas como foi no caso da implementação das funcionalidades

da MPU6050, a apresentação se dará de forma breve e tem como objetivo apenas garantir que todos os aspectos da implementação do *software* tenham sido abordados.

As funções apresentadas na Figura 97 tem como objetivo realizar a inicialização geral do sistema, assim como no caso da MPU6050 esse procedimento consiste em garantir que o sistema esteja operacional e com a configurações adequadas em relação a taxa de amostragem, modo de operação, FSR, e outros.

Uma peculiaridade dessas funções é que também são responsáveis por garantir toda a comunicação entre a MPU6050 e o QMC5883L seja possível através de *bypass* ou em modo mestre-escravo, em que o sensor QMC5883L é escravo da MPU6050, tal característica sendo essencial no processo para que a MPU6050 possa acessar diretamente os dados do sensor auxiliar e coloca-los na fila, garantindo assim que os dados do acelerômetro, giroscópio e magnetômetro sejam consistentes para um mesmo instante de tempo.

Figura 97 – Funções - Inicialização - QMC5883L

```
int qmc5883l_init();
int qmc5883l_reset();
```

Fonte: Produzido pelos autores.

As funções apresentadas na Figura 98, são responsáveis por realizar toda a configuração do dispositivo como, por exemplo, taxa de amostragem, filtro passa-baixas, modo de operação, e outro, gozando das mesmas características já abordadas na subseção 3.3.5.2.

Figura 98 – Funções - Configurações gerais - QMC5883L

```
int qmc5883l_setSampleRate(u8 qmc5883l_rate);
int qmc5883l_setOverSampleRate(u8 qmc5883l_orate);
int qmc5883l_setResolution(u8 qmc5883l_fsr);
int qmc5883l_setMode(u8 qmc5883l_mode);
int qmc5883l_setIntr(u8 state);
```

Fonte: Produzido pelos autores.

Por fim, foram garantidos também as funções de coleta de dados conforme Figura 99. Além disso, foram implementadas também funções para realização de configurações específicas e processo de *debug*, sendo essas omitidas por não serem de interesse para o projeto.

Figura 99 – Funções - Coleta de dados - QMC5883L

```
int qmc5883l_getData(float* receivedDataBuffer);
float qmc5883l_convertData(u16 encodedData2sComplement);
```

Fonte: Produzido pelos autores.

3.4 Seleção de algoritmos de odometria inercial e implementação

Para realizar o procedimento de estimação da posição e orientação utilizando os sensores inerciais existem uma grande quantidade de algoritmos disponíveis como, por exemplo, algoritmos de referência expostos por Madgwick, Harrison e Vaidyanathan (2011), e Mahony, Hamel e Pflimlin (2008), assim como diferentes formulações desses mesmos algoritmos. Algoritmos de odometria inercial são um tópico bastante extenso de pesquisa e desenvolvimento e têm sido aprimorados cada vez mais devido à crescente demanda por soluções autônomas e sua aplicabilidade em diversos setores da industria.

Apesar da ampla gama de possibilidades é necessário observar as diversas limitações existentes no projeto e na plataforma disponível para garantir que um algoritmo ou uma combinação de algoritmos e uma formulação adequada dos mesmos seja utilizada.

Entre os fatores críticos para o projeto se destacam: o tempo disponível para o desenvolvimento, o poder de processamento disponível, a memória utilizada e a robustez dos dispositivos sensores utilizados.

Em relação ao tempo disponível, é necessário a observância dos prazos estabelecidos para o projeto. Quando se trata de desenvolvimento, é notável o fato de que diversos testes, reformulações e integração de novas soluções são etapas essenciais a qualquer projeto. Portanto, para garantir que todos os prazos serão atendidos deve-se realizar um *trade-off* entre a complexidade dos algoritmos propostos e um processo de desenvolvimento adequado.

Já em relação a plataforma, apesar da Minized contar com um ARM Cortex-A9 *single-core* que possui poder de processamento relativamente elevado quando se comparado a outros dispositivos mais simples, ela ainda possui limitações bastante notáveis, e não somente em relação a esse aspecto, mas a diversos outros.

O processo de estimação geralmente envolve um número bastante expressivo de operações de alto custo e, quando se trata de algoritmos mais complexos e robustos é bastante comum uso constante de operações envolvendo matrizes como, por exemplo, multiplicação e inversão, que notavelmente possuem um custo computacional bastante elevado. Esse problema se torna ainda mais crítico quando se considera a integração dos algoritmos de odometria visual e inercial em conjunto na plataforma pois o sistema deverá ser capaz de atender as requisições tanto dos sensores inercias quanto da câmera e processar os dados relativos a cada um deles.

Com isso, é importante propor uma solução que seja adequada a plataforma através do uso de algoritmos mais simples, diferentes formulações, e a otimização do código utilizado para que a solução possa ser incorporada com sucesso na plataforma.

Apesar da memória não ser um fator tão crítico para esses algoritmos, ao menos em

formulações mais simples, tratando-se de um sistema embarcado é sempre recomendável otimizar o uso da memória, portanto, também pode ser considerado um fator a ser analisado durante a escolha e implementação desses algoritmos.

Por fim, os sensores utilizados são um fator crítico no projeto pois é através dos dados coletados por eles que o processo de estimação é feito, se os dados obtidos não tiverem precisão adequada e forem ruidosos, inexoravelmente os resultados obtidos durante o processo de estimação não serão adequados e, em muitos casos, o processo se torna inviável. Portanto, tratando-se de sensores de baixo custo, uma alternativa viável a implementação de algoritmos cada vez mais complexos é o desenvolvimento de algoritmos simples e posterior aprimoramento por meio do tratamento adequado dos dados disponíveis.

Com as considerações apresentadas, inicialmente não existe uma escolha única para um algoritmo que apresente definitivamente a melhor solução para o problema apresentado, porém, é possível associar implementações mais robustas e otimizadas a resultados com maior desempenho e precisão. Naturalmente, quanto mais robusta a solução, maior tende a ser o arcabouço teórico necessário para viabilizar a solução, assim como maior a complexidade do algoritmo a ser implementado.

Considerando então as questões apresentadas, devido a limitações de tempo e também da base teórica necessária para desenvolver algoritmos complexos, nesse projeto a solução será abordada de forma relativamente progressiva, abordando inicialmente algoritmos mais simples como o *dead reckoning* e o filtro complementar, e progredindo então para uma implementação simples do filtro de Kalman e, por fim, tentando promover o aprimoramento dessas soluções através de métodos mais simples, sendo possível então obter resultados relativamente satisfatórios para o projeto em questão sem recorrer a soluções complexas e de alto custo, até mesmo devido as limitações de recursos apresentadas.

3.4.1 Algoritmo 1 - *Dead reckoning*

Dead reckoning é o processo de estimar a posição ou orientação atual a partir das informações de posição e orientação anteriores e dos dados obtidas pelos sensores inerciais e também o tempo decorrido entre os instantes de análise.

Apesar de simples, esse tipo de estimação é bastante impreciso devido ao fato de utilizar apenas integrações durante o processo de estimação e não realizar nenhuma correção para mitigar a influência dos erros envolvidos no processo, dessa forma, o resultado tende a acumular erros significativos durante o tempo.

As Equações 3.3 e 3.4 apresentam exemplos discretos do procedimento. No primeiro caso θ_k representa a orientação relativa a um eixo específico em um instante de tempo k e ω_k a velocidade angular medida. No segundo caso, P_k representa a posição em relativa a um eixo específico em um instante de tempo k, $\mathbf{v}$ a velocidade linear obtida através da integração

das medidas da aceleração, e a **a** aceleração medida. Para ambos os casos T representa o período de amostragem dos dados.

$$\theta_{k+1} = \theta_k + \omega_k.T \tag{3.3}$$

$$P_{k+1} = P_k + v_k.T + 0.5.a_k.T^2 \tag{3.4}$$

Para ilustrar os problemas desse procedimento considera-se o problema de estimação da orientação em uma único eixo. Tome uma medida ω tal que $\omega = \omega_k + \omega_k^e$ em que ω representa o valor medido, ω_k o valor real e ω_k^e a contribuição dos erros associados. Note que para um instante de tempo t qualquer têm-se que:

$$\begin{aligned}\theta_{k+1} &= \theta_k + \omega.T \\ &= \theta_k + (\omega_k + \omega_k^e).T \\ &= \theta_k + \omega_k.T + \omega_k^e.T\end{aligned}$$

Para um instante k qualquer θ_k pode ser definido como $\theta_k = \theta_0 + \sum_{k=0}^{N} \omega_k.T + \sum_{k=0}^{N} \omega_k^e.T$. Em que $\theta_0 + \sum_{k=0}^{N} \omega_k.T$ representará a orientação real em relação a um referencial e $\sum_{k=0}^{N} \omega_k^e.T$ a contribuição do erro até aquele instante. Dessa forma, ao incorporar novos dados o erro tenderá a crescer indefinidamente.

Apesar de não ser realmente útil por conta própria, esse algoritmo é essencial não somente por permitir uma melhor compreensão dos problemas a qual esse tipo de sistema está sujeito, mas também por ser a base de muitos outros algoritmos pois representa a dinâmica fundamental desse tipo de sistema.

3.4.2 Algoritmo 2 - Filtro complementar

Comumente existem casos em que uma única informação ou variável de interesse pode ser medida por vários dispositivos ou meios diferentes. Quando se estima uma informação através do uso de diferentes dispositivos e cada um possui características distintas no que diz respeito ao ruído, caso essas características sejam complementares, ou seja, um apresente informações adequadas em baixas frequências e o outro em altas frequências, o filtro complementar pode ser utilizado para estimar tal informação de maneira bastante eficiente (BROWN; HWANG, 2012) (BAI et al., 2020).

Conforme apresentado são necessárias duas fontes diferentes que permitam obter uma mesma informação, no projeto inicialmente estão disponíveis apenas os sensores do dispositivo MPU6050, ou seja, acelerômetros e giroscópios, porém, é possível considerar também o uso de magnetômetros. Considerando que apenas o o acelerômetro pode fornecer informação efetiva sobre a posição esse algoritmo não é capaz de estimar tal informação nessas condições.

Em relação aos problemas de cada dispositivo, é notável o fato de que o acelerômetro é sensível a qualquer tipo de força que atua sobre ele e, portanto, apresenta medidas bastante ruidosas, principalmente em casos dinâmicos. Considerando isso, o acelerômetro tende a ser confiável apenas no longo prazo quando suas medidas tendem a se estabilizar, portanto, as medidas desses sensores são uma boa indicativa da orientação em condições estáticas.

Utilizando um filtro passa-baixas sobre os dados provenientes do acelerômetro é possível eliminar ou reduzir a contribuição de forças muito elevadas ou muito baixas que muito provavelmente são advindas de perturbações no dispositivo e consequentemente não devem ter muito peso no resultado.

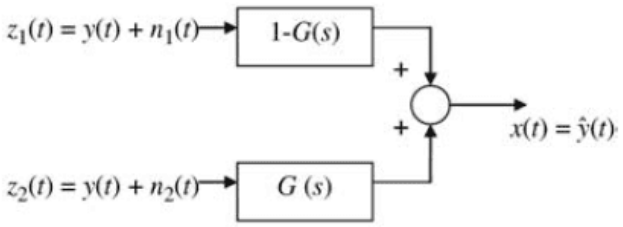
O giroscópio por sua vez não responde a todo tipo de força que atua sobre ele e, portanto, apresenta uma medida menos suscetível a ruídos. Utilizando as informações de taxa de variação angular para se obter orientação relativa a cada eixo é possível obter um resultado bastante preciso no curto prazo.

Devido as integrações inerentes ao processo, esse sensor fica sujeito a um desvio que tende a se intensificar ao longo do tempo e, portanto, não é confiável a longo prazo, porém, fornece uma boa indicativa da orientação em condições dinâmicas.

Utilizando um filtro passa-altas sobre os dados obtidos pelo giroscópios é possível dar um peso maior as informações atuais e relativamente precisas obtidas pelo giroscópio.

Somando os dois resultados, ou seja, incorporando a informação filtrada do acelerômetro a informação também filtrada do giroscópio, é possível corrigir os problemas de desvios apresentados mas ainda mantendo a precisão no curto prazo (BAI et al., 2020). A Figura 100 apresenta a construção básica do filtro complementar com comportamento definido pela Equação 3.5.

Figura 100 – Construção do filtro complementar



Fonte: Brown e Hwang (2012)

$$X(s) = Y(s) + N_1(s)[1 - G(s)] + N_2(s)[G(s)] \tag{3.5}$$

Apesar da construção apresentada na Equação 3.5, o sinal e ruído não estão disponíveis sem apresentar associação, dessa forma a Equação 3.5 se reduz a Equação 3.6 em que Y_g(s) é o resultado obtido através do giroscópio e Y_a(s) através do acelerômetro (BROWN; HWANG, 2012). O fator 1-G(s) geralmente toma valores próximos a 0,96 a 0,98 para dar ênfase nos valores atuais mas ainda realizar a correção do desvio que é essencial para o funcionamento do algoritmo.

$$X(s) = Y_g(s)[1 - G(s)] + Y_a[G(s)] \tag{3.6}$$

É importante salientar que somente os ângulos de *roll*, Equação 3.7, e *pitch*, Equação 3.8, podem ser fornecidos pelo acelerômetro, dessa forma, ao utilizar somente os dados do acelerômetro e do giroscópio o ângulo de *yaw* tenderá a continuar o desvio decorrente do processo de integração inerente ao processo e, portanto, apresentará erros bastante expressivos a medida que o tempo passa.

Uma forma de corrigir tal problema é introduzindo ao sistema um magnetômetro, pois esse é capaz de fornecer informações de direcionamento ou *yaw*, Equação 3.9, com precisão relativamente alta. As Equações 3.7, 3.8 e 3.9 aqui abordadas são definidas em Bai et al. (2020).

As informações de orientação advindas do giroscópio podem ser obtidas através dos método apresentados na subseção 3.4.1.

$$Roll(\phi) = tan^{-}1\left(\frac{a_k^y}{\sqrt{a_k^{x2} + a_k^{z2}}}\right) \tag{3.7}$$

$$Pitch(\theta) = tan^{-}1\left(\frac{a_k^{x2}}{\sqrt{a_k^{y2} + a_k^{z2}}}\right) \tag{3.8}$$

$$Yaw(\psi) = tan^{-}1\left(\frac{mag_k^z * sen(\phi) - mag_k^y * cos(\phi)}{mag_k^x * cos(\theta) + mag_k^y * sen(\theta) * sen(\phi) + mag_k^z * sen(\theta) * cos(\phi)}\right) \tag{3.9}$$

3.4.2.1 Implementação - Filtro complementar

O algoritmo pode ser implementado de diversas formas mas aqui serão especificadas 4(quatro) etapas distintas para realização de todo o procedimento.

Etapas: Filtro complementar

1. Receber e preparar os dados do giroscópio e do acelerômetro.
2. Obter a orientação a partir dos dados do acelerômetro de acordo com as Equações 3.7 e 3.8.
3. Obter orientação a partir dos dados do giroscópio de acordo com a Equação 3.3.
4. Filtrar os dados de [2] e [3] de acordo com a Equação 3.6 e atualizar o estado atual.
    - Opcional: Incorporar os dados do magnetômetro de acordo com Equação 3.9.

O algoritmo apresentado é bastante simples de ser implementado, tem um custo computacional bastante baixo devido as poucas operações necessárias, e utiliza pouco espaço na memória pois necessita armazenar somente a orientação atual estimada para para os três eixos. Porém, o fato do algoritmo utilizar poucos recursos não significa otimizar o uso de recursos, deve-se extrair o melhor do sistema com o menor impacto possível.

Devido a essas características esse algoritmo apresenta uma boa opção para ser utilizado no sistema proposto para estimação da orientação.

Considerando que esse algoritmo possui apenas um parâmetro ajustável e portanto não permite grande flexibilidade, é importante levar em consideração algoritmos que possam ser implementados de maneira inteligente e possam ser aprimorados eventualmente para se tornarem cada vez mais robustos considerando as limitações apresentadas.

3.4.3 Algoritmo 3 - Filtro de Kalman

Conforme apresentado na subseção 2.11.2, para estimar qualquer informação de um processo primeiro é necessário conhecer a dinâmica do sistema em questão por meio de uma descrição ou modelo apropriado.

Do ponto de vista de implementação, o filtro de Kalman é um algoritmo de tipo recursivo e é relativamente fácil de ser implementado em sistemas de processamento de diversos portes (ROMANIUK; GOSIEWSKI, 2014). Naturalmente, as diversas formulações existentes tanto para a representação do sistema quanto para o filtro levarão a diferentes custos computacionais. Em muitos casos, sistemas de processamento com capacidade muito limitada podem não ser capazes de realizar as iterações do algoritmo em tempo hábil, principalmente caso a taxa de amostragem de dados seja elevada.

Para modelar um sistema no espaço de estados é necessário definir essencialmente três tipos de variáveis, variáveis de entrada, variáveis de saída, e variáveis de estado. Todas essas variáveis estão envolvidas na dinâmica do sistema e representam diversas características.

Para que se possa realizar o processo de estimação tanto da posição quanto da orientação por meio do filtro de Kalman é necessário, portanto, definir quais as variáveis envolvidas, como cada informação se relaciona, e construir efetivamente um modelo para o sistema em análise.

Para iniciar o processo deve-se definir como extrair as informações desejadas das informações disponíveis dos sensores inerciais, ou seja, como obter a posição a partir de informações da aceleração linear, e a orientação a partir de informações da velocidade angular. Uma maneira bastante direta e simples já foi apresentada anteriormente na subseção 3.4.1, e será adotada inicialmente.

3.4.3.1 Modelagem do sistema - Orientação

Para estimação da orientação define-se as Equações 3.10, 3.11, e 3.12, em que ϕ, θ e ψ representam o ângulo de *roll*, *pitch* e *yaw* respectivamente.

$$\phi_{k+1} = \phi_k + \omega_k^x.T \tag{3.10}$$

$$\theta_{k+1} = \theta_k + \omega_k^y.T \tag{3.11}$$

$$\psi_{k+1} = \psi_k + \omega_k^z.T \tag{3.12}$$

Apesar do modelo apresentado ser bastante simples ele considera que as medidas para velocidade ângular em cada um dos eixos, ω_k^x, ω_k^y e ω_k^z, ou são isentas de erro, ou já foram corrigidas de alguma forma. A primeira alternativa naturalmente não é adequada tendo em vista que qualquer medida é corrompida de alguma forma por erros de diversas naturezas, porém, considerar que as medidas já foram corrigidas a priori também pode não ser uma alternativa interessante.

Conforme apresentado na subseção 2.11.2, o filtro de Kalman consegue utilizar toda a informação disponível para realizar as estimativas, dessa forma, incorporar novas informações pode ser uma maneira de promover ganhos significativos na precisão das informações obtidas no processo.

As informações dos giroscópio podem ser distorcidas por *biases*, esses *biases* tendem a variar com a temperatura e com o tempo e podem contribuir para diminuir a precisão das estimativas, portanto, podem e devem ser uma informação estimada pelo algoritmo de forma a promover melhores resultados durante o processo (ROMANIUK; GOSIEWSKI, 2014). Com isso, define-se as equações iniciais que descrevem a dinâmica do sistema, Equações 3.13, 3.14, e 3.15, em que, ϕ_k, θ_k e ψ_k representam o ângulo de *roll*, *pitch* e *yaw* para o instante de

tempo k, ω_k^x, ω_k^y e ω_k^z as velocidades angulares medidas nos eixos x, y e z no mesmo instante, por fim, $\omega_k^{b_x}$, $\omega_k^{b_y}$ e $\omega_k^{b_z}$ os *biases* para cada um dos eixos, x, y e z.

$$\begin{aligned} \phi_{k+1} &= \phi_k + (\omega_k^x - \omega_k^{b_x}).T \\ \omega_{k+1}^{b_x} &= \omega_k^{b_x} \end{aligned} \tag{3.13}$$

$$\begin{aligned} \theta_{k+1} &= \theta_k + (\omega_k^y - \omega_k^{b_y}).T \\ \omega_{k+1}^{b_y} &= \omega_k^{b_y} \end{aligned} \tag{3.14}$$

$$\begin{aligned} \psi_{k+1} &= \psi_k + (\omega_k^z - \omega_k^{b_z}).T \\ \omega_{k+1}^{b_z} &= \omega_k^{b_z} \end{aligned} \tag{3.15}$$

Dado o modelo expresso pelas Equações 3.13, 3.14, e 3.15, para continuar a modelagem do sistema as variáveis de estado, de entrada, e também de saída do sistema devem ser definidas, após isso, deve-se extrair as matrizes necessárias para colocar o sistema no espaço de estados.

Uma alternativa para realizar as definições apresentadas seria considerar um sistema que envolvesse a estimativa para todos os eixos simultaneamente, porém, ao manter uma instância separada do filtro para cada um dos eixos é possível diminuir significativamente o custo computacional envolvido pois as operações com matrizes tendem a ter um custo bem mais elevado quanto maior o número de entradas nas matrizes envolvidas (ROMANIUK; GOSIEWSKI, 2014).

Com isso, define-se as variáveis de estado, variáveis de entrada e saída do sistema, assim como o vetor de entrada, saída e de estado do sistema de acordo com as Equações 3.16 - 3.24. Os sobrescritos x, y e z, indicam a associação da respectiva variável ao eixo correspondente.

$$x^x = \begin{bmatrix} \phi & \omega^{b_x} \end{bmatrix}^T \tag{3.16}$$
$$x^y = \begin{bmatrix} \theta & \omega^{b_y} \end{bmatrix}^T \tag{3.17}$$
$$x^z = \begin{bmatrix} \psi & \omega^{b_z} \end{bmatrix}^T \tag{3.18}$$
$$u^x = \begin{bmatrix} \omega^x \end{bmatrix} \tag{3.19}$$
$$u^y = \begin{bmatrix} \omega^y \end{bmatrix} \tag{3.20}$$
$$u^z = \begin{bmatrix} \omega^z \end{bmatrix} \tag{3.21}$$
$$y^x = \begin{bmatrix} \phi \end{bmatrix} \tag{3.22}$$
$$y^y = \begin{bmatrix} \theta \end{bmatrix} \tag{3.23}$$
$$y^z = \begin{bmatrix} \psi \end{bmatrix} \tag{3.24}$$

Define-se ainda a matriz de estado A, a matriz de entrada B e a matriz de saída H de acordo com as Equações 3.25, 3.26 e 3.27.

$$A = \begin{bmatrix} 1 & -T \\ 0 & 1 \end{bmatrix} \tag{3.25}$$
$$B = \begin{bmatrix} T \\ 0 \end{bmatrix} \tag{3.26}$$
$$H = \begin{bmatrix} 1 & 0 \end{bmatrix} \tag{3.27}$$

As definições do modelo completo para cada um dos eixos são apresentadas nas Equações 3.28, 3.29 e 3.30.

$$\begin{bmatrix} \phi_{k+1} \\ \omega_{k+1}^{b_x} \end{bmatrix} = \begin{bmatrix} 1 & -T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \phi_k \\ \omega_k^{b_x} \end{bmatrix} + \begin{bmatrix} T \\ 0 \end{bmatrix} \begin{bmatrix} \omega_k^x \end{bmatrix} \quad (3.28) \qquad \begin{bmatrix} \theta_{k+1} \\ \omega_{k+1}^{b_y} \end{bmatrix} = \begin{bmatrix} 1 & -T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \theta_k \\ \omega_k^{b_y} \end{bmatrix} + \begin{bmatrix} T \\ 0 \end{bmatrix} \begin{bmatrix} \omega_k^y \end{bmatrix} \quad (3.29)$$

$$\begin{bmatrix} \psi_{k+1} \\ \omega_{k+1}^{b_z} \end{bmatrix} = \begin{bmatrix} 1 & -T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \psi_k \\ \omega_k^{b_z} \end{bmatrix} + \begin{bmatrix} T \\ 0 \end{bmatrix} \begin{bmatrix} \omega_k^z \end{bmatrix} \quad (3.30)$$

As matrizes de covariância do ruído associado ao processo, Q, e da covariância do ruído associado as medições, R, são definidas nas Equações 3.31 e 3.32. Em que σ_α representa o ruído na medida da orientação, e σ_{ω^i} o ruído na medida da velocidade angular.

$$Q = \begin{bmatrix} T^2 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} {\sigma_\alpha}^2 \end{bmatrix} \quad (3.31) \qquad R = \begin{bmatrix} \sigma_{\omega^i}^2 \end{bmatrix} \quad (3.32)$$

3.4.3.2 Modelagem do sistema - Posição

De forma bastante similar ao caso da orientação, é possível utilizar inicialmente a dinâmica definida na subseção 3.4.1 para definir o sistema para estimação da posição.

Conforme apresentado no caso da orientação, o *bias* é uma informação bastante interessante de ser incorporada na dinâmica do sistema para que durante a estimação seja atualizada junto as outras informações do filtro, mesmo que sensores de alta qualidade e bem calibrados estejam disponíveis e, portanto, será uma informação definida já inicialmente no modelo.

Para estimação da posição define-se as Equações 3.33, 3.34, e 3.35, em que p_k^x, p_k^y e p_k^z representam a posição relativa ao eixo x, y e z no instante k, respectivamente.

$$\begin{aligned} p_{k+1}^x &= p_k^x + v_k^x.T + 0.5(a_k^x - a_k^{b_x}).T^2 \\ v_{k+1}^x &= v_k^x + (a_k^x - a_k^{b_x}).T \\ a_{k+1}^{b_x} &= a_k^{b_x} \end{aligned} \quad (3.33)$$

$$\begin{aligned} p_{k+1}^y &= p_k^y + v_k^y.T + 0.5(a_k^y - a_k^{b_y}).T^2 \\ v_{k+1}^y &= v_k^y + (a_k^y - a_k^{b_y}).T \\ a_{k+1}^{b_y} &= a_k^{b_y} \end{aligned} \quad (3.34)$$

$$\begin{aligned} p_{k+1}^z &= p_k^z + v_k^z.T + 0.5(a_k^z - a_k^{b_z}).T^2 \\ v_{k+1}^z &= v_k^z + (a_k^z - a_k^{b_z}).T \\ a_{k+1}^{b_z} &= a_k^{b_z} \end{aligned} \quad (3.35)$$

Dado o modelo expresso pelas Equações 3.33, 3.34, e 3.35, as variáveis de estado, de entrada, e de saída do sistema devem ser definidas. Nesse caso, uma instância do filtro para cada eixo também será definida de forma a reduzir os custos computacionais envolvidos.

Com isso, define-se as variáveis de estado, variáveis de entrada e saída do sistema, assim como o vetor de entrada, saída e de estado do sistema de acordo com as Equações 3.36 - 3.44. Note que apesar dos vetores de entrada, saída e estado tomarem mesma definição daqueles apresentados nas Equações 3.16 - 3.24, u^x a u^z, y^x a y^z e x^x a x^z, esses são referentes as instâncias do filtro relacionados a estimação da posição e apenas representam a notação, não devendo ser confundidos com aqueles utilizados para estimação da orientação.

$$x^x = \begin{bmatrix} p^x & v^x & a^{b_x} \end{bmatrix}^T \quad (3.36) \qquad x^y = \begin{bmatrix} p^y & v^y & a^{b_y} \end{bmatrix}^T \quad (3.37) \qquad x^z = \begin{bmatrix} p^z & v^z & a^{b_z} \end{bmatrix}^T \quad (3.38)$$

$$u^x = \begin{bmatrix} a^x \end{bmatrix} \quad (3.39) \qquad u^y = \begin{bmatrix} a^y \end{bmatrix} \quad (3.40) \qquad u^z = \begin{bmatrix} a^z \end{bmatrix} \quad (3.41)$$

$$y^x = \begin{bmatrix} p^x \end{bmatrix} \quad (3.42) \qquad y^y = \begin{bmatrix} p^y \end{bmatrix} \quad (3.43) \qquad y^z = \begin{bmatrix} p^z \end{bmatrix} \quad (3.44)$$

Define-se ainda a matriz de estado A, a matriz de entrada B e a matriz de saída H de acordo com as Equações 3.45, 3.46 e 3.47.

$$A = \begin{bmatrix} 1 & T & -0.5.T^2 \\ 0 & 1 & -T \\ 0 & 0 & 1 \end{bmatrix} \quad (3.45) \qquad B = \begin{bmatrix} 0.5.T^2 \\ T \\ 0 \end{bmatrix} \quad (3.46) \qquad H = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \quad (3.47)$$

Por fim, as definições do modelo completo para o caso da posição relativo a cada um dos eixos são apresentadas nas Equações 3.48, 3.49 e 3.50.

$$\begin{bmatrix} p^x_{k+1} \\ v^x_{k+1} \\ a^{b_x}_{k+1} \end{bmatrix} = \begin{bmatrix} 1 & T & -0.5T^2 \\ 0 & 1 & -T \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p^x_k \\ v^x_k \\ a^{b_x}_k \end{bmatrix} + \begin{bmatrix} 0.5T^2 \\ T \\ 0 \end{bmatrix} \begin{bmatrix} a^x_k \end{bmatrix} \quad (3.48)$$

$$\begin{bmatrix} p^y_{k+1} \\ v^y_{k+1} \\ a^{b_y}_{k+1} \end{bmatrix} = \begin{bmatrix} 1 & T & -0.5T^2 \\ 0 & 1 & -T \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p^y_k \\ v^y_k \\ a^{b_y}_k \end{bmatrix} + \begin{bmatrix} 0.5T^2 \\ T \\ 0 \end{bmatrix} \begin{bmatrix} a^y_k \end{bmatrix} \quad (3.49)$$

$$\begin{bmatrix} p^z_{k+1} \\ v^z_{k+1} \\ a^{b_z}_{k+1} \end{bmatrix} = \begin{bmatrix} 1 & T & -0.5T^2 \\ 0 & 1 & -T \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p^z_k \\ v^z_k \\ a^{b_z}_k \end{bmatrix} + \begin{bmatrix} 0.5T^2 \\ T \\ 0 \end{bmatrix} \begin{bmatrix} a^z_k \end{bmatrix} \quad (3.50)$$

As matrizes de covariância do ruído associado ao processo, Q, e da covariância do ruído associado as medições, R, são definidas nas Equações 3.51 e 3.52. Em que σ_{p^i} representa o ruído na medida da posição, e σ_{a^i} o ruído na medida da aceleração.

$$Q = \begin{bmatrix} 0.25T^4 & 0.5T^3 & 0 \\ 0.5T^3 & T^2 & 0 \\ 0 & 0 & 0 \end{bmatrix} \left[\sigma_{p^i}{}^2\right] \quad (3.51) \qquad R = \left[\sigma_{a^i}^2\right] \quad (3.52)$$

As matrizes de covariância do ruído associado ao processo e da covariância do ruído associado as medições foram adaptadas ou retiradas de Romaniuk e Gosiewski (2014), referência também utilizada para derivação dos modelos apresentados.

3.4.3.3 Implementação - Filtro de Kalman

A priori o algoritmo foi implementado em uma de suas formas padrão. Para realização de uma única iteração são definidas as etapas abaixo:

Etapas: Filtro de Kalman

1. Receber uma nova informação.
2. Realizar uma previsão a priori para o estado e matriz de covariância do sistema.
3. Realizar uma atualização da estimativa incorporando os novos dados disponíveis.
    - Opcional: Incorporar os dados do magnetômetro e acelerômetro de acordo com a Equação 3.9, e Equações 3.7 e 3.8

No caso do filtro de Kalman a etapa denominada *Opcional* consiste em adicionar informações redundantes no processo para que a estimação tenha melhores resultados, considerando que as informações tem uma precisão relativamente aceitável. Apesar de possivelmente promoverem um ganho no desempenho do filtro essas informações não são essenciais, sendo possível utilizar parte da própria estimativa realizada como dado a ser incorporado na atualização, alternativa, porém, que tende a levar a resultados com menor precisão mas não necessariamente inadequados.

Considerando a implementação em *software*, o algoritmo utiliza diversas operações com matrizes e, portanto, implementar as funções mais características para tratar dessas operações é necessário. Como essas funções não são de interesse imediato para o projeto elas foram omitidas, porém, podem ser acessadas e visualizadas no código principal implementado.

Em suas formulações mais básicas o filtro de Kalman é um algoritmo relativamente simples de ser implementado, porém, tem um custo computacional bastante elevado. Além disso, apesar de não necessitar armazenar muitas informações durante a execução, em comparação aos algoritmos já apresentados o filtro de Kalman acaba utilizando uma quantidade bem mais significativa de memória.

Assim como foi o caso para os demais algoritmos apresentados, o código foi pensado e implementado de forma a utilizar a menor quantidade de recursos possíveis, porém, o foco da implementação foi o desempenho, de forma que em muitos casos optou-se por um maior uso de memória em prol de um menor custo computacional.

Já em relação a construção, o código foi implementado da maneira mais genérica possível, de forma que qualquer sistema pudesse ser definido. Dessa forma, para adaptar o algoritmo para uma nova aplicação basta que as matrizes associadas a representação do sistema sejam fornecidas, assim como as estimativas iniciais para o estado e covariância do erro do sistema.

O filtro de Kalman representa uma opção excelente para o projeto tendo em vista que é um algoritmo bastante versátil e possui diversas implementações como, por exemplo, UKF, EKF e *Square-root*, assim como várias formulações distintas como algoritmos baseados em quaternions ou *Joseph's Stabilized Form*.

O algoritmo não conta somente com diferentes formulações e implementações mas diversas formas de aprimoramento, por exemplo, através do uso de suavizadores (do inglês, *smoothers*), que também podem ser incorporados eventualmente.

Além das características apresentadas, o algoritmo dispõe de vários parâmetros ajustáveis para que se possa realizar o processo de *tuning*, assim como diversas formas de utilizar a informação disponível no processo.

3.4.4 Filtro de Kalman - *Joseph's Stabilized Form*

Ao longo do processo de estimação é comum observar erros de arrendondamento durante as operações, especialmente quando ocorrem inversões de matrizes. Esses erros podem ter uma tendência a se intensificar de forma significativa levando a um desempenho insatisfatório do algoritmo ou até mesmo a não convergência (BROWN; HWANG, 2012) (GREWAL; ANDREWS, 2008).

Esses erros podem estar associados a diversos fatores como, por exemplo, uso de dados de baixa precisão, ou seja, com representação em poucos *bits*, e um grande número de estados, sendo que quanto maior o número de estados pior tende a ser o problema.

Para mitigar a influência desses erros é bastante comum o uso de diversos métodos, por exemplo, utilizar dados com maior precisão como números de dupla precisão, reduzir o número de estados através da eliminação de estados não observáveis ou que representem pouca contribuição no modelo, ou utilizando técnicas de redução de estados efetivamente (GREWAL; ANDREWS, 2008). Entre outras alternativas destacam-se, por exemplo, o uso de métodos numericamente mais estáveis e o uso de algoritmos de inversão de maior precisão, como a decomposição de Cholesky.

Com o intuito de reduzir os problemas numéricos associados as operações com as matrizes e possivelmente permitir que os resultados obtidos fossem mais estáveis, além de tentar reduzir os problemas de não convergência e até mesmo divergência encontrados durante o desenvolvimento, a forma estabilizada de Joseph (do inglês, *Joseph's Stabilized Form*) para formulação do algoritmo foi implementada, através da adaptação daquilo exposto por Brown e Hwang (2012) e Grewal e Andrews (2008).

A formulação proposta por P. D. Joseph é uma variação para a equação de atualização da matriz de covariância que, apesar de computacionalmente mais pesada, é menos sensível a erros de arrendondamento (BAR-SHALOM; LI; KIRUBARAJAN, 2001). Em relação a implementação, a única alteração essencial é substituir a etapa de atualização pela nova forma definida, em questões de implementação, isso se traduz essencialmente em utilizar as mesma funções definidas para o filtro de Kalman comum, com a diferença de que utilizam a nova formulação. A formulação é definida de acordo com as equações abaixo:

Etapa de atualização: Filtro de Kalman - *Joseph's Stabilized Form*

$$
\begin{aligned}
S[k] &= (H * Pp[k] * H^T + Rk)^{-1} \\
K[k] &= Pp[k] * HT * S[k] \\
Xu[k] &= (I - K * H)Xp[k] + K[k] * Y[k] \\
Pu[k] &= (I - K * H * Pp[k])Pp[k](I - K * H * Pp[k])^T + K[k] * R * K[k]^T
\end{aligned}
$$

3.4.5 Filtro de Kalman - *Smoothers*

Ao contrário do Kalman que se utiliza apenas das informações atuais para realizar a estimativa dos estados futuros, suavizadores ou *smoothers* são algoritmos que se utilizam de toda informação disponível até um determinado intervalo de tempo para determinar os estados do sistema, tanto informação passada quanto futura (GREWAL; ANDREWS, 2008).

Naturalmente, devido ao fato de utilizar informações obtidas no futuro para corrigir as estimativas passadas esse algoritmo não tem característica de tempo real.

Existem diversos tipos de suavizadores disponíveis, por exemplo, suavizadores de intervalo fixo, de ponto fixo, e com atraso fixo (BROWN; HWANG, 2012) (BAR-SHALOM; LI; KIRUBARAJAN, 2001). Sendo que cada um apresenta características que os tornam interessantes para diferentes tipos de aplicação, além de apresentarem complexidades bastante diferentes.

O algoritmo abordado aqui será um algoritmo para um suavizador de intervalo fixo desenvolvido por Rauch, Tung, e Striebel, e é conhecido como *RTS smoother*.

O *RTS smoother* utiliza uma janela de informações de de intervalo fixo com tamanho N + 1. Nesse intervalo o algoritmo opera essencialmente em duas etapas, primeiro, é efetuado uma passagem de estimação de forma progressiva utilizando o Kalman comum e, para cada nova estimativa realizada as informações a priori e a posteriori, ou seja, de predição e atualização, devem ser armazenadas.

Uma vez que todos os dados necessários forem coletados, considerando o tamanho do intervalo definido, uma nova formulação é utilizada para realizar uma estimação de forma regressiva, incorporando a informação futura nos estados passados, levando a uma suavização dos estados obtidos anteriormente e resultados possivelmente mais precisos.

Para realizar a etapa de estimação de forma progressiva o Kalman definido na subseção 2.11.2 é utilizado. Para realizar a suavização de forma regressiva uma nova formulação é necessária. A etapa regressiva do algoritmo definida em Grewal e Andrews (2008), Brown e Hwang (2012) e Bar-Shalom, Li e Kirubarajan (2001), foi adaptada para a notação utilizada na subseção 2.11.2 e é apresentada abaixo:

Etapa: Suavização - Etapa regressiva

$$
\begin{aligned}
C_k &= P_k^u . A^T . P_k^{p-1} \\
X_k^s &= X_k^u + C_k . (X_{k+1}^s - X_k^p) \\
P_k^s &= P_k^u + C_k . (P_{k+1}^s - P_k^p)
\end{aligned}
$$

Nas equações definidas, P^u e P^p ainda indicam as matrizes de covariância advindas das etapas de atualização e predição, assim como x^u e x^p indicam os estados para essas respectivas etapas.

O ganho C_k faz um papel bastante similar ao ganho de Kalman, porém, nesse caso o ele serve para definir a contribuição da diferença entre aquilo que foi observado na prática e aquilo que se obteve através da suavização para o estado subsequente.

Por fim, X^s e P^s indicam o estado e a matriz de covariância suavizados, respectivamente. Note que a matriz de covariância suavizada não representa uma informação tão relevante e, portanto, pode ser eliminada do processo de suavização caso suas informações não sejam realmente utilizadas, levando a um ganho no desempenho do algoritmo.

Com isso, o algoritmo é então definido de acordo com as etapas abaixo:

Etapas: *RTS smoother*

1. Receber novos dados e estimar os estados de forma progressiva utilizando o filtro de Kalman definido na subseção 2.11.2.

2. Armazenar as informações de predição e atualização obtidas para cada instante de tempo no intervalo N + 1.

3. Realizar a suavização dos estados no intervalo de forma regressiva utilizando as equações definidas nessa seção: *Etapa: Suavização - Etapa regressiva*

Considerando a implementação em *software*, foi necessário definir uma nova estrutura para que os dados de estimação do intervalo definido fossem armazenados e, para realizar os procedimentos, 2 novas funções foram definidas, uma para inicializar o a estrutura e outra para realizar a etapa regressiva do algoritmo. Para a etapa de estimação progressiva o Kalman implementado anteriormente foi utilizado.

De um ponto de vista de projeto, a ideia de utilizar o algoritmo definido, *RTS smoother*, seria criar uma pequena janela de dados a ser suavizada e fornecer uma estimativa com atraso. Geralmente esse não é o uso comum para o algoritmo, principalmente quando se considera o uso em sistemas de tempo real.

Porém, esse método foi considerado uma opção a ser utilizada no projeto pois, caso a taxa de amostragem fosse suficientemente alta, ou seja, os dados fossem muito próximos temporalmente, e a janela de dados fosse pequena o suficiente, seria observado um pequeno impacto na resposta do algoritmo, mas possivelmente os requisitos de tempo real poderiam ser respeitados na prática.

Considerando as questões computacionais, o algoritmo consome bem mais recursos pois necessita armazenar diversas informações de todos os estados dentro da janela e não somente aquelas associadas a computação do Kalman simples. Além disso, o procedimento de suavização é realizado para todos os estados anteriores pertencentes a janela de dados e, portanto, tem um custo computacional bem mais alto.

3.4.6 Aprimorando dados - Calibração dos sensores inerciais

Realizar a calibração de qualquer tipo de sensor é um procedimento essencial para promover um melhor desempenho do dispositivo, mesmo quando esse já apresenta dados com precisão relativamente elevada.

Para realizar a de calibração dos sensores existem diversos procedimentos possíveis, assim como diversos modelos para representar os dados advindos desses sensores. Cada um desses procedimentos e modelos podem representar mais fielmente as características observadas na prática, mas não necessariamente existe uma forma única que seja mais adequada para qualquer solução.

Para definir a forma como a calibração deve ser realizada, deve-se primeiro definir qual o modelo a ser utilizado para os dados, pois é através desse modelo que serão extraídas as possíveis formas de obtenção dos dados associados.

Nessa seção serão ilustrados então algumas formas de realizar o procedimento e também diferentes modelos para os dados.

Possivelmente a forma mais simples de definir imperfeições nos dados é através da atribuição de um erro associado que representa um desvio do valor verdadeiro, as Equações 3.53 - 3.55 e 3.56 - 3.58, apresentam os dados para o giroscópio e para o acelerômetro respectivamente.

Note que esse é um dos modelos mais simples pois utiliza apenas um *bias* para representar toda a contribuição dos erros estáticos e ignora completamente a presença dos erros de natureza dinâmica.

$$\omega_m^x = \omega_r^x + \omega_b^x \quad (3.53) \qquad \omega_m^y = \omega_r^y + \omega_b^y \quad (3.54) \qquad \omega_m^z = \omega_r^z + \omega_b^z \quad (3.55)$$

$$a_m^x = a_r^x + a_b^x \quad (3.56) \qquad a_m^y = a_r^y + a_b^y \quad (3.57) \qquad a_m^z = a_r^z + a_b^z \quad (3.58)$$

Para as equações definidas e as demais nessa seção, ω_m e a_m representam os valores para velocidade angular e aceleração medidos, ω_r e a_r representam os valores reais, e ω_b e a_b o *bias* para as mesmas medidas. Os sobrescritos x, y e z demonstram a relação do dado com cada eixo específico.

Idealmente em estado estacionário as medidas dos giroscópios deveriam fornecer valor nulo como saída, o mesmo vale para o acelerômetro, com exceção aquele eixo colocado contra a aceleração da gravidade. Porém, não é o caso, sempre serão observados valores não nulos para cada um dos eixos, por exemplo, devido a ruídos ou imperfeições na montagem do dispositivo, esse valores são aqueles considerados *biases*.

Para realizar o procedimento de calibração de acordo com o modelo apresentado nas Equações 3.53 - 3.58 a forma mais simples é deixar o dispositivo MPU6050 inerte e fazer a coleta dos dados de cada um dos eixos por um tempo determinado. Tomando a média desses dados é possível então atribuir valor a cada um dos *biases*, utilizando esses valores para anular a contribuição dos termos definidos, teoricamente seria possível observar as medidas reais.

Apesar do procedimento não conseguir eliminar completamente os problemas, ele é uma forma bastante simples para realizar a calibração de um dispositivo, contribuindo, portanto, para a obtenção de dados mais significativos e precisos.

Um modelo bastante similar ao apresentado é abordado nas Equações 3.59 - 3.61 e 3.62 - 3.64. Apesar de considerar mais termos, esse modelo apenas deixa explícito o fato de que o sinal possui ruído e, portanto, realizar a calibração de forma estática resultaria na eliminação do *bias* mas na manutenção do ruído observado.

$$\omega_m^x = \omega_r^x + \omega_b^x + \epsilon_\omega^x \quad (3.59) \qquad \omega_m^y = \omega_r^y + \omega_b^y + \epsilon_\omega^y \quad (3.60) \qquad \omega_m^z = \omega_r^z + \omega_b^z + \epsilon_\omega^z \quad (3.61)$$

$$a_m^x = a_r^x + a_b^x + \epsilon_a^x \quad (3.62) \qquad a_m^y = a_r^y + a_b^y + \epsilon_a^y \quad (3.63) \qquad a_m^z = a_r^z + a_b^z + \epsilon_a^z - g \quad (3.64)$$

Os dados ϵ_ω e ϵ_a representam a contribuição do ruído para o giroscópio e acelerômetro respectivamente. Note que um termo representado por g foi incluído para modelar a contribuição da aceleração da gravidade no eixo z, esse termo poderia ser incluído em qualquer eixo a depender da montagem.

Apesar de afetar as medidas, não é uma opção interessante eliminar a gravidade durante o procedimento de estimação da orientação. A aceleração da gravidade fornece uma maneira bastante efetiva de se definir um referencial para a orientação do dispositivo e consequentemente pode ser utilizado durante o processo de estimação.

Porém, para estimar a posição a contribuição da gravidade representa um problema. Manter um valor constante em um eixo específico fará com que o algoritmo entenda que um movimento efetivo está ocorrendo naquele eixo específico e, portanto, a estimação se torna impossível, principalmente considerando a elevada magnitude dessa interferência.

Por fim, considera-se o modelo apresentado nas Equações 3.65 - 3.67 e 3.68 - 3.70. Nesse modelo é incorporado um termo chamado de fator de escala definido como sf_ω para o giroscópio e sf_{a^x} para o acelerômetro. Esse modelo assim como o processo de calibração aqui definido foram adaptados daqueles expostos por Bai et al. (2020).

$$\omega_m^x = sf_{\omega^x}.\omega_r^x + \omega_b^x + \epsilon_\omega^x \quad (3.65) \qquad \omega_m^y = sf_{\omega^y}.\omega_r^y + \omega_b^y + \epsilon_\omega^y \quad (3.66) \qquad \omega_m^z = sf_{\omega^z}.\omega_r^z + \omega_b^z + \epsilon_\omega^z \quad (3.67)$$

$$a_m^x = sf_{a^x}.a_r^x + a_b^x + \epsilon_a^x \quad (3.68) \qquad a_m^y = sf_{a^y}.a_r^y + a_b^y + \epsilon_a^y \quad (3.69) \qquad a_m^z = sf_{a^z}.a_r^z + a_b^z + \epsilon_a^z - g \quad (3.70)$$

Apesar de adicionar um único termo, o processo de calibração desse modelo é mais complexo pois o fator de escala essencialmente denota parte das características da dinâmica do dispositivo, dessa forma, é necessário obter as contribuições especificadas tanto de forma estática como de forma dinâmica.

Para o acelerômetro, como a gravidade age a todo momento no dispositivo, tanto a resposta dinâmica quanto estática pode ser obtida diretamente de um processo de calibração estática devido ao valor já conhecido da aceleração da gravidade. Para o giroscópio por outro lado, é necessário um valor conhecido de velocidade angular para que se estabeleça a contribuição das questões dinâmicas do sistema, dessa forma, a calibração deve ser realizada tanto de forma dinâmica quanto estática.

Como cada eixo responde de uma maneira distinta a um mesmo estímulo o processo de calibração deve ser feito separadamente para cada eixo. Além disso, um mesmo eixo

responde de maneira diferente a depender da direção do estimulo, deve-se portanto verificar a resposta tanto para o maior valor possível quanto para o menor em um mesmo eixo. Com as características apresentadas o processo necessita de um total de 6 pontos de teste, por esse motivo esse processo também é chamado de calibração de 6 pontos.

Para o acelerômetro, desconsiderando a contribuição do ruído e sabendo que o estímulo é a aceleração da gravidade, o fator de escala e *bias* para cada um dos eixos são definidos de acordo com as Equações 3.71 - 3.73 e 3.74 - 3.73, respectivamente. Nas equações definidas, a_{max} representa a aceleração máxima medida em um determinado eixo, e a_{min} mínimo.

$$sf_{a^x} = \frac{2}{a^x_{max} - a^x_{min}} g \quad (3.71) \qquad sf_{a^y} = \frac{2}{a^y_{max} - a^y_{min}} g \quad (3.72) \qquad sf_{a^z} = \frac{2}{a^z_{max} - a^z_{min}} g \quad (3.73)$$

$$a^x_b = -\frac{a^x_{max} + a^x_{min}}{a^x_{max} - a^x_{min}} g \quad (3.74) \qquad a^y_b = -\frac{a^y_{max} + a^y_{min}}{a^y_{max} - a^y_{min}} g \quad (3.75) \qquad a^z_b = -\frac{a^z_{max} + a^z_{min}}{a^z_{max} - a^z_{min}} g \quad (3.76)$$

Para o giroscópio se define-se a Equações 3.77 - 3.79 em que ω_{max} representa o valor máximo de velocidade angular medido em um determinado eixo, e ω_{min} o valor mínimo.

$$\omega^x_b = \frac{\omega^x_{max} + \omega^x_{min}}{2} \quad (3.77) \qquad \omega^y_b = \frac{\omega^y_{max} + \omega^y_{min}}{2} \quad (3.78) \qquad \omega^z_b = \frac{\omega^z_{max} + \omega^z_{min}}{2} \quad (3.79)$$

Por fim, para obter os fatores de escala do giroscópio é necessário utilizar algum dispositivo capaz de estimular o dispositivo com uma velocidade angular conhecida, comparando a velocidade medida com a ideal se define o fator de escala conforme as Equações 3.80 - 3.82, em que ω_{ideal} representa o valor ideal utilizado no processo.

$$sf_{\omega^x} = \frac{\omega^x - \omega^x_b}{\omega^x_{ideal}} \quad (3.80) \qquad sf_{\omega^y} = \frac{\omega^y - \omega^y_b}{\omega^y_{ideal}} \quad (3.81) \qquad sf_{\omega^z} = \frac{\omega^z - \omega^z_b}{\omega^z_{ideal}} \quad (3.82)$$

Com os modelos definidos é possível então realizar o procedimento de calibração dos sensores inerciais através de testes para definir um modelo mais adequado ao projeto.

3.4.7 Aprimorando dados - Reduzindo ruídos

Quando se considera o desempenho dos algoritmos propostos, uma forma bastante direta de obter resultados mais próximos aqueles esperados ou que se tem como reais é através da redução dos ruídos no processo. O ruído pode ser derivado de diversos fatores diferentes como, por exemplo, da qualidade dos sensores utilizados, da forma de montagem na plataforma, da natureza do processo de medição, entre outros.

No caso da orientação, os sensores que representam a maior contribuição na estima-

tiva são os giroscópios. Giroscópios, mesmo de baixa de qualidade, são pouco sensíveis a diversas formas de estimulo e, portanto, tendem a apresentar uma resposta relativamente precisa, porém, ainda tendem a ser afetados por questões de montagem e temperatura.

No caso da posição, o ruído já é uma questão bem mais crítica. O acelerômetro, sensor responsável por fornecer as informações necessárias para essa estimação, é bastante sensível a qualquer tipo de estimulo e, portanto, apresenta dados com bastante ruído associado, além disso, ele também é afetado pelos fatores mais comuns. Dessa forma, é necessário reduzir todas as fontes de ruídos que sejam possíveis e incorporar o maior número de informações de qualidade possível no processo.

O caso da estimação da posição é tão crítico que motivou o desenvolvimento e procedimentos abordados nessa seção. Durante o processo de desenvolvimento o ruído observado nos acelerômetros se provou um fator limitante no processo de estimação, pois mesmo com magnitude relativamente baixa levou o algoritmo a instabilidade, fazendo com que ocorresse divergência tanto lenta quanto rapidamente a depender dos parâmetros utilizados. Em muitos casos a entrada sequer era percebida devida a natureza já divergente da solução, naturalmente o impacto desse novo dado é mínimo para corrigir esse tipo de problema.

Para corrigir os problemas apresentados diversas medidas foram tomadas. Inicialmente, para reduzir os ruídos do processo optou-se por utilizar soluções que fossem simples de serem incorporadas no projeto e pudessem levar a um ganho de desempenho de forma bastante direta.

3.4.7.1 Solução - Configurações

Considerando as opções disponíveis, alterar as configurações do dispositivo responsável pela captura de dados é possivelmente a solução mais simples de ser incorporada. Dentre as diversas configurações disponíveis no dispositivo, aquelas que podem promover o resultado desejado são essencialmente três: sensibilidade dos sensores, filtro passa-baixas digital, e taxa de amostragem de dados.

Conforme abordado na subseção 3.3.5.2 a questão da sensibilidade é uma característica que pode contribuir para aprimorar o desempenho dos algoritmos de estimação tendo em vista que uma menor sensibilidade estabelece um limiar maior a ser superado para que movimentos sejam detectados e, consequentemente, sinais com menor oscilação sejam observados. Portanto, reduzir a sensibilidade é uma forma bastante simples de reduzir o nível de ruído nas medidas.

Também abordado na subseção 3.3.5.2, o filtro passa-baixas digital incorporado no dispositivo MPU6050 é outra alternativa para promover a redução do ruído no processo. Idealmente um filtro deveria eliminar completamente a contribuição de sinais acima de um

certo limiar, porém, não é o caso para filtros reais. Apesar de não eliminar completamente, utilizar limiares mais restritos permite reduzir o impacto de estímulos que muito possivelmente são advindos de perturbações no sistema, levando a uma redução efetiva no ruído observado.

Por fim, considerando a taxa de amostragem dos sensores, a priori seria possível assumir que quanto maior a taxa de amostragem melhor o desempenho dos algoritmos de estimação tendo em vista que mais informações estariam disponíveis num menor tempo e, portanto, seriam um reflexo mais próximo daquilo que se observa na realidade.

Porém, tal suposição não necessariamente é observada na prática. Os dados advindos dos sensores não são perfeitos e, mesmo que uma maior taxa se traduza em mais informação, informação ruidosa e constante pode levar até mesmo a resultados piores do que aqueles observados a menores taxas.

Em essência o processo de discretização apresenta intrinsecamente característica passa-baixas, sendo que quanto menor a taxa de amostragem do sinal original, menor a quantidade de informação recuperada e menor a contribuição de variações pontuais, o contrário também é verdade.

Reduzir a taxa de amostragem garante então que um menor nível de ruído seja incorporado nas medidas. Naturalmente deve ocorrer um *trade-off* de forma que a taxa configurada permita que a informação chegue de forma relativamente rápida para que as correções nas estimativas possam ser realizadas mais frequentemente, mas também se reduza o nível de ruído no processo.

3.4.7.2 Solução - Montagem

Levando em consideração as conexões dos sensores com a plataforma, é muito comum o surgimento de ruídos advindos de diversas fontes, seja de interferências externas, seja da própria construção dos conectores.

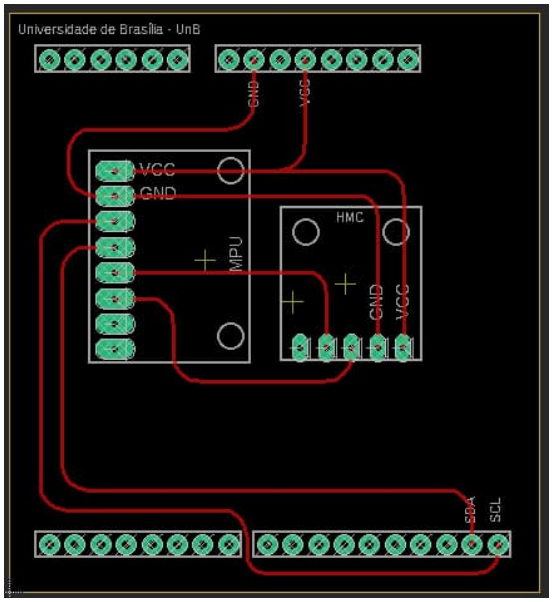
Ainda em consideração a montagem, é importante notar que o alinhamento dos sensores também é uma questão bastante importante nesse caso. Se um bom alinhamento dos sensores não for observado, mesmo que uma calibração seja realizada esse erro pode não ser eliminado devido ao fato de que cada um dos eixos dos sensores apresenta uma resposta diferente a um mesmo estímulo.

A forma escolhida para reduzir os ruídos relacionados a montagem foi o desenvolvimento de uma placa de circuito impresso (do inglês, *Printed Circuit Board* - PCB). Essa placa permite reduzir os ruídos mais característicos apresentados através, por exemplo, da redução do tamanho dos condutores, do uso de materiais com qualidade relativamente superior, e de uma construção mais robusta, além de garantir um melhor alinhamento dos sensores.

A placa mostrada na Figura 101 suporta até 1(um) dispositivo MPU6050 e 1(um)

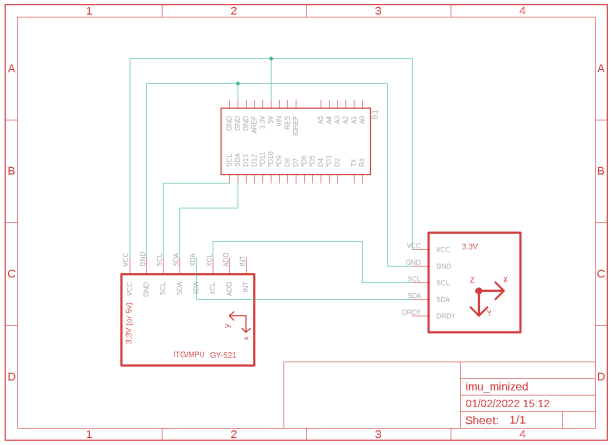
sensor externo. A priori a placa foi projetada para suportar como sensor externo o magnetômetro QMC5883L, um magnetômetro relativamente preciso e de baixo custo, porém, diversos sensores mais simples possuem conexão bastante similar e também podem ser incorporados utilizando tal conexão. O respectivo esquemático elétrico da PCB é ilustrado na Figura 102.

Figura 101 – *Board* da Placa de circuito impresso para os sensores inerciais



Fonte: Produzido pelos autores.

Figura 102 – Esquemático elétrico da Placa de circuito impresso para os sensores inerciais



Fonte: Produzido pelos autores.

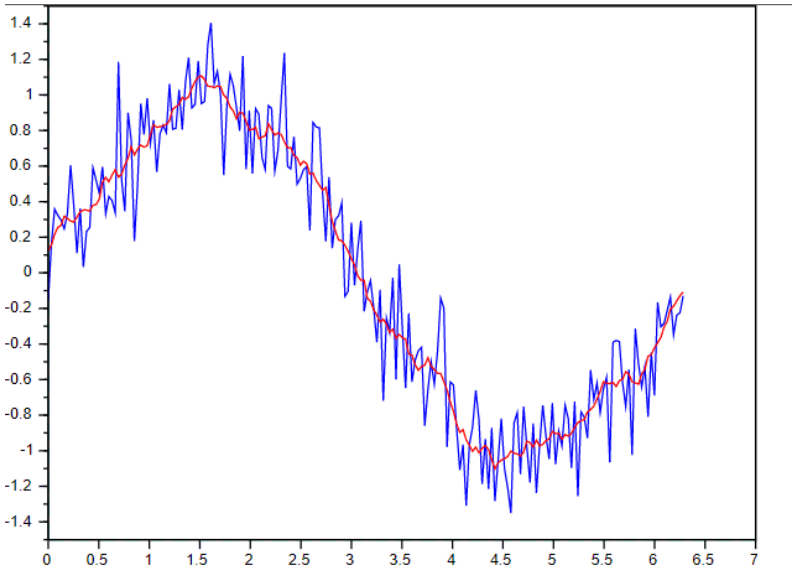
3.4.8 Aprimorando dados - Outros procedimentos

Ao contrário das soluções já apresentadas anteriormente, essa seção é composta por algoritmos que não atuam diretamente na fonte para reduzir os erros ou ruídos no processo, mas sim promover a mitigação da influência desses problemas no resultado final.

3.4.8.1 Solução - Média móvel

A Figura 103 apresenta um exemplo de aplicação da média móvel, a média móvel consiste essencialmente em definir uma janela de dados para que se aplique a média e, uma vez que o número de dados já seja suficiente para cobrir toda a janela de dados, a medida que novos dados são incorporados na média, os mais antigos são eliminados.

Figura 103 – Exemplo de média móvel



Fonte: Chan (2014)

A proposta de utilizar a média móvel no projeto tem como objetivo reduzir a influência das variações observadas no curto prazo e focar nas tendências de longo prazo dos dados, levando a uma curva de dados mais suave e resultados possivelmente mais precisos.

Apesar de ser uma solução simples e interessante de ser incorporada no sistema, a média móvel também gera novos problemas. Caso os dados já apresentem estabilidade no curto prazo, como é o caso do giroscópio, utilizar essa ferramenta pode levar a uma convergência mais lenta e a resultados imprecisos nos algoritmos propostos.

3.4.8.2 Solução - Limiar para detecção de movimento

Conforme já abordado anteriormente, os sensores apresentam problemas de desvio que tendem a crescer mesmo quando os sensores se mantém inertes. Isso ocorre devido ao fato do processo de integração adicionar continuamente a contribuição do ruído ao processo.

Esse problema é bastante notável no caso da estimação da posição em que o algoritmo pode tender a divergir indefinidamente devidos aos erros introduzidos no início do processo. Para reduzir tal problema é possível considerar o algoritmo definido em Lasmadi et al. (2017).

O algoritmo chamado de *Zero Velocity Compensation* (ZVC) reduz o problema apresentado através do uso de essencialmente três passos. Primeiro é tomada uma média móvel

a partir de uma janela bem definida de dados, utilizando a contribuição da operação em condição estática do dispositivo se corrige o valor observado, por fim, se compara o valor corrigido com um limiar previamente estabelecido, caso o valor seja inferior considera-se que a velocidade observada é nula.

Zero Velocity Compensation

1. Receber entrada atual e incorporar na média.
2. Corrigir entrada para adicionar contribuição dos valores de estado estacionário.
3. Se o valor corrigido for menor que o limiar, considerar que a valor é nulo.
4. Caso contrário, manter valor obtido para realizar a estimação.

3.4.8.3 Solução - Detecção de estado estacionário

Outra forma de reduzir a influência de ruídos é considerar um limiar mínimo para que um dado obtido seja considerado como parte um movimento efetivamente observado na realidade.

Uma forma de realizar esse procedimento é utilizando uma média móvel e o valor de desvio padrão associado as amostras. Comparando o desvio padrão das amostras com um valor de limiar previamente definido é possível definir se o sensor capturou movimento ou não. Esse algoritmo é chamado de algoritmo de detecção de estado estacionário e é definido em Lasmadi et al. (2017).

De um ponto de vista prático esse algoritmo permite que as estimativas geradas se mantenham relativamente constantes uma vez que o dispositivo não esteja em movimento e não tenha tendência a oscilar naquele estado a medida que o tempo passa devido a interferência dos ruídos.

Detecção de Estado Estacionário

1. Receber entrada atual já incorporada na média.
2. Comparar o desvio padrão σ observado com o limiar
3. Se σ for menor que o limiar, considerar que não houve alteração na posição e velocidade é nula.
4. Caso contrário, manter os resultados da estimação.

Esses algoritmos foram usados como inspiração para desenvolver um processo similar nesse projeto, garantindo que algumas dessas características fossem incorporadas ao processo de estimação.

3.5 Estudo sobre a fusão de dados

A odometria visual e inercial consiste, dentre outras etapas, na fusão de dados da odometria visual e da odometria inercial. Como o algoritmo de odometria visual foi implementado em uma CPU, devido às limitações expostas na seção 3.1, e a odometria inercial integrada na plataforma híbrida Minized, a fusão de dados não foi realizada neste projeto, já que ambos os sistemas devem estar integrados na mesma plataforma.

No sistema de odometria visual implementado, a ausência de escala é um dos fatores que mais causa imprecisões nos resultados. Conforme expresso na subseção 3.2.10, na configuração monocular não é possível obter uma escala utilizando apenas dados provenientes da câmera, é necessário uma fonte externa, como uma IMU. No sistema de odometria inercial implementado, é possível utilizar os dados de posição gerados pela IMU utilizada para obter a escala. Dessa forma, é possível mesclar dados da IMU e da câmera para obter um resultado final mais preciso, isto é, usar a escala obtida pela odometria inercial na trajetória reconstruída pelo sistema de odometria visual.

3.5.1 Obtenção da escala através de dados da odometria inercial

Uma vez implementado o algoritmo de odometria inercial, é possível extrair os dados da posição. Seja X_{n-1}, Y_{n-1} e Z_{n-1} as posições geradas pela IMU nos eixos X, Y e Z respectivamente, no instante *n-1*, e seja X_n, Y_n e Z_n as posições no instante seguinte, n, a escala, $S_{n,n-1}$, entre esses instantes consecutivos é dada por:

$$S_{n,n-1} = \sqrt{(X_n - X_{n-1})^2 + (Y_n - Y_{n-1})^2 + (Z_n - Z_{n-1})^2} \tag{3.83}$$

3.5.2 Reconstrução da trajetória com escala

Conforme ressaltado na seção 2.8 e repetido aqui por conveniência, a posição e orientação da câmera em um dado instante n é dada por Siciliano e Khatib (2007):

$$C_n = R_{n,n-1}C_{n-1} + T_{n,n-1} \tag{3.84}$$

em que $R_{n,n-1}$ é a matriz de rotação e $T_{n,n-1}$ é a matriz de translação. Com o valor da escala, S, em mãos, é possível obter resultados mais precisos da trajetória reconstruída, através da formulação:

$$C_n = R_{n,n-1}C_{n-1} + S_{n,n-1} \times T_{n,n-1} \tag{3.85}$$

4 Resultados

4.1 Resultados - Odometria Visual

A realização de testes para o algoritmo de odometria visual foi feita por meio da filmagem de um percurso conhecido. Conforme expresso na subseção 3.2.4, a câmera do Iphone 7 *Plus* da *Apple* foi utilizada para a filmagem e o trajeto foi percorrido de bicicleta com o celular na mão para ambientes controlados, e de motocicleta para ambientes não controlados.

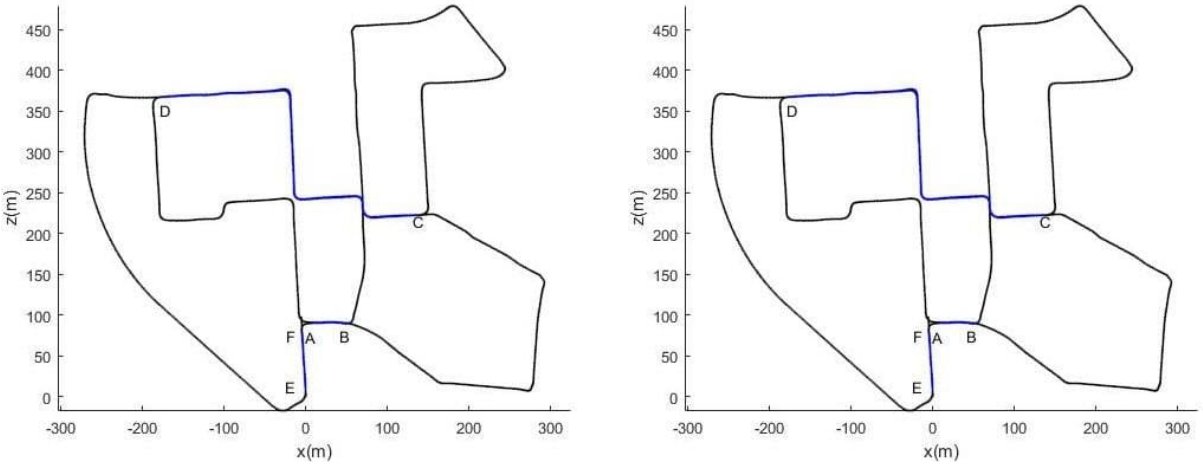
4.1.1 Conversão do vídeo em sequência de imagens

A câmera utilizada filmava à taxa de 30 FPS. Na etapa de conversão do vídeo em uma sequência de imagens, é de suma importância escolher um FPS adequado, isto é, a quantidade de *frames* convertidos para cada segundo de vídeo. Por exemplo, o KITTI *dataset* (GEIGER; LENZ; URTASUN, 2012) utilizou a taxa de 10 FPS na conversão. Primeiramente, o comportamento do algoritmo foi analisado com o valor de 30 FPS na conversão, porém, os resultados se mostraram distorcidos e aleatórios, pois como há um intervalo mínimo entre *frames*, a contribuição de movimentos indesejados se torna maior que a do próprio movimento da trajetória. Já com 10 FPS, o algoritmou apresentou resultados satisfatório, pois a posição das *features* representaram mais fielmente o movimento verdadeiro. Portanto, foi o valor escolhido.

4.1.2 Seleção de testes e *ground truth*

A seleção dos testes a serem realizados foi feita de forma a englobar a maior possibilidade de casos possíveis. Ao se analisar testes realizados em *datasets*, como o KITTI *dataset* (GEIGER; LENZ; URTASUN, 2012), percebe-se que as trajetórias percorridas (conhecidas como *ground truth*), que são realizadas em ambientes *outdoors*, contemplam, basicamente, curvas de 0 a 90º, semi-círculos e retas, conforme mostrado nas Figuras 104a e 104b.

Figura 104 – Sequências do KITTI *dataset*



(a) Sequência 00 do KITTI *dataset*

(b) Sequência 06 do KITTI *dataset*

Fonte: Ouerghi et al. (2018)

Além disso, a trajetória percorrida pela câmera (doravante chamada de agente) no KITTI *dataset* foi realizada em vias urbanas, onde há a presença de fluxo de pedestres e carros. Portanto, trata-se de um cenário dinâmico, que influencia no comportamento do algoritmo.

Dessa forma, para este projeto foram elaborados testes em ambiente controlado e não controlado. Nos testes em ambiente controlado, apenas o agente se move, enquanto os demais objetos do ambiente são estáticos ou possuem movimento desprezível. Estes testes foram realizados para analisar a precisão do algoritmo em curtas distâncias, sua repetibilidade e erros estatísticos associados e seu comportamento diante de falta de textura e fatores climáticos. Cada trajetória planejada foi percorrida 10 vezes. Já nos testes em ambiente não controlado, há a presença de objetos que se movem no ambiente. Estes testes foram realizados para analisar a precisão do algoritmo em longas distâncias, seu comportamento diante a influência de objetos móveis, terrenos irregulares e curvas acentuadas. Cada trajetória planejada foi percorrida uma única vez.

4.1.3 Testes em ambiente controlado

As trajetórias em ambiente controlado foram percorridas em um canteiro de obras abandonado de dimensão quadrada, cujo lado é de aproximadamente 25m, ilustrado na Figura 105. Nota-se que o pavimento de concreto do local é bastante degradado, com marcas de lodo, rachaduras e reparos, o que se torna um ambiente ideal, pois há textura em abundância para identificação de cantos no algoritmo.

Figura 105 – Espaço de 25 × 25 m utilizado nas trajetórias estáticas

Fonte: Produzido pelos autores.

Com o uso de uma trena manual, o trajeto a ser percorrido pelo agente na bicicleta foi demarcado. As marcações foram pintadas no pavimento, conforme ilustrado nas Figuras 106a, 106b, 106c e 106d.

Figura 106 – Marcações indicadores no pavimento

(a) Marcação indicadora de ângulos de 90º

(b) Marcação indicadora de ângulos de 90º

(c) Marcação indicadora de ângulos de 45º

(d) Marcação indicadora de retas

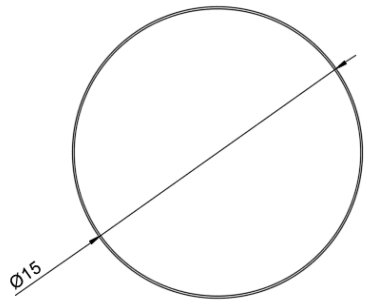
Fonte: Produzido pelos autores.

Quatro trajetórias foram testadas para ambiente controlado, compostas de retas, curvas em ângulos de 45 e 90º, círculos e semi-círculos, o que permite analisar a precisão de rotação e translação do algoritmo. Além disso, como há trajetórias que começam e terminam no mesmo ponto, pode-se verificar a capacidade de localização espacial do algoritmo. A trajetória verdadeira é conhecida na literatura como *ground truth* e doravante será chamada assim. Conforme ressaltado na subseção 3.2.4, o trajeto em ambiente não controlado foi percorrido em uma bicicleta, com velocidade média de 10 km/h.

4.1.3.1 Trajetória 1: Círculo

O *ground truth* da trajetória circular é ilustrado na Figura 107 e foi elaborado para avaliar o comportamento do algoritmo em trajetórias circulares.

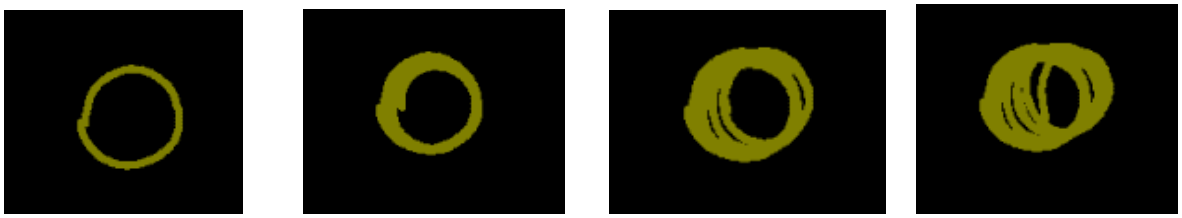
Figura 107 – Trajetória 1: círculo (Dimensões em metros)



Fonte: Produzido pelos autores.

O *ground truth* foi percorrido 10 vezes de forma contínua. O resultado acumulado é apresentado nas Figuras 108a, 108b, 108c e 108d. Pode-se perceber que, a cada volta, há um pequeno desvio entre o ponto de partida e o de chegada, que deveriam ser os mesmos. Dessa forma, o erro se propaga e se acumula ao longo do resultado.

Figura 108 – Resultado acumulado para trajetória 1



(a) Primeira volta do resultado acumulado para trajetória 1

(b) Terceira volta do resultado acumulado para trajetória 1

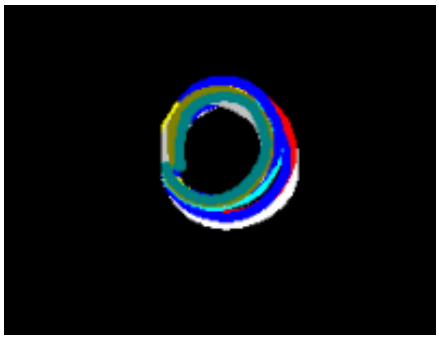
(c) Sexta volta do resultado acumulado para trajetória 1

(d) Décima volta do resultado acumulado para trajetória 1

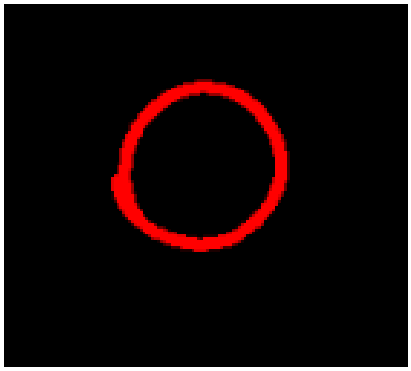
Fonte: Produzido pelos autores.

Ao se fixar o ponto de partida de cada volta, o resultado é conforme ilustrado na Figura 109a e a volta mais precisa é mostrada na Figura 109b. Nota-se que o resultado apresenta uma boa precisão para curvaturas, embora o resultado acumulado apresente desvios.

Figura 109 – Resultados em ambiente controlado: trajetória 1



(a) Resultado acumulado para a trajetória 1 com ponto de partida fixo



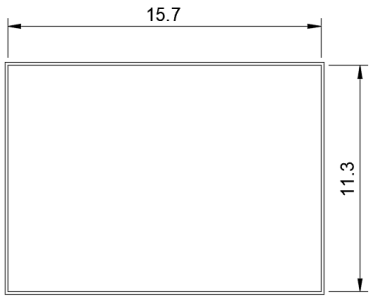
(b) Resultado mais preciso da trajetória 1

Fonte: Produzido pelos autores.

4.1.3.2 Trajetória 2: Retângulo

O *ground truth* da trajetória retangular é ilustrado na Figura 110 e foi elaborado para avaliar o comportamento do algoritmo em retas e ângulos de 90º. É válido salientar que como a trajetória foi realizado em uma bicicleta, ângulos retos não são possíveis de serem realizados, portanto, todos os cantos do retângulo são arredondados.

Figura 110 – Trajetória 2: retângulo (Dimensões em metros)



Fonte: Produzido pelos autores.

Assim como na trajetória circular, *ground truth* foi percorrido 10 vezes. O resultado acumulado é apresentado na Figura 111a e a volta mais precisa é apresentada na Figura 111b. Nota-se que embora o ponto de partida e de chegada verdadeiros sejam os mesmos, há uma leve desvio no resultado. Além disso, alguns ângulos de 90º e retas não apresentaram boa precisão, o que se atribui a falta de instrumentos adequados para demarcação do *ground truth*. Já o resultado acumulado demonstra uma boa repetibilidade dos resultados.

Figura 111 – Resultados em ambiente controlado: trajetória 2

(a) Resultado acumulado para a trajetória 2 (b) Resultado mais preciso da trajetória 2

Fonte: Produzido pelos autores.

Ao longo dos testes, procurou-se analisar também o comportamento do algoritmo diante de curvas realizadas de forma brusca e realizadas em cenas com baixa textura. A Figura 112a ilustra o resultado de um teste realizado em que em "1" a curva foi realizada de forma brusca. Nota-se que a curva resultante apresentou um ângulo muito maior que 90º, pois a suposição do rastreador Lucas-Kanade de que o movimento da cena entre *frames* consecutivos é pequeno não foi cumprida. Assim, como há uma grande quantidade de movimento, os *pixels* se deslocam mais frequentemente para fora da vizinhança local, consequentemente, o algoritmo não é capaz de detectá-los.

Além disso, foram realizadas curvas perto de paredes com baixa textura, como a parede ilustrada na Figura 112b. A respectiva curva é indicada em "2" na Figura 112a. Nota-se que a precisão da rotação foi prejudicada, pois com baixa textura, a quantidade de *features* detectáveis diminui ao ponto de se tornar insuficiente para obter uma boa precisão. Ao se realizar curvas mais perto ainda da parede, o algoritmo não funcionava, pois a quantidade mínima de *features* estabelecida na subseção 3.2.7.1 não era atingida, nem mesmo na redetecção e portanto a matriz essencial não era formada.

Figura 112 – Trajetória 2 em cenário com baixa textura

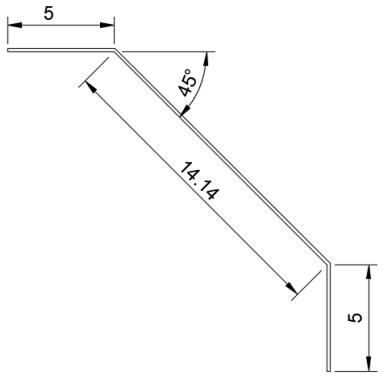
(a) Resultado da trajetória 2 com imprecisões (b) Resultado mais preciso da trajetória 2

Fonte: Produzido pelos autores.

4.1.3.3 Trajetória 3: Ângulos de 45º

O *ground truth* da trajetória 3 é ilustrado na Figura 113 e foi elaborado para avaliar o comportamento do algoritmo em retas e ângulos de 45º.

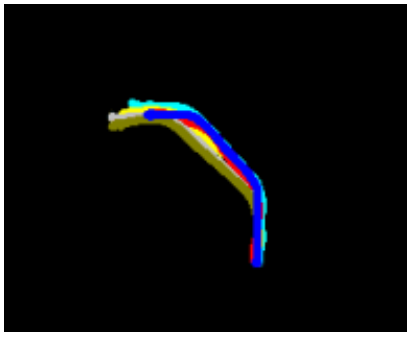
Figura 113 – Trajetória 3: ângulos de 45º (Dimensões em metros)



Fonte: Produzido pelos autores.

O *ground truth* foi percorrido 10 vezes. Diferentemente das trajetórias circular e retangular, o ponto de partida e de chegada não coincidem, logo, foram necessárias 10 filmagens diferentes. O resultado acumulado é apresentado na Figura 114a e o percurso mais preciso é apresentado na Figura 114b. Percebe-se que os ângulos de 45º foram precisos, assim como a repetibilidade dos resultados.

Figura 114 – Resultados em ambiente controlado: trajetória 3



(a) Resultado acumulado para a trajetória 3

(b) Resultado mais preciso da trajetória 3

Fonte: Produzido pelos autores.

É importante ressaltar as condições climáticas durante os testes em ambiente controlado realizados. Ao se realizar testes durante horários do dia em que luminosidade local é intensa devido à iluminação do sol, notou-se que a câmera suavizava automaticamente a cor após certo limiar de saturação, como pode ser observado nas Figuras 115a e 115b. Veja que na Figura 115a, o pavimento apresenta um cor saturada e no *frame* consecutivo, ilustrado pela Figura 115b, o pavimento já apresenta uma cor suave.

Figura 115 – trajetória com correção automática de cor

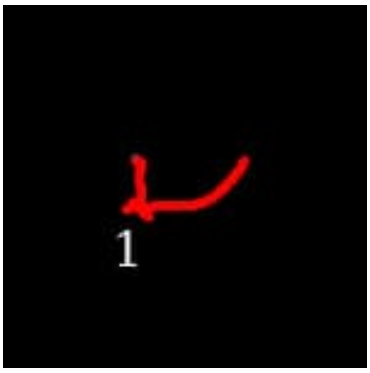
(a) *frame* inicial

(b) *frame* consecutivo com correção automática de cor

Fonte: Produzido pelos autores.

Como o rastreador de *features* Lucas-Kanade compara o vetor gradiente na vizinhança do *pixel* para estimar o *motion field*, a correção de cor altera todos os vetores gradiente, distorcendo o resultado da trajetória estimada, conforme ilustrado na Figura 116. Veja que a Figura 116 deveria ilustrar o resultado da trajetória 3, mas a posição e orientação é completamente alterada quando ocorre a correção de cor, indicado em "1" na Figura 116. Como esta funcionalidade automática da câmera não pode ser removida, os testes foram realizados em um intervalo anterior ao pôr-do-sol, em que a luminosidade local não é intensa.

Figura 116 – Resultado da trajetória 3 distorcida devido correção automática de cor

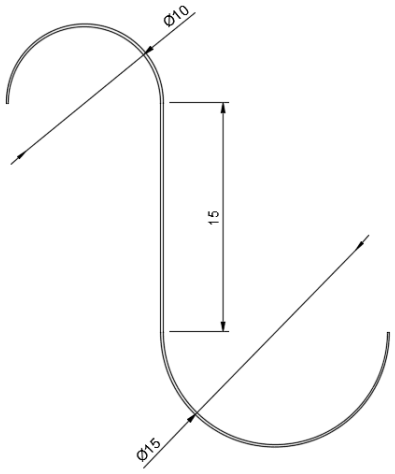


Fonte: Produzido pelos autores.

4.1.3.4 Trajetória 4: Semi-círculos

O *ground truth* da trajetória 4 é ilustrado na Figura 117 e foi elaborado para avaliar o comportamento do algoritmo em semi-círculos de raios diferentes e retas.

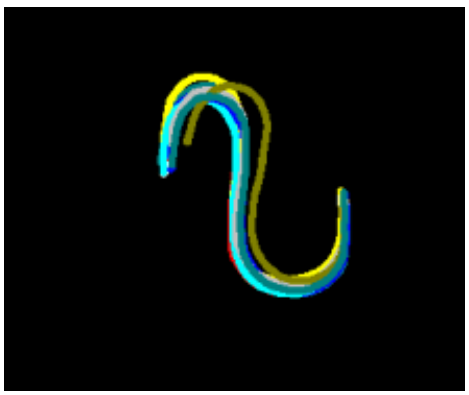
Figura 117 – Trajetória 4: semi-círculos (Dimensões em metros)



Fonte: Produzido pelos autores.

Assim como na trajetória 3, o *ground truth* foi percorrido 10 vezes e o ponto de partida e de chegada não coincidem, necessitando 10 filmagens diferentes. O resultado acumulado é apresentado na Figura 118a e o percurso mais preciso é apresentado na Figura 118b. Nota-se que o algoritmo apresentou uma boa precisão e repetibilidade dos resultados, apenas um única repetição, ilustrada pela cor verde-oliva na Figura 118a, apresentou um leve desvio.

Figura 118 – Ambiente controlado: trajetória 4



(a) Resultado acumulado para a trajetória 4

(b) Resultado mais preciso da trajetória 4

Fonte: Produzido pelos autores.

4.1.4 Testes em ambiente não controlado

As trajetórias em ambiente não controlado foram percorridas nas vias urbanas da cidade de Piripiri, no estado do Piauí, assim como em rodovias estadual e federal próximas. No percurso, pode-se encontrar vias niveladas e com acidentes geográficos, retas, curvas suaves e acentuadas, pavimentação asfáltica e calçamento de pedra, e redutores de velocidade do tipo tachão e sonorizador. Além disso, as trajetórias foram percorridas em horário de fluxo intenso

de pedestres e veículos. Esses elementos são essenciais para analisar o comportamento do algoritmo diante a influência de terrenos irregulares e objetos móveis no cenário.

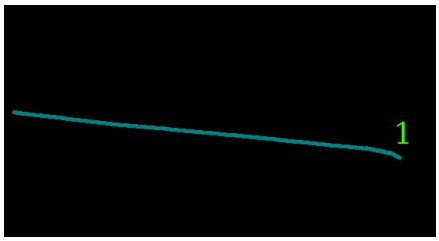
As trajetórias foram elaboradas com auxílio do *software Google Earth*, de forma que cada uma contenha pelo menos um destes elementos: reta de longa distância, curvas, retornos e terrenos irregulares. Ao todo, cinco trajetórias foram testadas. Além disso, foi dada a devida atenção de garantir que houvesse fluxo de veículos em cada uma das trajetórias, que foram percorridas uma única vez. Conforme ressaltado na subseção 3.2.4, o trajeto em ambiente não controlado foi percorrido em uma motocicleta, a uma velocidade média de 60 km/h.

4.1.4.1 Trajetória 1: Reta

O *ground truth* da trajetória 1 é ilustrado na Figura 119a e foi elaborado para avaliar o comportamento do algoritmo em trajetórias retilíneas. Trata-se de uma reta longa de aproximadamente 1920 m, porém com leves curvaturas. O resultado do algoritmo é ilustrado na Figura 119b.

Figura 119 – Ambiente não controlado: trajetória 1

(a) *Ground truth* da trajetória 1 (escala em metros). *Google Earth, earth.google.com/web/*



(b) Resultado da trajetória 1

Fonte: Produzido pelos autores.

O algoritmo obteve resultado satisfatório, no entanto, apresenta um desvio em uma das extremidades, indicada por “1” na Figura 119b. Esse desvio se deve à mudança de orientação de um veículo ao realizar uma curva, conforme mostrado nas Figuras 120a e 120b. Naturalmente, ao ultrapassar o agente, o veículo logo é identificado pelo algoritmo, e a medida que o veículo avança, sua orientação coincide com a orientação do agente. Porém, quando o veículo realiza uma curva à esquerda, sua orientação difere da orientação do agente, então o algoritmo, erroneamente, compreende que o agente está rotacionando à direita, pois o rastreador Lucas-Kanade entende que um conjunto de *pixels* se deslocando à esquerda em uma sequência de *frames* corresponde a um movimento de rotação à direita.

Figura 120 – Veículo alterando sua orientação durante percurso

(a) Veículo ultrapassando o agente

(b) Veículo realizando uma curva à esquerda

Fonte: Produzido pelos autores.

Ao longo da trajetória, houve a presença de diversos objetos móveis, como pedestre atravessando a via, ilustrado na Figura 121a, assim como motos e bicicletas sendo ultrapassadas, destacadas na Figura 121b. Diferentemente do caso anterior, a presença destes objetos móveis em cena não provocou anomalias no comportamento do algoritmo, já que ambos os objetos não variaram de forma significativa sua orientação de movimento, que é praticamente constante no intervalo em que eles aparecem no percurso.

Figura 121 – Objetos móveis no percurso

(a) Pedestre atravessando a via

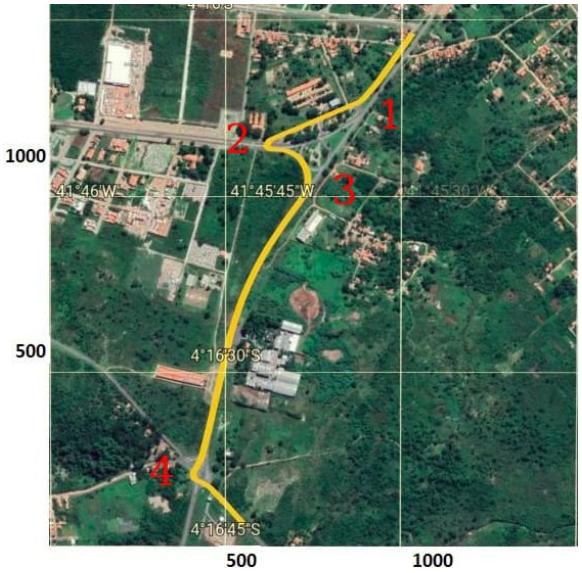
(b) Ciclista pedalando no mesmo sentido do agente

Fonte: Produzido pelos autores.

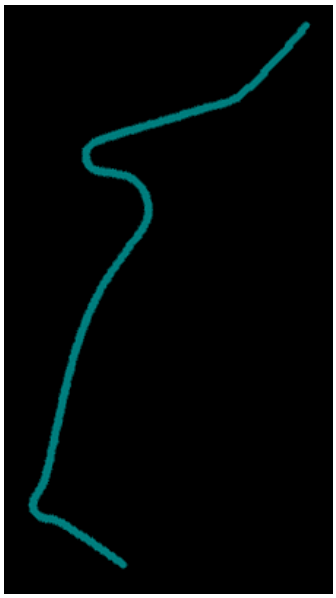
4.1.4.2 Trajetória 2: Curvas

O *ground truth* da trajetória 2 é ilustrado na Figura 122a e foi elaborado para avaliar o comportamento do algoritmo em curvas. Na Figura 122a, os elementos de interesse que compõem a trajetória são indicados em “1”: curva; em “2”: curva acentuada; em “3” e “4”: curvas decorrentes de rotatórias. O resultado do algoritmo é ilustrado na Figura 122b. Por inspeção visual, analisa-se que o resultado apresentou uma boa precisão, exceto em “2”, no qual a curva acentuada, na verdade, apresentou um aspecto suave.

Figura 122 – Trajetória 2: Curvas



(a) *Ground truth* da trajetória 2 (escala em metros). *Google Earth, earth.google.com/web/*



(b) Resultado da trajetória 2

Fonte: Produzido pelos autores.

O trecho compreendido entre os números “3” e “4” indicados na Figura 122a é parte de uma rodovia federal com intenso fluxo de veículos. Ao percorrê-lo, havia veículos que trafegavam no sentido oposto do agente, conforme ilustrado na Figura 123. Nota-se também, na figura, um sonorizador, que são pequenas ondulações impressas no asfalto com o objetivo de provocar trepidação, que causou movimentos aleatórios e indesejáveis na gravação do vídeo. Porém, tais fatores não influenciaram no resultado do algoritmo, demonstrando sua robustez.

Figura 123 – Veículo em sentido oposto e sonorizador na pista

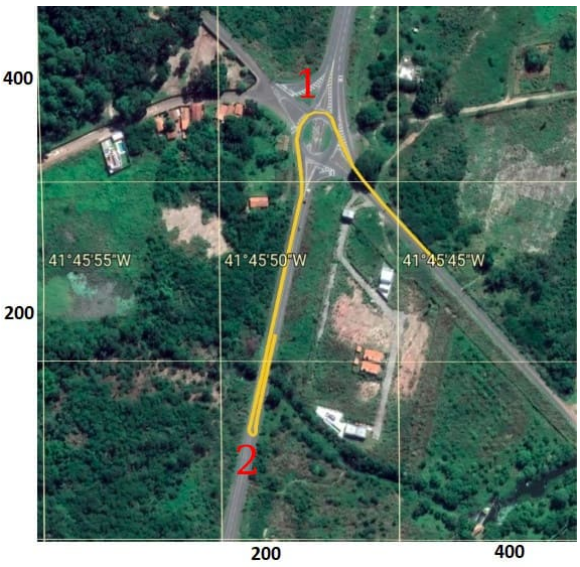
Fonte: Produzido pelos autores.

4.1.4.3 Trajetória 3: Retorno acentuado

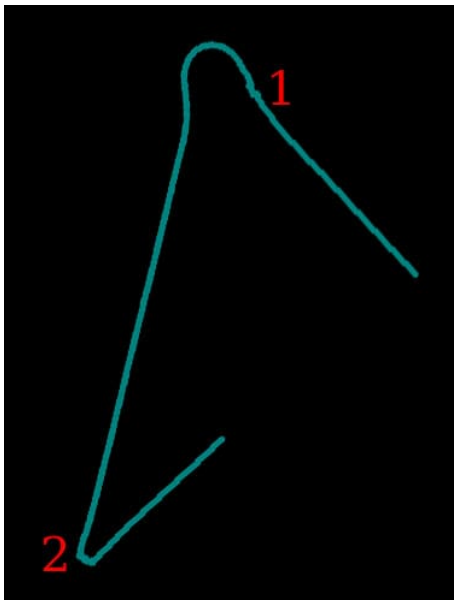
O *ground truth* da trajetória 3 é ilustrado na Figura 124a e foi elaborado para avaliar o comportamento do algoritmo em um retorno acentuado. Na Figura 124a, “1” indica uma

rotatória e “2” indica um retorno em uma rodovia estreita. O resultado do algoritmo é apresentado na Figura 124b.

Figura 124 – Trajetória 3: Retorno acentuado



(a) *Ground truth* da trajetória 3 (escala em metros). *Google Earth, earth.google.com/web/*



(b) Resultado da trajetória 3

Fonte: Produzido pelos autores.

Nota-se que o comportamento do algoritmo em retas e curvas suaves, como a da rotatória, apresenta uma boa precisão, no entanto, o resultado foi comprometido em dois pontos: a influência de um veículo já trafegando na rotatória e seguindo por uma via de saída diferente da do agente, e retornos acentuados, indicados por “1” e “2”, respectivamente, na Figura 124b.

No primeiro caso, nota-se que no trajeto indicado por “1” na Figura 124b, o resultado apresentou pequenas distorções, onde a orientação foi levemente comprometida. A distorção ocorreu no instante em que o agente se aproximava da entrada da rotatória, enquanto um veículo já trafegava nela, conforme destacado na Figura 125. Pode-se perceber o momento em que o veículo trafega na saída à direta da rotatória, enquanto o agente continua trafegando na via à esquerda. O resultado foi levemente comprometido pois como em determinado instante o veículo trafegava na mesma orientação do agente e logo em seguida altera sua própria orientação, o algoritmo processa esse deslocamento de *pixels* para a direita como se o agente estivesse rotacionando à esquerda. Caso semelhante foi apresentado na subseção 4.1.4.1.

Figura 125 – Veículo trafegando na rotatória

(a) Veículo trafegando na rotatória

(b) Veículo seguindo em um saída diferente da do agente

Fonte: Produzido pelos autores.

O segundo caso trata de um retorno realizado pelo agente em uma rodovia estreita. Nota-se que o retorno processado pelo algoritmo, indicado por “2” na Figura 124b, foi impreciso. Dentre os fatores que contribuíram para o erro de estimação da rotação, destaca-se a grande quantidade de movimento entre *frames* consecutivos. A Figura 126 mostra quatro *frames* consecutivos correspondentes a uma parte do retorno realizado. Note que as cenas mudam em cada *frames* de forma acentuada. Assim como descrito na subseção 4.1.3.2, quando há grande quantidade de movimento, a suposição do rastreador Lucas-Kanade de que o movimento da cena entre *frames* consecutivos é pequena não é cumprida, portanto, imprecisões de orientação e posição são esperadas. Além disso, como a rodovia em que o retorno foi realizado era bastante estreita, houve *frames* em que o pavimento asfáltico da rodovia não era mostrado, apenas a vegetação circundante, como, por exemplo, no *frame* ilustrado pela Figura 126a. Como a vegetação-local se movimenta de forma aleatória, pois está sujeita à ação do vento, o vetor gradiente era prejudicado, já que as *features*-referência se tornam inadequadas.

Figura 126 – *Frames* consecutivos do agente realizando retorno em rodovia

(a) *frame* 1

(b) *frame* 2

(c) *frame* 3

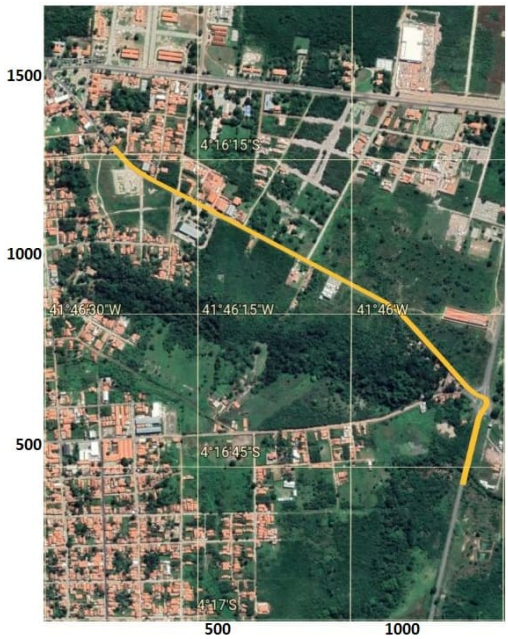
(d) *frame* 4

Fonte: Produzido pelos autores.

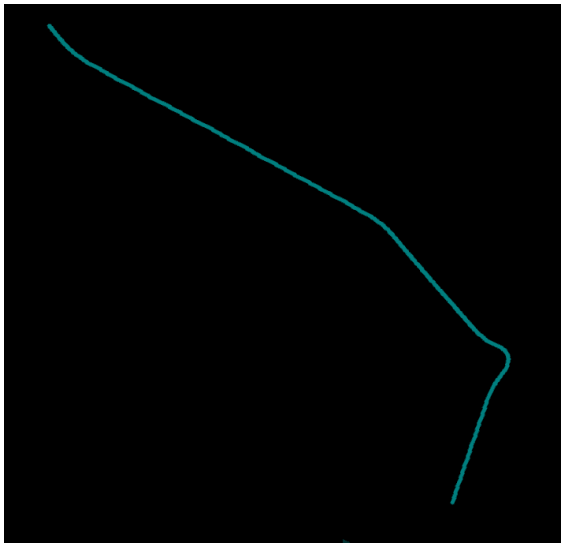
4.1.4.4 Trajetória 4

O *ground truth* da trajetória 4 é ilustrado na Figura 127a. O resultado do algoritmo é ilustrado na Figura 127b e apresentou uma boa precisão, sem alterações significativas.

Figura 127 – Trajetória 4



(a) *Ground truth* da trajetória 4 (escala em metros). *Google Earth, earth.google.com/web/*



(b) Resultado da trajetória 4

Fonte: Produzido pelos autores.

Em determinado trecho da trajetória, havia na pista redutores de velocidade do tipo tachões, ilustrado na Figura 128. Assim como no sonorizador descrito na trajetória 2 (vide subseção 4.1.4.2), não houve alterações perceptíveis no resultado, o que demonstra a robustez do algoritmo.

Figura 128 – Tachões no pavimento asfáltico

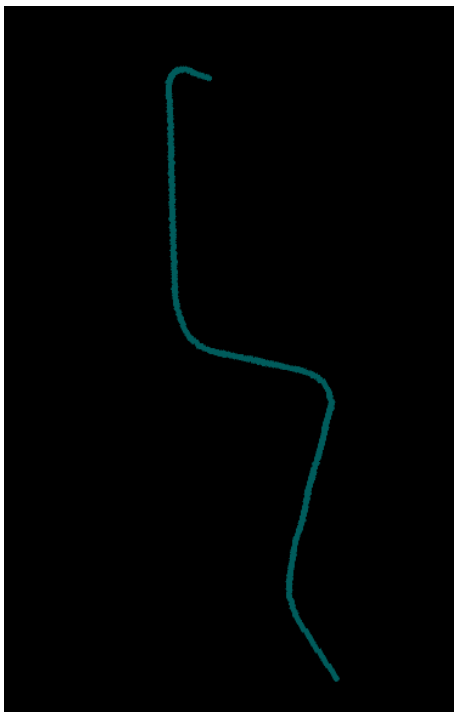
Fonte: Produzido pelos autores.

4.1.4.5 Trajetória 5

O *ground truth* da trajetória 5 é ilustrado na Figura 129a. O resultado do algoritmo é ilustrado na Figura 129b e, assim como na trajetória 4, apresentou uma boa precisão, sem alterações significativas.

Figura 129 – Trajetória 5

(a) *Ground truth* da trajetória 5 (escala em metros). *Google Earth, earth.google.com/web/*



(b) Resultado da trajetória 5

Fonte: Produzido pelos autores.

O percurso foi realizado nas vias urbanas da cidade. Em determinado trechos, não havia pavimento asfáltico, apenas calçamento de pedra, ilustrado na Figura 130. Mesmo trafegando em calçamento de pedra, que é um pavimento irregular, o algoritmo não sofreu alterações nos resultados.

Figura 130 – Via composta por calçamento de pedra

Fonte: Produzido pelos autores.

Além disso, havia trechos na trajetória compostos por acidentes geográficos, como o morro ilustrado na Figura 131. Assim como no terreno irregular anterior, o algoritmo não sofreu alterações nos resultados.

Figura 131 – Acidente geográfico na via

(a) Via com aclive

(b) Via com declive

Fonte: Produzido pelos autores.

4.1.5 Análise dos requisitos temporais do sistema de odometria visual

Reitera-se que, nesta implementação, assim como na validação de projetos com o uso de *datasets*, a captura de *frames* e processamento de estimação da trajetória são realizados de forma assíncrona, uma vez que já está disponível a sequência de imagens correspondente à trajetória. No entanto, em um sistema embarcado em tempo real, o processo de odometria visual e captura de *frames* deve ocorrer de forma síncrona. Nesse contexto, é válido analisar o tempo de execução das trajetórias estimadas a fim de servir como referência para uma

implementação futura de um sistema de odometria visual em tempo real, de forma que os requisitos temporais sejam cumpridos. No caso da Minized, é utilizado o *buffer* triplo para armazenamento dos *frames*, conforme descrito na seção 3.1, assim, a cada ciclo, são capturados e armazenados três *frames*, que serão substituídos no ciclo seguinte, em que serão capturados mais três *frames*. Dessa forma, é crucial garantir que o maior tempo de execução para estimação da trajetória da sequência de três *frames* seja menor que o tempo de duração do respectivo ciclo.

O tempo de execução é o tempo decorrido desde a etapa detecção de *features* até a etapa de reconstrução da trajetória, englobando o rastreamento de *features*, cálculo da matriz essencial e estimação da rotação e translação. A calibração da câmera é uma etapa de *setup*, portanto, é realizada antes do início do processamento.

O tempo de execução para cada trajetória dos testes em ambiente não controlado foi calculado, mostrado na Tabela 2. O algoritmo foi executado em uma CPU Intel Core i7-6500U 2.50GHz (4 núcleos), 8GB RAM.

Tabela 2 – Tempo de execução para testes em ambiente não controlado

Teste em Ambiente não controlado	Tempo de execução médio (ms)	Tempo de execução no melhor caso (ms)	Tempo de execução no pior caso (ms)
Trajetória 1	99,048	55,901	591,627
Trajetória 2	145,612	55,717	801,112
Trajetória 3	170,527	52,549	1015,604
Trajetória 4	157,497	53,600	624,783
Trajetória 5	153,305	59,864	766,158

Fonte: Produzido pelos autores.

É possível perceber que o tempo médio de execução é de aproximadamente 145ms. O tempo de execução no melhor caso ocorre quando a cena apresenta pouca textura, como ilustrado na Figura 132b, mas sem comprometer o funcionamento do algoritmo. Já o tempo de execução no pior caso ocorre quando a cena apresenta altíssima textura, principalmente devido à presença de vegetação, como ilustrado na Figura 132a. Em alta textura, uma maior quantidade de *features* são detectados e, consequentemente, rastreados, o que aumenta o tempo de execução consideravelmente. Note que a trajetória 3 apresentou tanto o maior tempo de execução no pior caso (1015,604 ms), quanto o menor tempo de execução no melhor caso (52,549 ms), sendo este aproximadamente 20 vezes menor que aquele. Assim, é importante que a duração do ciclo seja maior que o tempo de execução no pior caso e com uma margem de erro adequada, senão haverá perda de sincronia, afetando o funcionamento do sistema.

Figura 132 – *Frames* com pouca e grande quantidade de *features* detectados (representados por pontos verde-claros)

(a) *Frame* com pouca quantidade de *features* detectados

(b) *Frame* com grande quantidade de *features* detectados

Fonte: Produzido pelos autores.

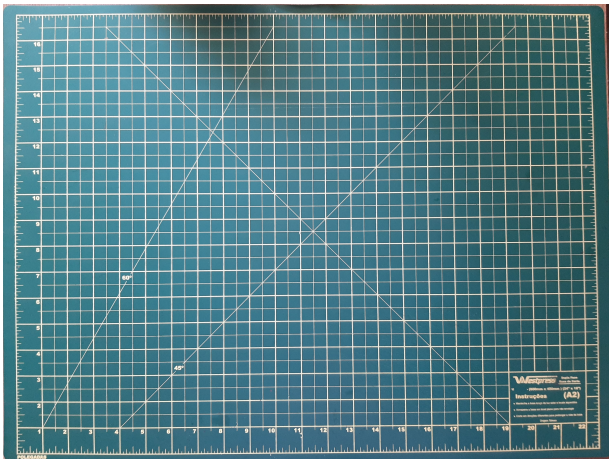
4.2 Resultados - Odometria Inercial

Para realizar cada um dos testes para os algoritmos implementados foi necessário utilizar alguma referência conhecida para que os dados relativos a orientação e posição pudessem ser extraídos.

Existem atualmente diversos dispositivos com precisão extremamente elevada capazes de realizar uma série de movimentos com velocidade e aceleração conhecidas, porém, nesse projeto esses equipamentos não estão disponíveis, tanto para realizar a calibração dinâmica dos sensores, quanto para realização dos testes. Dessa forma, foi necessário encontrar uma outra forma de garantir ao menos certo nível de precisão nos testes.

A forma encontrada para solucionar ou mitigar o problema apresentado foi utilizar uma base de corte A2 com dimensões de 600x450 mm para realização dos testes. Essa base possui graduações em cada um dos eixos que podem servir de referência para a posição e também referências de ângulo, mais especificamente 45° e 60°, naturalmente, as linhas dispostas perpendicularmente permitem também ter um referências para 90°, 180°, 270°, 360°/0°. A Figura 133 apresenta a base utilizada no projeto.

Figura 133 – Base de corte



Fonte: Produzido pelos autores.

Para garantir que as rotações para ângulo de arfagem e rolagem possuíssem uma referência, a plataforma com a PCB e os sensores foi introduzida em um caixa de dimensões 10,9x9,7x3,7cm, conforme a Figura 134. Além de fornecer uma referência, essa montagem garante também maior repetibilidade para os testes, porém, permite obter somente informações para os limites de cada uma dessas orientações, 90º, 0º e -90º. Por fim, essa solução permite também uma maior facilidade no direcionamento da plataforma para movimentações no plano.

Figura 134 – Montagem da plataforma na caixa para testes

Fonte: Produzido pelos autores.

As Tabelas 3, 4 e 5 apresentam as especificações utilizadas durante os testes para cada um dos sensores. Essas especificações são em parte aspectos construtivos para cada um desses sensores como, por exemplo, a não linearidade e taxa de ruído, e em parte aspectos relacionados a configurações realizadas de fato.

As configurações utilizadas são aquelas que, após análise considerando os aspectos abordados na subseção 3.4.7, possivelmente proporcionariam os melhores resultados na

teoria, garantindo um menor ruído e alta disponibilidade de dados para os algoritmos. Note que optou-se por sincronizar os dados dos sensores, pois isso permite não somente simplificar as etapas da estimação mas também garantir que toda a informações disponível seja utilizada. Essa sincronização, entretanto, não é necessária, sendo possível eventualmente incorporar na solução outros dispositivos que gozem de taxa de amostragem mais baixa como, por exemplo, um GPS.

Vale ressaltar também que o magnetômetro apresentou instabilidade na operação quando configurado para 8 Gauss, quiçá devido a alguma falha no dispositivo ou a algum evento esporádico. Quando configurado da forma apresentada porém, o dispositivo apresentou um comportamento adequado e, portanto, os testes foram realizados utilizando essa unidade do sensor, pois a mesma era a única disponível no projeto.

Tabela 3 – Especificações do sensor - Giroscópio

Configuração	Valor definido
Fundo de escala (FSR)	2000 (°/s)
Sensibilidade	16.4 (LSB/°/s)
Taxa de amostragem	200 (Hz)
Não linearidade	0.2 (% FSR)
Filtro passa-baixas digital	5 (Hz)
Rate Noise Spectral Density	0.005 (°/s/$\sqrt{Hz}$)

Fonte: Produzido pelos autores.

Tabela 4 – Especificações do sensor - Acelerômetro

Configuração	Valor definido
Fundo de escala (FSR)	16 (g)
Sensibilidade	2048.0 (LSB/g)
Taxa de amostragem	200 (Hz)
Não linearidade	0.5 (% FSR)
Filtro passa-baixas digital	5 (Hz)
Rate Noise Spectral Density	400 (µg /$\sqrt{Hz}$)

Fonte: Produzido pelos autores.

Tabela 5 – Especificações do sensor - Magnetômetro

Configuração	Valor definido
Fundo de escala (FSR)	2 Gauss (G)
Sensibilidade	12000.0 (LSB/G)
Taxa de amostragem	200 (Hz)
Não linearidade	0.1 (% FSR)
Filtro passa-baixas digital	Não específicado - Banda mínima

Fonte: Produzido pelos autores.

Note também que cada um dos algoritmos especificados possuem parâmetros que devem ser especificados para sua descrição, nessa seção serão apresentados esses parâmetros assim como os parâmetros de calibração obtidos após os processos apresentados na subseção 3.4.6.

Os valores dos *offsets* para o giroscópio, assim como os valores dos fatores de escala e dos *offsets* para o acelerômetro são especificados abaixo:

$$[\omega_b^x] = [-3.109756] \qquad [sf_{a^x}, a_b^x] = [1.000047, -0.0928]$$
$$[\omega_b^y] = [-0.182927] \qquad [sf_{a^y}, a_b^y] = [0.9806, -0.033625]$$
$$[\omega_b^z] = [0.487805] \qquad [sf_{a^z}, a_b^z] = [0.95835, 0.01069]$$

Os valores dos fatores de escala e dos *offsets* para o magnetômetro foram definidos de forma similar aos anteriores, porém, o fator de escala deve ser aplicado após a aplicação do *offset*. Note que o processo de calibração não foi realizado de forma efetiva em z, esses valores apresentados são as médias para estado estacionário obtidas e estão aqui definidos pelo fato de terem sido utilizados nos testes.

$$[sf_{m^x}, m_b^x] = [1.0084, 0.0246]$$
$$[sf_{m^y}, m_b^y] = [1.0, -0.0174]$$
$$[sf_{m^z}, m_b^z] = [1.0, 0.0111]$$

Para o filtro de Kalman foram utilizados os parâmetros definidos abaixo para os testes relativos a estimação da orientação:

$$Q = \begin{bmatrix} 0.5T^2 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0.05 \end{bmatrix} \qquad (4.1) \qquad R = \begin{bmatrix} 0.05 \end{bmatrix} \qquad (4.2)$$

$$P_o = \begin{bmatrix} 0.5 & 0 \\ 0 & 2{,}5 \end{bmatrix} \quad (4.3) \qquad X_o = \begin{bmatrix} 0 \\ \omega_b \end{bmatrix} \quad (4.4)$$

Para o filtro de Kalman utilizando a formulação de Joseph foi necessário alterar a a matriz de covariância do ruído associado as medições, R, de tal forma que $\sigma_\omega^2 = 0.0025$, devido a oscilações observadas com ao utilizar os parâmetros definidos. O valor apresentado garantiu a convergência da solução de forma que os testes pudessem ser realizados. Tal problema será abordado em análise posterior.

Verificou-se também que a matriz P_o definida abaixo apresentou melhores resultados que a definida anteriormente, porém, ela não foi utilizada devido a falha em alterar o parâmetro antes de iniciar os testes finais.

$$P_o = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.005 \end{bmatrix} \quad (4.5)$$

Para estimação da posição os seguintes parâmetros foram utilizados:

$$Q = \begin{bmatrix} 0.25T^4 & 0.5T^3 & 0 \\ 0.5T^3 & T^2 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0.05 \end{bmatrix} \quad (4.6) \qquad R = \begin{bmatrix} 0.0025 \end{bmatrix} \quad (4.7)$$

$$P_o = \begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 1.25 & 0 \\ 0 & 0 & 0.05 \end{bmatrix} \quad (4.8) \qquad X_o = \begin{bmatrix} 0 \\ 0 \\ a_b \end{bmatrix} \quad (4.9)$$

Para o filtro complementar os parâmetros são tais que 1-G(s) = 0.96 e G(s) = 0.04.

4.2.1 Resultados - Estimação da orientação

Conforme já abordado anteriormente, para realizar os testes para orientação para ângulo de arfagem e rolagem a referência será a própria caixa na qual a plataforma está inserida. Para ângulo de guinada as referências serão as demarcações na base de corte.

A movimentação da plataforma foi realizada de forma manual, portanto, existem erros não-caracterizados associados ao processo, esses erros naturalmente podem levar a resultados diferentes a cada novo teste devido ao fato de não se atingir com precisão a referência estabelecida, e não se obter uma movimentação padronizada durante o processo. Entre esses problemas é possível destacar, por exemplo, o estímulo inicial de aceleração que a plataforma é submetida, a velocidade mantida até a referência estabelecida, a duração do período de espera durante o processo de estimação, e no caso da estimação do ângulo de guinada, a força aplicada sobre a plataforma capaz de gerar atrito entre a plataforma e o plano, provocando oscilações que porventura não seriam observadas na realidade.

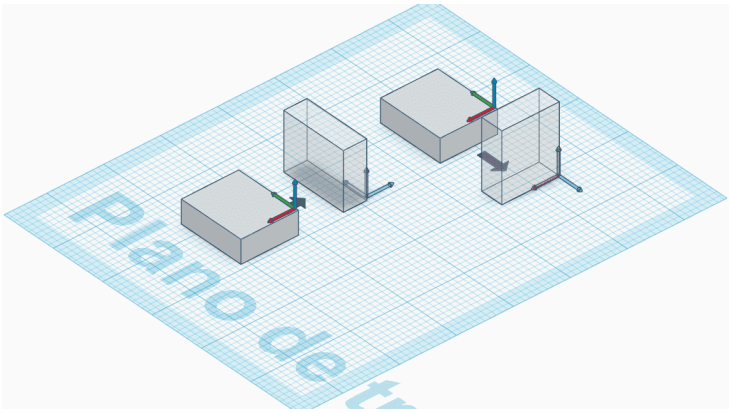
Para mitigar esses problemas os testes foram repetidos diversas vezes de forma a reduzir as discrepâncias observadas em relação ao tempo de espera nas movimentações e, caso alguma ação de movimentação grosseira fosse observada durante o teste, esse era descartado.

Por fim, ressalta-se que que a caixa não é completamente rígida, apresentando também leve convexidade nas extremidade laterais e, dependendo de como o teste for realizado essa pode levar a pequenas diferenças nas posições tomadas como referência tendo em vista que pode ocorrer ou não a deformação da estrutura a depender da força aplicada.

4.2.1.1 Testes - Avaliação dos resultados em curto prazo

Com o intuito de observar o comportamento dos algoritmos no curto prazo, incluindo questões de convergência e precisão foi proposto um teste relativamente simples, realizar a rotação da plataforma rapidamente em direção a uma referência de 90° para ângulo de arfagem e rolagem, conforme Figura 135.

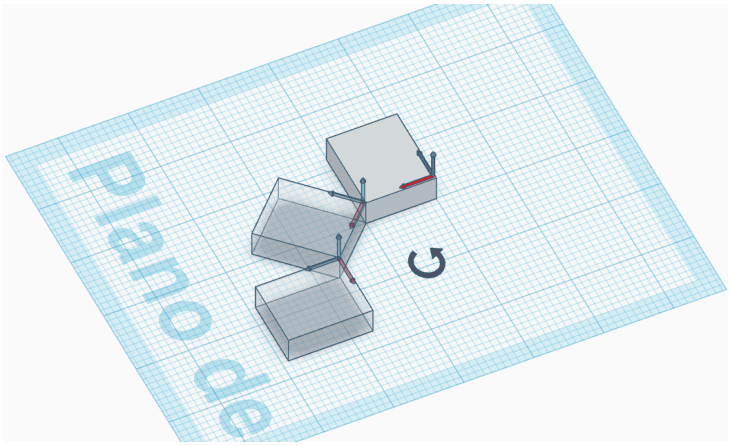
Figura 135 – Testes - Estimação da orientação em curto prazo para ângulo de arfagem e rolagem



Fonte: Produzido pelos autores.

Para estimação do ângulo de guinada a plataforma é rotacionada rapidamente em direção a uma referência de 45° na qual permanece por aproximadamente 4 segundos, em seguida, a mesma é rotacionada em direção a uma referência de 90° na qual permanece até o fim do teste, conforme a Figura 136.

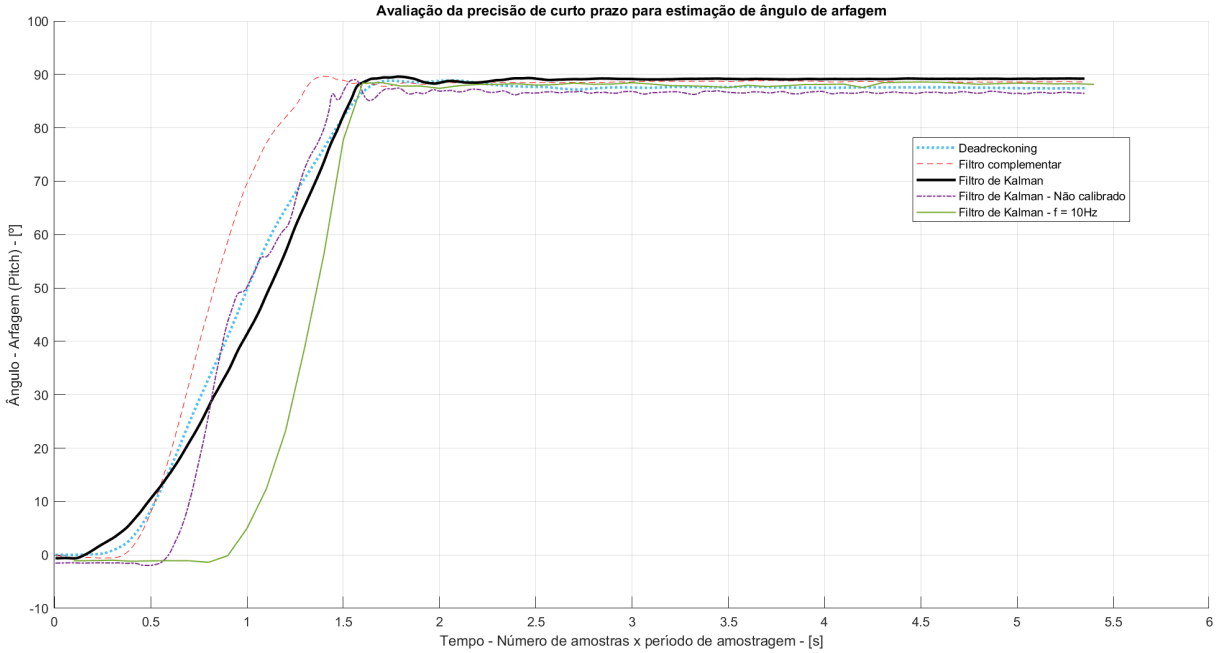
Figura 136 – Teste - Estimação da orientação em curto prazo para ângulo de guinada



Fonte: Produzido pelos autores.

Para o teste proposto foram avaliados os algoritmos apresentados na seção 3.4, mais especificamente o *dead reckoning*, o filtro complementar, e o filtro de Kalman. Além disso também foram avaliadas duas situações para o filtro de Kalman: sem calibração, e operando com taxa de amostragem de 10Hz. Nas Tabelas 6, 7 e 8, os dados denominados 'Média' e 'RMSE' foram obtidos através do cálculo do valor médio e raiz do erro quadrático médio utilizando os dados a partir do instante em que se atinge 98% do valor de pico, denominado 'Valor de pico', esse valor foi definido de forma arbitrária para padronizar os testes. O erro final, denominado 'Erro - Final' é o último valor observado na curva respectiva no fim do período de estimação.

Figura 137 – Teste - Estimação de curto prazo para ângulo de arfagem



Fonte: Produzido pelos autores.

Tabela 6 – Dados de estimação em curto prazo para ângulo de arfagem

Algoritmo	Referência (°)	Média (°)	RMSE (°)	Erro (%)	Valor de pico (°)	Erro - Final (°)
Dead reckoning	90	87.636	2.3881	2.63	88.89	2.5786
Filtro Complementar	90	88.614	1.3902	1.54	89.64	1.4193
Filtro de Kalman	90	89.106	0.9104	0.99	89.58	0.7835
Filtro de Kalman - Não calibrado	90	86.631	3.3761	3.74	89.06	3.5429
Filtro de Kalman - f = 10Hz	90	88.115	1.9342	2.09	88.58	1.8584

Fonte: Produzido pelos autores.

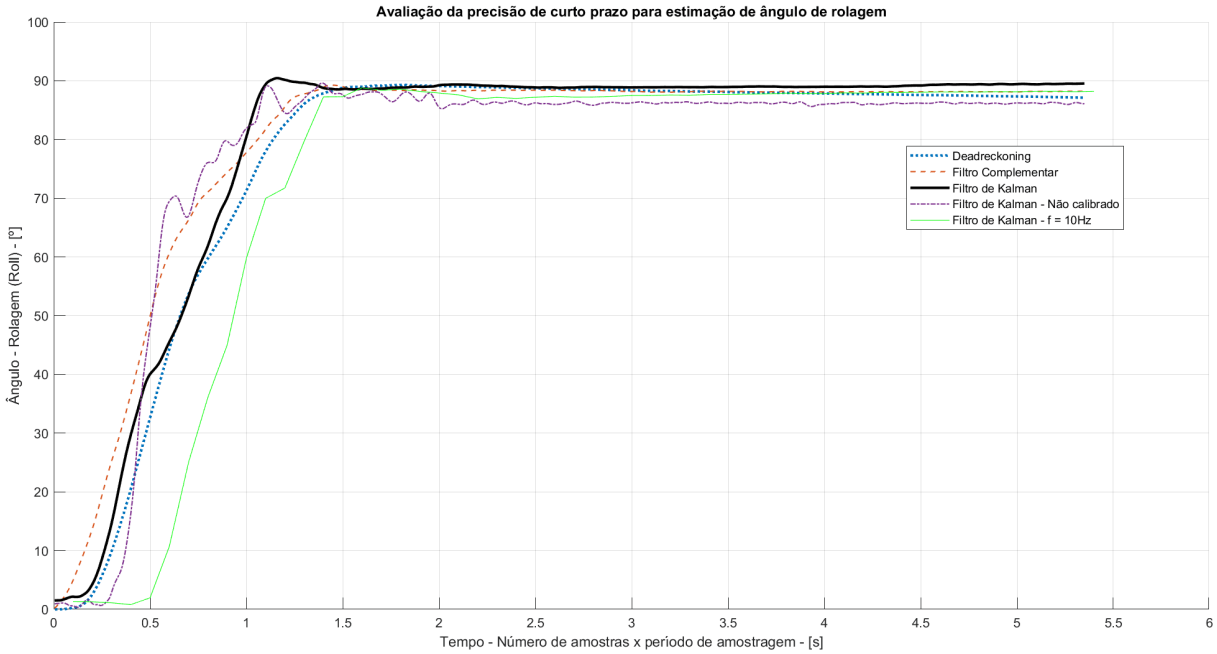
Para a estimação do ângulo de arfagem, independentemente do caso avaliado é possível notar que nenhum dos algoritmos apresenta fortes discrepâncias em relação ao valor de referência definido, conforme a Figura 137, apresentando erro final máximo de 3.5429° para o filtro de Kalman não calibrado, e mínimo de 0.7835° para o filtro de Kalman, conforme Tabela 6.

Ainda conforme Tabela 6, é possível observar também que existe ganhos significativos de desempenho ao utilizar o filtro de Kalman e o filtro complementar quando comparados ao *dead reckoning*, cerca 0.978° para o filtro complementar e 1.47° para o filtro de Kalman. Além disso, conforme esperado, o RMSE do filtro de Kalman tende a ser mínimo, apresentando valores relativamente baixos e bem abaixo da média dos outros algoritmos apresentados, indicando que possivelmente os parâmetros utilizados conseguem levar a ganhos próximos aqueles considerados ótimos.

Nota-se também que o *dead reckoning* e o filtro de Kalman não calibrados mesmo em curto prazo já apresentam tendências de acumulo de erro, condizente com a natureza do algoritmo *dead reckoning* e dos constantes estímulos de magnitude relativamente elevada quando os sensores não estão calibrados, apresentando ao final do processo um acréscimo de, respectivamente, 0.214° e 0.174° em relação ao valor médio.

Por fim, avaliando de uma forma mais qualitativa, no geral os algoritmos apresentaram velocidade de convergência relativamente próximas. Para o filtro de Kalman operando a taxa de amostragem de 10Hz esperave-se observar uma velocidade de convergência mais lenta e um atraso na resposta ao estímulo de movimento tendo em vista que o sistema receberia informações com menor frequência e tenderia a manter velocidades menores por mais tempo, a primeira colocação parece não valer, a segunda porém, pode ser observada, sendo necessário verificar se isso advém de erros durante a realização dos testes ou é uma resposta natural, dessa forma, é necessário avaliar os outros resultados antes de qualquer conclusão.

Figura 138 – Teste - Estimação de curto prazo para ângulo de rolagem



Fonte: Produzido pelos autores.

Tabela 7 – Dados de estimação em curto prazo para ângulo de rolagem

Algoritmo	Referência (°)	Média (°)	RMSE (°)	Erro (%)	Valor de pico (°)	Erro - Final (°)
Dead reckoning	90	88.041	2.0356	2.18	89.29	2.8768
Filtro Complementar	90	88.202	1.8018	1.997	89.25	1.7713
Filtro de Kalman	90	89.098	0.9279	1.002	90.45	0.4628
Filtro de Kalman - Não calibrado	90	86.168	3.8391	4.26	89.62	3.9172
Filtro de Kalman - f = 10Hz	90	87.732	2.3313	2.528	88.8	1.7972

Fonte: Produzido pelos autores.

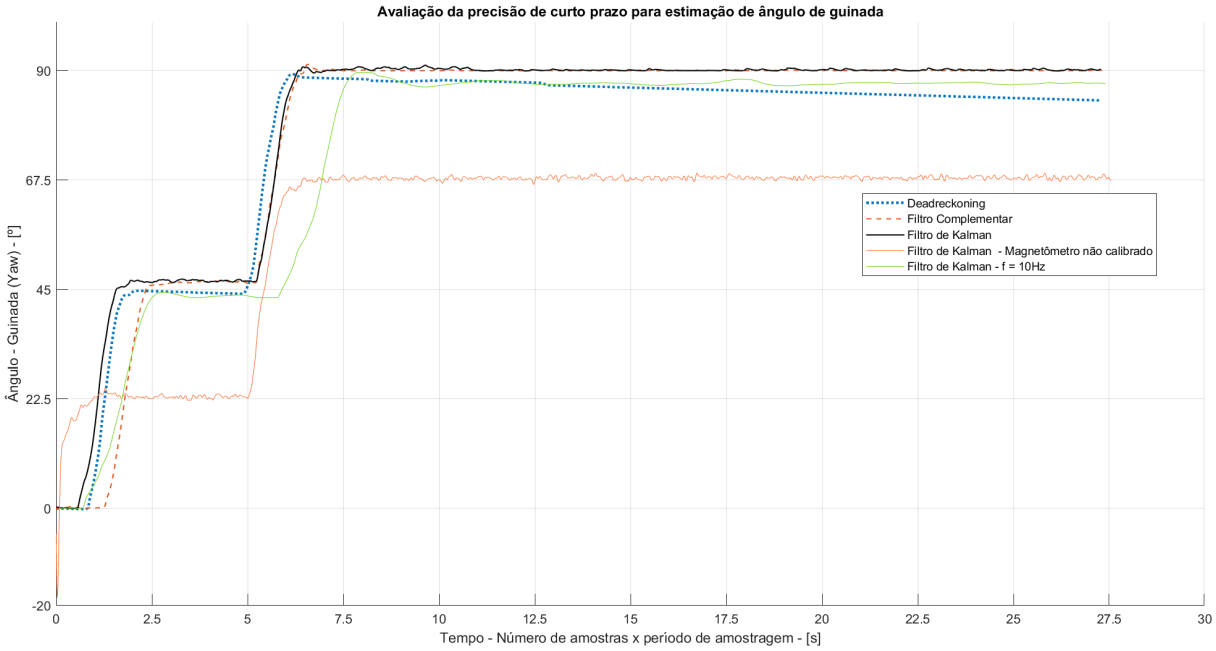
Assim como no caso da estimação do ângulo de arfagem, para o caso da rolagem, conforme Figura 138, os algoritmos apresentaram valores relativamente próximos a referência, com erro final máximo de 3.9172° para o filtro de Kalman não calibrado, e mínimo de 0.4628° para o filtro de Kalman, conforme Tabela 7. Note porém que após cerca de 4.3s após alcançar certa estabilidade a curva relativa ao filtro de Kalman apresenta uma nova tendência, indicando que algum estímulo deve ter sido fornecido, possivelmente devido a algum problema no teste, dessa forma espera-se observar na realidade um erro mais próximo aquele definido pelo RMSE de 0.9279°.

No geral esperava-se observar valores próximos aqueles apresentados para arfagem tendo em vista que os dois eixos de referência para cada um desses ângulos tem características similares. Os valores médios foram relativamente próximos, por exemplo, a diferença entre os valores médios para o filtro de Kalman para os 2 casos é de apenas 0.008, e um valor mais significativo de 0.463 para o filtro de Kalman não calibrado.

Nesse caso, porém, observa-se um acúmulo mais significativo de erros assim como oscilações mais pronunciadas. Uma avaliação mais adequada das diferenças apresentadas nesse caso é considerar uma particularidade da realização dos testes nesse eixo, como o cabo que alimenta a plataforma está inserido bem no centro de uma das laterais onde uma das referências para o ângulo de rolagem é definida, o teste é realizado nas extremidade da superfície na qual a base de corte está colocada para que a rotação possa ser realizada de forma completa, porém, isso implica em movimento mais bruscos e menos estabilidade da plataforma, possivelmente levando aos problemas apresentados.

Por fim, observa-se novamente os tempos de convergência similares na maioria dos casos, além disso, ainda existe atraso na resposta no caso do filtro de Kalman operando a 10Hz.

Figura 139 – Teste - Estimação de curto prazo para ângulo de guinada



Fonte: Produzido pelos autores.

Tabela 8 – Dados de estimação em curto prazo para ângulo de guinada

Algoritmo	Referência (°)	Média (°)	RMSE (°)	Erro (%)	Valor de pico (°)	Erro - Final (°)
Dead reckoning	45 90	44.3568 86.0726	0.7082 4.1458	1.43 4.36	44.7196 89.2151	6.1377
Filtro Complementar	45 90	46.3708 90.0073	1.4116 0.0454	-3.05 -0.01	46.7913 91.3154	-0.0927
Filtro de Kalman	45 90	46.6428 90.1668	1.6918 0.2583	-3.65 -0.19	47.127 91.1176	-0.2287
Filtro de Kalman - Não calibrado	45 90	22.9144 67.9267	22.1098 22.0785	49.08 24.53	24.7045 68.9357	22.5116
Filtro de Kalman - f = 10Hz	45 90	43.6534 87.4701	1.4086 2.5898	2.99 2.81	44.4686 89.6317	2.6682

Fonte: Produzido pelos autores.

Para finalizar os testes de curto prazo avalia-se então a estimação da orientação para ângulo de guinada, conforme Figura 139. Nesse caso duas referências são utilizadas, 45° e 90°, note também que esse teste é mantido por tempo mais significativo que os 2 anteriores, apesar de não ser o adequado, o objetivo é ressaltar algumas diferenças presentes nesse caso.

Inicialmente, algo que fica bastante claro é a diferença entre a necessidade de calibração nos casos apresentados, nos casos anteriores apesar da calibração ser essencial para promover uma maior estabilidade e garantir uma maior precisão, a estimação ainda pode ser realizada com certa precisão sem realizar o processo de calibração.

Nesse caso, porém, realizar a estimação utilizando as informações do magnetômetro sem calibração não é possível, note que para um ângulo de 45° é verificado um valor médio de 22.914° e de 67.927° para 90°, conforme a Tabela 8. Note que esses erros são bastante expressivos e, portanto, o valor estimado não é reflexo da realidade observada.

Note também que, além dos erros apresentados, existe também uma variação diferente em cada um dos eixos, associando os dois fatores observa-se que não é possível descrever uma volta completa utilizando o sensor sem calibração, pois na prática o que se observa não é um desvio, mas diferentes faixas de valores para cada quadrante da volta descrita.

Dessa forma é imperativo realizar a calibração para incorporar as informações do magnetômetro no processo de estimação, caso contrário, qualquer um dos outros algoritmos apresentados e suas variações apresentariam resultados mais adequados.

Outra questão interessante a ser observada é que, nos algoritmos que utilizam as informações do magnetômetro no processo de estimação, existe uma diferença significativa de precisão entre as referências, por exemplo, para o filtro complementar existe um erro de -3.05% para referência de 45° e de apenas -0.01% para 90°, algo similar é observado no filtro de Kalman, com um erro de -3.65% para referência de 45° e de apenas -0.19% para 90°.

Esse problema advém do processo de correção utilizado para resolver o problema das diferentes variações apresentadas que, atrelado ao processo de calibração, faz com que as variações sejam aproximadas em cada eixo, porém, para ângulos intermediários, por exemplo, entre 0° e 90°, existirão erros mais significativos, ao passo de que para os ângulos de referência 0°, 90°, 180° e 270° serão observados os menores erros possíveis, dessa forma existe um *trade-off* que deve ser avaliado de forma mais adequada para verificar se a correção deve ser efetuada ou apenas o processo de calibração.

Independente de outras questões, é possível notar que os resultados para os ângulos de referência seguem a proposta apresentada, possuindo os menores erros possíveis dentre os ângulos estimados, por exemplo, um RMSE de 0.0454° para o filtro complementar, e de 0.2583° para o filtro de Kalman, conforme a Tabela 8.

Avaliando a questão do erro, nota-se que existe uma tendência de desvio bem mais

significativa nesse caso, por exemplo, no caso do *dead reckoning* existe um desvio de valores bem próximos a referência para um valor final com desvio de 6.1377°. Apesar de ser difícil de notar a curto prazo, esse desvio de característica mais acentuada é oriundo da construção do dispositivo, sendo esse eixo mais sensível a erros e a transmissão de estímulos aplicados a outros eixos de referência.

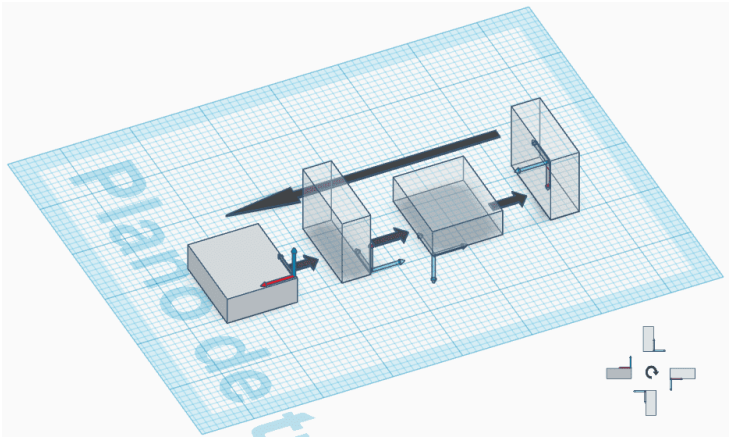
Por fim, foi observado que nesse caso não existe diferença apreciável no tempo de resposta para o filtro de Kalman utilizando taxa de amostragem de 10Hz, porém, existe diferença significativa na velocidade de convergência, exatamente o contrário daquilo que foi observado nos outros casos, dessa forma os testes acabaram sendo inconclusivos para esses aspectos.

4.2.1.2 Testes - Avaliação dos resultados em longo prazo

Para entender melhor o comportamento dos algoritmos apresentados no longo prazo, é proposto um teste de maior duração que o apresentado anteriormente, com duração próxima a 1 minuto, apesar de parecer um tempo curto, o número de amostras é relativamente elevado devido a taxa de amostragem relativamente alta, 200Hz, dessa forma é já é possível observar os erros e características desejadas.

O teste consiste na realização de movimentos periódicos sem deslocamento com rotação em torno de um mesmo eixo de referência, evitando movimentações nos planos de referência. Para os ângulos de guinada e arfagem são realizados movimentos de 90° a -90° por 4 vezes consecutivas, conforme a Figura 140.

Figura 140 – Testes - Estimação da orientação em longo prazo para ângulo de arfagem e rolagem com estímulos periódicos

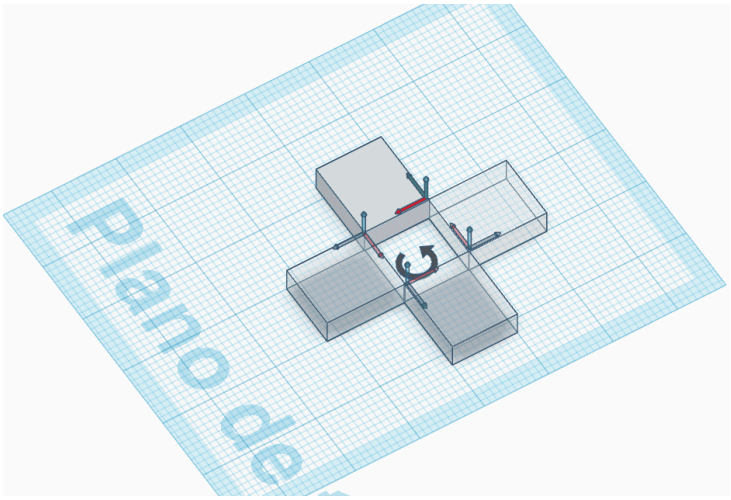


Fonte: Produzido pelos autores.

Para o ângulo de guinada são realizados movimentos para 90°, 180°, 270° e 360°, por fim retornando a 0°. O processo é repetido 4 vezes. A Figura 141 apresenta o procedimento.

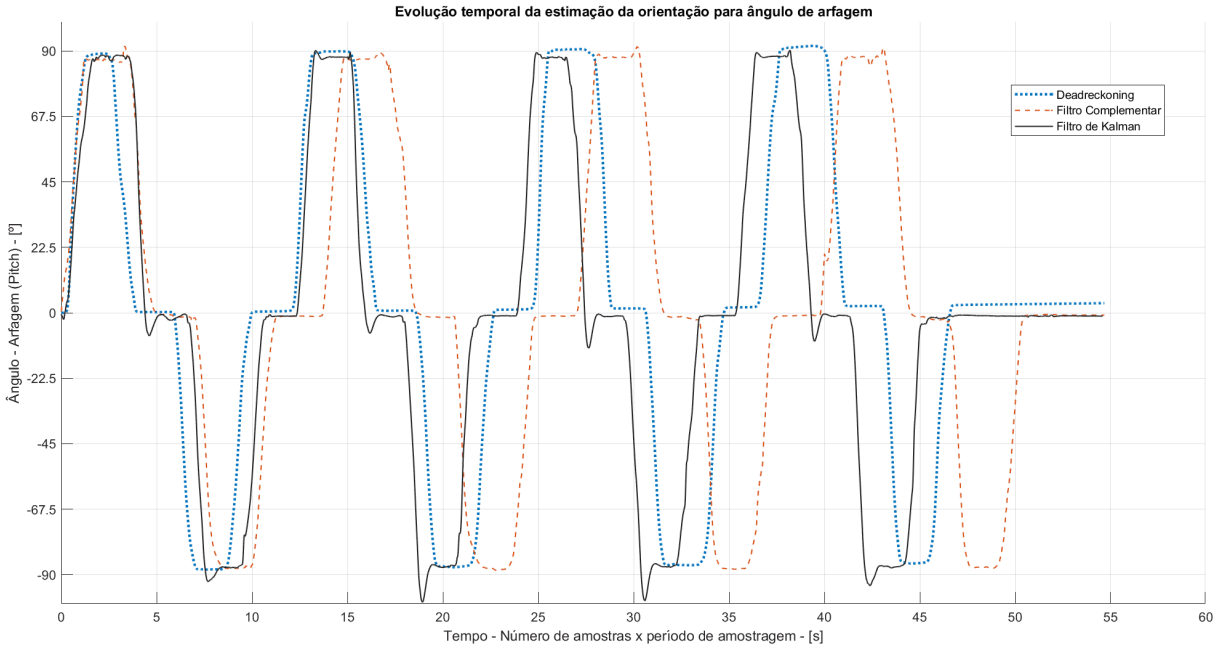
Note que para todos os casos se estabeleceu um limite de 4 repetições, isso advém do fato do cabo de conexão que fornece a energia a plataforma ser mecanicamente frágil devido a sua seção nominal bastante reduzida, realizar muitas repetições poderia levar a falha mecânica do cabo. Tendo em vista que o cabo era o único disponível esse problema poderia impossibilitar a realização dos testes. Além disso, devido a montagem precária da plataforma na caixa foi observado um rápido aquecimento, dessa forma, para não danificar o dispositivo, evitou-se utilizar a plataforma por um longo período tempo.

Figura 141 – Teste - Estimação da orientação em longo prazo para ângulo de guinada com estímulos periódicos



Fonte: Produzido pelos autores.

Figura 142 – Teste - Estimação de longo prazo com estímulos periódicos para ângulo de arfagem



Fonte: Produzido pelos autores.

Tabela 9 – Dados de estimação de longo prazo com estímulos periódicos para ângulo de arfagem

Algoritmo	Referência (°)	Tempo (s)	Média (°)	RMSE (°)	Erro - Inicial (°)	Erro - Final (°)
Dead reckoning	0	54.67	2.999	3.0044	0.00052	-3.312
Filtro Complementar	0	54.67	-0.7709	0.7754	-0.3908	0.802
Filtro de Kalman	0	54.585	-1.1384	1.1628	0.4375	1.137

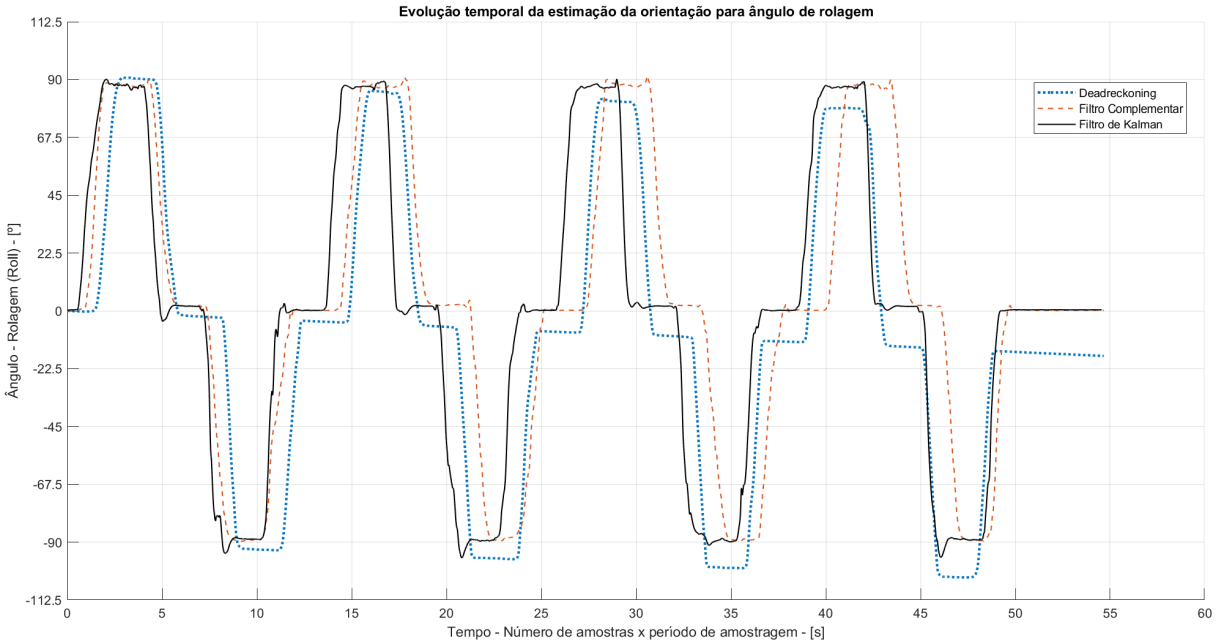
Fonte: Produzido pelos autores.

Conforme Figura 142 é possível observar que os algoritmos apresentaram comportamento relativamente similar ao longo de todo o processo de estimação para ângulo de arfagem, porém, existe uma leve diferença para o filtro de Kalman, conforme é possível notar, existem sobressaltos de magnitudes variadas ao alcançar a referência.

Esperava-se observar esses sobressaltos em maior ou menor escala durante os processo, como o algoritmo apresenta uma natureza preditiva, caso se estabeleça uma tendência de movimento e o algoritmo interprete que essa tendência continuará, uma manobra que leve a uma quebra nessa tendência, por exemplo, a parada que ocorre ao chegar na referência, forçará com que o algoritmo tenha que reavaliar se a previsão ainda é válida ou se a nova informação obtida irá estabelecer uma nova tendência, com isso, é observado então a um atraso até que ocorra a transição para a nova tendência observada, que no caso é manter-se constantemente em uma posição específica.

Em relação a evolução do erro no processo, conforme Tabela 9, comparando os resultados para erro inicial e final já se observa uma tendência de desvio mais acentuado para o *dead reckoning*, além disso, observa-se que esse desvio tem uma tendência a crescer de forma relativamente rápida com cada novo estímulo.

Figura 143 – Teste - Estimação de longo prazo com estímulos periódicos para ângulo de rolagem



Fonte: Produzido pelos autores.

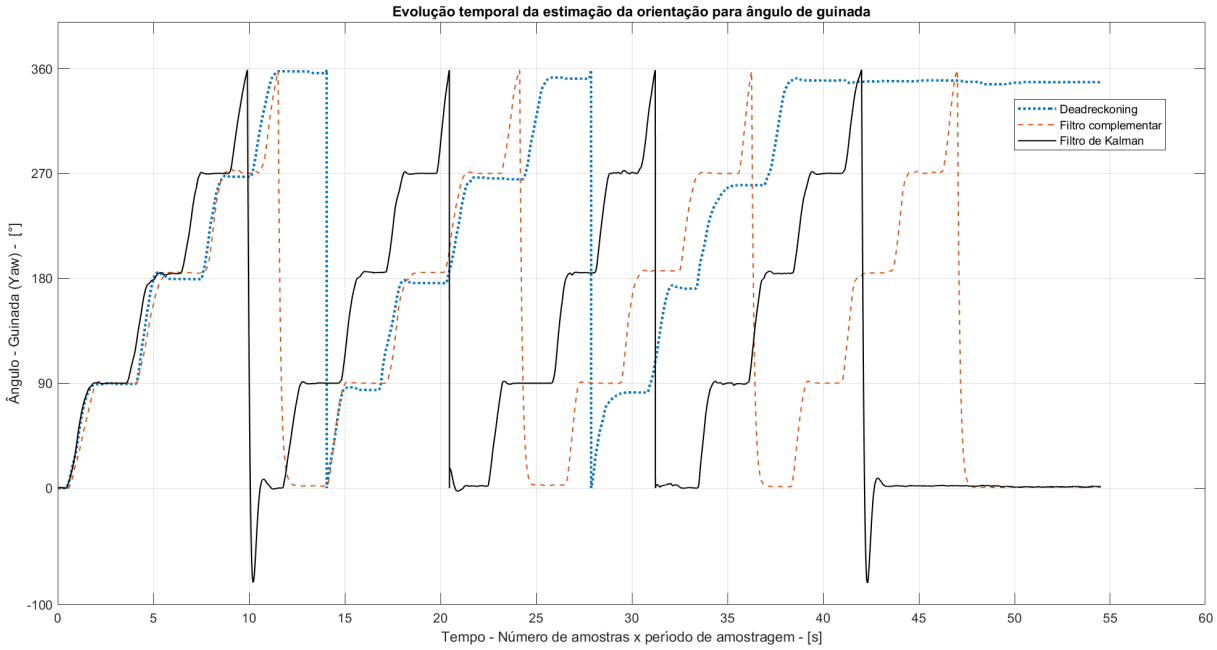
Tabela 10 – Dados de estimação de longo prazo com estímulos periódicos para ângulo de rolagem

Algoritmo	Referência (°)	Tempo (s)	Média (°)	RMSE (°)	Erro - Inicial (°)	Erro - Final (°)
Dead reckoning	0	54.66	-16.5994	16.6016	0.00017	17.554
Filtro Complementar	0	54.66	0.2202	0.2314	-0.0196	0.183
Filtro de Kalman	0	54.555	0.3746	0.3846	-0.1251	0.4113

Fonte: Produzido pelos autores.

Conforme a Figura 143 e os dados expressos na Tabela 10 é interessante notar que quando existem erros mais expressivos e instabilidade durante o processo, a tendência de desvio se apresenta bem mais rapidamente e de forma bem mais expressiva no caso do *dead reckoning*, com um erro acumulado de 17.554° ao final do processo, algo que não é observado nos outros algoritmos.

Figura 144 – Teste - Estimação de longo prazo com estímulos periódicos para ângulo de guinada



Fonte: Produzido pelos autores.

Tabela 11 – Dados de estimação de longo prazo com estímulos periódicos para ângulo de guinada

Algoritmo	Referência (°)	Tempo (s)	Média (°)	RMSE (°)	Erro - Inicial (°)	Erro - Final (°)
Dead reckoning	0	54.5	348.8195	348.765	-0.00005	11.752
Filtro Complementar	0	54.5	0.7365	0.7515	-0.00003	-1.077
Filtro de Kalman	0	54.5	1.2716	1.3302	-0.1627	-0.9122

Fonte: Produzido pelos autores.

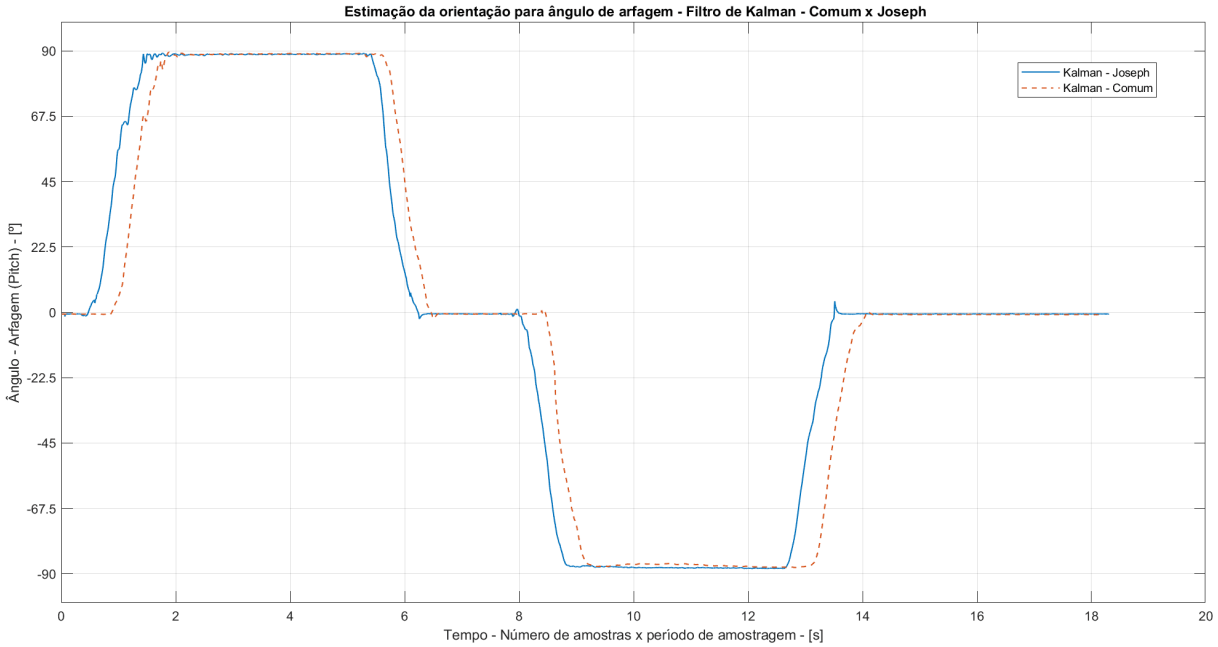
Por fim, analisa-se o teste proposto para ângulo de guinada. Conforme a Figura 144 é possível notar que o problema dos sobressaltos já apresentado tende a ser mais severo quando uma variação muito brusca ocorre, que é o caso da transição de 360° a 0°. Em todos os casos desse teste é possível observar uma série de fatores que tendem a se repetir em maior ou menor escala, portanto, conclusões bastante similares podem ser tiradas em todos os casos e não serão avaliadas aqui.

Em relação a precisão dos algoritmos avaliados, é possível notar que o filtro complementar tende a apresentar uma precisão ligeiramente maior que o filtro de Kalman em alguns casos, por exemplo, com erro de 0.183° contra 0.4113° para estimação do ângulo de rolagem, conforme a Tabela 10. O contrário também pode ser observado em outros casos, por exemplo, -1.077° contra -0.9122° para estimação do ângulo de guinada, conforme a Tabela 11. Além disso, ambos apresentam uma evolução bastante similar para o erro observado, dessa forma, considerando a relação custo-benefício dos algoritmos, possivelmente o uso do filtro de Kalman não se justifica, ao menos não antes de introduzir melhorias.

4.2.1.3 Testes - Avaliação do filtro de Kalman com uso da forma estabilizada de Joseph

A introdução da forma estabilizada de Joseph tinha como principal objetivo melhorar o desempenho da estimação da posição, porém, é importante introduzir essa variação para estimar a orientação para verificar a diferença efetiva observada.

Figura 145 – Teste - Filtro de Kalman - Comum x Joseph para ângulo de arfagem



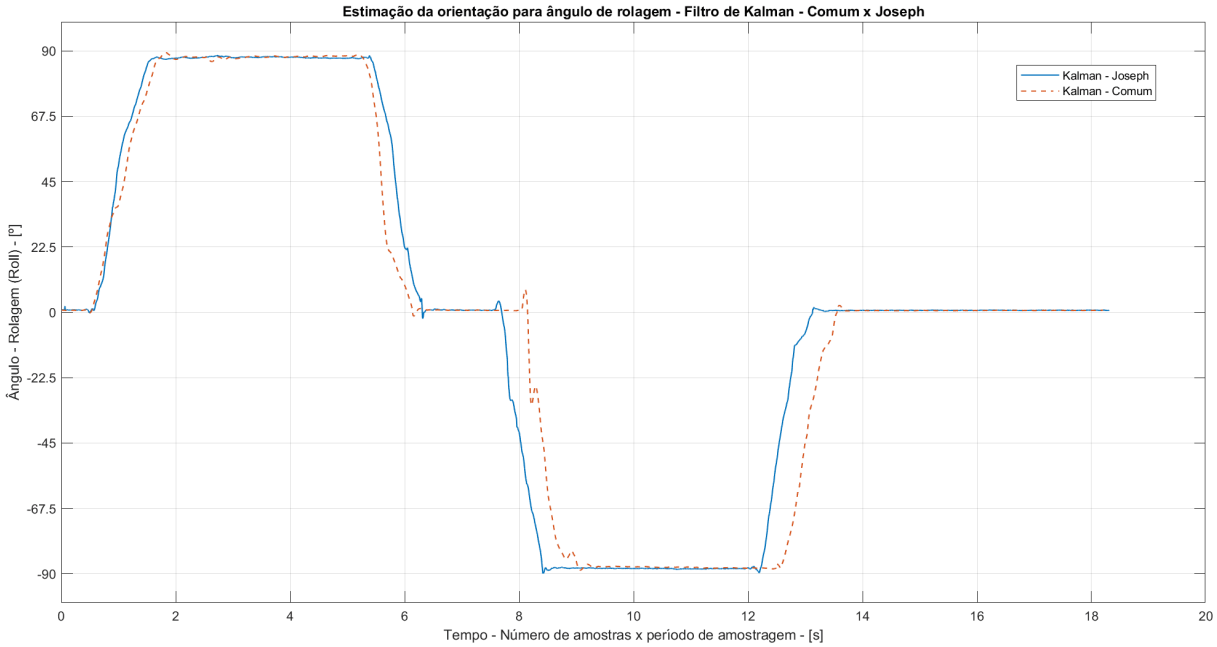
Fonte: Produzido pelos autores.

Tabela 12 – Dados de estimação - Filtro de Kalman - Comum x Joseph para ângulo de arfagem

Algoritmo	Referência (°)	Média (°)	RMSE (°)	Erro (%)	Valor de pico (°)	Erro - Final (°)
Filtro de Kalman	90 -90	88.9075 -87.0828	1.1065 2.9458	1.21 3.24	89.5725 -87.8659	0.754179
Filtro de Kalman - Joseph	90 -90	88.8094 -87.9499	1.198 2.0575	1.32 2.28	89.2091 -88.2416	0.62498

Fonte: Produzido pelos autores.

Figura 146 – Teste - Filtro de Kalman - Comum x Joseph para ângulo de rolagem



Fonte: Produzido pelos autores.

Tabela 13 – Dados de estimação - Filtro de Kalman - Comum x Joseph para ângulo de rolagem

Algoritmo	Referência (°)	Média (°)	RMSE (°)	Erro (%)	Valor de pico (°)	Erro - Final (°)
Filtro de Kalman	90	87.8631	2.1753	2.37	89.4295	-0.703
	-90	-87.7957	2.2153	2.45	-88.7967	
Filtro de Kalman - Joseph	90	87.7703	2.2419	2.48	88.3881	-0.67
	-90	-88.241	1.7629	1.95	-89.9481	

Fonte: Produzido pelos autores.

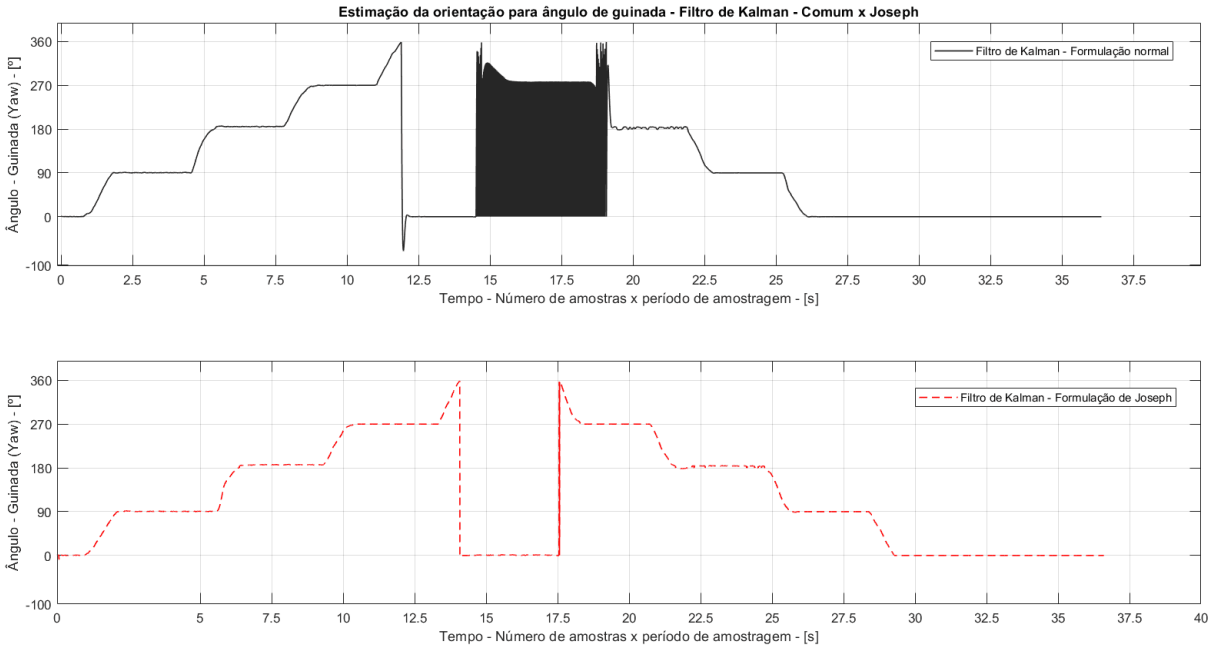
Para estimação do ângulo de arfagem e de rolagem para esse teste são analisadas as Figuras 145 e 146. Em ambos os casos é possível notar que o comportamento é essencialmente o mesmo, porém existe ligeira diferença na precisão, conforme as Tabelas 12 e 13 é possível notar que o filtro de Kalman com a forma de Joseph apresenta precisão maior quando estimando para referência de -90°, com média de -87.0828 e -87.7957° para o filtro de Kalman e de -87.9499° e -88.241° para o filtro de Kalman utilizando Joseph.

Além disso, observa-se também que o acúmulo de erro é menor no caso do filtro de Kalman utilizando a forma estabilizada de Joseph, 0.625° contra 0.754° do filtro de Kalman para ângulo de arfagem, e -0.67° contra -0.703° do filtro de Kalman para o ângulo de rolagem, o que é condizente com a redução dos problemas numéricos promovidos por essa formulação que se traduzem em maior estabilidade e precisão.

Note que apesar de apresentar certo ganho, essa formulação tem custo ainda maior que a formulação normal do filtro de Kalman que, conforme analisado anteriormente, parece não apresentar ganhos significativos em relação ao filtro complementar, ao menos não com

os parâmetros, modelos, e aprimoramentos atuais, portanto, é interessante analisar qual o custo computacional aproximado para cada caso para uma avaliação mais adequada.

Figura 147 – Teste - Filtro de Kalman - Comum x Joseph para ângulo de guinada



Fonte: Produzido pelos autores.

Para realizar a análise da diferença entre o filtro de Kalman comum e utilizando a forma de Joseph, é interessante introduzir um problema associado ao uso dos ângulos de Euler para definir a orientação do corpo, conforme a Figura 147.

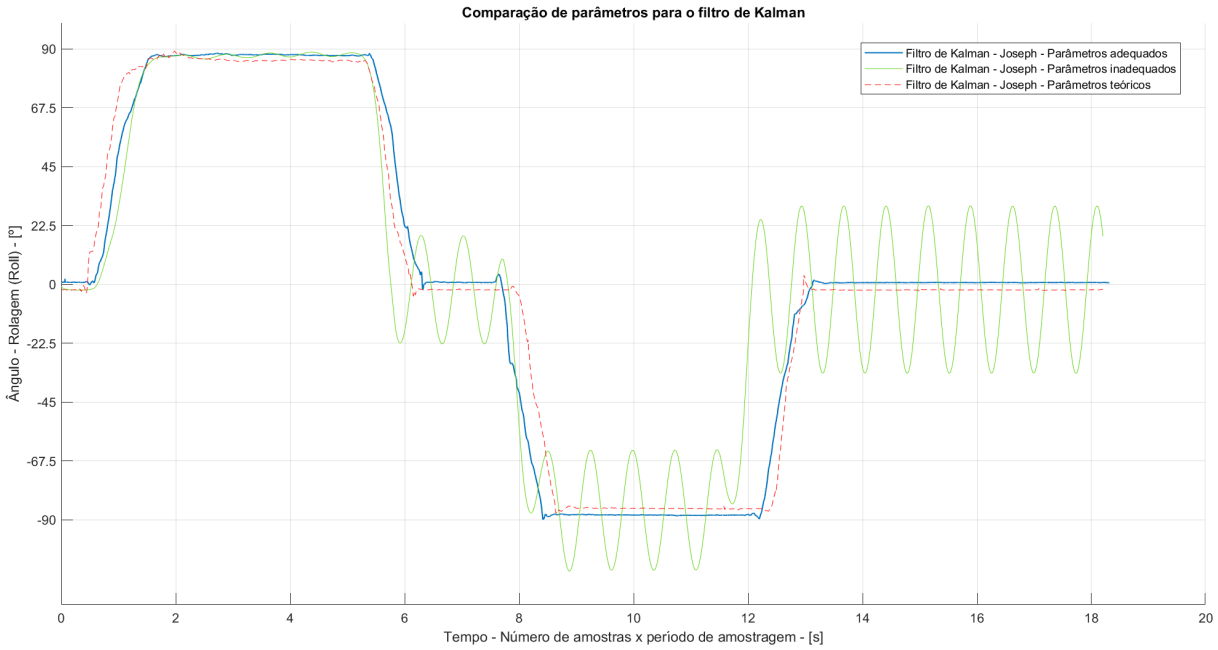
Note que na figura que existe um trecho com alta taxa de variação o algoritmo apresenta instabilidade, isso ocorre devido a um estado indeterminado para orientação entre 0° e 360°, caso o sistema permanece inerte e o processo tenha confiança significativa nas medidas, como o magnetômetro tende a apresentar ruídos significativos esses podem ocasionar uma transição continua entre 0° e 360° fazendo com que o algoritmo apresente uma tendência a acompanhar o movimento gerando instabilidade. Além disso, note que tal condição de indeterminismo gera problema numéricos que, caso permaneçam podem eventualmente levar a problemas na matriz de covariância do erro do sistema impedindo que o sistema volte a convergir.

Avaliando as diferenças entre o filtro de Kalman com e sem a forma de Joseph conforme a Figura 147, é possível então notar que a forma de Joseph consegue mitigar o problema apresentado, sendo bem mais estável já que promove a redução de problemas numéricos e eventuais problemas que causem divergência ou não convergência na matriz da covariância do erro tendo em vista que garante que a mesma seja sempre uma matriz definida positiva de forma que sempre ocorra a convergência para uma solução finita em estado estacionário.

Note que não há como afirmar o exposto sem que se analise as matrizes especificadas, porém, tal consideração foi tomada como adequada.

Com os resultados apresentados é possível notar portanto, que utilizar o filtro de Kalman com a formulação de Joseph nesse caso é uma alternativa bastante adequada.

Figura 148 – Teste - Diferença entre respostas para diferentes parâmetros



Fonte: Produzido pelos autores.

Tabela 14 – Dados de estimação - Resposta a diferentes parâmetros

Algoritmo	Referência (°)	Média (°)	RMSE (°)	Erro (%)	Valor de pico (°)	Erro - Final (°)
Joseph - Parâmetros adequados	90 -90	87.7703 -88.241	2.2419 1.7629	2.48 1.95	88.3881 -89.9481	-0.67
Joseph - Parâmetros teóricos	90 -90	85.9064 -85.6612	4.1530 4.3448	4.55 4.82	89.2173 -87.7101	1.9168

Fonte: Produzido pelos autores.

É importante notar também que uma escolha adequada de parâmetros é essencial para um bom funcionamento do sistema no caso do filtro de Kalman, considere, por exemplo, o teste realizado utilizando o filtro de Kalman com formulação de Joseph expresso na Figura 148.

Nesse caso foram feitos testes com três parâmetros distintos para a matriz de covariância do ruído associado as medições, R, tal que $R = [\sigma_\omega^2]$. Para os testes para a curva denominada 'Filtro de Kalman - Joseph - Parâmetros adequados' utilizou-se os mesmos valores de σ_ω^2 que para os outros testes que envolvem a formulação de Joseph, $\sigma_\omega^2 = 0.0025$, para a curva denominada 'Filtro de Kalman - Joseph - Parâmetros inadequados' utilizou-se um valor arbitrário para exemplificar os problemas $\sigma_\omega^2 = 1$, por fim, a curva denominada

'Filtro de Kalman - Joseph - Parâmetros teóricos', utiliza os parâmetros especificados de acordo com a as características construtivas do dispositivo apresentadas pelo fabricante, de acordo com a *Rate Noise Spectral Density* definida na Tabela 3.

É importante notar nesse caso que nem sempre os parâmetros teóricos são adequados a um determinado sistema pois na prática o comportamento dinâmico observado pode ser bastante diferente daquele previsto pelo simples fato de qualquer modelo ser meramente uma aproximação.

Dessa forma é necessário realizar testes para adaptar os parâmetros para que esses reflitam melhor a realidade observada, mesmo que apenas de forma empírica.

Por exemplo, considere a Tabela 14, nesse caso note que os parâmetros considerados adequados e obtidos de forma empírica apresentam precisão bem superior aqueles especificados de forma teórica.

Quando se tem um bom modelo para o sistema, ou seja, quando a matriz de covariância do erro tem entradas de pequena magnitude, e valores adotados para covariância do ruído associado as medições são altos, o ganho de Kalman, tenderá a ser pequeno, dando ênfase nas predições realizadas e não mais nas medições.

Quando as condições apresentadas são verdadeiras porém as medições contém ainda informação importante para o processo de estimação, o filtro tenderá a apresentar uma operação não adequada. Com ganhos pequenos o filtro não responde mais tão bem aos estímulos e passa a ter uma resposta mais lenta de tal forma que não consegue mais seguir o processo, levando então a não-convergência ou divergência do processo, isso pode ser representado na curva de parâmetros considerados inadequados em que existe uma tendência de oscilação crescente a medida que o processo de estimação avança.

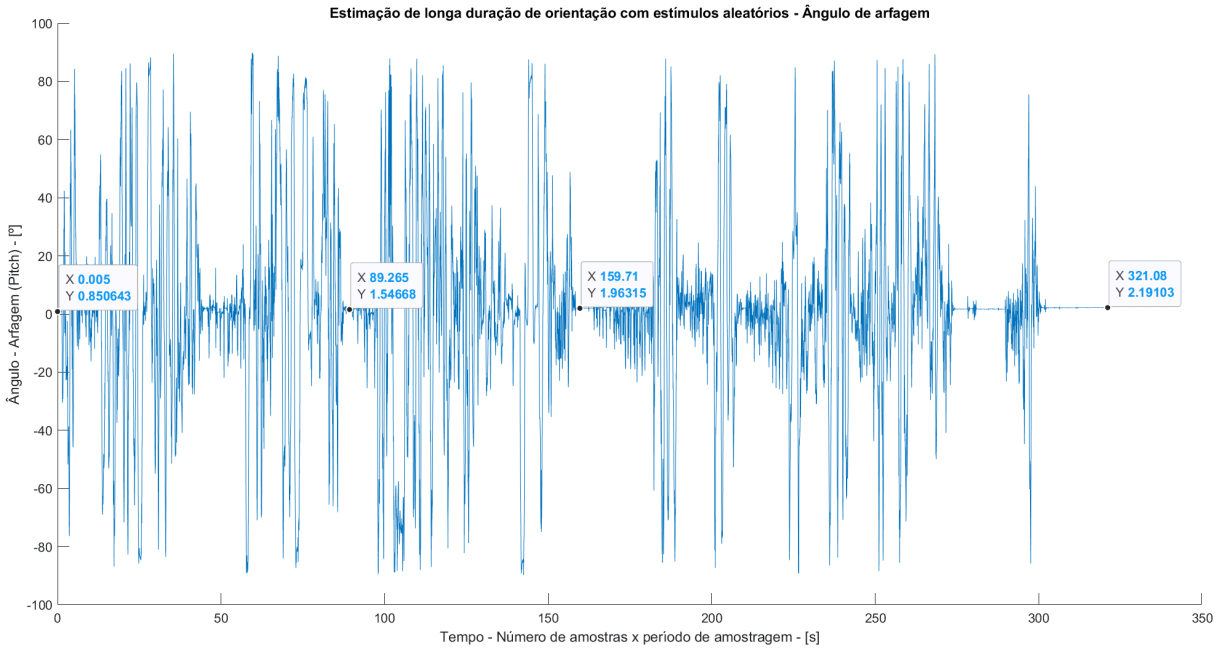
Algo similar pode ser observado para a matriz de covariância do ruído associado ao processo, e para as condições iniciais fornecidas ao processo. Todas essas características apresentadas tem grande impacto na dinâmica durante o processo de estimação.

Com o resultados apresentados se estabelece então a importância de realizar o processo de *tuning* dos parâmetros do filtro mesmo que promovendo alguma sub-otimalidade para o filtro, até mesmo porque tal condição já é inerente ao sistema tendo em vista que os modelos jamais se adéquam perfeitamente a realidade observada.

4.2.1.4 Testes - Avaliação do considerando movimentos gerais aleatórios por longa duração

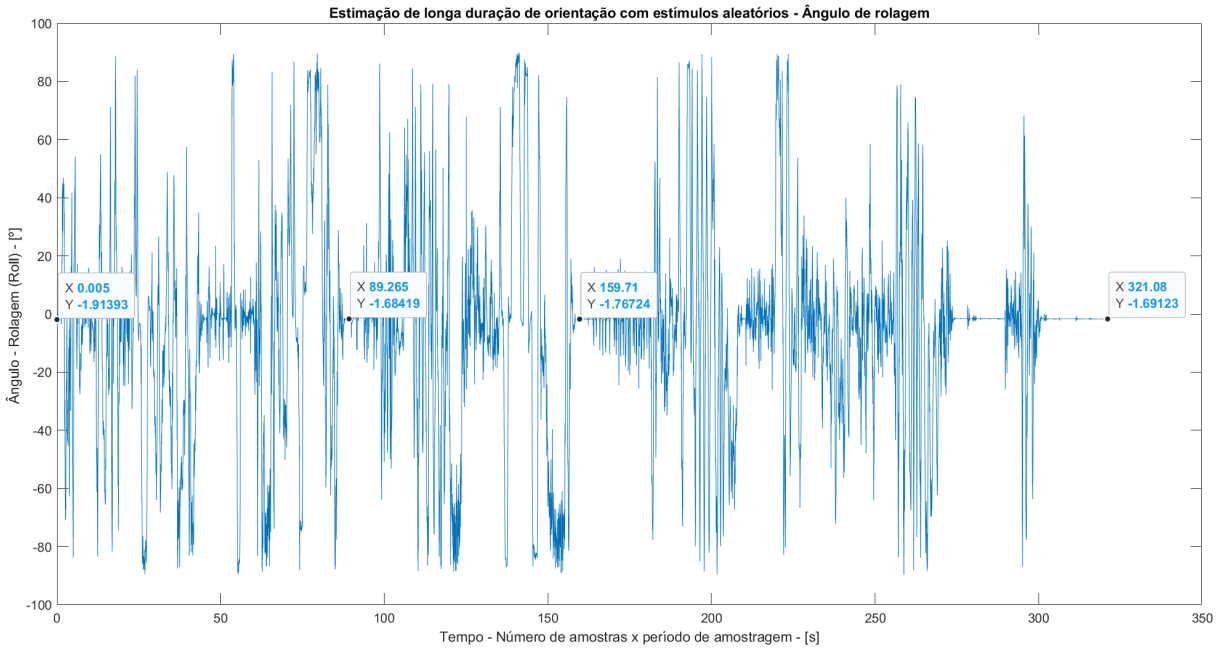
Para concluir os testes para orientação e levantar os últimos pontos foi realizado um teste de longa duração com estímulos aleatórios. O teste é apresentado nas Figuras 149, 150 e 151. O teste tem duração de 5,35 minutos e foi realizado utilizando o filtro de Kalman com formulação de Joseph.

Figura 149 – Teste - Estimação de longo prazo com estímulos aleatórios para ângulo de guinada



Fonte: Produzido pelos autores.

Figura 150 – Teste - Estimação de longo prazo com estímulos aleatórios para ângulo de arfagem



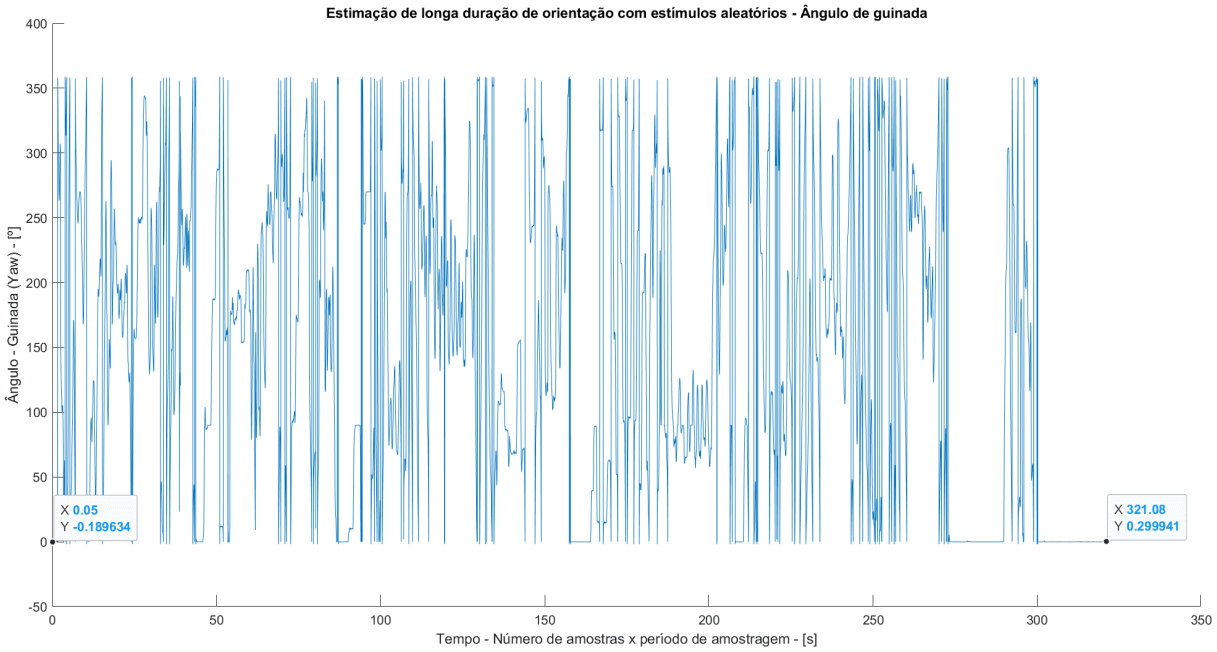
Fonte: Produzido pelos autores.

Note que apesar de todas as questões levantadas durante essa seção, na prática nem sempre o que se observa pode ser reflexo daqueles testes mais mais simples e controlados, por exemplo, note nas marcações nas figuras que nem sempre ocorre uma evolução significativa do erro no processo, por vezes, a depender dos movimentos descritos pode ser que o erro tenha uma tendência a acumular em direção contrária ao erro já presente no sistema, o que leva a uma redução efetiva do erro de uma forma geral, o que gera maior estabilidade para

o processo, permitindo que o mesmo possa continuar de por mais tempo e com precisão relativamente satisfatória.

Além disso, movimentações um pouco mais complexas geralmente envolvem estímulos diversos, dessa forma o comportamento dinâmico do processo de estimação pode ser bastante diferente considerando, por exemplo, aspectos de velocidade de convergência e quão estável é a solução.

Figura 151 – Teste - Estimação de longo prazo com estímulos aleatórios para ângulo de guinada



Fonte: Produzido pelos autores.

Outro fator importante a ser citado é o comportamento da estimação do ângulo de guinada no espaço, note que como a calibração do sensor foi realizada apenas no plano desconsiderando diversos outros fatores, uma vez que movimentos sejam realizados no espaço e não mais no plano, ou seja, sejam descrito movimentos que promovam alteração nos ângulos de rolagem e arfagem, o magnetômetro perde sua referência, enquanto o processo de integração do filtro de Kalman continua, dessa forma duas informações conflitantes são observadas, levando a uma maior instabilidade, e também a valores completamente incorretos para o ângulo de guinada, porém, é possível retornar ao processo de estimação uma vez que a plataforma volte ao plano de referência, o que pode ser observado ao fim do processo na Figura 151.

4.2.1.5 Considerações finais - Estimação da orientação

Por fim, é relevante citar que os algoritmos utilizados já incorporam grande parte dos aspectos abordados na subseção 3.4.7 para promover ganhos de desempenho no processo,

mais especificamente no que tange ao uso de configurações adequadas e da montagem da plataforma.

Devido a uma abordagem indevida no processo de desenvolvimento não foram obtidos dados para os resultados durantes as etapas inicias do processo para que a análise de ganho de desempenho pudesse ser realizada com uma referência bem estabelecida, dessa forma, esses pontos serão abordados brevemente de forma qualitativa de acordo com aquilo que se observou durante os testes.

Diminuir a sensibilidade do dispositivo e utilizar uma banda mais restrita para o filtro passa-baixas digital a priori não levou a ganhos significativos no resultado, porém, utilizando as novas configurações e conduzindo um novo processo de calibração foi possível observar uma redução significativa no ruído.

No caso do acelerômetro foi possível promover uma redução por um fator de aproximadamente 10, já no caso do giroscópio a redução foi bastante insignificante, o que já era esperado tendo em vista que esse apresenta um comportamento relativamente estável.

A montagem do dispositivo na PCB desenvolvida foi a medida que promoveu o maior ganho de precisão e estabilidade, nesse caso o ganho observado foi de pouco menos de 1° de precisão.

Uma montagem mais robusta permitiu um melhor alinhamento dos sensores garantindo um menor desvio da posição inicial definida, além disso, também garantiu que os sensores ficassem expostos a menos estímulos indevidos que, em conjunto com as conexões elétricas mais robustas, permitiu uma redução significativa das oscilações observadas.

Em relação a taxa de amostragem, observou-se que quanto menor a taxa de amostragem menor a precisão do processo de estimação, o que é normal tendo em vista que os algoritmos ficam sem nenhuma informação para realizar correções durante todo o período em que os sensores não realizam uma nova amostragem, nesse caso essa questão já foi apresentada e pode ser verificada nas Figuras 137, 138 e 139.

Considerando um aumento na taxa de amostragem, verificou-se que não necessariamente uma taxa de amostragem mais alta leva a resultados mais adequados, em alguns casos o contrário é verdade, tal colocação a priori aparenta ser contra-intuitiva, porém, dois fatores são bastante importantes nesse caso, primeiro, uma taxa de amostragem muito elevada promove a capacidade de que variações pontuais de frequência e magnitude relativamente elevadas possam ser percebidas de forma mais recorrente, segundo, taxas muitos elevadas podem levar ao *overflow* na fila de dados, o que resultará na reinicialização da estrutura, o que leva bastante tempo, além de eliminar todos os dados capturados e, caso tal condição seja observada constantemente, o processo de estimação se torna inviável. Considerando a implementação em questão, foi observado na prática que com taxas de amostragem próximas a 1kHz problemas de instabilidade já são observados.

No total as medidas apresentadas levaram a um ganho aproximado de 1.3° na precisão.

Por fim, considera-se a aplicação dos algoritmos de média móvel, detecção de limiar, e *RTS Smoother*. Para o caso da estimação da orientação a aplicação da média móvel e do limiar degradaram o desempenho de forma significativa, para a estimação da posição a aplicação das duas medidas foi essencial, tal questão é abordada na subseção 4.2.2.

Para estimação da orientação utilizar a média móvel promoveu a suavização da curva de estimativas durante a estimação, porém, promoveu uma redução na precisão.

Esses resultados podem ser ser explicados devido ao comportamento já estável do giroscópio no curto prazo. Aplicar a média móvel que tem a proposta de reduzir a influência de variações no curto prazo para focar em tendências de longo prazo para os dados tem como único efeito diminuir a velocidade média observada durante os movimentos realizados garantindo então que um movimento diferente daquele observado na prática seja estimado. Quanto maior for a janela de dados utilizada para obtenção da média, mais grave se torna o problema.

No caso do uso do limiar, foram observadas oscilações no processo devido a eliminação de dados durante movimentações válidas do dispositivo.

Para concluir essa subseção considera-se então o uso do suavizador *RTS Smoother*. Não foi possível utilizar o algoritmo para testes devido ao fato de não ter sido possível fazer com que o algoritmo convergisse de nenhuma forma, diversas medidas foram aplicadas, entre elas os aprimoramentos e algoritmos já citados nessa subseção, alteração da estrutura da implementação e, por fim, tentando realizar o *tuning* dos parâmetros do filtro de Kalman, porém, em todos os casos poucos instante de amostragem após o início do processo ocorre a divergência tal que o valor estimado tende ao infinito.

O código final para implementação que não apresentou resultados é apresentado na Figura 152. O código tem como base de funcionamento capturar os dados de uma instância comum do filtro de Kalman, armazenar as informações necessárias para a suavização e, quando todos os dados necessários para realizar a suavização de acordo com a janela de dados definidas forem capturados, o processo de suavização é iniciado e permanece sendo realizado de forma contínua.

Figura 152 – Código - *RTS Smoother*

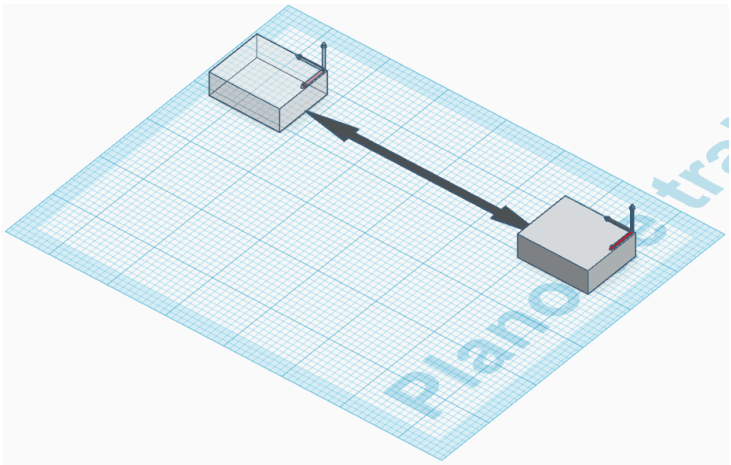
```
//Function: rts_smooth_init(...);
//Description: RTS smoother initialization
//Arg: rtss - rst_smoother instance pointer, Xo, Po - initial state and covariance matrix
void rts_smooth_init(rts_smoother* rtss, matrix Xo, matrix Po){
    rtss->w_size_reached = 0;
    for(int k = 0; k < W_SIZE; k++){
        Xu_s[k] = Xp_s[k] = null_mat(Xo.size[0], Xo.size[1]);
        Pu_s[k] = Pp_s[k] = null_mat(Po.size[0], Po.size[1]);
        Y_s[k] = null_mat(1, 1); //(*)
    }
}

//Function: rts_smooth(...);
//Description: Run RTS smoother algorithm associated with a single Kalman filter instance.
//Arg: rtss - RTS smoother instance pointer, kfilter - Kalman filter instance pointer associated with smoother instance.
static inline __attribute__((always_inline)) void rts_smooth(rts_smoother* rtss, kfilter_t* kfilter){
    matrix Cs[W_SIZE];
    matrix Xs[W_SIZE];
    //matrix Ps[W_SIZE+1]; //- Shouldn't be used if you don't really need covariance matrix information. (*)
    Xs[W_SIZE - 1] = kfilter->Xu[0];
    //Ps[W_SIZE] = kfilter->Pu[0]; - (*)
    matrix At = transpose(*kfilter->A);
    if(rtss->w_size_reached == W_SIZE - 2){
        for(int k = W_SIZE - 2; k >= 0; k--){
            Cs[k] =  mult(mult(Pu_s[k], At), inv(Pp_s[k]));
            Xs[k] =  sum(Xu_s[k], mult(Cs[k], sub(Xs[k + 1], Xp_s[k])));
            //Ps[k] = sum(Pu[k], mult(C[k], sub(Ps[k+1] - Pp_s[k]))); - (*)
        }
        rtss->w_size_reached = 0;
    }
    else{
        Xu_s[rtss->w_size_reached] = kfilter->Xu[0];
        Pu_s[rtss->w_size_reached] = kfilter->Pu[0];
        Xp_s[rtss->w_size_reached] = kfilter->Xp[0];
        Pp_s[rtss->w_size_reached] = kfilter->Pp[0];
        rtss->w_size_reached += 1;
    }
}
```

Fonte: Produzido pelos autores.

4.2.2 Resultados - Estimação da posição

Para realizar os testes de estimação de posição para os eixos de referência x e y foram utilizados 38cm ou 0,38m da base de corte, esse valor essencialmente representa parte da dimensão livre após a colocar a plataforma sobre a base de forma que ficasse alinhada com a referência e ainda mantivesse o apoio necessário. A Figura 153 apresenta uma ilustração do procedimento.

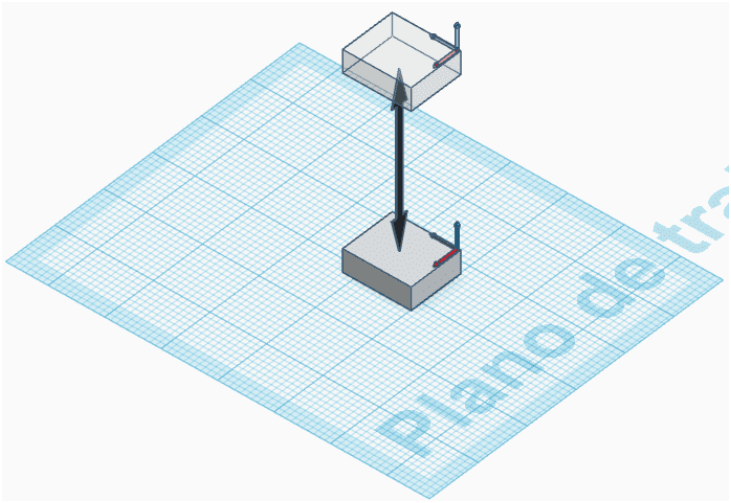
Figura 153 – Testes - Estimação da posição para referências x e y



Fonte: Produzido pelos autores.

De forma similar foram utilizados 30cm ou 0,3m para referência em z. Para realização desse teste foi necessário apoiar a plataforma ao lado de uma superfície rígida com demarcações. A Figura 154 apresenta uma ilustração do procedimento.

Figura 154 – Testes - Estimação da posição para referência z



Fonte: Produzido pelos autores.

Em todos os casos o teste consiste em um movimento retilíneo da posição inicial, arbitrariamente definida e igual para todos os casos, até a posição de referência estabelecida, após chegar na posição de referência, aguarda-se cerca de 1 segundo e retorna-se a posição inicial pelo mesmo caminho.

Note que o teste é extremamente simples pois representa um movimento unidimensional de curta duração. Essa abordagem foi uma decisão de projeto realizada após dezenas de testes em que diversas configurações e algoritmos foram utilizados, em todos os casos foram observados erros bastante grosseiros. Devido ao número de curvas necessárias e também do fato de cada uma apresentar suas peculiaridades, uma análise adequada dos dados não seria possível. Dessa forma optou-se por testes mais simples para que os problemas pudessem ser observados apropriadamente, assim como as diferenças entre as abordagens, e para que os resultados pudessem ser de fácil compreensão.

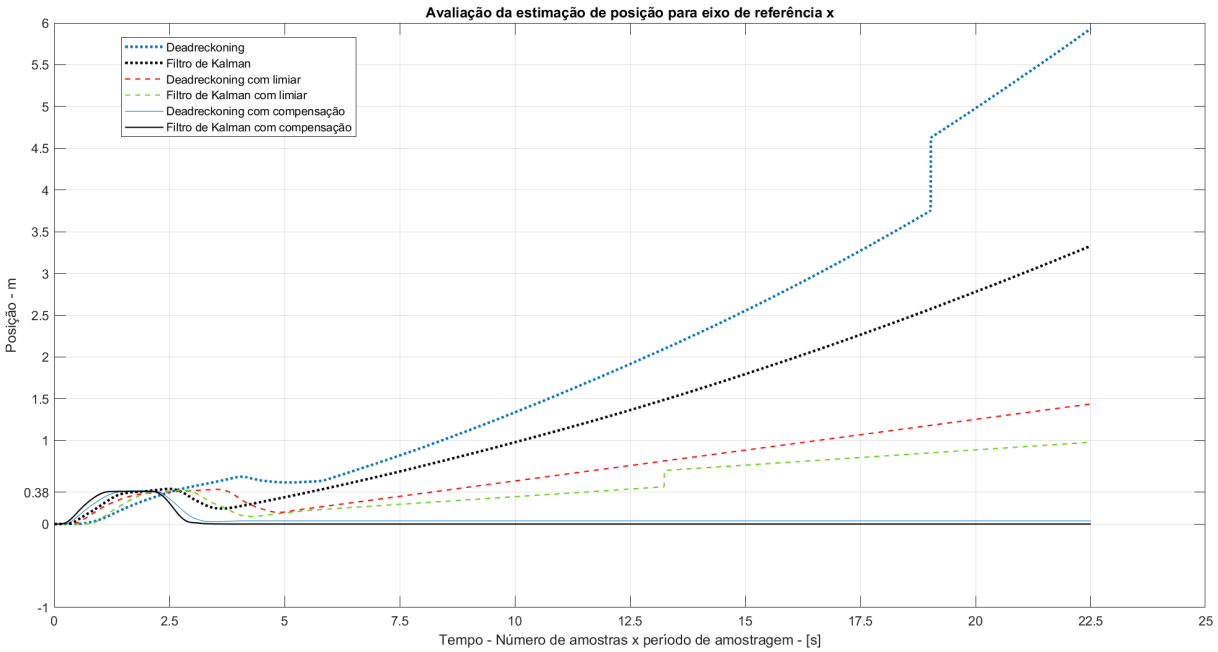
Assim como foi no caso da orientação, a movimentação da plataforma foi realizada de forma manual, dessa forma, existem erros não-caracterizados associados ao processo, o que pode levar a resultados diferentes para cada novo teste. Também conforme os procedimentos adotados no caso dos testes para orientação para mitigar os problemas, os testes foram repetidos diversas vezes para tentar reduzir as discrepâncias nos tempos de movimentação e também testes que apresentassem algum problema grosseiro durante sua realização eram interrompidos.

As Figuras 155, 156 e 157, apresentam os resultados para os testes realizados para estimação da posição para os eixos de referência x, y e z respectivamente. Em todos os casos

foram utilizados 2 algoritmos, sendo esses o *dead reckoning* e o filtro de Kalman, e 3 variações, comum, com limiar, e com compensação.

Os algoritmos aqui utilizados são aqueles definidos na seção 3.4, a variação denominada '- com limiar' se refere a introdução do algoritmo para detecção de dados válidos advindos do acelerômetro através do uso de um limiar pré-definido no processo de estimação, algoritmo esse definido na subseção 3.4.8.2, por fim, a variação '- com compensação' se refere a um procedimento específico a ser abordado posteriormente.

Figura 155 – Teste - Estimação da posição em curto prazo para eixo de referência x



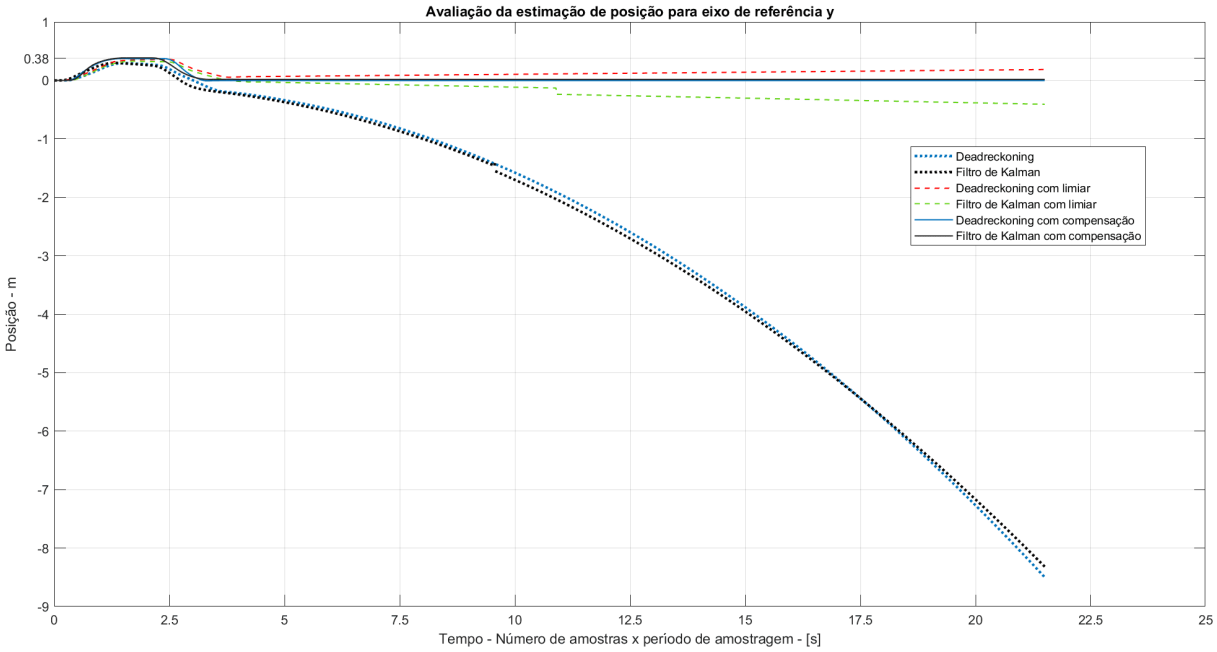
Fonte: Produzido pelos autores.

Tabela 15 – Dados de estimação em curto prazo para posição em relação ao eixo de referência x

Algoritmo	Referência (m)	Média (m)	RMSE (m)	Erro (%)	Erro - Final (m)
Dead reckoning	0.38	0.3883	0.1227	-2.17	-5.931
Filtro de Kalman	0.38	0.3926	0.0245	-3.32	-3.33
Dead reckoning com limiar	0.38	0.3905	0.0197	-2.76	-1.438
Filtro de Kalman com limiar	0.38	0.3863	0.017	-1.67	-0.979
Dead reckoning com compensação	0.38	0.389	0.0157	-2.36	-0.0383
Filtro de Kalman com compensação	0.38	0.3893	0.0097	-2.46	0.00067

Fonte: Produzido pelos autores.

Figura 156 – Teste - Estimação da posição em curto prazo para eixo de referência y



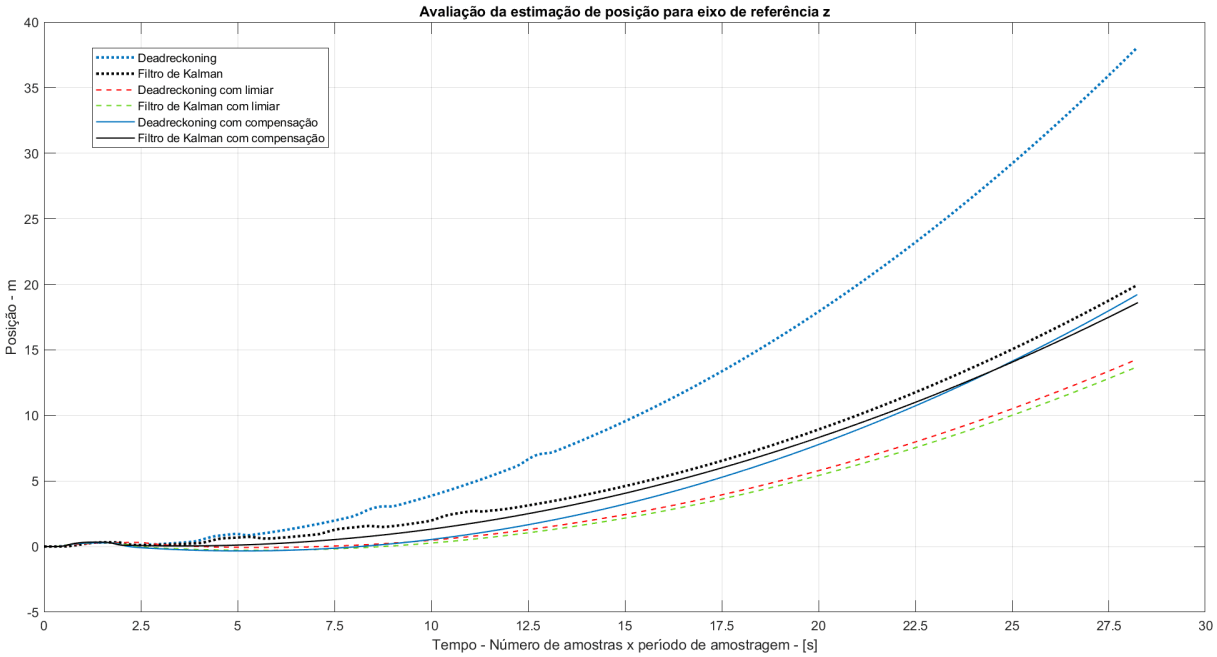
Fonte: Produzido pelos autores.

Tabela 16 – Dados de estimação em curto prazo para posição em relação ao eixo de referência y

Algoritmo	Referência (m)	Média (m)	RMSE (m)	Erro (%)	Erro - Final (m)
Dead reckoning	0.38	0.2805	0.1005	26.19	8.498
Filtro de Kalman	0.38	0.2786	0.103	26.68	8.318
Dead reckoning com limiar	0.38	0.3513	0.0299	7.56	-0.1863
Filtro de Kalman com limiar	0.38	0.3226	0.0577	25.09	0.4079
Dead reckoning com compensação	0.38	0.3712	0.0091	2.33	0.0023
Filtro de Kalman com compensação	0.38	0.3819	0.0061	-0.51	-0.017

Fonte: Produzido pelos autores.

Figura 157 – Teste - Estimação da posição em curto prazo para eixo de referência z



Fonte: Produzido pelos autores.

Tabela 17 – Dados de estimação em curto prazo para posição em relação ao eixo de referência z

Algoritmo	Referência (m)	Média (m)	RMSE (m)	Erro (%)	Erro - Final (m)
Dead reckoning	0.3	0.3147	0.0237	-4.89	-39.0093
Filtro de Kalman	0.3	0.3098	0.0119	-3.26	-19.879
Dead reckoning com limiar	0.3	0.2982	0.0147	0.6	-14.2846
Filtro de Kalman com limiar	0.3	0.275	0.0286	8.32	-13.6978
Dead reckoning com compensação	0.3	0.24706	0.0341	9.8	-19.2167
Filtro de Kalman com compensação	0.3	0.3289	0.0317	-9.64	-18.6094

Fonte: Produzido pelos autores.

Inicialmente, utilizando o modelo expresso pelas Equações 3.33, 3.34, e 3.35 foram feitos os testes iniciais. Com esses testes foi possível observar um tendência de divergência acentuada dos dados durante o processo de estimação, além disso, o sistema não conseguia responder a nenhum tipo de estímulo externo.

A priori, cogitou-se as seguintes possibilidades: primeiro, a intensidade do ruído era muito maior do que a aceitável levando a forte tendência de divergência e incapacidade de resposta, principalmente tendo em vista que os estímulos aplicados para acelerar o dispositivos não são de alta magnitude, segundo, o modelo utilizado possuía algum erro grosseiro ou parâmetros inadequados, por fim, o modelo proposto não era adequado ao processo, sendo esse dois últimos fatores bastante comuns quando a divergência ou a não-convergência é observada.

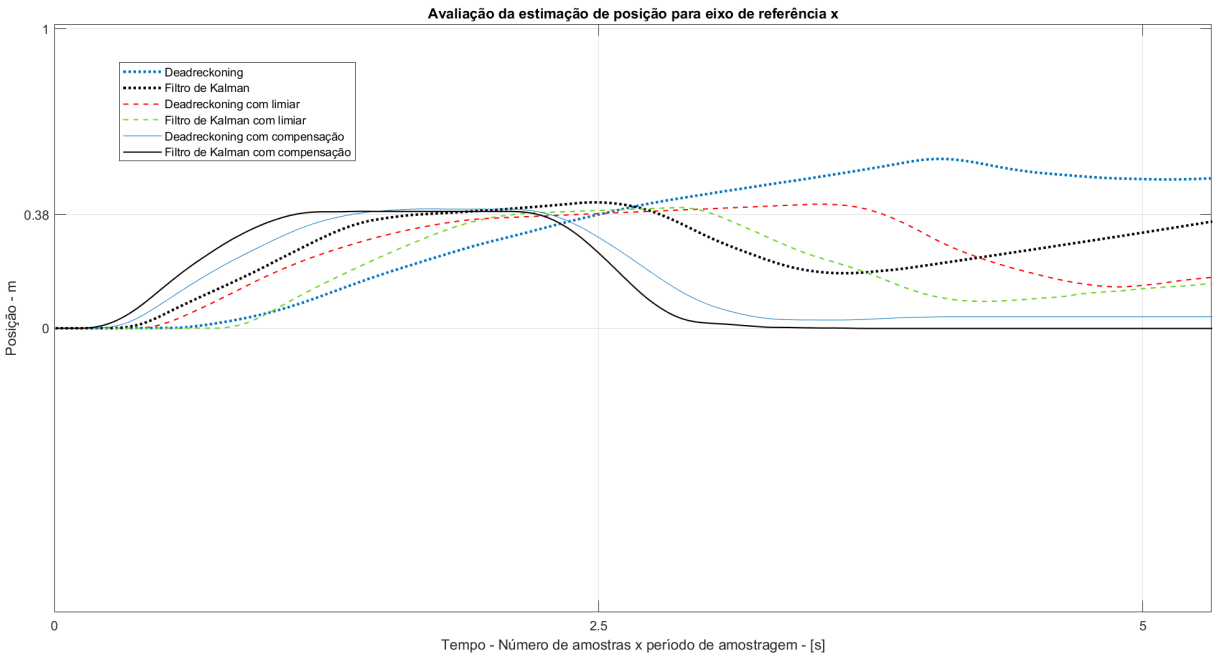
Para explorar a primeira alternativa foram adotadas as medidas apresentadas nas seções 3.4.7 e 3.4.8, por exemplo, diminuir a sensibilidade dos sensores, utilizar bandas mais restritas para o filtro passa-baixas digital, utilizar uma faixa de amostragem mais adequada e, por fim, incorporar os sensores a plataforma por meio da PCB desenvolvida.

Utilizando todos as correções e com os parâmetros especificados nas Tabelas 3, 4 e 5 obteve-se então os resultados expressos nas curvas denominadas 'Deadreckoning' e 'Filtro de Kalman', conforme as Figuras 155, 156 e 157.

Em todos os casos é possível observar erros expressivos. Note que em apenas 22,5s, 21,5, e 28,2 de estimação para x, y, e z, respectivamente, é observado um erro acumulado de -5,931m e -3,33m para x, 8,498m e 8,318m para y, e -39,009m e -19,879m para z para os algoritmos *dead reckoning* e filtro de Kalman, conforme as Tabelas 15, 16 e 17.

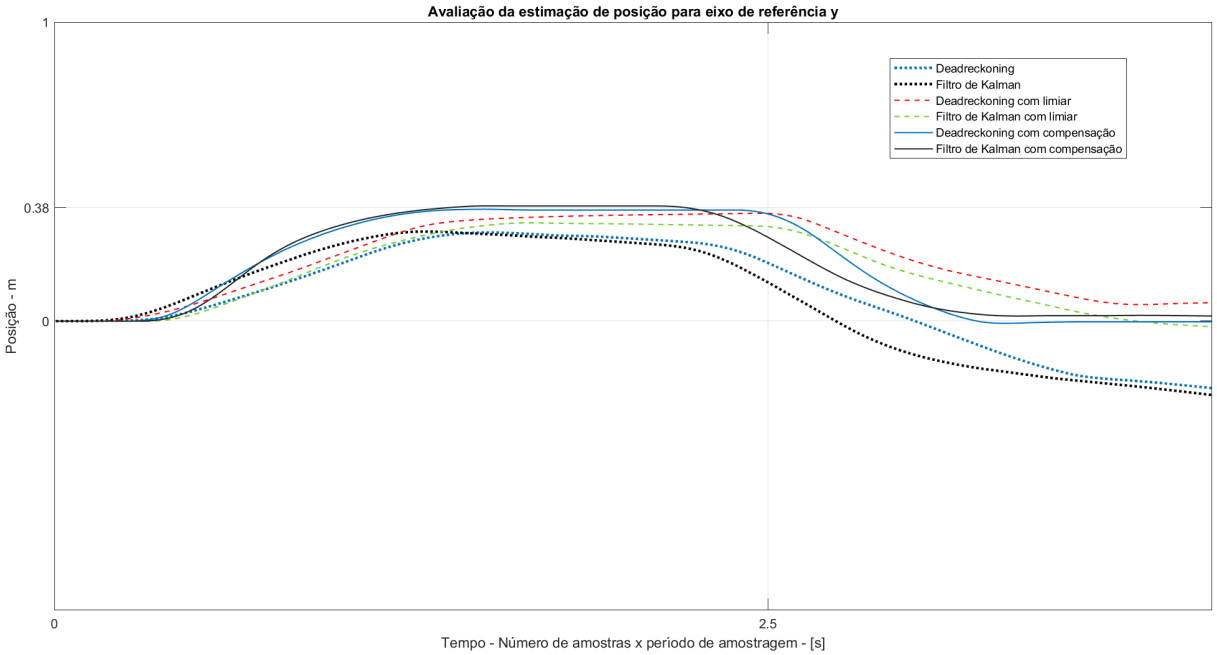
Além dos erros observados, nas Figuras 158, 159, e 160 é possível perceber que o sistema passou a responder aos estímulos introduzidos, o que indica que a hipótese inicial de que o sistema não respondia devido a quantidade de ruído parece ser válida, porém, a hipótese de que leva a divergência não se sustenta, tendo em vista que ainda existe forte tendência de desvio conforme apresentado nas Figuras 155, 156 e 157.

Figura 158 – Teste - Estimação da posição em curto prazo para eixo de referência x - Comportamento durante estímulo



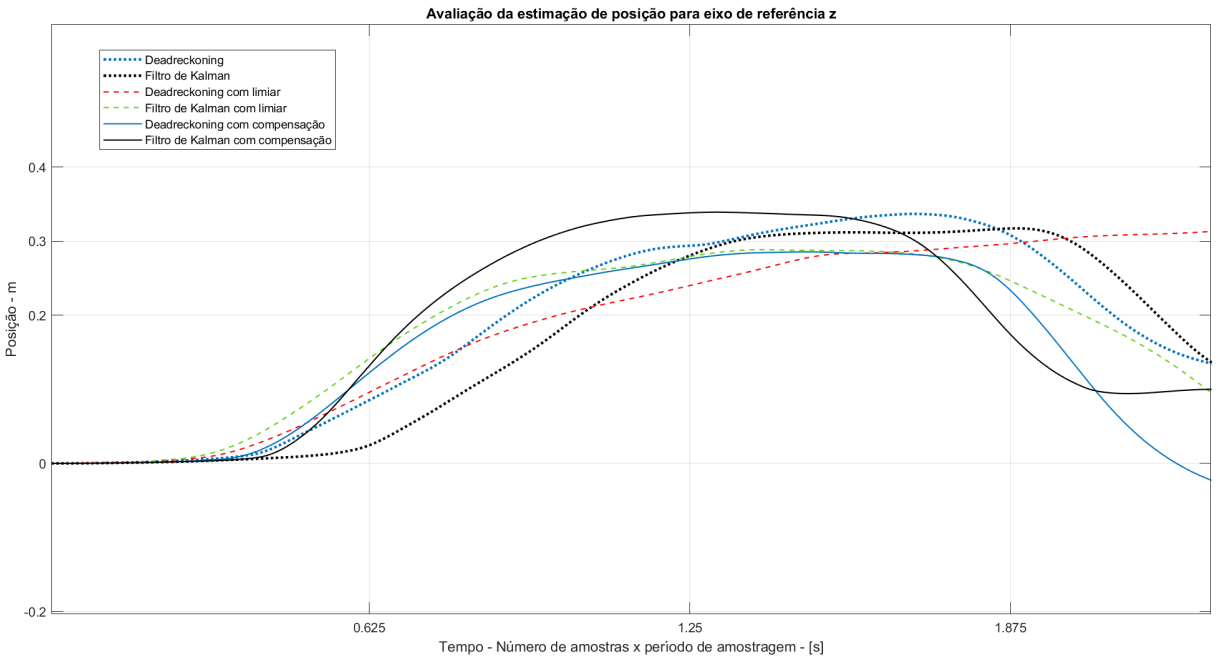
Fonte: Produzido pelos autores.

Figura 159 – Teste - Estimação da posição em curto prazo para eixo de referência y - Comportamento durante estímulo



Fonte: Produzido pelos autores.

Figura 160 – Teste - Estimação da posição em curto prazo para eixo de referência z - Comportamento durante estímulo



Fonte: Produzido pelos autores.

Para avaliar o comportamento do algoritmo do filtro de Kalman, foram feitas alterações na modelagem e parâmetros de forma a avaliar, entre essas mudanças destacam-se a alteração do vetor de estado X_o e da matriz de covariância do erro, P_o, inicialmente fornecidos ao sistema. Considerou-se ainda a alteração na matriz de covariância do ruído associado ao

processo, Q, e na matriz de covariância do ruído associado as medições, R, que por sua vez também levarão a alterações da matriz de covariância do erro e do ganho do filtro durante o processo, sendo esses os fatores essenciais para definir o comportamento do processo de estimação.

Ao se alterar as condições iniciais para o estado do sistema não se observou grandes diferenças de resposta, apenas pequena alteração no tempo de resposta e precisão, o que é condizente com o esperado tendo em vista que essa informação geralmente representa a melhor estimativa que temos antes de iniciar o processo de estimação. Como trabalhamos com um referencial relativo, a estimativa começa sempre exatamente no ponto inicial e, portanto, qualquer alteração nesse parâmetro leva apenas a um pequeno desvio do valor real.

Alterar a estimativa inicial para a matriz de covariância do erro por outro lado é algo bem mais crítico, esse estímulo inicial é responsável por controlar essencialmente todo o comportamento futuro da estimação, claro, considerando um mesmo modelo dinâmico. Alterar esse parâmetro, em conjunto com outros fatores como, por exemplo, as matrizes Q e R citadas, controla a não-convergência, divergência, velocidade de convergência e quão preciso e estável é o resultado da estimação, conforme já comentado na seção anterior. Para realizar esses testes os valores de σ_p^2 em Q foram mantidos constantes e os de σ_a^2 em R foram alterados, com valores entre $\sigma_a^2 = 0.005$ e $\sigma_a^2 = 5$. Algo similar foi feito para os valores de σ_p^2. Para P_o foram introduzidos valores relativamente pequenos de mesma magnitude que os apresentados anteriormente, porém, foram variados de forma aleatória dentro do intervalo.

Realizando os testes para esses novos parâmetros e com o resultado da estimação utilizando o algoritmo *dead reckoning*, observou-se que o problema muito possivelmente não estava no modelo utilizado, mas sim no processo.

Como ambas as alternativas apresentadas inicialmente como possível causa do problema não se provaram efetivas para garantir a convergência da solução, apenas para que o sistema respondesse aos estímulos, antes de partir para a última alternativa, ou seja, uma reformulação mais complexa do problema, optou-se por desenvolver algoritmos ou soluções mais simples capazes de mitigar o problema.

Dentre esses algoritmos se destaca o algoritmo apresentado na subseção 3.4.8.2. Conforme abordado na seção em que foi introduzido, esse algoritmo é capaz de estabelecer um limiar a ser superado para que a detecção de movimento ocorra, levando a uma maior estabilidade.

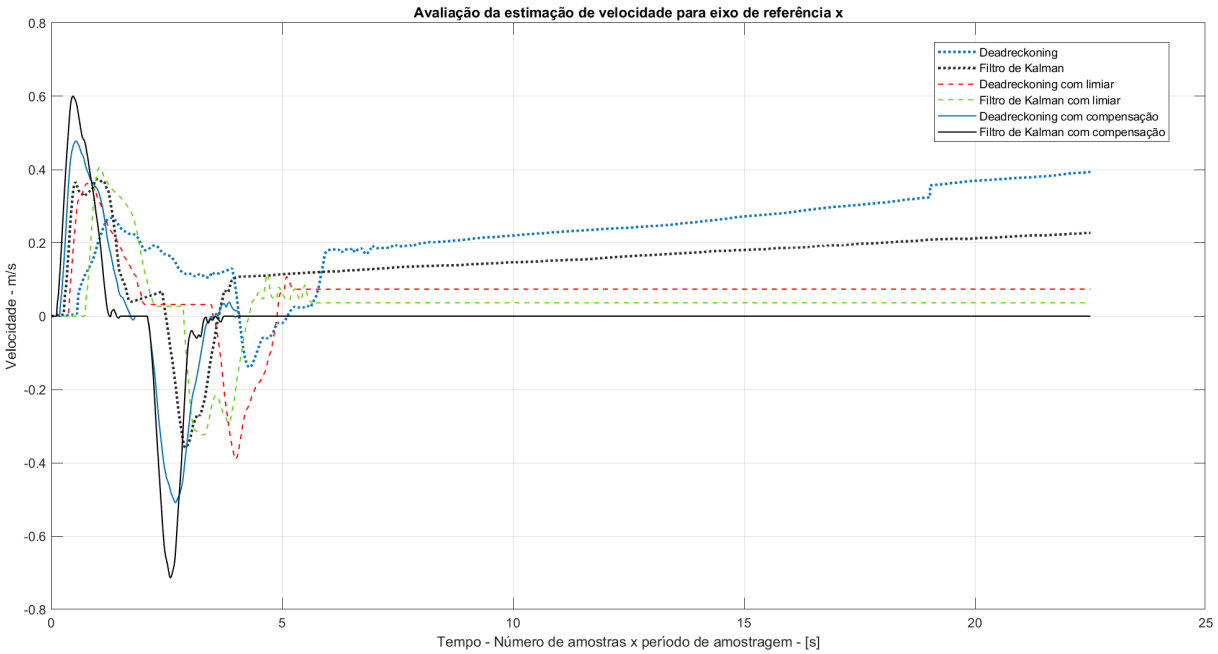
Nas Figuras 155, 156 e 157 é possível observar o resultado do uso desse algoritmo nas curvas denominadas 'Deadreckoning com limiar' e 'Filtro de Kalman com limiar' para os eixos x, y e z respectivamente.

Note que ao introduzir esse algoritmo foi possível reduzir de forma significativa o

erro observado, de -5,931m para -1,438m em x, de 8,498m para -0,1863m em y, e de -39,009m para -14,285m em z, no caso do 'Deadreckoning com limiar'. No caso do 'Filtro de Kalman com limiar' algo similar é observado, redução de -3,33m para -0,979m em x, de 8,138m para 0,4079m em y, e de -19,879m para -13,6978m em z, conforme as Tabelas 15, 16 e 17.

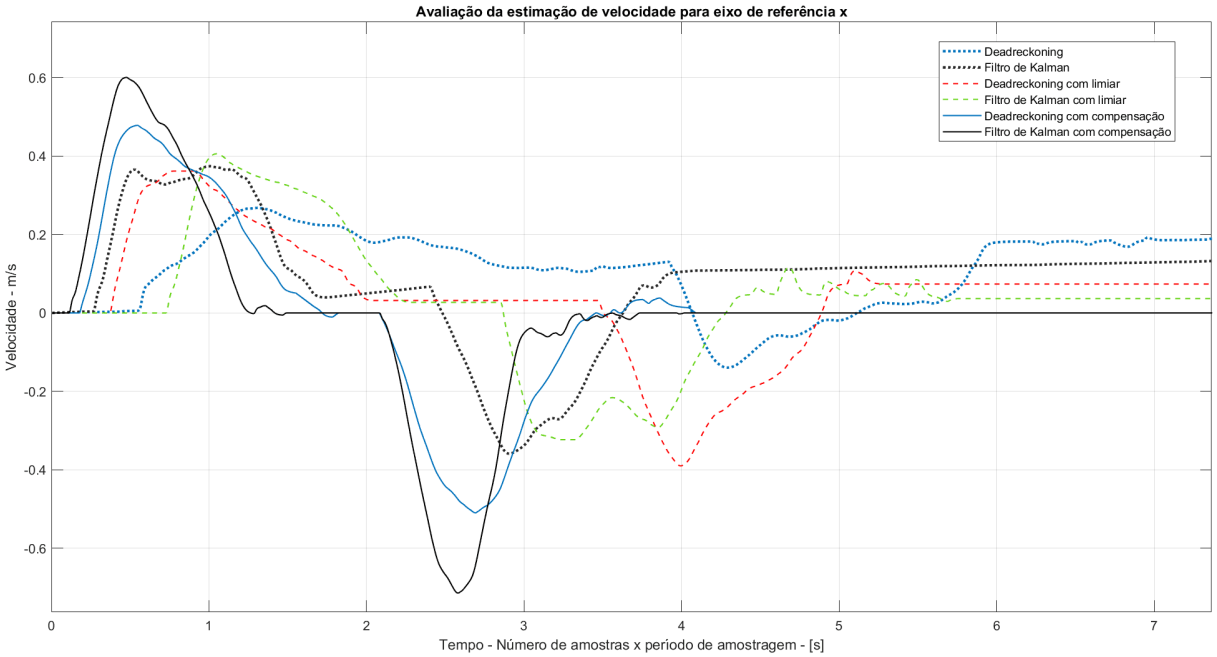
Com isso verificou-se que a condição de desaceleração a qual a plataforma é submetida no fim do movimento não leva a a uma velocidade significativamente baixa como esperado e, portanto, o algoritmo continua a operar com uma velocidade com leve variação ao longo do tempo no caso do *dead reckoning* e do filtro de Kalman, e com velocidade constante para os casos em que se introduz o limiar, conforme pode ser observado na Figura 161 para as curvas respectivas, os dados para estimação da velocidade em y e z foram omitidos. Note que esses dados são relativos ao teste expresso na Figura 155 para estimação da posição. Na Figura 162 pode ser observado o comportamento da estimação da velocidade durante os instantes em que o sistema sofreu excitação.

Figura 161 – Teste - Estimação da velocidade em curto prazo para eixo de referência x



Fonte: Produzido pelos autores.

Figura 162 – Teste - Estimação da velocidade em curto prazo para eixo de referência x - Comportamento durante estímulo



Fonte: Produzido pelos autores.

As Tabelas 18, 19 e 20 indicam os erros finais relacionado a estimação da velocidade para cada um dos eixos de referência para os casos citados.

Note que para os eixos de referência x e y o procedimento se provou bastante proveitoso tendo em vista que a redução do erro foi bastante significativa, por exemplo, considere o caso do *dead reckoning*, uma redução de -0.393m/s para -0.073m/s em relação ao eixo de referência x é observada, algo similar é observado em relação a y, de 0.8526m/s para -0.007m/s.

Em relação a z o erro final observado ainda foi bastante significativo, indicando que o limiar estabelecido é insuficiente e o processo está sujeito a maiores interferências nesse caso.

Tabela 18 – Dados de estimação em curto prazo para velocidade - Eixo de referência x

Algoritmo	Referência (m/s)	Erro - Final (m/s)
Dead reckoning	0	-0.393
Filtro de Kalman	0	-0.227
Dead reckoning com limiar	0	-0.073
Filtro de Kalman com limiar	0	-0.036
Dead reckoning com compensação	0	0
Filtro de Kalman com compensação	0	0

Fonte: Produzido pelos autores.

Tabela 19 – Dados de estimação em curto prazo para velocidade - Eixo de referência y

Algoritmo	Referência (m/s)	Erro - Final (m/s)
Dead reckoning	0	0.8526
Filtro de Kalman	0	0.796
Dead reckoning com limiar	0	-0.007
Filtro de Kalman com limiar	0	0.016
Dead reckoning com compensação	0	0
Filtro de Kalman com compensação	0	0

Fonte: Produzido pelos autores.

Tabela 20 – Dados de estimação em curto prazo para velocidade - Eixo de referência z

Algoritmo	Referência (m/s)	Erro - Final (m/s)
Dead reckoning	0	-2.921
Filtro de Kalman	0	-1.6328
Dead reckoning com limiar	0	-1.2628
Filtro de Kalman com limiar	0	-1.2329
Dead reckoning com compensação	0	-1.693
Filtro de Kalman com compensação	0	-1.49

Fonte: Produzido pelos autores.

Note que com uma velocidade constante e com magnitude relativamente elevada a estimação definitivamente não seria possível tendo em vista que o sistema tenderia a continuar evoluindo no mínimo a uma taxa constante considerando o caso em que se introduz o limiar.

Com isso numa ultima tentativa de tentar verificar a possibilidade de resolver os problemas no processo e levantar possíveis alternativas para que a estimação da posição fosse realizada, foi introduzido um algoritmo de compensação capaz de detectar aproximadamente condições de desaceleração da plataforma.

Esse algoritmo foi implementado de forma bastante grosseira e sem otimizações e tem como principal foco abordar o problema e mostrar que existem medidas que podem ser tomadas para garantir o funcionamento do processo de estimação.

Os resultados da aplicação do algoritmo podem ser visualizados nas Figuras 155, 156 e 157 nas curvas respectivas, *dead reckoning* com compensação e filtro de Kalman com compensação.

Note que para os eixos de referência x e y o algoritmo é capaz de reduzir o erro de forma bastante significativa, apresentando o melhor resultado dentre os procedimentos

apresentados, sendo esses erros de -0.0383m e 0,00067m para o eixo x, e 0.0023m e -0.017m para o eixo y, conforme Tabelas 15 e 16.

Observe na Figura 157 que os algoritmos com compensação apresentaram resultados piores que aqueles obtidos com a aplicação do limiar simples para o eixo de referência z, com erros de -19.2167m e -18.6094m, conforme Tabela 17. Atribui-se a esse resultado possivelmente o fato de que esse eixo de referência é mais suscetível ao erros, inclusive de estímulos em outros eixos, além disso, nesse caso observa-se um estímulo constante advindo da ação da aceleração da gravidade, por fim, note que em nenhum momento o limiar estabelecido consegue atuar, dessa forma, grande partes dos estímulos são considerados válidos, o que não é adequado, porém, existe ainda mitigação dos erros observados no processo.

Por fim, é possível notar nas Figuras 158, 159, e 160 que, pesar de apresentarem na média valores próximos, a resposta dinâmica de cada algoritmo é bastante diferente. A aplicação direta do *dead reckoning* e do filtro de Kalman leva a um comportamento indesejável durante o processo de estimação, porém, com a aplicação das variações é possível ver um comportamento bem mais adequado e precisão relativamente satisfatória.

No caso de y, é possível notar, conforme Tabela 16, que a média evolui de 0,2805m para 0,3819m ao incorporar as correções apresentadas, enquanto o erro quadrático médio diminui de 0.1005m para 0.0061m condizente com os resultados observados. Para x os resultados estão bem próximos a média para todos os casos, com mínimo de 0,3863m e máxima de 0,3629m, conforme Tabela 15, além disso, o erro quadrático médio observado manteve a mesma relação observada para y, com o máximo sendo associado ao algoritmo *dead reckoning* e mínimo associado ao filtro de Kalman com compensação, com valores de 0.1227m e 0.0097m, respectivamente.

Os resultados para eixo de referência z podem ser visualizados na Tabela 17, note que, apesar de alguns algoritmos apresentarem valores que na média aparentam ser adequados, na realidade nenhum deles apresenta um resultado realmente preciso e estável.

4.2.2.1 Considerações finais - Estimação da posição

Com os resultados apresentados é então estabelecido o fato de que não foi possível estimar a posição de forma adequada, considera-se aqui como adequado um algoritmo capaz de realizar um estimação com precisão satisfatória e com evolução temporal aceitável, ou seja, com erros de pequena magnitude para longos períodos de tempo, tal que seja possível realizar a correção ou mitigação desses através da inserção de novas informações, e calibração ou compensação em *runtime*.

Com os dados obtidos, porém, é possível notar que o sistema é sim capaz de realizar a estimação caso as devidas correções sejam introduzidas ao processo.

4.2.3 Resultados - Tempo de execução

Essa seção tem como intuito apresentar brevemente resultados sobre o tempo de execução dos algoritmos apresentados, note que a plataforma não tem capacidade para obter dados relativos ao tempo e, portanto, uma avaliação precisa do desempenho dos algoritmo é impossível nesse caso, dessa forma optou-se por realizar um procedimento para fazer uma análise relativa.

Primeiro, são estabelecidos valores para o número de iterações do código, note que essas iterações não são as iterações dos algoritmos, mas do ciclo de verificação da disponibilidade de dados e posterior análise, caso esses últimos estejam disponíveis. Esses ciclos são repetidos para realização da coleta de dados sem que nenhum algoritmo esteja operando, estabelecendo então um valor de referência para o número de ciclos que podem ser realizados para um número específico de amostras, após isso os diversos algoritmos são inseridos no processo e o processo é repetido, sendo possível então comparar os algoritmos com a referência estabelecida.

Tabela 21 – Tempo estimado para estimação

Algoritmo	Número de iterações para 0.5x10^5 amostras	Tempo estimado (s)
Sem algoritmo	17722	-
Dead reckoning	17725	0.015
Filtro complementar	17767	0.225
Filtro de Kalman - Orientação	18123	2.005
Filtro de Kalman Joseph - Orientação	18233	2.555
Filtro de Kalman - Posição	18599	4.385
Filtro de Kalman Joseph - Posição	18950	6.14

Fonte: Produzido pelos autores.

Na Tabela 21 é importante notar que quanto mais lento o algoritmo mais iterações ele consegue realizar, o que pode parecer contra-intuitivo quando não se conhece o sistema, porém, o que acontece efetivamente é que o atraso promovido pelos algoritmos permite que quando se entre no ciclo de verificação de disponibilidade de dados, os dados estejam disponíveis com uma maior frequência permitindo, consequentemente, que mais iterações possam ser realizadas.

Dessa forma o tempo estimado para realizar todas as iterações consiste no atraso promovido pelo algoritmo mais o tempo para coletar a diferença de amostras, sendo que quanto maior essa diferença menor a precisão do valor estimado.

É possível notar que os tempos são relativamente pequenos para um número de amostras significativos, porém, existe diferença significativa nos tempos de estimação avaliados para cada um dos algoritmos, note que o filtro de Kalman com formulação de Joseph

tem um custo relativamente superior ao filtro de Kalman comum, porém, não tão elevado, portanto, sua aplicação é interessante para o projeto.

Além disso, o tempo para estimação da posição é bem mais significativo que para a orientação, o que já era esperado tendo em vista que é necessário primeiramente obter dados referentes a posição para então realizar a estimação da posição de forma adequada.

5 Conclusões

O objetivo inicial do projeto é pautado na estimação em tempo real da posição e orientação da câmera através do processamento de imagens (odometria visual) e de dados inerciais provenientes de uma IMU (odometria inercial) em uma plataforma híbrida ARM+FPGA, com fusão de ambos os dados (odometria visual e inercial) para aprimorar a acurácia dos resultados do sistema.

A câmera foi integrada à plataforma com êxito, no entanto, devido à complexidade de implementação do algoritmo de odometria visual na plataforma, e limitações, como transferência lenta de dados para a CPU, citada na seção 3.1, optou-se por implementar o algoritmo de odometria visual em uma CPU e integrar a IMU na plataforma minized. Estudos acerca da fusão de dados e da implementação do algoritmo de odometria visual na plataforma foram realizados e podem ser encontrados na seção 3.5 e no Apêndice A, respectivamente.

De forma similar ao apresentado, os sensores inerciais foram integrados à plataforma com êxito, englobando os principais aspectos necessários para fazer uso de praticamente todas as funcionalidades disponíveis no dispositivo, conforme abordado brevemente na subseção 3.3.5.2.

Ao contrário do caso da integração da câmera à plataforma, a integração dos sensores inercias à plataforma pode ser feito de forma mais simples tendo em vista que o processo, apesar de bastante complexo, pode ser realizado utilizando uma abordagem bastante direta. Os sensores inercias trabalham de forma bastante harmoniosa quando operando diretamente na plataforma, podendo utilizar taxas de amostragem relativamente altas para o processo de estimação, e também taxas de transferência extremamente rápidas para realizar a coleta de dados pela UART, tendo em vista que os valores avaliados são essencialmente números de ponto flutuante de precisão simples ou dupla.

Todos os códigos relativos a implementação estão disponíveis em repositório e podem ser encontrados em:

`https://github.com/embedded-computing/visual-inertial-odometry`

Na implementação do algoritmo de odometria visual, foram utilizadas funções compatíveis com a biblioteca da *Xilinx* de forma a facilitar trabalhos futuros de implementação na plataforma. Uma das principais contribuições do algoritmo implementado foi o uso de sequências de *frames* próprios, em vez de *datasets*, pois assim foi possível analisar o comportamento do sistema diante de condições adversas relacionadas à textura do ambiente filmado e à configuração da câmera, além de ser uma etapa necessária para o trabalho futuro

de fusão de dados da câmera e da IMU.

Os resultados da implementação da odometria visual foram divididos em ambiente controlado e não controlado. Os testes em ambiente controlado demonstraram que, para distâncias curtas (unidades de metros) o algoritmo apresenta uma boa precisão em curvas e retas, assim como uma boa repetibilidade. O resultado do algoritmo é prejudicado em cenários de baixíssima textura, já que a quantidade mínima de *features* detectáveis não era atingida na etapa de detecção de *features*, nem mesmo na re-detecção. Além disso, é possível concluir que as funcionalidades automáticas da câmera, como a correção automática de cor, que não pode ser desativada, também exerce influência negativa nos resultados, já que afeta diretamente a intensidade dos *pixels*.

Por outro lado, os testes em ambientes não controlados demonstraram que, para longas distâncias (centenas de metros), o algoritmo apresenta uma precisão satisfatória em trajetórias curva e retilíneas. Visto que há a presença de objetos móveis em cena, foi possível concluir que o algoritmo é robusto para objetos que se movem na cena sem alterar seu sentido e direção iniciais, como veículos se deslocando em sentido contrário (vide subseção 4.1.4.2) e pedestres atravessando a via (vide subseção 4.1.4.1). No entanto, para objetos que mudam de direção e sentido durante a cena, como o caso do veículo trafegando em uma trajetória (vide subseção 4.1.4.3) ou realizando uma curva (vide subseção 4.1.4.1), a orientação do algoritmo é prejudicada, já que nestes casos o algoritmo compreende que uma rotação foi realizada. Além disso, concluiu-se que o algoritmo é robusto diante de terrenos irregulares, como pavimentos de calçamento de pedra (vide subseção 4.1.4.5), vias com redutores de velocidade, como sonorizadores (vide subseção 4.1.4.2) e tachões (vide subseção 4.1.4.4), e vias com aclives e declives acentuados. Já em retornos acentuados, o algoritmo não foi capaz de estimar a orientação da câmera, o que demonstra falta de robustez para grandes quantidades de movimento, que já era esperado. Com relação ao tempo de execução, nota-se que o algoritmo, no geral, é rápido, mas em cenários com altíssima quantidade de textura, o custo computacional é alto ao ponto de poder comprometer uma aplicação em tempo real, caso não seja levado em conta o tempo de execução no pior caso.

Apesar dos resultados imprecisos do algoritmo de odometria visual em casos específicos, conclui-se que ele é preciso e robusto o suficiente para aplicações gerais, sendo adequado para uso.

Por fim, considerando então a odometria inercial, os algoritmos implementados, citados na seção 3.4, assim como os demais algoritmos para aprimoramento do processo de estimação, como aqueles citados na subseção 3.4.7, e demais que porventura foram omitidos, foram desenvolvidos de forma relativamente simples e podem ser facilmente adaptados a outros modelos e formulações, permitindo então abranger diversas aplicações distintas.

Para odometria inercial os resultados foram divididos em estimação da posição, conforme subseção 4.2.2, e estimação da orientação, conforme subseção 4.2.1, tendo em

vista que esses processos apresentam comportamentos bastante distintos.

Na estimação da orientação foi possível observar que os algoritmos apresentam comportamento bastante similar no curto prazo (vide subseção 4.2.1.1), porém, podendo apresentar desvios e uma leve diferença no comportamento dinâmico a depender das condições impostas. Já no longo prazo e com presença de estímulos de forma mais recorrente (vide subseção 4.2.1.2), os algoritmos já começam a apresentar algumas de suas características como, por exemplo, os sobressaltos oriundos das predições no filtro de Kalman, que tendem a ser mais pronunciados caso ocorram variações bruscas, e os desvios acentuados no caso do *dead reckoning* com tendência de acúmulo cada vez maior a depender dos erros já acumulados e da dinâmica descrita no processo.

Em relação a precisão, foi possível observar ganhos substanciais com o uso do filtro complementar e do filtro de Kalman, promovendo ainda maior estabilidade para processos de longa duração.

Durante os testes de curta duração (vide subseção 4.2.1.1) foi possível observar também que existem variações desses algoritmos e parâmetros que, apesar de apresentarem desempenho inferior, podem ser uma alternativa viável para realizar o processo de estimação como, por exemplo, operar com taxas de amostragem bastante baixas, o que pode ser fundamental para reduzir o custo computacional da odometria inercial, permitindo que a integração da odometria visual e inercial em uma plataforma de baixo custo possa ser realizada de forma mais simples.

Ainda considerando as variações, durante os testes com a formulação de Joseph para o filtro de Kalman (vide subseção 4.2.1.3), foi possível perceber a importância que as formulações, o modelo, e os parâmetros utilizados, tem em toda a dinâmica do processo, podendo definir questões como: convergência da solução, velocidade de convergência, precisão, e também estabilidade. Vale ressaltar ainda que, conforme observado nos testes gerais de orientação (vide subseção 4.2.1.1 e subseção 4.2.1.2), e no teste de tempo de execução (vide subseção 4.2.3), o filtro complementar apresentou resultados excelentes, sendo rápido e preciso, considerando ainda que o algoritmo é de fácil ajuste, e de simples incorporação no processo, decidir entre o uso do filtro de Kalman e do filtro complementar pode se tornar efetivamente uma questão de projeto.

Com isso, conclui-se que os algoritmos desenvolvidos são adequados para realização do processo de estimação da orientação, sendo bastante robustos e possuindo precisão relativamente elevada, podendo operar por longos períodos de tempo mantendo alta precisão (vide subseção 4.2.1.4).

Já no caso da estimação da posição, através dos testes realizados para estimação em cada um dos eixos de referência (vide subseção 4.2.2), foi possível verificar que o processo de estimação da posição não pode ser realizado de forma adequada. Com a introdução de

soluções para uma montagem mais robusta e com o uso de configurações adequadas para os sensores inerciais, foi possível promover aprimoramentos na solução, porém, observando ainda a maioria dos problemas iniciais. Com a introdução de algoritmos para aprimorar o processo de obtenção e preparação dos dados para os algoritmos já foi possível introduzir soluções mais precisas e que apresentavam redução significativa na divergência observada, porém, ainda contavam com diversos problemas, seja devido a natureza da operação, seja devido as características da implementação.

Com isso, foi possível concluir que a estimação da posição não pode ser realizada de forma adequada, sendo necessário promover o aprimoramento da solução através do uso de novos modelos, estudos dos parâmetros, e do aprimoramento dos algoritmos apresentados.

5.1 Perspectivas Futuras

Visto que o sistema de odometria visual foi implementado em uma CPU, sugere-se, primeiramente, a adaptação do algoritmo de odometria visual para a plataforma da *Xilinx*. O Apêndice A apresenta a relação entre as funções utilizadas da biblioteca OpenCV e suas respectivas correspondências com a *Vitis Vision Library*, já que as funções do algoritmo foram selecionadas de forma que a adaptação fosse possível. É importante ressaltar que se deve dar prioridade à implementação de algoritmos de cálculo da matriz essencial e sua decomposição para extração da rotação e translação na Minized, a fim de que o sistema de odometria visual seja possível de forma embarcada. Ainda, deve-se levar em conta que é necessário implementar o sistema de transferência de dados via Wi-Fi na Minized, a fim de que possa ser realizado o *debug* do algoritmo implementado. Além disso, é necessário analisar o sistema de estabilidade da câmera do Iphone 7 *plus* utilizada neste projeto, para que mecanismo semelhante possa ser implementando na câmera OV7670 já integrada na Minized, a fim de que a robustez do algoritmo em terrenos irregulares seja mantida. Neste cenário, deve-se analisar as funcionalidades disponíveis na OV7670 de forma que se garanta que funcionalidades automáticas, como correção de cor, estejam desativadas. Ainda no contexto do sistema embarcado, é de suma importância garantir que a duração do ciclo de captura de imagens seja maior que o pior tempo de execução do algoritmo de estimação da trajetória.

Outra sugestão válida é o acoplamento da câmera a um carro, a fim de se obter uma maior estabilidade na captura de *frames* e eliminar ângulos de inclinação em curvas. Com a câmera fixa ao carro, é possível obter também a escala absoluta da trajetória, através da altura da câmera em relação ao solo. Essa escala pode ser comparada com a escala gerada pelos dados da IMU, descrito na subseção 3.5.1, para obter uma escala mais precisa e, consequentemente, resultados mais precisos.

Ademais, sugere-se o estudo e implementação da etapa de otimização da *pose* esti-

mada, que não foi implementada neste projeto. Pode-se utilizar técnicas de otimização como *Graph-pose* e *Windowed Bundle Adjustments* (SCARAMUZZA; FRAUNDORFER, 2012). Ainda, recomenda-se que sejam testadas novas configurações de parâmetros utilizados nas funções da biblioteca OpenCV a fim de se obter resultados mais precisos. Por fim, é de suma importância que, uma vez que o sistema de odometria visual e o de odometria inercial esteja embarcado, a fusão de dados entre ambos os sensores seja implementada para alcançar o objetivo final, isto é, um sistema de odometria visual e inercial embarcado para estimar a posição e orientação em tempo real do agente.

Para aprimorar o processo de odometria inercial, sugere-se que a primeira modificação a ser incorporada no projeto seja garantir que a plataforma possa ser alimentada por uma fonte dedicada e que o processo de armazenamento dos dados seja realizada diretamente na plataforma.

Com essas duas medidas é possível então eliminar as restrições de movimentação na plataforma, permitindo que os dados coletados sejam mais confiáveis e, por consequência, que a análise possa ser realizada de forma mais adequada, tanto de forma qualitativa quanto quantitativa, permitindo então propor modificações com base em métricas mais criteriosas.

Para aprimorar o processo de estimação da orientação, inicialmente é necessário garantir que a calibração do magnetômetro seja realizada de forma completa, permitindo então que a estimação do ângulo de guinada possa ser feita no espaço, e não somente no plano de referência, tornando completo o processo de estimação da orientação. Com a devida correção feita, deve-se estudar a adoção de parâmetros mais adequados para o processo utilizando metodologias com um critério bem estabelecido, possivelmente garantindo desempenhos melhores que os apresentados nesse projeto.

Ressalta-se ainda que os modelos aqui apresentados para o processo de calibração e para o filtro de Kalman são modelos simplificados, recomenda-se a avaliação de modelos que utilizem formulações mais abragentes e tratem das características gerais do processo através do uso de quaternions ou do EKF. Os trabalhos de Li e Yang (2019) e Avrutov, Geraimchuk e Xiangming (2017) permitem obter uma introdução a alguns dos problemas dos modelos mais complexos.

Para aprimorar o processo de estimação da posição deve ser feita uma análise adequada do problema de forma a verificar a melhor solução a ser adotada para eliminar o problema de divergência observado.

Recomenda-se aqui algumas medidas que podem conduzir a uma solução apropriada.

Primeiro, uma melhor calibração do algoritmo que estabelece um limiar de detecção para o movimento pode promover melhores resultados através de uma avaliação mais precisa daquilo que é considerado um movimento válido, permitindo então obter informações mais adequadas da aceleração e desaceleração da plataforma.

O algoritmo ou solução apresentado para realizar a detecção de condição de desaceleração e posterior compensação apresentou resultados satisfatórios, dessa forma, um estudo mais adequado e posterior aprimoramento desse algoritmo pode levar a um processo de estimação superior.

Note que ambas soluções apontadas tem como objetivo principal a atuação na redução do problemas relacionados a estimação da velocidade e sua posterior propagação ao resultado da estimação da posição.

Por fim, considera-se então a inserção de novas informações ao processo. De forma mais imediata, para promover o aprimoramento do processo é possível considerar tanto as informações advindas do algoritmos de odometria visual, quanto as informações de controle utilizadas para navegação do VANT, tendo em vista que o objetivo final seria a integração ao veículo citado.

Referências

AGUIRRE, Luis Antonio. **Introdução à Identificação de Sistemas. Técnicas Lineares e não Lineares Aplicadas a Sistemas. Teoria e Aplicação**. 4. ed. [S.l.]: UFMG, 2015. ISBN 978-8542300796. Citado nas pp. 54, 64, 66, 67.

ALADEM, Mohamed; RAWASHDEH, Samir A. Lightweight Visual Odometry for Autonomous Mobile Robots. **Sensors**, v. 18, n. 9, 2018. ISSN 1424-8220. DOI: 10.3390/s18092837. Disponível em: <https://www.mdpi.com/1424-8220/18/9/2837>. Citado na p. 20.

ALBARBAR, Alhussein et al. Suitability of MEMS Accelerometers for Condition Monitoring: An experimental study. **Sensors**, v. 8, n. 2, p. 784–799, 2008. DOI: 10.3390/s8020784. Citado nas pp. 56, 57.

ANAC. **Veículo Aéreo Não Tripulado**. [S.l.: s.n.], 2014. Acesso: 23-09-2021. Disponível em: <https://www2.anac.gov.br/anacpedia/sig_por/tr735.htm>. Citado na p. 18.

APPLE. **iPhone 7 Plus - Technical Specifications**. [S.l.: s.n.], 2021. Acesso: 08-03-2022. Disponível em: <https://support.apple.com/kb/SP744?locale=en_US>. Citado na p. 84.

ARBABMIR, Mohammadvali; EBRAHIMI, Masoud. Visual–inertial state estimation with camera and camera–IMU calibration. **Robotics and Autonomous Systems**, v. 120, p. 103249, 2019. ISSN 0921-8890. DOI: https://doi.org/10.1016/j.robot.2019.103249. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0921889019300466>. Citado nas pp. 63, 64.

AVNET; XILINX. **Developing Zynq Software With Xilinx Software Development Kit 2017.4**. [S.l.: s.n.], 2017a. Citado nas pp. 73–75, 77.

AVNET; XILINX. **MiniZed Hardware User Guide Version 2.0**. [S.l.: s.n.], 2020. Acesso: 09-11-2021. Disponível em: <https://www.avnet.com/wps/wcm/connect/onesite/87117cdc-81ad-4af7-986a-d3e9c71aee8b/MiniZed-HW-UG-v1-0-V2.pdf?MOD=AJPERES&CACHEID=ROOTWORKSPACE.Z18_NA5A1I41L0ICD0ABNDMDDG0000-87117cdc-81ad-4af7-986a-d3e9c71aee8b-nNnW-Ie>. Citado nas pp. 72–75, 77.

AVNET; XILINX. **Minized Product Brief**. [S.l.: s.n.], 2017b. Acesso: 09-11-2021. Disponível em: <https://il.farnell.com/avnet/aes-minized-7z007-g/dev-board-single-core-zynq-7000/dp/2775207>. Citado na p. 23.

AVRUTOV, V.; GERAIMCHUK, M.; XIANGMING, Xing. 3D-Calibration for IMU of the Strapdown Inertial Navigation Systems. **MATEC Web of Conferences**, v. 114, p. 01013, jan. 2017. DOI: 10.1051/matecconf/201711401013. Citado na p. 200.

BAI, L. et al. Low Cost Inertial Sensors for the Motion Tracking and Orientation Estimation of Human Upper Limbs in Neurological Rehabilitation. **IEEE Access**, v. 8, p. 54254–54268, 2020. DOI: 10.1109/ACCESS.2020.2981014. Citado nas pp. 117–119, 131.

BAR-SHALOM, Yaakov; LI, X.-Rong; KIRUBARAJAN, Thiagalingam. **Estimation With Application to Tracking and Navigation**. 1. ed. [S.l.]: John Wiley Sons, 2001. ISBN 0-471-41655-X. Citado nas pp. 127, 128.

BARRON, John; FLEET, David; BEAUCHEMIN, S. Performance Of Optical Flow Techniques. **International Journal of Computer Vision**, v. 12, p. 43–77, fev. 1994. DOI: 10.1007/BF01420984. Citado nas pp. 36–38, 40, 82.

BENTLEY, John P. **Principles of Measurement Systems**. 4. ed. [S.l.]: Pearson Education, 2005. ISBN 0-130-43028-5. Citado nas pp. 55, 56.

BOUGUET, Jean-Yves. **Pyramidal Implementation of the Lucas Kanade Feature Tracker Description of the algorithm**. [S.l.]: Intel Corporation Microprocessor Research Labs, 2000. Disponível em: <http://robots.stanford.edu/cs223b04/algo_tracking.pdf>. Citado nas pp. 46, 88, 89.

BRADSKI, G. The OpenCV Library. **Dr. Dobb's Journal of Software Tools**, 2000. Citado na p. 80.

BROWN, Robert Grover; HWANG, Patrick Y. C. **Introduction to Random Signals and Applied Kalman Filtering With MATLAB Exercises**. 4. ed. [S.l.]: John Wiley Sons, 2012. ISBN 978-0-470-60969-9. Citado nas pp. 68–71, 117–119, 126–128.

CABRAL, Eduardo L. L. **Visão Computacional: Calibração de câmeras**. [S.l.: s.n.], 2016. Acesso: 23-10-2021. Disponível em: <https://edisciplinas.usp.br/pluginfile.php/4433641/mod_resource/content/0/V4%20-%20Calibra%C3%A7%C3%A3o%20de%20cameras.pdf>. Citado na p. 31.

CATTANEO, C; MAINETTI, Giacomo; SALA, R. The Importance of Camera Calibration and Distortion Correction to Obtain Measurements with Video Surveillance Systems. **Journal of Physics: Conference Series**, v. 658, p. 012009, nov. 2015. DOI: 10.1088/1742-6596/658/1/012009. Citado nas pp. 30, 31.

CHAN, Christophe Dang Ngoc. **Smoothing of a noisy sine with a moving average.** [S.l.: s.n.], 2014. Acesso: 17-02-2022. Disponível em: <https://commons.wikimedia.org/w/index.php?curid=31805720>. Citado na p. 136.

CORKE, P.; LOBO, J.; DIAS, J. An Introduction to Inertial and Visual Sensing. **The International Journal of Robotics Research**, v. 26, n. 6, p. 519–535, 2007. DOI: 10.1177/0278364907079279. Citado na p. 21.

DELIGIANNIDIS, L.; ARABNIA, Hamid. Emerging Trends in Image Processing, Computer Vision and Pattern Recognition, p. 1–609, jan. 2014. Citado na p. 36.

DORF, Richard C. **The Control Systems Handbook - Control System Advanced Methods**. 2. ed. [S.l.]: CRC Press, 2011. ISBN 978-1-4200-7364-5. Citado nas pp. 54, 64.

ELING, Christian; KLINGBEIL, Lasse; KUHLMANN, Heiner. Real-time single-frequency GPS/MEMS-IMU attitude determination of lightweight UAVs. **Sensors**, v. 15, p. 26212–26235, out. 2015. DOI: 10.3390/s151026212. Citado na p. 19.

EMBARCADOS. **ARM e FPGA**. [S.l.: s.n.], 2014. Acesso: 23-09-2021. Disponível em: <https://www.embarcados.com.br/arm-e-fpga/>. Citado na p. 22.

EMBARCADOS. **Introdução do mundo do hardware reconfigurável: Conhecendo as FPGAs**. [S.l.: s.n.], 2018. Acesso: 10-05-2022. Disponível em: <https://www.embarcados.com.br/introducao-do-mundo-do-hardware-reconfiguravel-conhecendo-as-fpgas/>. Citado na p. 23.

FAN, Zheng. **Optical Flow Estimation With Horn-Schunck Method**. [S.l.: s.n.], 2015. Acesso: 05-11-2021. Disponível em: <https://fzheng.me/2015/03/25/optical-flow/>. Citado na p. 29.

FÁTIMA BENTO, Maria de. Unmanned Aerial Vehicles: An overview. **Unmanned Aerial Vehicles: An overview**, 2008. Citado na p. 19.

FISCHLER, Martin A.; BOLLES, Robert C. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. **Commun. ACM**, Association for Computing Machinery, New York, NY, USA, v. 24, n. 6, p. 381–395, jun. 1981. ISSN 0001-0782. DOI: 10.1145/358669.358692. Citado nas pp. 47, 48, 52, 90.

GÁBOR, Bernát. **Camera calibration With OpenCV**. [S.l.: s.n.], 2013. Acesso: 19-01-2022. Disponível em: <https://docs.opencv.org/4.x/d4/d94/tutorial_camera_calibration.html>. Citado na p. 85.

GEBRE, Mez. **Monocular visual Odometry C+**. [S.l.: s.n.], 2018. Acesso: 18-04-2022. Disponível em: <https://github.com/mez/monocular-visual-odometry>. Citado nas pp. 82, 83.

GEIGER, Andreas; LENZ, Philip; URTASUN, Raquel. Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In: CONFERENCE on Computer Vision and Pattern Recognition (CVPR). [S.l.: s.n.], 2012. Citado nas pp. 83, 139.

GREWAL, Mohinder S.; ANDREWS, Angus P. **Kalman Filtering Theory and Practice Using MATLAB**. 3. ed. [S.l.]: John Wiley Sons, 2008. ISBN 978-0-470-17366-4. Citado nas pp. 69, 126–128.

GROVES, Paul D. **Principles of GNSS, Inertial, and Multisensor Integrated Navigation Systems**. 1. ed. [S.l.]: Artech House, 2008. ISBN 978-1-58053-255-6. Citado nas pp. 54, 55, 58–61, 65.

HANLY, Steve. **Accelerometers: Taking the Guesswork out of Accelerometer Selection**. [S.l.: s.n.], 2020. Acesso: 11-11-2021. Disponível em: <https://blog.endaq.com/accelerometer-selection>. Citado na p. 56.

HANLY, Steve. **Piezoelectric Accelerometers: Mysteries On How They Work... Revealed!** [S.l.: s.n.], 2016. Acesso: 11-11-2021. Disponível em: <https://blog.endaq.com/piezoelectric-accelerometers-how-they-work-and-where-to-buy>. Citado na p. 55.

HARTLEY, R. I.; ZISSERMAN, A. **Multiple View Geometry in Computer Vision**. 2. ed. [S.l.]: Cambridge University Press, 2004. ISBN 0521540518. Citado nas pp. 47, 52, 90.

HARTLEY, Richard I. Chirality. **Int. J. Comput. Vision**, Kluwer Academic Publishers, USA, v. 26, n. 1, p. 41–61, jan. 1998. ISSN 0920-5691. DOI: 10.1023/A:1007984508483. Citado na p. 52.

HEIKKILA, J.; SILVEN, O. A four-step camera calibration procedure with implicit image correction. In: PROCEEDINGS of IEEE Computer Society Conference on Computer Vision and Pattern Recognition. [S.l.: s.n.], 1997. P. 1106–1112. DOI: 10.1109/CVPR.1997.609468. Citado na p. 32.

HUSSEINI, Sahar. A Survey Of Optical Flow Techniques For Object Tracking. In. Citado na p. 46.

IBM. **What is Computer Vision?** [S.l.: s.n.], 2021. Acesso: 05-10-2021. Disponível em: <https://www.ibm.com/topics/computer-vision>. Citado na p. 25.

INSTRUMENTS, Texas. **Stereo vision — Facing the challenges and seeing the opportunities for ADAS applications**. [S.l.: s.n.], 2020. Acesso: 01-11-2021. Disponível em: <https://www.ti.com/lit/wp/spry300a/spry300a.pdf?ts=1635790979220&ref_url=https%253A%252F%252Fwww.google.com%252F>. Citado nas pp. 34, 35.

INVENSENSE. **MPU-6000 and MPU-6050 Product Specification Revision 3.3**. [S.l.: s.n.], 2012. Acesso: 11-11-2021. Disponível em: <https://www.alldatasheet.com/datasheet-pdf/pdf/517744/ETC1/MPU-6050.html>. Citado nas pp. 57, 61, 66.

INVENSENSE. **MPU-6000/MPU-6050 Product Specification Revision 3.4**. [S.l.: s.n.], 2013. Acesso: 11-03-2022. Disponível em: <https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf>. Citado nas pp. 94, 95.

JO, Marcelo. **Calibração de câmeras – Parte 1**. [S.l.: s.n.], 2015. Acesso: 23-10-2021. Disponível em: <https://www.embarcados.com.br/calibracao-de-cameras-parte-1/>. Citado nas pp. 31, 32.

KROLL, Noam. **3 Tips for Dealing With Rolling Shutter**. [S.l.: s.n.], 2015. Acesso: 08-11-2021. Disponível em: <https://www.premiumbeat.com/blog/3-tips-for-dealing-with-rolling-shutter/>. Citado na p. 28.

LASMADI et al. Inertial Navigation for Quadrotor Using Kalman Filter with Drift Compensation. **International Journal of Electrical and Computer Engineering (IJECE)**, Institute of Advanced Engineering e Science (IAES), Yogyakarta, Indonesia, v. 7, n. 5, p. 2596–2604, out. 2017. ISSN 2088-8708. DOI: 10.11591/ijece.v7i5.pp2596-2604. Citado nas pp. 136, 137.

LAZEBNIK, S. et al. **Corners, Blobs Descriptors**. [S.l.: s.n.], 2012. Acesso: 06-11-2021. Disponível em: <https://cs.nyu.edu/~fergus/teaching/vision_2012/3_Corners_Blobs_Descriptors.pdf>. Citado nas pp. 41, 42, 44.

LEIBSON, Steve. **IMUs for Precise Location: Part 2 – How to Use IMU Software for Greater Precision**. [S.l.: s.n.], 2019. Acesso: 11-11-2021. Disponível em: <https://www.digikey.com.br/pt/articles/imus-for-precise-location-part-2-how-to-use-imu-software-for-greater-precision>. Citado nas pp. 54, 55, 65.

LI, Hongdong; HARTLEY, R. Five-Point Motion Estimation Made Easy. In: 18TH International Conference on Pattern Recognition (ICPR'06). [S.l.: s.n.], 2006. v. 1, p. 630–633. DOI: 10.1109/ICPR.2006.579. Citado nas pp. 50–52.

LI, Joel; YANG, Van. Strapdown Inertial Navigation System Based on an IMU and a Geomagnetic Sensor. **Analog Dialogue**, v. 53, mar. 2019. Disponível em: <https://www.analog.com/en/analog-dialogue/articles/strapdown-inertial-navigation-system-based-on-an-imu-and-a-geomagnetic-sensor.html>. Citado na p. 200.

LI, Zhongke; WAN, Changsheng. Visual tracking with re-detection based on feature combination. In: 2018 Tenth International Conference on Advanced Computational Intelligence (ICACI). [S.l.: s.n.], 2018. P. 655–660. DOI: 10.1109/ICACI.2018.8377537. Citado na p. 87.

LIMA, Arthur Mendes. **Repository for the reconfigurable-camera project: Camera-Module+Zynq**. [S.l.: s.n.], 2021. Acesso: 09-05-2022. Disponível em: <https://github.com/embedded-computing/reconfigurable-camera/tree/vdma3buff>. Citado na p. 80.

LONGUET-HIGGINS, Hugh Christopher. A computer algorithm for reconstructing a scene from two projections. **Nature**, v. 293, p. 133–135, 1981. Citado na p. 51.

LUCAS, Bruce; KANADE, Takeo. An Iterative Image Registration Technique with an Application to Stereo Vision (IJCAI). In: v. 81. Citado na p. 36.

MADGWICK, Sebastian O.H.; HARRISON, Andrew J.L.; VAIDYANATHAN, Ravi. Estimation of IMU and MARG orientation using a gradient descent algorithm. **IEEE International Conference on Rehabilitation Robotics**, p. 1–7, 2011. DOI: 10.1109/ICORR.2011.5975346. Citado na p. 115.

MAHONY, Robert; HAMEL, Tarek; PFLIMLIN, Jean-Michel. Nonlinear Complementary Filters on the Special Orthogonal Group. **IEEE Transactions on Automatic Control, Institute of Electrical and Electronics Engineers**, v. 53, n. 5, p. 1203–1218, jun. 2008. DOI: 10.1109/TAC.2008.923738. Citado na p. 115.

MAJUMDER, Aditi; GOPI, M. **Introduction to Visual Computing- Core Concepts in Computer Vision, Graphics, and Image Processing**. 1. ed. [S.l.]: CRC Press, 2018. ISBN 978-1-4822-4491-5. Citado na p. 49.

MUNICH, Technical University of. **Monocular Visual Odometry Dataset**. [S.l.: s.n.], 2016. Acesso: 18-03-2022. Disponível em: <https://vision.in.tum.de/data/datasets/mono-dataset>. Citado na p. 83.

NAVY, United States. **The Navy Electricity and Electronics Training Series - Introduction To Test Equipment**. 1. ed. [S.l.]: Createspace Independent Publishing Platform, 2016. ISBN 978-1530962402. Citado nas pp. 58, 59.

NISTER, D. An efficient solution to the five-point relative pose problem. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 26, n. 6, p. 756–770, 2004. DOI: 10.1109/TPAMI.2004.17. Citado nas pp. 50–52, 90, 91.

NÚÑEZ, Pedro; VÁZQUEZ-MARTÍN, Ricardo; BANDERA, Antonio. Visual Odometry Based on Structural Matching of Local Invariant Features Using Stereo Camera Sensor. **Sensors**, v. 11, n. 7, p. 7262–7284, 2011. ISSN 1424-8220. DOI: 10.3390/s110707262. Disponível em: <https://www.mdpi.com/1424-8220/11/7/7262>. Citado na p. 35.

OMNIVISION. **OV7670 datasheet**. [S.l.: s.n.], 2006. Acesso: 08-03-2022. Disponível em: <http://web.mit.edu/6.111/www/f2016/tools/OV7670_2006.pdf>. Citado na p. 78.

OPENCV. **Camera Calibration**. [S.l.: s.n.], 2013. Acesso: 19-01-2022. Disponível em: <https://docs.opencv.org/3.4/dc/dbb/tutorial_py_calibration.html>. Citado na p. 30.

OTSU, Kyohei et al. An Examination of Feature Detection for Real-Time Visual Odometry in Untextured Natural Terrain. In: **Robot Intelligence Technology and Applications 2012: An Edition of the Presented Papers from the 1st International Conference on Robot Intelligence Technology and Applications**. Edição: Jong-Hwan Kim. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. P. 405–414. DOI: 10.1007/978-3-642-37374-9_39. Disponível em: <https://doi.org/10.1007/978-3-642-37374-9_39>. Citado nas pp. 43, 44, 82.

OTUYAMA, Júlio M. **Curso de Visão Computacional: Visão estéreo**. [S.l.: s.n.], 1998. Acesso: 02-11-2021. Disponível em: <http://www.inf.ufsc.br/~aldo.vw/visao/1998/otuyama/1.htm>. Citado na p. 35.

OUERGHI, Safa et al. Visual Odometry and Place Recognition Fusion for Vehicle Position Tracking in Urban Environments. **Sensors**, v. 18, n. 4, 2018. ISSN 1424-8220. DOI: 10.3390/s18040939. Disponível em: <https://www.mdpi.com/1424-8220/18/4/939>. Citado na p. 140.

PAUL, Johnathan. **Rolling Shutter vs Global Shutter: What's the difference?** [S.l.: s.n.], 2016. Acesso: 07-10-2021. Disponível em: <https://www.premiumbeat.com/blog/know-the-basics-of-global-shutter-vs-rolling-shutter/>. Citado nas pp. 27, 28.

PRATAMA, Andre Tegar. Augmented Reality Land Transportation Using FAST Corner Detection and Lucas Kanade. **JATISI (Jurnal Teknik Informatika dan Sistem Informasi)**, v. 8, n. 3, 2021. Citado na p. 82.

QU, Chengchao et al. Capturing ground truth super-resolution data. In: p. 2812–2816. DOI: 10.1109/ICIP.2016.7532872. Citado na p. 27.

REBERT, Martin et al. A review of the dataset available for visual odometry. In: p. 77. DOI: 10.1117/12.2521750. Citado na p. 83.

ROMANIUK, Sławomir; GOSIEWSKI, Zdzisław. Kalman Filter Realization for Orientation and Position Estimation on Dedicated Processor. **Acta Mechanica et Automatica**, v. 8, n. 2, p. 88–94, 2014. DOI: 10.2478/ama-2014-0016. Citado nas pp. 70, 120–122, 125.

ROSTEN, E.; DRUMMOND, T. Fusing points and lines for high performance tracking. In: TENTH IEEE International Conference on Computer Vision (ICCV'05) Volume 1. [S.l.: s.n.], 2005. v. 2, 1508–1515 vol. 2. DOI: 10.1109/ICCV.2005.104. Citado na p. 86.

ROSTEN, Edward; DRUMMOND, Tom. Machine Learning for High-Speed Corner Detection. In: LEONARDIS, Aleš; BISCHOF, Horst; PINZ, Axel (Ed.). **Computer Vision – ECCV 2006**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006. P. 430–443. DOI: 10.1007/11744023_34. Citado nas pp. 41, 42, 86.

SCARAMUZZA, Davide; FRAUNDORFER, Friedrich. Visual Odometry. Part I: The First 30 Years and Fundamentals. **IEEE Robotics Automation Magazine**, v. 18, n. 4, p. 80–92, dez. 2011. DOI: 10.1109/MRA.2011.943233. Citado nas pp. 20, 33, 36, 38, 39, 53, 81.

SCARAMUZZA, Davide; FRAUNDORFER, Friedrich. Visual Odometry. Part II: Matching, Robustness, Optimization, and Applications. **IEEE Robotics Automation Magazine**, v. 19, n. 2, p. 78–90, jun. 2012. DOI: 10.1109/MRA.2012.2182810. Citado nas pp. 39, 41, 43, 44, 47, 48, 81, 87, 91, 200.

SCARAMUZZA, Davide; FRAUNDORFER, Friedrich; SIEGWART, Roland. Real-time monocular visual odometry for on-road vehicles with 1-point RANSAC. In: 2009 IEEE International Conference on Robotics and Automation. [S.l.: s.n.], 2009. P. 4293–4299. DOI: 10.1109/ROBOT.2009.5152255. Citado na p. 33.

SCARAMUZZA, Davide; MARTINELLI, Agostino; SIEGWART, Roland. A Toolbox for Easily Calibrating Omnidirectional Cameras. In: 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems. [S.l.: s.n.], 2006. P. 5695–5701. DOI: 10.1109/IROS.2006.282372. Citado na p. 32.

SCARAMUZZA, Davide; ZHANG, Zichao. Visual-Inertial Odometry of Aerial Robots. **Springer Encyclopedia of Robotics**, 2019. Citado nas pp. 21, 22, 33, 63, 64.

SCHWEBER, Bill. **Gyroscopes, Part 2: Optical and MEMS implementations**. [S.l.: s.n.], 2018. Acesso: 11-11-2021. Disponível em: <https://www.analogictips.com/gyroscopes-part-2-optical-and-mems-implementations-faq/>. Citado na p. 60.

SHAFIQUE, Khurram Hassan. **CAP5415 - Computer Vision**. [S.l.: s.n.], 2003. Acesso: 28-01-2022. Disponível em: <http://www.cs.ucf.edu/courses/cap6411/cap5415/>. Citado nas pp. 45, 88, 89.

SHAN, Mo et al. A brief survey of visual odometry for micro aerial vehicles. In: p. 6049–6054. DOI: 10.1109/IECON.2016.7793198. Citado na p. 33.

SHARMA, Mathanraj. **Understanding Images with skimage-Python**. [S.l.: s.n.], 2019. Acesso: 06-10-2021. Disponível em: <https://towardsdatascience.com/understanding-images-with-skimage-python-b94d210afd23>. Citado nas pp. 25, 26.

SHI, Jianbo; TOMASI. Good features to track. In: 1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition. [S.l.: s.n.], 1994. P. 593–600. DOI: 10.1109/CVPR.1994.323794. Citado nas pp. 45, 82.

SICILIANO, Bruno; KHATIB, Oussama. **Springer Handbook of Robotics**. 3. ed. Berlin, Heidelberg: Springer-Verlag, 2007. ISBN 354023957X. Citado nas pp. 53, 138.

SIEGWART, Roland; NOURBAKHSH, Illah R.; SCARAMUZZA, Davide. **Introduction to Autonomous Mobile Robots**. 2. ed. [S.l.]: The MIT Press, 2011. ISBN 0262015358. Citado na p. 41.

SILVA, Bruno A. da et al. A Manycore Vision Processor for Real-Time Smart Cameras. **Sensors**, v. 21, n. 21, 2021. ISSN 1424-8220. DOI: 10.3390/s21217137. Disponível em: <https://www.mdpi.com/1424-8220/21/21/7137>. Citado na p. 78.

SINGH, Avi. **Monocular Visual Odometry**. [S.l.: s.n.], 2015. Acesso: 17-03-2022. Disponível em: <http://avisingh599.github.io/assets/ugp2-report.pdf>. Citado na p. 82.

STURM, J. et al. A Benchmark for the Evaluation of RGB-D SLAM Systems. In: PROC. of the International Conference on Intelligent Robot Systems (IROS). [S.l.: s.n.], out. 2012. Citado na p. 83.

TRUCCO, Emanuele; VERRI, Alessandro. **Introductory techniques for 3-D computer vision**. 1. ed. [S.l.]: Pearson, 1998. ISBN 978-0-13-261108-4. Citado nas pp. 25–29, 31, 35, 36, 38, 40, 50, 53.

UNICAMP, IME -. **O Sistema GPS**. [S.l.: s.n.], 2020. Acesso: 23-09-2021. Disponível em: <http://www.ime.unicamp.br/~apmat/o-sistema-gps/>. Citado na p. 18.

WANG, Sen et al. DeepVO: Towards End-to-End Visual Odometry with Deep Recurrent Convolutional Neural Networks. In: p. 2043–2050. DOI: 10.1109/ICRA.2017.7989236. Citado na p. 91.

WIKIPEDIA. **Inertial Measurement Unit**. [S.l.: s.n.], 2021. Acesso: 23-09-2021. Disponível em: <https://en.wikipedia.org/wiki/Inertial_measurement_unit>. Citado na p. 19.

WILSON, Jon S. **Sensor Technology Handbook**. 1. ed. [S.l.]: ELSEVIER, 2005. ISBN 0-7506-7729-5. Citado nas pp. 55, 62, 63.

XILINX. **AXI IIC Bus Interface v2.1 LogiCORE IP Product Guide**. [S.l.: s.n.], 2021a. Acesso: 11-03-2022. Disponível em: <https://www.xilinx.com/support/documentation/ip_documentation/axi_iic/v2_1/pg090-axi-iic.pdf>. Citado nas pp. 98, 99.

XILINX. **Field Programmable Gate Array (FPGA)**. [S.l.: s.n.], 2017. Acesso: 08-11-2021. Disponível em: <https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html>. Citado na p. 22.

XILINX. **Vitis Vision Library**. [S.l.: s.n.], 2021b. Acesso: 08-03-2022. Disponível em: <https://xilinx.github.io/Vitis_Libraries/vision/2021.2/index.html>. Citado nas pp. 80, 213.

XILINX. **Vitis Vision Library functions**. [S.l.: s.n.], 2021c. Acesso: 18-04-2022. Disponível em: <https://xilinx.github.io/Vitis_Libraries/vision/2021.2/api-reference.html#vitis-vision-library-functions>. Citado na p. 213.

ZHANG, Hongmei; XIAO, Li; XU, Guangyan. A Novel Tracking Method Based on Improved FAST Corner Detection and Pyramid LK Optical Flow. In: 2020 Chinese Control And Decision Conference (CCDC). [S.l.: s.n.], 2020. P. 1871–1876. DOI: 10.1109/CCDC49329.2020.9164332. Citado na p. 82.

ZHANG, Z. A flexible new technique for camera calibration. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 22, n. 11, p. 1330–1334, 2000. DOI: 10.1109/34.888718. Citado na p. 32.

ZHANG, Zhe; HAN, Deqiang et al. A New Image Registration Algorithm Based on Evidential Reasoning. **Sensors**, v. 19, p. 1091, mar. 2019. DOI: 10.3390/s19051091. Citado nas pp. 39, 40, 81.

Apêndices

APÊNDICE A – Adaptação do algoritmo de Odometria Visual para a Minized

Conforme expresso na Seção 3.1, as funções do algoritmo de Odometria Visual foram selecionadas levando em conta, dentre outros fatores, a sua implementação na biblioteca de visão computacional da *Xilinx* baseada em OpenCV - *Vitis Vision Library* (XILINX, 2021b). Uma descrição completa das funções disponíveis pode ser encontrada em (XILINX, 2021c).

Na etapa de detecção de *features*, foi utilizado o detector de cantos FAST, que pode ser encontrada na *Vitis Vision* através da função *fast()*. Há a possibilidade de se processar 1 ou 8 *pixels* por ciclo. A Figura 163 ilustra a performance estimada da função *fast* usando a FPGA Xczu9eg-ffvb1156-1-i-es1 para processar uma imagem em escala de cinza com supressão não máxima.

Figura 163 – Performance estimada da função FAST

Operating Mode	Operating Frequency (MHz)	Filter Size	Latency Estimate
			Max (ms)
1 pixel	300	3x3	7
8 pixel	150	3x3	1.86

Fonte: Xilinx (2021c)

Na etapa de rastreamento de *features* foi utilizado o rastreador Lucas-Kanade piramidal, que pode ser encontrado na *Vitis Vision* através da função *densePyrOpticalFlow()*. É importante ressaltar que o algoritmo implementado usa o método esparso, enquanto a função, o denso. A Figura 164 ilustra a performance estimada da função *densePyrOpticalFlow* para 5 iterações sobre 5 níveis piramidais diminuído em um fator de 2 em cada nível. Os testes foram realizados na placa zcu102 (XILINX, 2021c).

Figura 164 – Performance estimada da função *densePyrOpticalFlow*

Operating Mode	Operating Frequency (MHz)	Image Size	Latency Estimate
			Max (ms)
1 pixel	300	1920x1080	49.7
1 pixel	300	1280x720	22.9
1 pixel	300	1226x370	12.02

Fonte: Xilinx (2021c)

As etapas de cálculo da matriz essencial e a estimação de rotação e translação utilizam as funções implementadas em OpenCV: *findEssentialMat()* e *recoverPose()*, respectivamente. Até a data de desenvolvimento deste projeto, estas funções ainda não foram disponibilizadas na *Vitis Vision Library*.