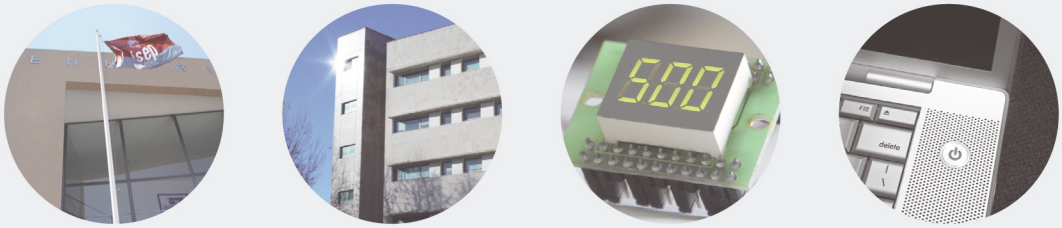
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES



Exploração de metodologias de Odometria e Mapeamento 3D baseado em LiDAR

EDUARDO DANIEL MACHADO MARQUES
novembro de 2022

POLITÉCNICO
DO PORTO

Politécnico do Porto
Instituto Superior de Engenharia do Porto

Exploração de Metodologias de Odometria e Mapeamento 3D baseado em LiDAR

Eduardo Daniel Machado Marques

Mestrado em Engenharia Electrotécnica e de Computadores
Área de Especialização em Sistemas Autónomos

Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto

Novembro, 2022

Esta dissertação satisfaz, parcialmente, os requisitos que constam da Ficha de Unidade Curricular de Tese/Dissertação, do 2º ano, do Mestrado em Engenharia Electrotécnica e de Computadores, Área de Especialização em Sistemas Autónomos.

Candidato: Eduardo Daniel Machado Marques, Nº 1150666,
1150666@isep.ipp.pt

Orientação Científica: Professor André Dias, apd@isep.ipp.pt

Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto
Rua Dr. António Bernardino de Almeida, 431, 4200–072 Porto

Novembro, 2022

Aos meus pais, por tudo o que fizeram para me tornarem a pessoa que sou hoje.

Agradecimentos

Gostaria de começar por agradecer ao meu orientador nesta dissertação, o Professor André Dias, pela orientação, por todo o conhecimento transmitido e colaboração na resolução de todos os problemas que surgiram na realização desta dissertação.

Gostaria também de agradecer a toda a minha família, com especial menção aos meus pais e às minhas irmãs que, desde sempre, me ajudaram e incentivaram a ser a minha melhor versão, e neste percurso académico, desde o primeiro dia me apoiaram e compreenderam nos momentos em que não pude estar presente.

Por fim, gostaria de agradecer a todos os meus amigos e colegas que me acompanharam na vida académica, que apoiaram nos momentos de maior dificuldade e ajudaram a criar memórias que levarei para a vida. Não podendo deixar de nomear três pessoas com especial importância.

Ao João Moreira, que me acompanhou ao longo destes últimos anos, nas dificuldades e nas boas memórias, mas acima de tudo, como um companheiro de jornada independentemente das circunstâncias ou adversidades.

Ao Diogo Xavier que, para além da sua amizade e companheirismo, no meu momento mais difícil deixou todas as suas preocupações de parte e ajudou a tornar esta dissertação possível.

E por fim, mas não menos importante, à Rita Varejão, por toda a paciência, apoio e bondade demonstrada, pelas suas palavras de alento e boa disposição que sempre conseguiu transmitir.

A todos os mencionados e aqueles que me possa ter esquecido de mencionar, o meu muito obrigado.

Resumo

Devido a esta utilização mais alargada dos UAV's, esta dissertação pretende analisar a possibilidade de utilização de mapeamento 3D e odometria baseado apenas em LiDAR como forma de substituir outros sensores, como o GPS em caso de falha de conecção ou *jamming* do mesmo.

De forma a responder a esta questão, esta dissertação irá analisar a possibilidade de utilização de implementações previamente desenvolvidas e baseadas em LOAM, até então testadas em veículos terrestres. Para além de testar esta possibilidade, pretende ainda identificar a implementação indicada para utilizar no contexto pretendido, uma vez que a sua utilização num UAV com processamento em tempo real apresenta maiores restrições do que as impostas aos veículos terrestres. Estas restrições advém principalmente da reduzida versatilidade na adição de capacidade de processamento.

De forma a realizar a análise, foram assim realizados dois *datasets* que permitiram perceber as restrições e cuidados a ter com o plano de voo caso o UAV esteja equipado com uma implementação baseada em LOAM. Estes teste, permitiram ainda compreender qual das implementações seria a mais indicada para uma possível implementação num UAV.

Palavras-Chave: UAV, LiDAR, Odometria, Mapeamento, Benchmark.

Abstract

Due to this wider use of UAV's, this dissertation aims to analyze the possibility of using 3D mapping and odometry based only on LiDAR as a way to replace other sensors, such as GPS in case of connection failure or jamming of the signal.

In order to answer this question, this dissertation will analyze the possibility of using implementations previously developed and based on LOAM, hitherto tested in land vehicles. In addition to testing this possibility, it also intends to identify the appropriate implementation to use in the intended context, since its use in an UAV with real-time processing presents greater restrictions than those imposed on land vehicles. These restrictions stem mainly from the reduced versatility in adding processing power.

In order to do the analyses, two datasets where made what enabled us to understand the constraints to look into when planning a flight plan for an UAV equipped with one of the LOAM based implementations. This tests, enabled us to also understand wich of the implementations would the the most suitable to apply on an UAV.

Keywords: UAV, LiDAR, Odometry, Mapping, Benchmark.

Índice

Lista de Figuras

Lista de Tabelas

Lista de Acrónimos

BDS *BeiDou Navigation Satellite System*

CPU *Central Processing Unit*

DEE Departamento de Engenharia Electrotécnica

ECEF *Earth-centered, Earth-fixed coordinate system*

ENU *East-North-Up coordinate system*

GLONASS *GLObalnaya NAvigatsionnaya Sputnikovaya Sistema*

GNSS *Global Navigation Satellite System*

GPS *Global Positioning System*

ICP *Iterative Closest Point*

IMU *Inertial Measurement Unit*

INESC-TEC Instituto de Engenharia de Sistemas e Computadores, Tecnologia e Ciência

INS *Inertial Navigation System*

ISC *Intensity Scan Context*

ISEP Instituto Superior de Engenharia do Porto

LiDAR *Light Detection and Ranging*

LOAM *LiDAR Odometry and Mapping*

MAE *Mean Absolute Error*

MAPE *Mean Absolute Percentage Error*

MEEC Mestrado em Engenharia Electrotécnica e Computadores

MSE *Mean Squared Error*

R2 *R-Squared*

RAM *Random Access Memory*

RMSE *Root Mean Squared Error*

ROS *Robot Operating System*

RTK *Real-Time Kinematic*

UAV *Unmanned Aerial Vehicle*

VLP-16 *Velodyne Puck 16*

Lista de Símbolos

Símbolo	Descrição	Unidades
c	Rugosidade da superfície	
$\mathcal{D}$	Conjunto de locais previamente visitados	
d	Distância	
$\mathcal{E}$	Conjunto de pontos de esquina	
$\mathcal{E}_{k+1}$	Conjunto de pontos de esquina	
$\widehat{\mathcal{E}}_{k+1}$	Conjunto de pontos de esquina reprojectados	
η	Leitura da intensidade	
$\mathcal{H}$	Conjunto de pontos de plano	
$\mathcal{H}_{k+1}$	Conjunto de pontos de plano	
$\widehat{\mathcal{H}}_{k+1}$	Conjunto de pontos de plano reprojectados	
i	Representa um ponto	
k	Iteração do *scan*/frame	
$\{L\}$	Conjunto dos pontos $\widehat{\mathcal{P}}$	
λ	Fator determinado pelo método de *Levenberg-Marquardt*	
$\widehat{\omega}$	Matriz anti-simétrica da matriz da magnitude de rotação[1]	
ω	Vetor unitário da direção da rotação	
Ω	Intensidade	
$\mathcal{P}_k$	*Point Cloud* gerada pelos diversos $\widehat{\mathcal{P}}$	

Símbolo	Descrição	Unidades
$\mathcal{P}_{k+1}$	*Point Cloud* gerada pelos diversos $\widehat{\mathcal{P}}_{k+1}$	
$\widehat{\mathcal{P}}_{k+1}$	*Point Cloud* gerada pelos diversos $\widehat{\mathcal{P}}_{k+1}$	
$\widehat{\mathcal{P}}_k$	Reprojecção de $\mathcal{P}_k$ em t_{k+1}	
$\widehat{\mathcal{P}}$	Pontos recolhidos durante um *Scan*	
R	Matriz de rotação definida pela formula de Rodrigues[1]	
$\mathcal{S}$	Pontos consecutivos de i retornados num mesmo *scan*	
θ	Magnitude de rotação	
θ_x	Rotação em torno de X, segundo a regra da mão direita	
θ_y	Rotação em torno de Y, segundo a regra da mão direita	
θ_z	Rotação em torno de Z, segundo a regra da mão direita	
t_k	Tempo inicial do *scan* k	
t_{k+1}	Tempo inicial do *scan* k+1	
$T^L_{k}{}_{+1}$	Transformada entre t_{k+1} e t	
$T^L_{(}{}_{k+1,i)}$	Transformação entre [t_{k+1}, ti]	
t_x	Transformada em torno de x	
t_y	Transformada em torno de y	
t_z	Transformada em torno de z	
φ	Função de mapeamento	
$\widehat{\mathcal{X}}^L_{(}{}_{k+1,i)}$	Coordenadas do ponto i	
$\widehat{\mathcal{X}}^L_{(}{}_{k+1,j)}$	Coordenadas do ponto j	
$\widehat{\mathcal{X}}^L_{(}{}_{k+1,l)}$	Coordenadas do ponto l	
$\widehat{\mathcal{X}}^L_{(}{}_{k+1,m)}$	Coordenadas do ponto m	

Símbolo	Descrição	Unidades
${}^{\mathcal{S}}X_i^k$	Coordenadas do ponto i	

Capítulo 1

Introdução

Este documento tem como intuito explicar e explorar os diferentes requisitos apresentados sobre o tema proposto para a realização da dissertação do 2º ano do Mestrado em Engenharia Electrotécnica e Computadores (MEEC), do Departamento de Engenharia Electrotécnica (DEE), do Instituto Superior de Engenharia do Porto (ISEP).

1.1 Contextualização

Motivada pelos diversos conflitos que acompanham a historia da humanidade, esta, procurou desde sempre descobrir novas formas de obter vantagem sobre os seus adversários. Foi desta forma que em 1849, 54 anos antes do voo do primeiro avião desenvolvido pelos irmãos Wright, foram lançados os predecessores dos atuais UAV's. Estes consistiam em balões carregados com explosivos, que eram transportados pelas correntes de ar, este fator levou a uma baixa taxa de sucesso na sua utilização, mas despoletou o interesse para utilização de sistemas semelhantes.

Desde então a utilização de UAV's apresentou um crescimento exponencial no dia a dia da humanidade, desda finalidade que despoletou o interesse pela área, à automatização e redução de custos de tarefas como verificação de danos em estruturas, agricultura ou mesmo entrega de pequenas mercadorias.

Com a expansão da utilização de sistemas autónomos que complementam e resolvem tarefas cada vez mais variadas, cresce também a procura por soluções que

aumentem a segurança e robustez dos sistemas, sem descurar das capacidades já disponíveis.

Desta forma, e no contexto em que o MEEC se encontra inserido, foi proposta a possibilidade de realizar um *benchmark*, de forma a compreender de entre as implementações baseadas em *LiDAR Odometry and Mapping* (LOAM) já disponíveis no mercado, aquela que poderia ser mais interessante para a aplicação da mesma num UAV.

Atualmente, a maioria das implementações existentes são desenvolvidas tendo em mente a sua aplicação em veículos terrestres, por diversos motivos, entre os quais:

- Custos mais reduzidos no caso de erro;
- Menor risco de danos no veículo;
- Ambientes mais controlados e mais conhecidos;
- Facilidade de adição de *hardware*, de forma a responder à necessidade de uma maior capacidade de processamento.

Na dissertação pretende-se avaliar a possibilidade de utilização das implementações baseadas em LOAM desenvolvidas para veículos terrestres em UAV's, ainda que estes apresentem mais limitações quando comparados ao descrito acima, nomeadamente:

- Em caso de erro poderá ocorrer colisão do UAV com um objeto, o que poderá resultar na queda do equipamento;
- Apresentarem maiores limitações em termos de espaço e peso que pode ser adicionado e como tal, maiores limitações na adição de *hardware* de processamento e armazenamento;
- Maior erro nas medidas devido ao modo de movimento;
- Maior suscetibilidade a que os dados sejam afetados pelas condições atmosféricas, nomeadamente o vento.

Apesar das limitações anteriormente apresentadas, a possibilidade de utilizar o mapeamento 3D e odometria processada através dos dados de um LiDAR já equipado no UAV apresentam vantagens de segurança na execução de missões. Estas vantagens prendem-se pela possibilidade de executar missões em localizações onde o sinal de *Global Positioning System* (GPS) é fraco ou inexistente.

Adicionalmente, a metodologia em estudo poderá ser aplicada em caso de *jamming* do sinal de *GPS*, tornando agora possível para que o UAV navegue até um sítio de segurança e aterre de emergência de forma segura.

1.2 Definição do Problema

O ISEP tem vindo a desenvolver um UAV que endereça os cenários de busca e salvamento, inspeção de ativos elétricos, a inspeção de infraestruturas como barragens ou eólicas e a reconstrução de locais históricos (Heritage).

De forma a responder a estes desafios, o objetivo passa por dotar o UAV da capacidade de efetuar reconstrução 3D em cenários onde a qualidade de posicionamento e atitude são afetadas pelas infraestruturas existentes no local. Isto advém da possibilidade dos dados do GPS serem afetados de multi caminho, não permitindo a reconstrução das nuvens de pontos extraídas do LiDAR.

A solução estudada nesta dissertação poderá, de futuro, servir como uma forma adicional de tornar os dados de posicionamento e atitude mais robustos a situações de *jamming*[2].

1.2.1 Objetivos

Com base no problema apresentado no Capítulo 1.2, esta dissertação propõe-se a:

- Analisar diversas técnicas baseadas em LiDAR que permitam estimação de posição e atitude, assim como o alinhamento das nuvens de pontos adquiridos pelo UAV;
- Avaliar qual a técnica mais adequadas para UAV's;
- Efetuar a produção de um *dataset* que permita a realização do *benchmark* das técnicas em estudo;
- *Benchmark* com base nos dados recolhidos com o UAV do ISEP.

1.3 Plano de Trabalho

O plano de elaboração desta dissertação pode ser visto na Figura 1.1.

	Tarefas	Duração
1	Desenvolvimento da dissertação	178 dias
2	Estado da Arte	91 dias
3	Compreensão do projecto	7 dias
4	LOAM	7 dias
5	ISC-LOAM	7 dias
6	FLOAM	7 dias
7	VLOAM	7 dias
8	Lego-LOAM	7 dias
9	KITTI Framework	7 dias
10	Trabalhos semelhantes	7 dias
11	Métricas de avaliação de erro	30 dias
12	Implementação	60 dias
13	Preparação das diversas implementações	14 dias
14	Desenvolvimento do node de rotação	14 dias
15	Desenvolvimento de ferramenta de visualização dos dados	14 dias
16	Desenvolvimento de ferramentas de suporte	54 dias
17	Recolha de dados	21 dias
18	Realização dos testes	21 dias

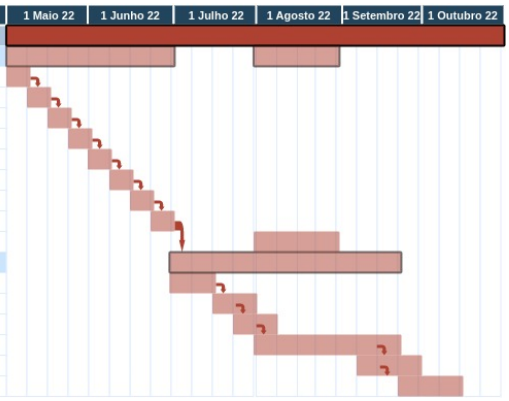


Figura 1.1: Diagrama de Gantt.

1.4 Organização da Dissertação

Nesta secção é apresentada a estrutura deste relatório, assim como uma breve descrição de cada capítulo.

No capítulo 2 é apresentado um estado da arte, no qual serão analisados os diversos trabalhos sobre os quais esta dissertação se irá debruçar. São ainda apresentados alguns trabalhos semelhantes na área e também alguns métodos que possibilitam uma análise dos erros associados aos resultados de cada implementação em estudo.

No capítulo 3 são apresentadas algumas ferramentas desenvolvidas de forma a possibilitar o desenvolvimento desta dissertação. É ainda apresentado o *hardware* utilizado para a recolha de um dos *datasets* utilizados para testar os diversos trabalhos.

No capítulo 4 é apresentado um primeiro teste que teve por objetivo detetar as fragilidades dos diversos trabalhos de forma a posteriormente recolher um *dataset* que possibilitasse uma análise mais correta dos resultados. De seguida é apresentado o teste realizado com base nas conclusões anteriormente retiradas de forma a analisar as mesmas. Por fim, é ainda feito um terceiro teste, com o objetivo de testar as diversas implementações num caso diferente do anterior.

No capítulo 5 são por fim apresentadas as conclusões finais desta dissertação, tendo por base os resultados anteriormente recolhidos. Com base nestas conclusões é ainda apresentado o trabalho futuro.

Capítulo 2

Estado da Arte

Neste capítulo é apresentada a base teórica para a elaboração do projeto proposto, focando-se principalmente no estudo de diversas implementações da estimação da odometria com recurso a LiDAR e nos métodos de avaliação do erro das mesmas. Este Capítulo procura explicar de forma simplificada o funcionamento de cada uma das metodologias em análise, de forma a possibilitar uma compreensão mais aprofundada dos resultados.

Iremos explorar as diferentes implementações *LiDAR Odometry and Mapping* disponíveis, não só em contexto de veículos aéreos mas também noutros cenários de implementação.

LOAM é a proposta de um método de odometria e mapeamento em tempo real, sem a necessidade de recorrer a sensores como o GPS, *encoders* ou semelhantes tradicionalmente utilizados para o cálculo da posição, mas que recorre ao processamento dos dados recolhidos por um LiDAR (Figura 2.1).

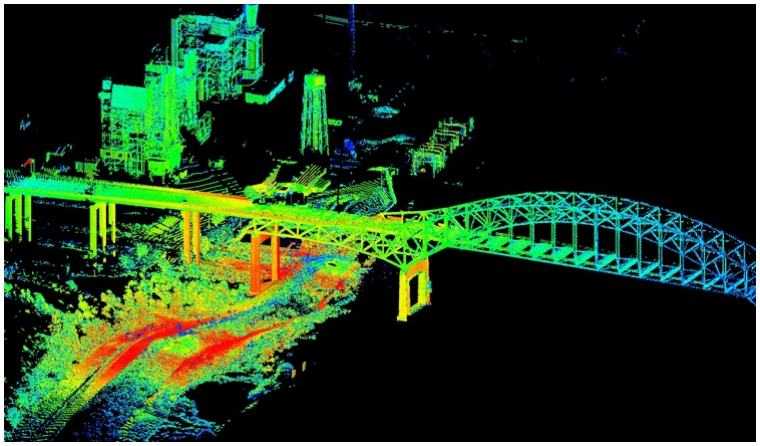


Figura 2.1: Mapeamento feito com um LiDAR.[3]

Tradicionalmente esta utilização do LiDAR apresenta um elevado desvio da realidade e um elevado custo computacional, sendo que estas metodologias se propõem a melhorar ambos os fatores.

A escolha do LiDAR advém da sua grande utilização na recolha de dados, uma vez que a alta frequência de leitura que possui é pouco afetada por fatores como a distância, textura e condições de iluminação do local

Posteriormente, foram criadas diversas versões da implementação inicial, cada uma delas com o objetivo de responder a problemas não atendidos inicialmente ou de melhorar aspetos específicos da implementação inicial.

2.1 *LOAM - LiDAR Odometry and Mapping in Real-time*

A implementação [4] detalhada neste capítulo foi a primeira de entre as que serão apresentadas que, como descrito anteriormente, se propõe a otimizar o processamento dos dados recolhidos por um LiDAR, possibilitando que o mesmo fosse utilizado em tempo real. Para tal, esta implementação propõe a utilização de dois algoritmos em paralelo, com diferentes objetivos e que correm a diferentes frequências, com o objetivo de calcular partes complementares do resultado final.

No primeiro, é-nos apresentado um algoritmo de elevada frequência (10Hz) mas baixa fiabilidade cujo objetivo é estimar a velocidade do LiDAR, que será utilizada para corrigir a distorção provocada na nuvem de pontos.

O segundo, trata-se de um algoritmo de baixa frequência (1Hz) que tem por objetivo fazer a correspondência entre a posição e a nuvem de pontos recolhida.

Em conjunto estes produzem o resultado à frequência de 10Hz.

Por fim, este método não usa *loop closure* uma vez que o seu objetivo é testar a utilização do método em estudo sem o suporte de outras técnicas.

2.1.1 Arquitetura de *Software*

A arquitetura de *Software* da implementação em estudo é apresentada na Figura 2.2, onde $\widehat{\mathcal{P}}$ representa os pontos recolhidos durante um *scan*, sendo que cada um destes é registado em $\{L\}$.

Os diversos *scans* durante uma dada iteração (k) formam $\mathcal{P}_k$ que é por sua vez processada nos dois algoritmos anteriormente mencionados.

No bloco *LiDAR Odometry*, a nuvem de pontos é utilizada para calcular o movimento entre duas iterações consecutivas que irá corrigir a distorção na nuvem de pontos. De seguida a informação é processada pelo passo do *LiDAR Mapping* que posiciona o veículo no mapa.

Finalmente, a posição é aplicada na nuvem de pontos, colocando a mesma numa dada posição do mundo, gerando uma representação deste.

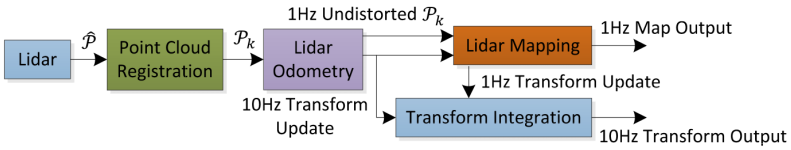


Figura 2.2: Arquitetura de *software*.[4]

2.1.2 Odometria pelo LiDAR

Este bloco do algoritmo pode ser dividida em quatro partes.

Na primeira, temos a extração de pontos de interesse de $\mathcal{P}_k$, onde são selecionados pontos que pertençam a planos ou esquinas. Para conseguir fazer esta extração, inicialmente é avaliada a suavidade da superfície através da Equação 2.1.

$$c = \frac{1}{|S| * ||X^L_{(k,i)}||} * \left|\left| \sum_{j \in S, j \neq i} (X^L_{(k,i)} - X^L_{(k,j)}) \right|\right| \tag{2.1}$$

Em que c representa a rugosidade da superfície, $\mathcal{S}$ os pontos de um dado *scan* e por fim $X^L_{(k,i)}$ e $X^L_{(k,j)}$ que representam as coordenadas dos ponto i e j, respetivamente.

Posteriormente, os valores de c são organizados por ordem crescente, sendo que os valores mais elevados representam as esquinas e os valores menores, representam os planos.

De forma a distribuir as zonas de interesse no mundo, a cada leitura o mundo observado é dividido em quatro partes iguais, sendo que cada uma dessas partes apenas pode apresentar dois pontos de esquina e quatro de plano.

Para diminuir a possibilidade de erro, algumas restrições extra são aplicadas [5]. Duas delas são possíveis de entender através da Figura 2.3.

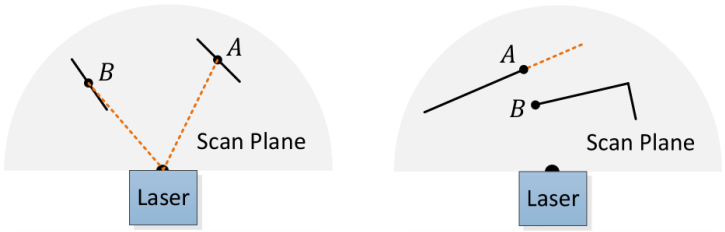


Figura 2.3: Pontos a descartar.[4]

A imagem apresentada à esquerda, representa a eliminação de pontos paralelos ao *Beam* do laser, como é possível observar no caso do ponto B. Enquanto que à direita temos os pontos de zonas tapadas por outro objeto, ponto A.

Em ambos os casos os pontos são descartados pois são considerados de baixa fiabilidade, uma vez que o primeiro caso pode provocar erros na leitura e o segundo pode estar a considerar uma esquina que é apenas a sombra do objeto mais próximo.

Na segunda parte, temos o estabelecimento de correspondência entre os pontos e as zonas de interesse correspondentes (Figura 2.4). Para tal, o algoritmo de odometria estima o movimento do LiDAR, em que t_k representa o tempo inicial do *scan* k que cria a nuvem de pontos $\mathcal{P}_k$, sendo esta re-projetada em t_{k+1} e renomeada como $\widehat{\mathcal{P}}_k$. De seguida, $\widehat{\mathcal{P}}_k$ em conjunto com $\mathcal{P}_{k+1}$ possibilitam a estimação do movimento do LiDAR.

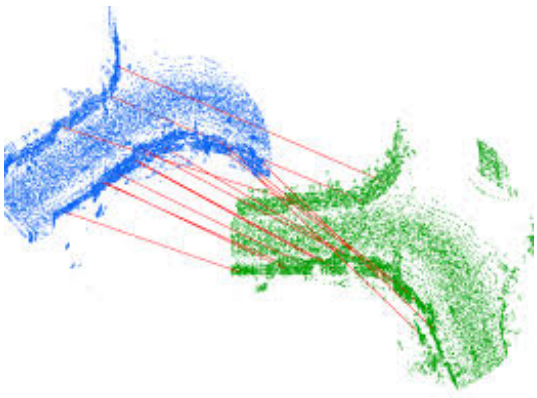


Figura 2.4: Correspondência entre zonas de interesse.[6]

A existência destas duas nuvem de pontos em paralelo possibilitam também a implementação da primeira parte descrita neste capítulo, pois em $\mathcal{P}_{k+1}$ detetamos os pontos de esquinas e planos, definidos nos conjuntos $\mathcal{E}_{k+1}$ e $\mathcal{H}_{k+1}$ que serão comparados com os pontos recolhidos durante o *scan* $\mathcal{P}_k$.

Por fim, nesta fase é ainda minimizada a distorção da nuvem de pontos através da correção da estimação da posição dos pontos de interesse nos diferentes *scans*. Assim, para um dado ponto de esquina i $\in \widehat{\mathcal{E}}_{k+1}$ e (j, l) pontos correspondentes da

esquina que pertencem a $\widehat{\mathcal{P}}_k$. Com estes dados podemos obter a distância entre os pontos e a esquina, representado na Equação 2.2.

$$d_{\mathcal{E}} = \frac{\left|(\widehat{\mathcal{X}}^L_{(k+1,i)} - \widehat{\mathcal{X}}^L_{(k,j)}) * (\widehat{\mathcal{X}}^L_{(k+1,i)} - \widehat{\mathcal{X}}^L_{(k,l)})\right|}{\left|(\widehat{\mathcal{X}}^L_{(k,j)} - \widehat{\mathcal{X}}^L_{(k,l)})\right|} \tag{2.2}$$

Onde, $\widehat{\mathcal{X}}^L_{(k+1,i)}$, $\widehat{\mathcal{X}}^L_{(k+1,j)}$ e $\widehat{\mathcal{X}}^L_{(k+1,l)}$ representam as coordenadas dos pontos i, j e l, respetivamente.

Para os pontos de plano, temos um dado ponto do plano i $\in \widehat{\mathcal{H}}_{k+1}$ e (j, l, m) pontos correspondentes do plano que pertencem a $\widehat{\mathcal{P}}_k$. Com estes dados podemos obter a distância entre os pontos e o plano, representado na Equação 2.3.

$$d_{\mathcal{H}} = \frac{\left|\begin{matrix}(\widehat{\mathcal{X}}^L_{(k+1,i)} - \widehat{\mathcal{X}}^L_{(k,j)}) \\ (\widehat{\mathcal{X}}^L_{(k,j)} - \widehat{\mathcal{X}}^L_{(k,l)}) * (\widehat{\mathcal{X}}^L_{(k,j)} - \widehat{\mathcal{X}}^L_{(k,m)})\end{matrix}\right|}{\left|(\widehat{\mathcal{X}}^L_{(k,j)} - \widehat{\mathcal{X}}^L_{(k,l)}) * (\widehat{\mathcal{X}}^L_{(k,j)} - \widehat{\mathcal{X}}^L_{(k,m)})\right|} \tag{2.3}$$

Nesta equação observamos a mesma nomenclatura utilizada anteriormente para as coordenadas dos pontos, com a adição de $\widehat{\mathcal{X}}^L_{(k+1,m)}$, que por sua vez, representa as coordenadas do ponto m.

Na terceira parte, temos a estimação do movimento. Assumindo que a velocidade angular e linear do LiDAR são constantes, uma vez que esta assunção se aproxima da realidade e permite fazer a interpolação da posição que servirá de base para a transformação dos pontos de diferentes *scans*, calculada através da Equação 2.4.

$$T^L_{(k+1,i)} = \frac{t_i - t_{k+1}}{t - t_{k+1}} T^L_{k+1} \tag{2.4}$$

Onde temos que T^L_{k+1} representa a transformação entre [t_{k+1}, t_k] e contém as restrições do LiDAR nos seus seis graus de liberdade definidos por [t_x, t_y, t_z, θ_x, θ_y, θ_z]^T. Desta forma, obtemos $T^L_{(k+1,i)}$ como a transformação entre [t_{k+1}, t_i].

Para estabelecer um movimento, é necessário primeiro estabelecer uma relação geométrica entre os pontos extraídos em $\mathcal{E}_{k+1}$ e $\mathcal{H}_{k+1}$, respetivamente. Para tal, podemos derivar a partir da Equação 2.4 onde iremos obter a Equação 2.5.

$$X^L_{(k+1,i)} = R * \widehat{X}^L_{(k+1,i)} + T^L_{(k+1,i)}(1:3) \tag{2.5}$$

Sabendo que $\widehat{\mathcal{X}}^L_{(k+1,i)}$ simboliza as coordenadas do ponto i em $\mathcal{E}_{k+1}$ ou $\mathcal{H}_{k+1}$, $\widehat{X}^L_{(k+1,i)}$ que representa as coordenadas do ponto i em $\widehat{\mathcal{E}}_{k+1}$ ou $\widehat{\mathcal{H}}_{k+1}$, $T^L_{(k+1,i)}(a:b)$ que simboliza a matriz de rotação entre a entrada a e b de $T^L_{(k+1,i)}$ e por fim R que representa a matriz de rotação definida pela formula de Rodrigues, presente na Equação 2.6.

$$R = e^{\widehat{\omega}\theta} = \mathbf{I} + \widehat{\omega}\sin\theta + \widehat{\omega}^2(1 - \cos\theta) \tag{2.6}$$

$$\theta = \left\| T^L_{(k+1,i)}(4:6) \right\| \tag{2.7}$$

$$\omega = \frac{T^L_{(k+1,i)}(4:6)}{\left\| T^L_{(k+1,i)}(4:6) \right\|} \tag{2.8}$$

Combinando a Equação 2.2/2.3 e as Equações 2.4 a 2.8 conseguimos derivar a relação geométrica entre o ponto de esquina e a própria esquina (Equação 2.9) ou entre o ponto de plano e o próprio plano (Equação 2.10).

$$f_{\mathcal{E}}(X^L_{(k+1,i)}, T^L_{k+1)}) = d_{\mathcal{E}}, i \in \mathcal{E}_{k+1} \tag{2.9}$$

$$f_{\mathcal{H}}(X^L_{(k+1,i)}, T^L_{k+1)}) = d_{\mathcal{H}}, i \in \mathcal{H}_{k+1} \tag{2.10}$$

Por fim, é possível resolver o movimento do LiDAR aplicando o método de *Levenberg-Marquardt*[7] nas Equações 2.9 e 2.10, obtendo uma equação não linear (Equação 2.11):

$$f(T^L_{k+1}) = d \tag{2.11}$$

Em que temos que cada linha de f representa um ponto de interesse e d a distância correspondente.

Para resolvermos a Equação 2.11 podemos assumir que J é definido na Equação 2.12 e aplicando o limite de d para zero, obtemos assim a Equação 2.13.

$$J = \frac{\partial f}{\partial T^L_{k+1}} \tag{2.12}$$

$$T^L_{k+1} \leftarrow T^L_{k+1} - (J^T J + \lambda diag(J^T J))^{-1} J^T d \tag{2.13}$$

Em que λ é o fator determinado pelo método de *Levenberg-Marquardt*.

2.1.3 Mapeamento por LiDAR

Esta parte do algoritmo corre a uma frequência mais baixa, como referido anteriormente, que é apenas chamado uma vez por cada *scan*. É também nesta fase que a distorção da nuvem de pontos é removida, sendo assim $\widehat{\mathcal{P}}_{k+1}$ gerada.

Utilizando a nomenclatura presente na Figura 2.5 onde temos que T^L_{k+1} representa o *scan* entre $[t_{k+1}, t_{k+2}]$, Q_k a nuvem de pontos aplicada no mapa e por fim T^W_k a posição do LiDAR no fim de um dado *scan*.

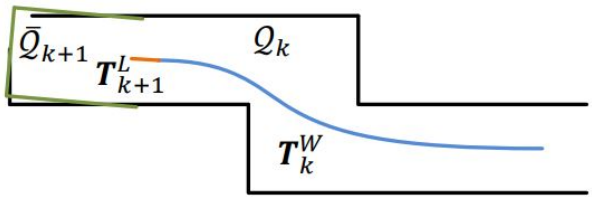


Figura 2.5: Processo de mapeamento.[4]

Com a conjugação destes dados temos então $\widehat{Q}_{k+1}$ como sendo os novos dados recolhidos, projetados no mapa. Sincronizando por fim estes pontos com os do *scan* anterior.

Consecutivamente os dados são armazenados em volumes de 10 metros cúbicos, sendo que aqueles que se intersetam em *scans* consecutivos são armazenados numa *3D KD-trees* [8].

Para os testes em ambientes *indoor*, o LiDAR foi colocado num carro que foi empurrado ao longo de um corredor estreito e comprido (Figura 2.6(a)) e ao longo de um átrio largo e curto (Figura 2.6(b)).

Para o caso dos testes em ambientes externos, o LiDAR foi colocado na frente de um carro que percorreu ruas rodeadas de árvores (Figura 2.6(c)) e um pomar (Figura 2.6(d)).

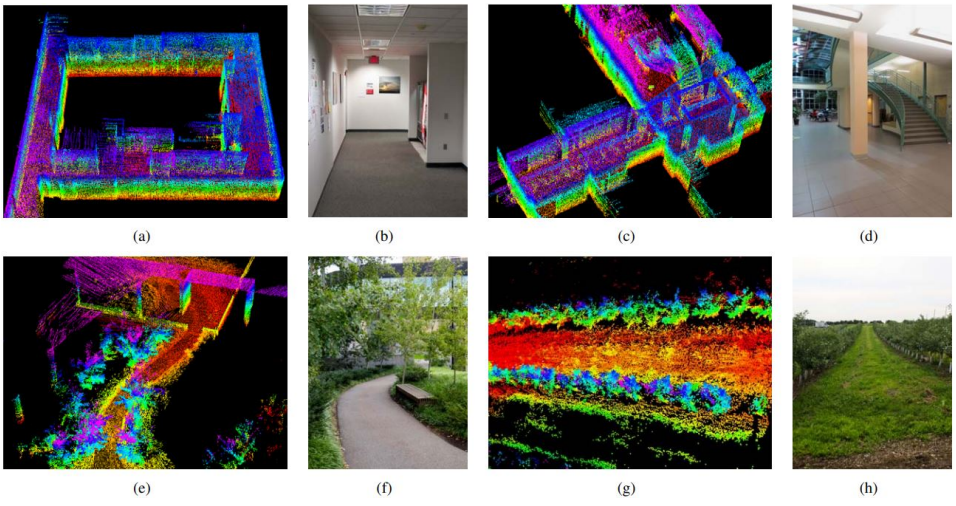


Figura 2.6: (a)-(b) Locais de recolha de *dataset* em ambiente interno;
(c)-(d) Locais de recolha de *dataset* em ambiente externo.[4]

Por fim, foi também utilizado um LiDAR estacionário utilizado como *ground truth*, fazendo uma comparação entre ambos e criando uma distribuição do erro, como é possível observar na Figura 2.7.

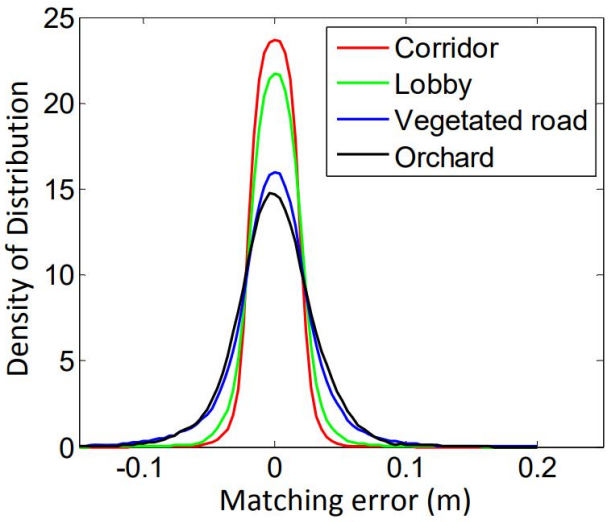


Figura 2.7: Distribuição do erro.[4]

Como esperado, podemos observar que os erros em ambiente externos são maiores, uma vez que os pontos de interesse são mais difíceis de distinguir e de isolar.

Como forma de melhoria de resultados, foi ainda testada a utilização de um IMU em conjunto com a implementação do autor em [4]. A utilização deste tem por objetivo reduzir a distorção causada pelas mudanças de direção provocadas pelo veículo através da fusão dos dados do acelerômetro, com os dados estimados pela implementação, podendo assim estimar a atitude do UAV, melhorando desta forma os dados do mapeamento e da estimação da odometria.

Comparando a Figura 2.8, que representa uma nuvem de pontos processada sem a utilização do IMU, com a Figura 2.9 que foi gerada com a utilização dos dados recolhidos pelo IMU, é possível perceber que a segunda apresenta zonas de interesse mais definidas, como tal, com menos ruído.

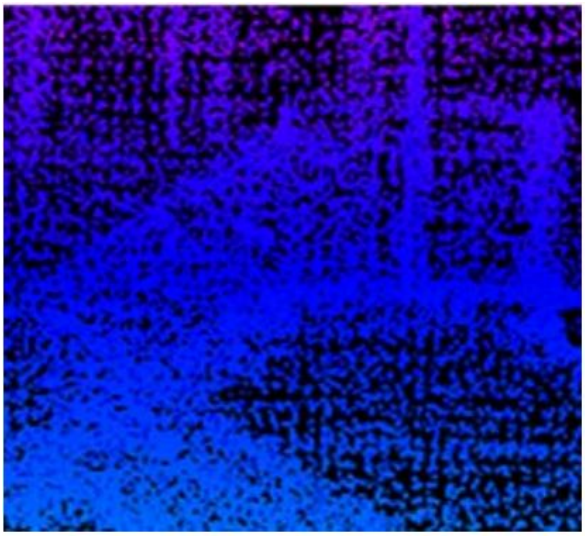


Figura 2.8: Nuvem de pontos final sem utilização do IMU.[4]

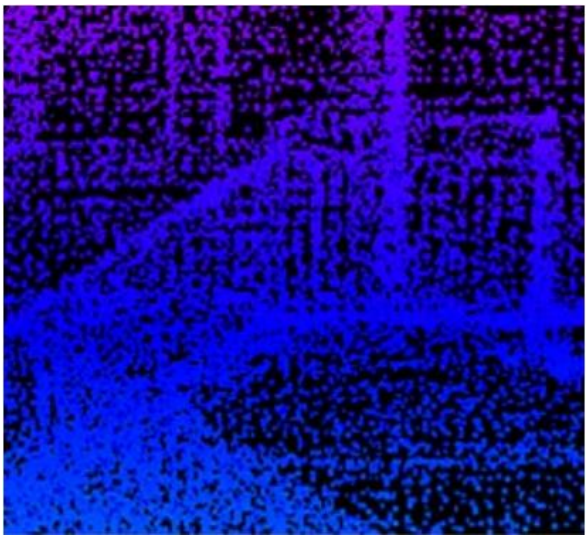


Figura 2.9: Nuvem de pontos final com utilização do IMU.[4]

Ambiente	Distância	Erro IMU	Erro da implementação	Erro combinado
Corredor	32m	16.7%	2.1%	0.9%
Átrio	27m	11.7%	1.7%	1.3%
Estrada	43m	13.7%	4.4%	2.6%
Pomar	51m	11.4%	3.7%	2.1%

Tabela 2.1: Estimação no erro do movimento com a introdução do IMU.[4]

Finalmente, foi também calculado o desvio que ocorreu na odometria. Para esse fim, no ambiente externo foi escolhido o pomar, uma vez que o veículo se encontra equipado com um *GPS/Inertial Navigation System (INS)* que possibilita o cálculo do desfasamento entre pontos coincidentes. Com isso obtemos os valores presentes na Tabela 2.1.

Ambiente	Distância	Erro	Distância	Erro
Corredor	58m	0.9%	46m	1.1%
Pomar	52m	2.3%	67m	2.8%

Tabela 2.2: Erro relativo da estimação da posição em dois testes consecutivos.[4]

Por fim, esta implementação foi também testada com o *dataset* do *KITTI Benchmark*, onde ao momento de redação desta dissertação, ainda se encontra como terceiro classificado de cento e cinquenta e seis avaliadas.

2.2 *Intensity Scan Context: LiDAR Odometry and Mapping in Real-time*

Nesta secção será apresentada uma versão mais elaborada da implementação anterior[4], com a inclusão de *loop closure* à implementação anterior. Para além da proposta de melhoria da abordagem normalmente utilizada de deteção de correspondência entre objetos geométricos, explora também as capacidades e possibilidades de utilização das leituras de intensidade disponibilizadas pelo LiDAR, mostrando e aplicando as mesmas de forma a atingir um melhor resultado final.

2.2.1 Calibração da intensidade e pré-processamento

Para potencializar a utilização das leituras de intensidade do LiDAR é necessário começar por calibrar as mesmas, uma vez que as leituras deste podem ser afetadas por inúmeros fatores, entre os quais, o material de que são compostos, rugosidade,

refletividade, distância (Figura 2.10 e Figura 2.11), restrições do equipamento, entre outros [9].

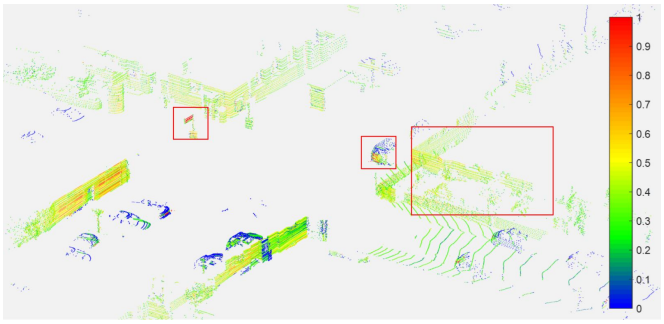


Figura 2.10: Comportamento em diferentes materiais.[10]

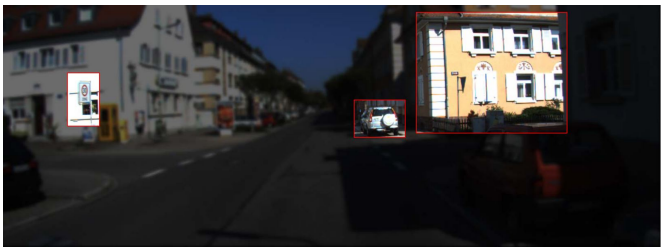


Figura 2.11: Representação no mundo real.[10]

Para executar a calibração do mesmo, é utilizada a Equação 2.14[10].

$$\eta_{cal} = \varphi(\eta_r, d) \tag{2.14}$$

Nesta equação temos η_r como sendo a leitura da intensidade, d a distância e por fim φ uma função de mapeamento, que descreve a influência da distância na intensidade recebida. Esta função serve de calibração com base em leituras prévias.

Além disso, é sabido que com a distância o ruído aumenta significativamente, como tal, também é adicionado um *threshold* L_{max} para remover pontos a distâncias com baixa fiabilidade.

2.2.2 *Intensity Scan Context*

Nesta secção é explicado como a intensidade é utilizada para ajudar na correlação entre pontos de interesse[11][12], uma vez que na maior parte do trabalho na área se aplica apenas à correlação de pontos através das suas características geométricas.

Tendo η, $[x, y, z]$ a posição do ponto e n a quantidade de pontos. Cada *scan* do LiDAR é definido por $\{L\}=\{\widehat{\mathcal{P}}_1, \widehat{\mathcal{P}}_2, ..., \widehat{\mathcal{P}}_n\}$ com cada ponto a ser definido por $\widehat{\mathcal{P}}=[x_k, y_k, z_k, \eta_k]$.

Para converter estes dados para coordenadas polares temos as Equações 2.15 a 2.17.

$$\widehat{\mathcal{P}} = [\rho_k, \theta_k, z_k, \eta_k] \tag{2.15}$$

$$\rho_k = \sqrt{x_k^2 + y_k^2} \tag{2.16}$$

$$\theta_k = \arctan \frac{y_k}{x_k} \tag{2.17}$$

Após aplicadas estas funções, as coordenadas são divididas com base nas suas coordenadas azimute e direções radiais em sectores N_s e anéis N_r, sendo cada segmento representado pela Equação 2.18.

$$\begin{aligned} \mathcal{S}_{ij} = \{\widehat{\mathcal{P}} \in \mathcal{P} | & \frac{i * L_{max}}{N_r} \leqslant \rho_k < \frac{(i+1) * L_{max}}{N_r}, \\ & \frac{j * 2\pi}{N_s} - \pi \leqslant \theta_k < \frac{(j+1) * 2\pi}{N_s} - \pi\} \end{aligned} \tag{2.18}$$

Onde i $\in$ [|1, N_s|] e $j \in$ [|1, N_r|].

A nuvem de pontos é então subdividida em N_s * N_r sub-espaços. Após isso, é criada uma função de codificação $\mathcal{K}$ que é aplicada na intensidade (Figura 2.19).

$$\begin{aligned} \eta_{ij} &= \mathcal{K}(\mathcal{S}_{ij}) \\ &= max_{\widehat{\mathcal{P}} \in \mathcal{S}_{ij}} * \eta_k \end{aligned} \tag{2.19}$$

Podendo assim gerar a Equação 2.20.

$$\Omega(i, j) = \eta_{ij} \tag{2.20}$$

Desta forma temos Ω, uma matriz 2D, que define a geometria e a intensidade do ambiente.

2.2.3 Identificação de pontos de interesse em *closed-loop*

O objetivo da identificação de pontos de interesse é fazer uma correspondência entre o local atual, $\mathcal{P}_k$, e os locais previamente visitados, $\mathcal{D}$=$\{\mathcal{P}_{k1}, \mathcal{P}_{k2}, ..., \mathcal{P}_{kn-1}\}$.

Com o aumento dos locais guardados, $\mathcal{D}$ também aumenta, o que consequentemente acresce o custo computacional. Para resolver este problema, esta implementação sugere uma solução em dois passos.

No primeiro passo, nomeado de *"Fast Geometry Re-identification"*, é utilizado um método de operações binária para indexação dos locais de interesse. Com esse propósito temos a descrição de um dado Ω, onde a sua distribuição geométrica pode ser descrita pela matriz binária $\mathcal{I}$ (Equação 2.21).

$$\mathcal{I}(x,y) = \begin{cases} false,\ if\ \Omega(x,y) = 0 \\ true,\ otherwise \end{cases} \tag{2.21}$$

Para um dado ponto de intensidade Ω^q, um dado candidato Ω^c e as suas relativas rotações binárias $\mathcal{I}^q$ e $\mathcal{I}^c$, a similaridade pode ser derivada através da Equação 2.22.

$$\varphi_g(\mathcal{I}^q, \mathcal{I}^c) = \frac{XOR(\mathcal{I}^q, \mathcal{I}^c)}{|\mathcal{I}^q|} \tag{2.22}$$

Como a coluna do vetor de *Iterative Closest Point* (ICP) representa uma direção azimutal, a rotação do *laser* transforma-se na coluna de *shift* Ω. Assim, para anular o ângulo do qual aquele ponto de interesse foi detetado temos a Equação 2.23.

$$\Phi_g(\mathcal{I}^q, \mathcal{I}^c) = max_{i \in [|1, Ns|]} \varphi_g(\mathcal{I}^q_i, \mathcal{I}^c) \tag{2.23}$$

Onde $\mathcal{I}^q_i$ é $\mathcal{I}^q$ com um *shift* aplicado pela coluna i.

Num segundo passo, com o nome de *"Intensity Structure Matching"* que procura a semelhança entre a intensidade de dois *Intensity Scan Context* (ISC)[12] Ω^q e Ω^c comparando coluna a coluna.

Sendo que v^q_i e v^c_i são a coluna i de cada um dos ICP, obtemos então a Equação 2.24.

$$\varphi_g(\Omega^q, \Omega^c) = \frac{1}{N_s} \sum_{i=0}^{N_s - 1} \left(\frac{v^q_i * v^c_i}{||v^q_i|| * ||v^c_i||}\right) \tag{2.24}$$

Novamente, para corrigir o ângulo de leitura dos dados é necessário aplicar $\mathcal{K}$, como mencionado no passo anterior. Com esse efeito, é feita uma comparação entre Ω^q_k e Ω^c onde o primeiro representa Ω^q *shiftado* $\mathcal{K}$ colunas, que é calculado com base na Equação 2.25.

$$\Phi_i(\Omega^q, \Omega^c) = \varphi_i(\Omega^q_k, \Omega^c) \tag{2.25}$$

2.2.4 Validação da consistência das medidas obtidas

Com os passos discutidos anteriormente, é inevitável que ocorra perda de informação. Assim sendo é importante verificar a consistência dos dados antes de fechar o *loop*. Para esse fim, são feitos dois tipos de verificação.

A primeira é uma verificação de consistência temporal apresentada na Equação 2.26.

$$P(\mathcal{P}_m, \mathcal{P}_n) = \frac{1}{N} \sum_{k=1}^{N} (\Phi_g(\mathcal{I}_{m-k}, \mathcal{I}_{m-k}) + \Phi_i(\Omega_{m-k}, \Omega_{n-k})) \tag{2.26}$$

Em que N representa o número de *frames*.

A segunda verificação é a de consistência geométrica, semelhante à aplicada em [13]. Posteriormente é ainda aplicado iterativamente o método de *ICP*[14] tentando assim encontrar o menor erro entre o ponto de interesse e o candidato proveniente do *scan*.

De forma a testar a implementação proposta, foram executados testes de interior num armazém, e testes de exterior recorrendo ao *dataset* disponibilizado pelo *KITTI*.

No caso dos testes em ambiente interior, os dados foram recolhidos num ambiente real, com máquinas e humanos a circular no ambiente. A odometria utilizada no veículo foi a odometria do LiDAR assim como a dos *encoders* das rodas.

Como *groud-truth* foi utilizada uma câmara montada na frente do veículo, que devido a diferentes ângulos de visão, não foi capaz de perceber que o robô tinha começado e terminado a recolha de dados no mesmo local. Pelo contrário a abordagem proposta conseguiu detetar esta coincidência devido à abordagem independente do ângulo (Figura 2.12) de captação de dados e posterior aplicação de *ICP*[14].

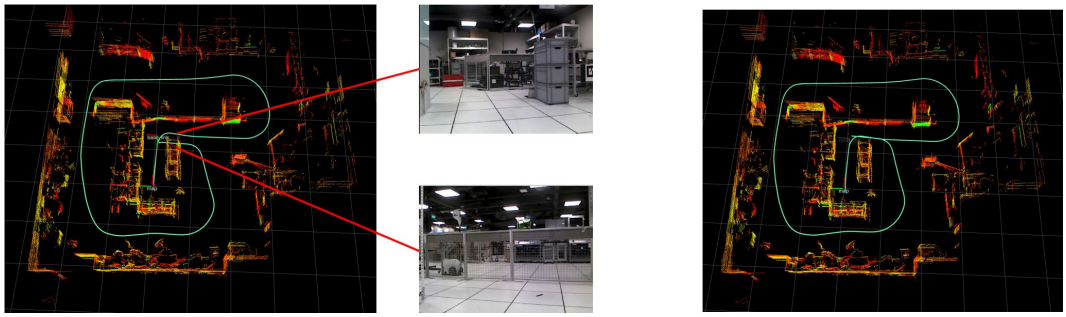


Figura 2.12: Da esquerda para a direita podemos ver, o mapeamento executado sem *loop closure*, ao centro o local q foi observado de ângulos diferentes e por fim o mapeamento executado com base na implementação sugerida.[10]

Nos caso dos testes com o *dataset* do *KITTI*, como esperado, apresenta bons resultados comparativamente a outros métodos, apresentado principalmente boa performance na identificação de partes do mundo onde ocorre *close loop*.

2.3 *Fast LiDAR Odometry and Mapping*

Nesta secção, assim como na anterior, é apresentada um versão baseada no método de LOAM, mas ao contrário da anterior, esta implementação [15] foca-se na otimização do mapeamento de pontos de interesse, uma vez que de modo a efetuar a compensação da distorção, utiliza um método não iterativo dividido em duas etapas, por oposição aos métodos anteriormente apresentados nos quais é aplicado um método iterativo.

2.3.1 Modelo de sensor e extração de pontos de interesse

Durante um dado *scan* k a nuvem de pontos detetada é nomeada de $\mathcal{P}_k$ e cada um dos seus pontos como $\widehat{\mathcal{P}}^{(m,n)}$ onde $m \in [1,M]$ e $n \in [1,N]$.

Devido à disposição do *hardware* a nuvem de pontos captada é mais densa horizontalmente do que verticalmente, como tal é mais produtivo focar o processamento na deteção de pontos de interesse horizontais, assim inicialmente é avaliada a rugosidade com base na Equação 2.27.

$$c_k^{(m,n)} = \frac{1}{|\mathcal{S}_k^{(m,n)}|} \sum_{\widehat{\mathcal{P}}^{(m,n)} \in \mathcal{H}_k^{(m,n)}} (||\widehat{\mathcal{P}}^{(m,j)} - \widehat{\mathcal{P}}^{(m,n)}||) \tag{2.27}$$

Onde, $c_k^{(m,n)}$ representa a rugosidade, $\mathcal{S}_k^{(m,n)}$ é o ponto horizontalmente adjacente de $\widehat{\mathcal{P}}^{(m,n)}$ e $|\mathcal{S}_k^{(m,n)}|$ o número de pontos da nuvem de pontos.

Após o cálculo da rugosidade do local, pontos de esquina $\mathcal{E}_k$ e de plano $\mathcal{H}_k$ são extraídos, através do seu valor de c assim como apresentado no Capítulo 2.1 para a implementação que serve de base.

2.3.2 Estimação do movimento e compensação de distorção

No trabalho apresentado em [4] o calculo da distorção é efetuada entre cada *scan*. Já o trabalho [15] propõe uma correção de distorção repartida em dois passos, com o objetivo de diminuir os requisitos computacionais.

Na primeira parte, uma vez que tratamos de *hardware* conhecido, é possível assumir as velocidades angular e linear como constantes.

Numa segunda parte, a distorção é recalculada após a estimação da posição e o resultado final é que será atualizado no mapa.

Para atingir este fim temos a Equação 2.28, que permite calcular a transformada entre dois *frames* consecutivos, sendo esse valor representado por $\mathcal{E}_{k-1}^k$. A Equação 2.29 que calcula a interpolação linear entre dois *scans* consecutivos, representado por $\mathbf{T}_k(\delta t)$, e a Equação 2.30 que corrige o *scan* $\mathcal{P}_k$, apresentado como $\widehat{\mathcal{P}}_k$.

$$\mathcal{E}_{k-1}^k = log(\mathbf{T}_{k-2}^{-1}\mathbf{T}_{k-1}) \tag{2.28}$$

$$\mathbf{T}_k(\delta t) = \mathbf{T}_{k-1} exp\left(\frac{N-n}{N}\mathcal{E}_{k-1}^k\right) \tag{2.29}$$

$$\widehat{\mathcal{P}}_k = \{\mathbf{T}_k(\delta t)\widehat{\mathcal{P}}^{(m,n)} | \widehat{\mathcal{P}}^{(m,n)} \in \mathcal{P}_k\} \tag{2.30}$$

Os pontos aos quais foi aplicada a matriz de transformação serão usados para estimar a posição do robô.

2.3.3 Estimação da posição

Para estimar a posição, é efetuada uma correspondência entre os pontos de interesse encontrados e os já existentes no mapa. Este processo requeria bastante tempo de processamento caso fosse feito iterativamente, contudo para otimizar este processo, esta informação é armazenada em *3D KD-trees*, da mesma forma que a implementação do trabalho [4].

Posteriormente, para calcular a posição estimada dos pontos de interesse do último *scan* no mundo, é efetuada a redução da distância entre o ponto de interesse e a zona de interesse correspondente, para tal, no caso dos pontos de esquina é utilizada a Equação 2.31 onde p_n é o vetor unitário definido na Equação 2.32.

$$f_{\mathcal{E}}(p_{\mathcal{E}}) = p_n \cdot ((T_k p_{\mathcal{E}} - p^g_{\mathcal{E}}) * n^g_{\mathcal{E}}) \tag{2.31}$$

$$p_n = \frac{(T_k p_{\mathcal{E}} - p^g_{\mathcal{E}}) * n^g_{\mathcal{E}}}{||(T_k p_{\mathcal{E}} - p^g_{\mathcal{E}}) * n^g_{\mathcal{E}}||} \tag{2.32}$$

Por outro lado, no caso dos pontos de plano temos a Equação 2.33.

$$f_{\mathcal{S}}(p_{\mathcal{H}}) = (T_k p_{\mathcal{H}} - p^g_{\mathcal{H}}) * n^g_{\mathcal{H}} \tag{2.33}$$

De forma a otimizar os cálculos anteriores, é ainda efectuada uma distribuição de pesos entre os pontos, uma vez que foi detetado que pontos de plano com um c mais elevado e pontos de esquina com c mais baixos são mais consistentes entre *scans*.

Desta forma a sua rugosidade é novamente utilizada para atribuir pesos a cada um dos pontos, tal como definido nas Equações 2.34 e 2.35.

$$W(p_{\mathcal{E}}) = \frac{exp(-c_{\mathcal{E}})}{\sum_{p^{(i,j)} \in \widehat{\mathcal{E}}_k} exp(-{c_k}^{(i,j)})} \tag{2.34}$$

$$W(p_{\mathcal{H}}) = \frac{exp(c_{\mathcal{H}})}{\sum_{p^{(i,j)} \in \widehat{\mathcal{H}}_k} exp({c_k}^{(i,j)})} \tag{2.35}$$

Em seguida, a nova posição do ponto de interesse é calculada com os novos valores, utilizando a Equação 2.36, que pode ser derivada utilizando o método de *Gauss-Newton* apresentado na Equação 2.37.

$$min_{T_k} \sum W(p_{\mathcal{E}}) f_{\mathcal{E}}(p_{\mathcal{E}}) + \sum W(p_{\mathcal{H}}) f_{\mathcal{H}}(p_{\mathcal{H}}) \tag{2.36}$$

$$J_p = \frac{\partial Tp}{\partial \delta \mathcal{E}} = \lim_{\delta_{\mathcal{E}} \to 0} \frac{(exp(\delta \mathcal{E}) Tp - Tp)}{\delta \mathcal{E}} = \begin{bmatrix} I_{3*3} & -[Tp] \\ 0_{1*3} & 0_{1*3} \end{bmatrix} \tag{2.37}$$

Para finalizar, este processo é resolvido iterativamente até que ambos os valores convirjam, obtendo assim a posição atual T_k^*.

2.3.4 Mapeamento e atualização da compensação de distorção

No caso desta implementação, a correção da distorção é efetuada com base nos pontos de plano e de esquina, quando a distorção destes ultrapassa um dado *threshold* previamente definido. Esta abordagem é mais eficiente computacionalmente, mas menos precisa que a utilizada da implementação do Capítulo 2.1[15].

Para melhorar os resultados do passo anterior, T_k^*, é utilizado pela Equação 2.38 que por sua vez é utilizada pela Equação 2.39 para uma otimização final do mapa.

$$\Delta\mathcal{E}^* = log({T_{k-1}}^{-1} \cdot \ T_k^*) \tag{2.38}$$

$$\widehat{P}_k^* = \{exp\left(\frac{(N-n)}{N} \cdot \Delta\mathcal{E}^*\right) \widehat{\mathcal{P}}^{(m,n)} | \widehat{\mathcal{P}}^{(m,n)} \in \mathcal{P}_k\} \tag{2.39}$$

Por fim, após cada atualização o mapa é passado por um filtro *3D Voxel Grid* [16] para reduzir os requisitos de armazenamento em memória.

Assim como nas implementações anteriores, também nesta foram executados testes com o *dataset* fornecido pela *KITTI framework* e com vários *dataset* próprios para o caso do ambiente *indoor*, dos quais uma simulação, um armazém e uma sala.

No caso dos testes com os *datasets* fornecidos pela *KITTI framework*, foi feito um estudo comparativo (Figura 2.13) entre diversas implementações, das quais LOAM[4], e LeGO-LOAM[17].

Para uma comparação mais justa, não foram utilizados métodos de *loop closure* ou os dados do IMU.

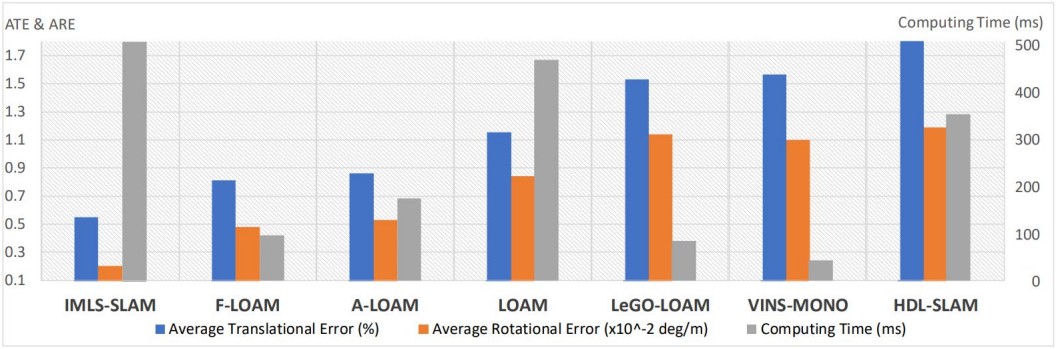


Figura 2.13: Comparação entre as diversas implementações.[15]

Através da comparação efetuada, é possível perceber que, mesmo o trabalho [15] não sendo a implementação com o tempo de computação mais baixo, é aquela que apresenta a melhor conjugação entre o tempo de computação e os erros de translação e rotação.

No caso do *dataset* recolhido pelos autores no trabalho [15] no armazém com ambiente fabril (Figura 2.14), foram executados três testes, o primeiro sem a compensação de distorção aplicado no *F-LOAM*[15] , o segundo com a compensação da distorção e por fim a implementação do trabalho [4]. Sendo que os resultados finais são apresentados na Tabela 2.3.

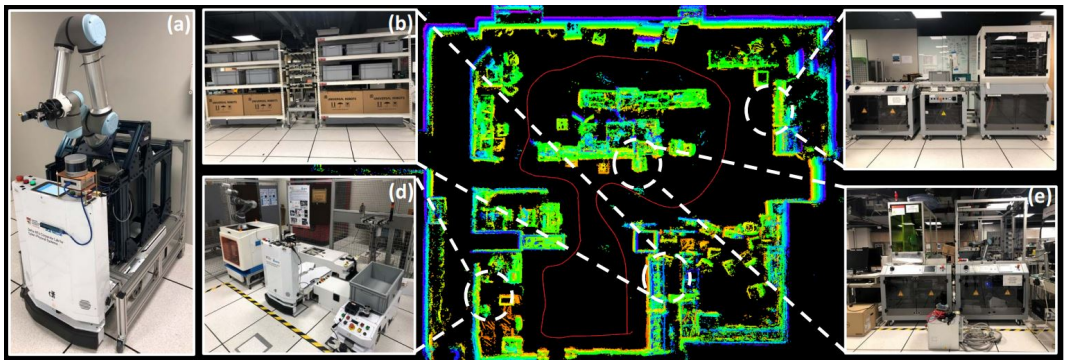


Figura 2.14: Ambiente de recolha do *dataset* em ambiente *indoor*.[15]

Implementação	Tempo de computação(ms/frame)	Precisão(cm)
F-LOAM sem compensação de distorção	66.33	2.132
F-LOAM[15]	69.07	2.037
LOAM[4]	84.52	2.052

Tabela 2.3: Comparação entre tempos de computação e precisão das várias abordagens.[15]

2.4 Visual-LiDAR Odometry and Mapping: Low-drift, Robust and Fast

A implementação [18] apresentada nesta secção consiste na combinação do método [4] com informação visual[19], com o objetivo otimizar a identificação de pontos de interesse(Figura 2.15).

A câmara que capta dados a uma frequência de 60Hz, estima o movimento, por outro lado, o LiDAR com uma frequência de 1Hz redefine a estimação da posição e remove distorções na nuvem de pontos.

Assim como no método [4], não foi utilizado *Loop Closure* para testar mais profundamente a implementação em causa.

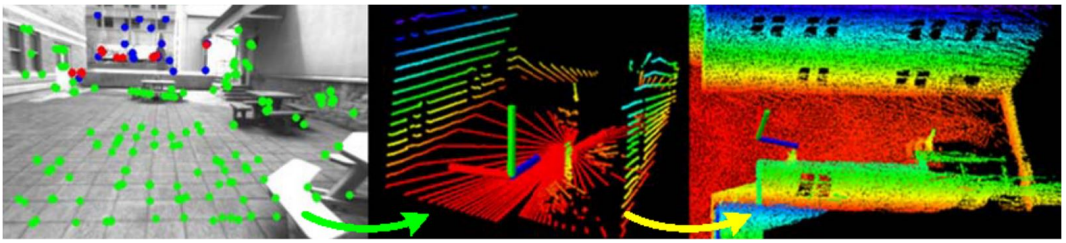


Figura 2.15: Conjugação dos dados captados pela câmera, assim como pelo LiDAR.[18]

2.4.1 Arquitetura de *Software*

Na Figura 2.16 podemos ver a arquitetura do *software*, onde temos a explicação do tratamento dos dados de cada um dos sensores.

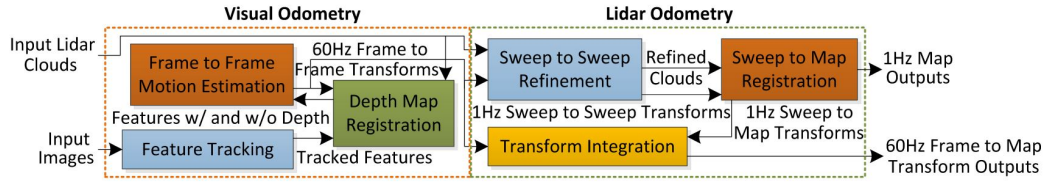


Figura 2.16: Arquitetura de *software*.[18]

No primeiro bloco "*Visual Odometry*", é executado o processamento *frame* a *frame* do movimento do robô. Para essa finalidade são extraídos pontos de interesse das imagens recolhidas e é guardado um mapa de profundidades com a informação dos mesmos. Posteriormente é feita a correspondência destes de forma a calcular o movimento.

No segundo bloco "*LiDAR Odometry*", executado a uma taxa de 1Hz, ocorre o processamento da nuvem de pontos. Numa primeira iteração temos o processamento do ajuste entre *scans* consecutivos e a remoção da distorção da nuvem de pontos percecionada, de seguida essa nuvem de pontos é associada a uma posição do mapa e por fim temos o cálculo da posição do sensor com base em ambos os sensores.

2.4.2 Métodos de estimação da odometria com base em sistemas de visão

Neste capítulo será detalhado os métodos de estimação da odometria com base em sistemas de visão. Como falado anteriormente, a odometria retirada das câmaras consiste na fusão da extração de pontos de interesse e um mapa de profundidades (Figura 2.17) das mesmas calculado através dos dados recolhidos da camera monocular. Este mapa consiste em distâncias $^{\mathcal{S}}$z[18].

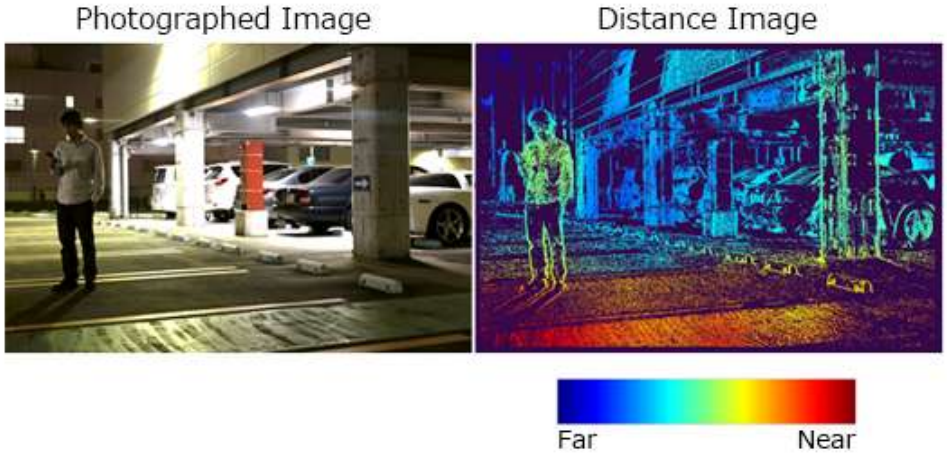


Figura 2.17: Exemplo de criação de um mapa de profundidades através de uma imagem.[18]

A estimação do movimento advém também de dois outros fatores, como triangulação da profundidade com base na estimação da distância entre pontos de interesse coincidentes em diferentes *frames*.

Na Equação 2.40 temos a formulação do movimento.

$$^{\mathcal{S}}X_i^k = R \ {}^{\mathcal{S}}X_i^{k-1} + T \tag{2.40}$$

Sendo que k representa o *frame* em questão, i representa um dado ponto de interesse e $^{\mathcal{S}}X_i^k$ as coordenadas desse mesmo ponto em $\mathcal{S}^k$ e que pode ser representado como apresentado na Equação 2.41 quando o mapa de distâncias ao ponto de interesse é conhecido, ou quando este não é conhecido pela Equação 2.42 onde $||^{\mathcal{S}}\widehat{X}_i^k|| = 1$.

$$^{\mathcal{S}}X_i^k = [^{\mathcal{S}}x_i^k, ^{\mathcal{S}}y_i^k, ^{\mathcal{S}}z_i^k] \tag{2.41}$$

$$^{\mathcal{S}}\widehat{X}_i^k = [^{\mathcal{S}}\widehat{x}_i^k, ^{\mathcal{S}}\widehat{y}_i^k, ^{\mathcal{S}}\widehat{z}_i^k] \tag{2.42}$$

No caso da distância estar disponível, é possível associa-la com $^{\mathcal{S}}\widehat{X}_i^{k-1}$, pois a distância ao estado atual é ainda desconhecida, uma vez que os dados do movimento entre *frames* e a triangulação ainda não são conhecidos.

Definindo a distância desconhecida como $^{\mathcal{S}}\widehat{d}_i^k$ temos então que $^{\mathcal{S}}\widehat{d}_i^k = ||^{\mathcal{S}}\widehat{X}_i^k||$ o que substituindo na Equação 2.40 e combinando a primeira e a segunda linha com a terceira, conseguimos eliminar $^{\mathcal{S}}\widehat{d}_i^k$ obtendo assim as Equações 2.43 e 2.44.

$$(^{\mathcal{S}}\widehat{z}_i^k R_1 - {}^{\mathcal{S}}\widehat{x}_i^k R_3)^{\mathcal{S}}X_i^{k-1} + {}^{\mathcal{S}}\widehat{z}_i^k T_1 - {}^{\mathcal{S}}\widehat{x}_i^k T_3 = 0 \tag{2.43}$$

$$({}^{\mathcal{S}}\widehat{z}_i^k R_2 - {}^{\mathcal{S}}\widehat{y}_i^k R_3)\,{}^{\mathcal{S}}X_i^{k-1} + {}^{\mathcal{S}}\widehat{z}_i^k T_2 - {}^{\mathcal{S}}\widehat{y}_i^k T_3 = 0 \tag{2.44}$$

No caso da distância em k-1 ser desconhecida, tanto ${}^{\mathcal{S}}\widehat{X}_i^k$ como ${}^{\mathcal{S}}\widehat{X}_i^{k-1}$ são desconhecidos. Como tal substituindo na Equação 2.40 pelos valores ${}^{\mathcal{S}}\widehat{d}_i^k$ ${}^{\mathcal{S}}\widehat{X}_i^k$ e ${}^{\mathcal{S}}\widehat{d}_i^{k-1}$ ${}^{\mathcal{S}}\widehat{X}_i^{k-1}$, respetivamente. Obtemos então a Equação 2.45.

$$\begin{bmatrix} -{}^{\mathcal{S}}\widehat{y}_i^k T_3 + {}^{\mathcal{S}}\widehat{z}_i^k T_2 \\ {}^{\mathcal{S}}\widehat{x}_i^k T_3 - {}^{\mathcal{S}}\widehat{z}_i^k T_1 \\ -{}^{\mathcal{S}}\widehat{x}_i^k T_2 + {}^{\mathcal{S}}\widehat{y}_i^k T_1 \end{bmatrix} R\,{}^{\mathcal{S}}\widehat{X}_i^{k-1} = 0 \tag{2.45}$$

Para finalizar, são atribuídos pesos com base nos resultados anteriores, onde pontos de interesse com valores residuais baixos tem um maior peso e aqueles com valores mais elevados do que o *threshold* são considerados *outliers*.

Para evitar *loops* infinitos o processo termina quando os valores convergem ou o numero máximo de iterações é atingido.

Para esta implementação dos dados são guardados numa *2D KD-tree* utilizando uma distância e dois ângulos.

2.4.3 Odometria por LiDAR

O procedimento explicado neste capítulo irá ajudar a refinar informação captada pela câmara, sendo que é dividida em dois passos, o primeiro que consiste na correspondência de informação *scan* a *scan* e um segundo que consiste na correspondência entre a informação do *scan* com a informação do mapa.

Numa primeira iteração esta implementação segue o mesmo procedimento da implementação apresentada no Capítulo 2.1, onde são extraídos pontos de esquina e de plano. A partir desses dados recolhidos, temos então a correção do *drift* utilizando a Equação 2.46.

$$T_i' = T' \frac{t_i - t^m}{t^{m+1} - t^m} \tag{2.46}$$

Onde T_i' é a posição do ponto i sem a distorção, T' é a matriz de transformação e t representa o tempo em que aqueles dados foram percecionados. Toda esta informação é armazenada em duas *3D KD-trees*, uma para pontos de plano e outras para pontos de esquina.

Através da correspondência dos pontos de esquina com os de plano é possível extrair a Equação 2.47 que em conjunto com a Equação 2.46 nos permite determinar T'.

$$f({}^{\mathcal{S}}\widehat{X}_i^m, T_i') = d_i \tag{2.47}$$

Onde ${}^{\mathcal{S}}\widehat{X}_i^m$ são as coordenadas do ponto i e d_i a distância ao seu correspondente.

Por fim a segunda parte da remoção da distorção no mapa, é aplicado o método de ICP entre a nova nuvem de pontos percecionada e a associada ao mapa, de forma a fazer o *match* do resultado de ambas.

Os resultados foram divididos em duas partes, numa primeira fase foram feitos testes de precisão, com duas câmaras, uma com uma lente de 76º de abertura horizontal(H) e uma com lente *fisheye* de 185º de abertura horizontal (F).

Para cada um dos casos foram ainda desenvolvidos três testes diferentes, sendo que no primeiro teste, o trajeto continha dois *loops*, os quais eram utilizados para medir o erro em diversos pontos. No segundo teste, no qual existia apenas um *loop* sendo que o trajeto começava e terminava na mesma localização, . E por fim, o terceiro teste, que não continha qualquer *loop* e o erro foi calculado através da posição relativa da partida e da chegada.

Teste Nº	Distância	H-V	F-V	H-VL	F-VL
Teste 1 (Loop 1)	49m	1.1%	1.8%	0.31%	0.31%
Teste 1 (Loop 2)	47m	1.0%	2.1%	0.37%	0.37%
Teste 2	186m	1.3%	2.7%	0.63%	0.64%
Teste 3	538m	1.4%	3.1%	0.71%	0.73%

Tabela 2.4: Erro relativo. V-Apenas odometria visual, VL - ambas em conjunto.[18]

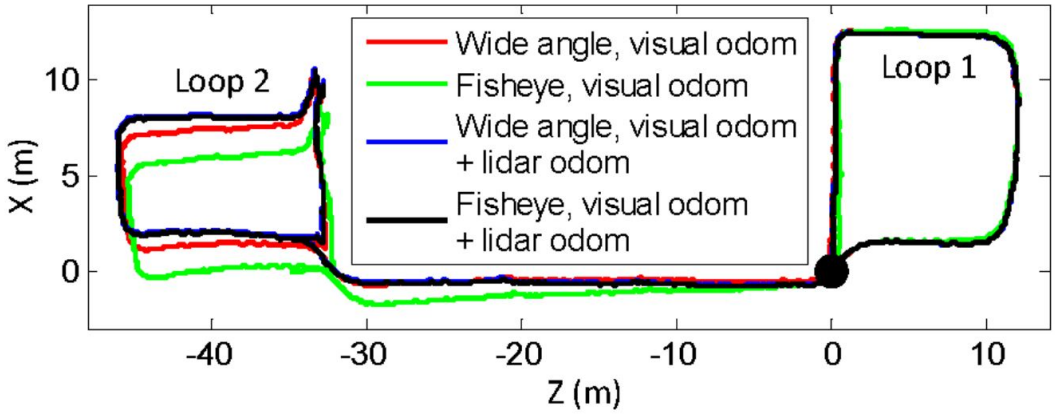


Figura 2.18: Dados de odometria visual (H-V e F-V) e da combinação de ambas (H-VL e F-VL).[18]

Como é possível observar na Figura 2.18, quando é apenas utilizada odometria visual a câmara *fisheye* apresenta um erro maior, mas quando esta é utilizada em conjunto com o LiDAR o erro diminui significativamente, aproximando-se dos valores finais da outra câmara utilizada quando também é utilizada a implementação completa.

Numa segunda parte temos os testes de robustez, em que o ambiente em causa são escadas em caracol onde o ângulo muda constantemente, sendo este o teste número 4 e um teste um corredor em que é percorrido inicialmente a uma baixa velocidade e posteriormente a uma velocidade mais elevada, apresentado na Tabela 2.4 como teste cinco.

Teste Nº	Distância	H-L	F-L	H-R	F-R
Teste 4	66m	0.67%	0.68%	Falhou	1.3%
Teste 5	54m	0.27%	0.28%	Falhou	0.39%

Tabela 2.5: Erro relativo. L - Movimento lento, R - Movimento rápido.[18]

Finalmente foi testado em condições extremas de iluminação, nos quais a luz estava apagada e era ligada a meio de percurso. Nos percursos efetuados com a luz apagada, o veículo apenas se guiava pelo LiDAR. Embora eficaz, a sugestão do autor para casos de baixa luminosidade é utilizar a implementação apresentada no trabalho [4], uma vez que apresenta melhores resultados[18].

2.5 Lego-LOAM - Lightweight and Ground-Optimized LiDAR Odometry and Mapping on Variable Terrain

Nesta secção iremos apresentar uma versão diferente e melhorada do trabalho [4] mas com especial foco na otimização para possibilitar a utilização da mesma em *hardware* com uma performance menor.

Esta abordagem [17] potencializa a segmentação da nuvem de pontos para descartar pontos que possam representar zonas de interesse de baixa confiança, sendo estas a deteção do chão a uma elevada densidade, mas sem pontos de interesse reais. Utiliza também algo a que o autor chama de *ground-optimized*, que é introduzido a partir de uma otimização a dois passos onde, na primeira fase, os pontos de interesse retirados do chão são utilizados para obter [t_z, θ_{roll}, θ_{pitch}], na segunda fase obtemos [t_x, t_y, θ_{yaw}] fazendo a correspondência de pontos de interesse de esquinas. A implementação apresenta ainda *loop closure*.

2.5.1 Arquitetura de Software

Na Figura 2.19 temos um pequeno resumo da arquitetura do sistema implementado.

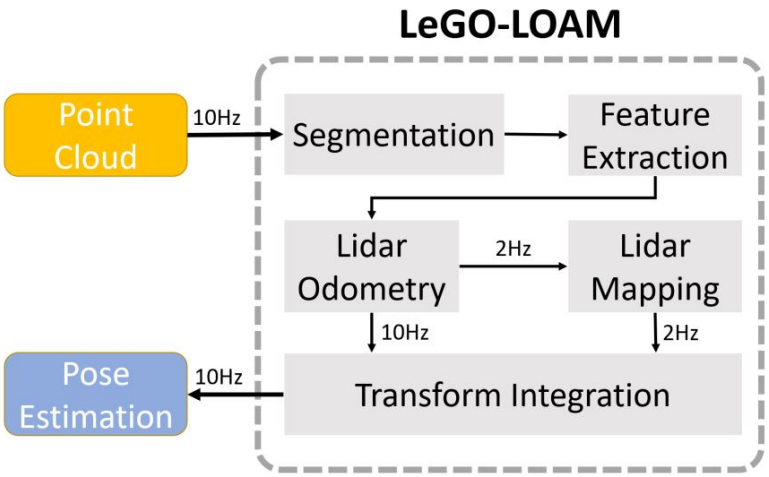


Figura 2.19: Arquitetura de *software*.[17]

Como é possível observar, a implementação é dividida em cinco módulos. O módulo "*Segmentation*", temos que $\mathcal{P}_k$=$\{\widehat{\mathcal{P}}_1, \widehat{\mathcal{P}}_2, ..., \widehat{\mathcal{P}}_n\}$ (Figura 2.20.a) são os pontos de um dado *scan* de seu tempo *t*. Após validar quais destes pontos são considerados válidos, são associadas distâncias r_i a cada um deles.

De seguida, é aplicado um método de segmentação da imagem onde todos os pontos [20] exceto os de chão serão agregados [21] em *clusters* e posteriormente, cada um desses *clusters* será reduzido a um único ponto (Figura 2.20.b), excetuando aqueles com menos de trinta pontos. A cada um desses pontos será ainda adicionada uma *label* que será discutida posteriormente.

De seguida ocorre a fase da "*Feature Extraction*"onde são extraídos pontos de interesse dos *clusters* anteriormente gerados assim como pontos de chão. Para esse fim é utilizada a Equação 2.48, onde é calculada a suavidade de uma dada superfície.

$$c = \frac{1}{|S| * ||r_i||} \left|\left| \sum_{j \in S, j \neq i} (r_i - r_j) \right|\right| \tag{2.48}$$

Em que c representa a rugosidade da superfície, $\mathcal{S}$ os pontos de um dado *scan* e por fim r_i e r_j que representam as coordenadas dos ponto i e j, respetivamente.

Assim como na implementação apresentada no Capítulo 2.1, também esta divide a imagem observada em diversas partes, para desta forma tentar distribuir os pontos de interesse pelo ângulo de visão. Posteriormente, em cada uma dessas secções, os pontos são ordenados pela sua suavidade, seguindo a abordagem da implementação acima mencionada. Por fim, o resultado obtido pode ser observado na Figura 2.20.d.

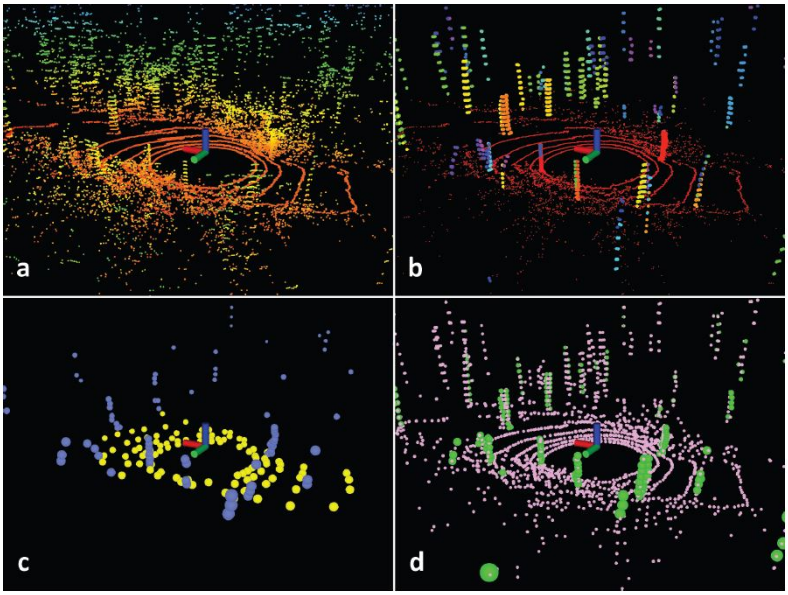


Figura 2.20: Diversas fases de processamento.[17]

Na parte nomeada "*LiDAR Odometry*", assim como nas implementações já descritas, ocorre o cálculo do movimento do robô. Esta implementação foi uma melhoria da apresentada em [4], sendo o foco dessa melhoria dois pontos principais:

- Correspondência de etiquetas - através das etiquetas anteriormente atribuídas, uma pré-seleção é efetuada de forma a que apenas pontos de interesse com etiquetas semelhantes sejam considerados para um dado caso;
- Otimização em dois passos do método de *Levenberg-Marquardt* - no caso desta implementação, primeiro serão processados os pontos de plano, possibilitando o cálculo de $[t_z, \theta_{roll}, \theta_{pitch}]$, de seguida serão processados os pontos de esquina com os valores anteriores utilizados como condições, obtendo assim os restantes valores $[t_x, t_y, \theta_{yaw}]$. Por fim, através dos valores calculados é então obtido o movimento entre dois *scans*.

Através de testes feitos pelo autor, foi possível perceber que com esta abordagem o tempo de processamento reduz-se em até 35%.

Na última parte, nomeada de "*LiDAR Mapping*"o objetivo é semelhante ao descrito nas implementações anteriores, sendo este refinar os dados criados no passo anterior. Assim como no passo anterior, também foi baseado na implementação de [4], sendo que neste caso, a principal diferença prende-se pela forma como a nuvem de pontos final é armazenada. Neste caso, cada um dos conjuntos de dados é armazenado com base nos conjuntos de pontos de interesse, sendo os pontos do instante t-1, M^{t-1}=$\{\{\mathcal{E}^1_e, \mathcal{H}^1_p\}, ..., \{\mathcal{E}^{t-1}{}_e, \mathcal{H}^{t-1}{}_p\}\}$, sendo que $\mathcal{E}^1_e$ representa os pontos de esquina e $\mathcal{H}^1_p$ os de superfície.

Desta forma o autor propões duas soluções para obter $\widehat{Q}^{t-1}$, sendo esta a nuvem de pontos que observa.

- A primeira sugestão é semelhante à utilizada na implementação que serviu de base [4] e baseia-se em filtrar todas os pontos de interesse que se encontram num raio de 100 metros para criar um mapa único com estes;
- A segunda sugestão, que apenas foi utilizada num dos testes, baseia-se em pegar num conjunto k de *scans* e aplica-los no mundo, só depois remover o *drift* que ocorreu entre cada um dos *scans*. Para obter este resultado utiliza técnicas como *ICP* e *Loop Closure*.

Os resultados apresentados pelo autor no trabalho [17] focam-se na comparação da implementação proposta com a implementação apresentada no trabalho [4], a base de desenvolvimento para todas as outras implementações.

O primeiro teste foi feito com recurso a um veículo de pequenas dimensões, em ambiente exterior. A primeira comparação que podemos retirar acenta na quantidade de pontos de interesse captados, com base na Figura 2.21. Como era esperado, a implementação discutida neste capítulo descarta inúmeros pontos com baixo interesse (Figura 2.21.d e Figura 2.21.e), como pontos na relva ou folhas, comparativamente aos recolhidos pela implementação apresentada no Capítulo 2.1 (Figura 2.21.b e Figura 2.21.c).

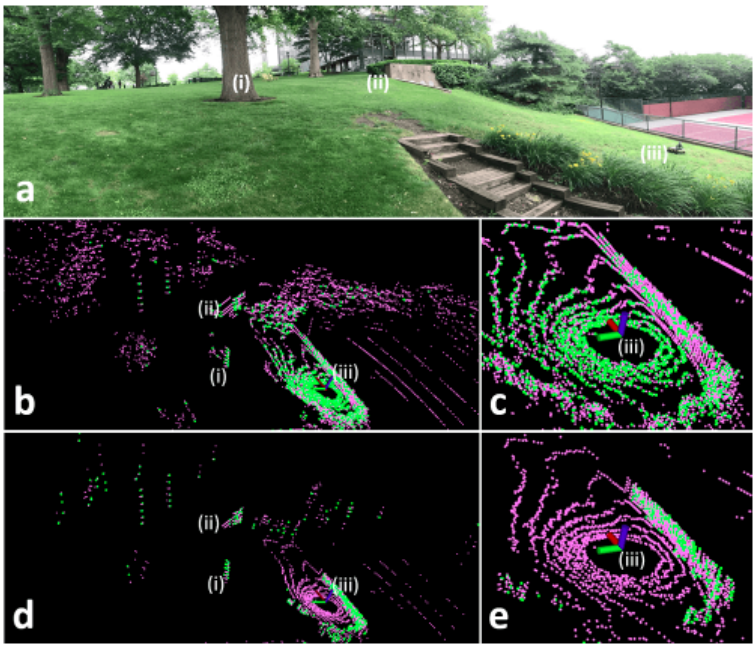


Figura 2.21: Pontos de interesse recolhidos.[17]

Para além da comparação anterior, temos também a estimação dos percursos percorridos, onde podemos ver que a implementação aqui sugerida (Figura 2.23) apresenta uma maior precisão do que a sua precedente (Figura 2.22).

Algo que é bastante nítido passa pela precisão do percurso gerado, mas também pela multiplicação da mesma árvore, quando utilizando o trabalho [4].

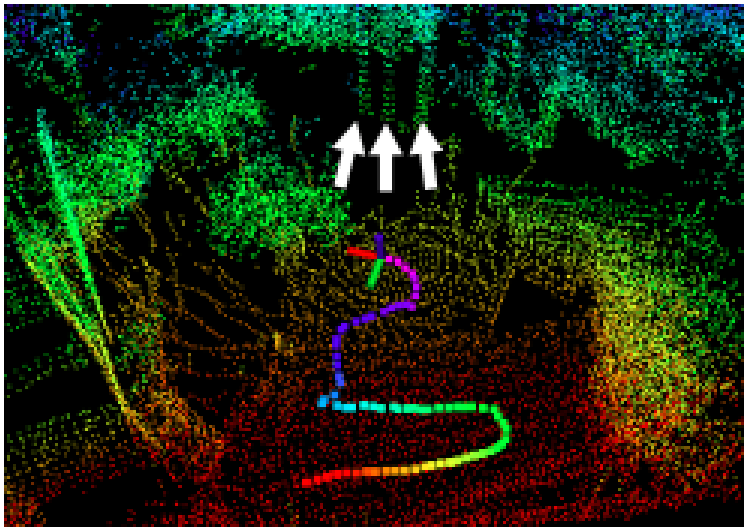


Figura 2.22: Reconstrução do percurso utilizando o trabalho [4].[17]

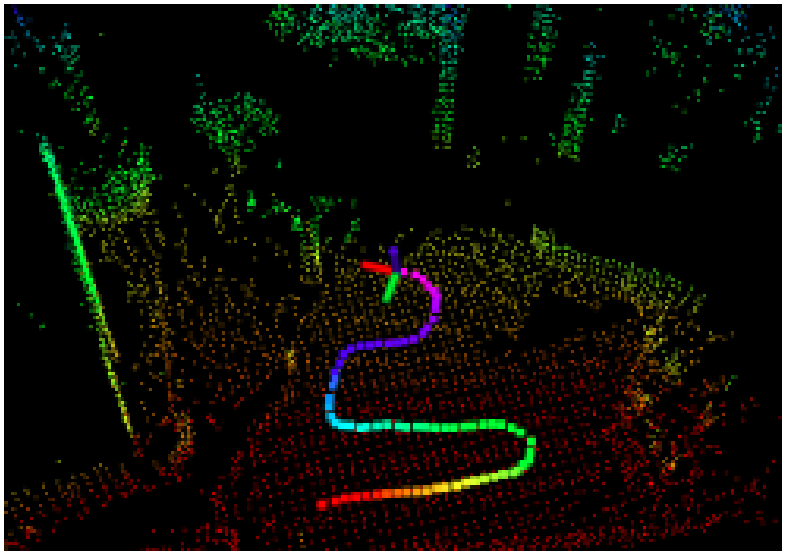


Figura 2.23: Reconstrução do percurso utilizando LeGO-LOAM.[17]

Para o segundo teste fora utilizados três *datasets* de grandes dimensões. Os dois primeiros foram recolhidos no campus da faculdade (Figura 2.24), num ambiente citadino comum, enquanto que o terceiro foi num terreno mais semelhante a trilhos, como tal mais irregular e com pontos de interesse menos distintos (Figura 2.25).

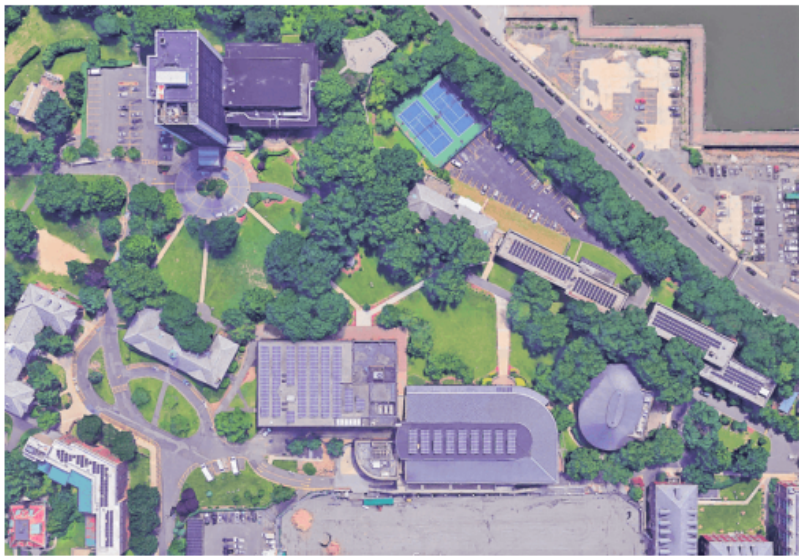


Figura 2.24: Campus da faculdade.[17]

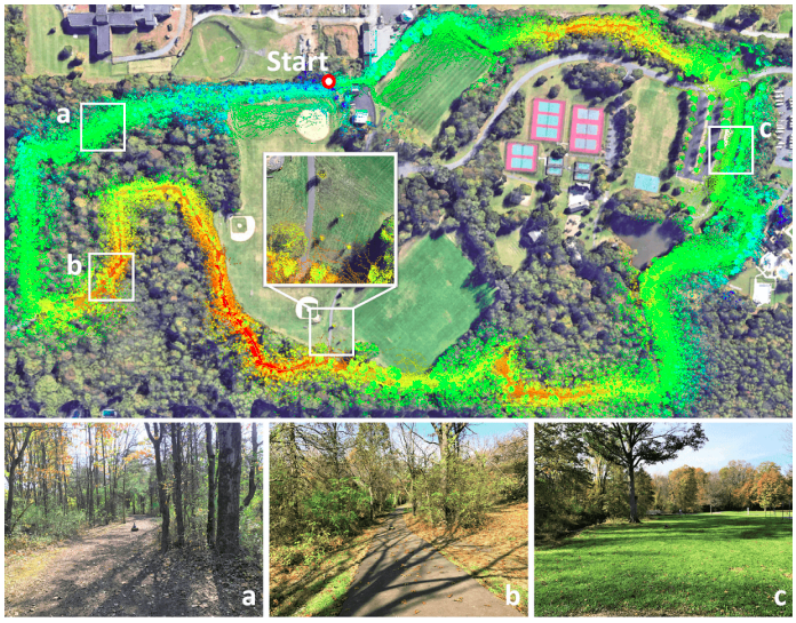


Figura 2.25: Trilha onde foi recolhido o *dataset*.[17]

- *Dataset* 1 - Este *dataset* foi recolhido com o objetivo de mostrar que a baixas velocidades e sem necessidade de movimentos bruscos, tanto a implementação do autor, como a apresentada no trabalho [4] apresentam valores muito semelhantes. Como é possível observar na Figura 2.26;

- *Dataset* 2 - Neste *dataset*, os dados foram recolhidos de forma muito semelhante ao *dataset* anterior, com uma passagem extra numa zona de passeio e com pontos de mais difícil identificação. Neste *dataset*, como é possível observar nas Tabelas 2.5 a 2.8 e representado na Figura 2.27, a implementação do autor supera-se à sua antecessora.

- *Dataset* 3 - Por fim, o terceiro *dataset* tem por objetivo testar ambos em condições extremas, estradas em gravilha, folhas, árvores.

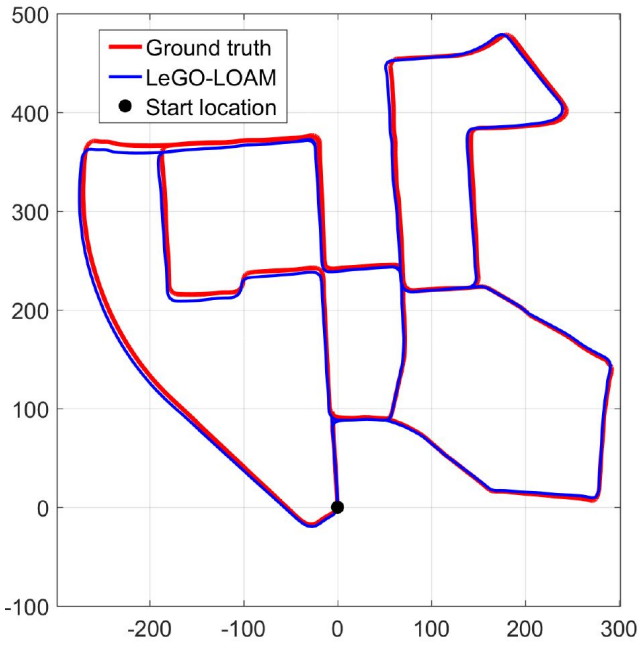


Figura 2.26: Resultados com base no *dataset* 1.[17]

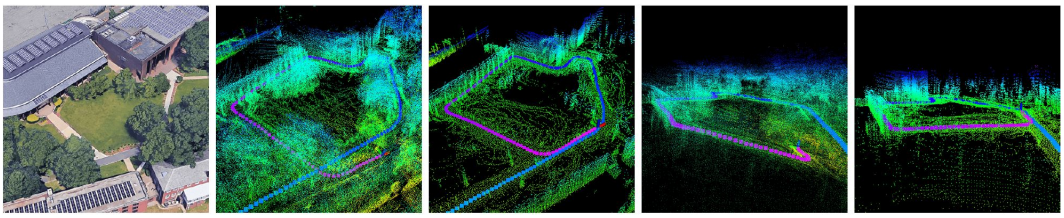


Figura 2.27: Da esquerda para a direita, na primeira imagem podemos observar uma imagem real do ambiente em que os dados foram recolhidos, na segunda e quarta imagens temos o comportamento do trabalho [4], enquanto que na terceira e quinta temos o comportamento do trabalho [17], o qual podemos observar que apresenta uma maior definição comparativamente com o trabalho [4].[17]

Como resumo destas experiências temos as Tabelas 2.5 a 2.8, onde nas duas primeiras tabelas se apresenta-se os valores dos dados recolhidos quando passados por ambas as implementações, que se encontram a correr num *Jetson JX-2* com um *Cortex-A57*. Enquanto que as duas últimas, representam o mesmo, mas utilizando como *hardware* um computador *i7-4710MQ*.

Dataset	Implementação	Roll	Pitch	Yaw	Total Rot.(º)
1	LOAM[4]	1.16	2.63	2.50	3.81
	Lego-LOAM[17]	0.46	0.91	1.98	2.23
2	LOAM[4]	7.05	5.06	9.40	12.80
	Lego-LOAM[17]	0.61	0.70	0.32	0.99
3	LOAM[4]	7.55	3.20	26.12	27.38
	Lego-LOAM[17]	4.62	5.45	2.95	7.73

Tabela 2.6: Erro relativo da estimação da rotação no momento de *loop closing* utilizando *Jetson*.[17]

Dataset	Implementação	X	Y	Z	Total Trans.(m)
1	LOAM[4]	1.33	2.91	0.43	3.23
	Lego-LOAM[17]	0.12	0.07	1.26	1.27
2	LOAM[4]	7.71	6.31	4.32	10.86
	Lego-LOAM[17]	0.04	0.10	0.34	0.36
3	LOAM[4]	34.61	56.19	21.46	69.40
	Lego-LOAM[17]	5.35	7.95	10.11	13.93

Tabela 2.7: Erro relativo da estimação da translação no momento de *loop closing* utilizando *Jetson*.[17]

Dataset	Implementação	Roll	Pitch	Yaw	Total Rot.(º)
1	LOAM[4]	0.28	1.98	1.74	2.65
	Lego-LOAM[17]	0.33	0.17	2.06	2.09
2	LOAM[4]	21.49	4.86	4.34	22.46
	Lego-LOAM[17]	0.18	0.85	0.64	1.08
3	LOAM[4]	6.27	3.08	4.83	8.50
	Lego-LOAM[17]	4.57	5.39	3.68	7.96

Tabela 2.8: Erro relativo da estimação da rotação no momento de *loop closing* utilizando *i7*.[17]

Dataset	Implementação	X	Y	Z	Total Trans.(m)
1	LOAM[4]	0.39	0.03	0.21	0.44
	Lego-LOAM[17]	0.03	0.02	0.22	0.22
2	LOAM[4]	1.39	2.59	11.63	11.99
	Lego-LOAM[17]	0.04	0.12	0.04	0.14
3	LOAM[4]	16.84	58.81	10.74	62.11
	Lego-LOAM[17]	6.69	7.79	10.76	14.87

Tabela 2.9: Erro relativo da estimação da translação no momento de *loop closing* utilizando *i7*.[17]

Por fim temos os testes realizados com o *dataset* disponibilizado pelo *KITTI Benchmark*, que assim como no caso anterior, foi feito de forma a comparar a performance da implementação do autor com a apresentada no Capítulo 2.1.

Desta comparação foram extraídas quatro métricas de comparação.

- Número de pontos de interesse - neste campo foi possível observar que a implementação do autor, devido aos filtros e abordagem acima explicados, reduziram o número de pontos de interesse em 29% a 72%;
- Número de iterações necessárias - neste caso foi testada a vantagem da utilização do método de *Levenberg-Marquardt* em dois passos, nos quais foi possível observar uma melhoria do tempo de processamento de entre 34% a 48%.
- Tempo de processamento - com esta métrica o objetivo foi comparar os tempos de processamento de ambos, sendo que foi possível observar dois pontos principais:
  - Com o *Hardware Jetson*, a implementação do Capítulo 2.1 ultrapassava os 100ms, o que levava à perda de leituras;
  - Com *Hardware* mais potente, a melhoria de tempo de processamento atingiu os 60%.
- Erro final na estimação da posição - Dados apresentados nas Tabelas 2.5 a 2.8.

2.6 Resultados do *KITTI benchmark*

Neste capítulo é apresentada a Tabela 2.9, onde é apresentado um resumo dos resultados finais dos testes para a *KITTI framework*. Desta forma, constitui uma primeira síntese de comparação entre algumas das implementações acima apresentadas.

	LOAM[4]	ISC-LOAM[10]	F-LOAM[15]	V-LOAM[18]
Posição no ranking KITTI	3	23	90	2
Tempo de processamento	0.1s	0.1s	0.1s	0.05s
Hardware	2 cores @ 2.5 Ghz (C/C++)	2 cores @ 2.5 Ghz (C/C++)	4 cores @ 3.0 Ghz (C/C++)	4 cores @ 3.0 Ghz (C/C++)
Parâmetros	Point to line /plane scan matching	Visual odometry: Harris corner features LiDAR odometry: Point to line /plane scan matching	geometry threshold =0.67 intensity threshold =0.93	- output frequency of visual odometry: 60 Hz - output frequency of LiDAR odometry: 20 Hz

Tabela 2.10: Resultados do *KITTI Benchmark*.[22]

Nas Tabelas 2.11, 2.12 2.13 e 2.14 os melhores valores de cada tabela estarão identificados a negrito, enquanto que os piores serão sublinhados

2.6.1 Erro de translação com base na distância

Distância (m)	LOAM[4]	ISC-LOAM[10]	F-LOAM[15]	V-LOAM[18]
100.0	0.008574%	0.009446%	0.022542%	**0.008477**%
200.0	0.006750%	0.007554%	0.019031%	**0.006640**%
300.0	0.005707%	0.006703%	0.018030%	**0.005615**%
400.0	0.005031%	0.006297%	0.017426%	**0.004960**%
500.0	0.004528%	0.006239%	0.017558%	**0.004470**%
600.0	0.004120%	0.006497%	0.017835%	**0.004042**%
700.0	0.003778%	0.006862%	0.017968%	**0.003665**%
800.0	0.003637%	0.007356%	0.018203%	**0.003482**%

Tabela 2.11: Erro médio de translação com base na distância, nas sequencias 11-21 segundo a *KITTI framework*.[22]

2.6.2 Erro de rotação com base na distância

Distância (m)	LOAM[4]	ISC-LOAM[10]	F-LOAM[15]	V-LOAM[18]
100.0	**0.000045**deg/m	0.000057deg/m	0.000153deg/m	**0.000045**deg/m
200.0	**0.000030**deg/m	0.000043deg/m	0.000101deg/m	**0.000030**deg/m
300.0	**0.000023**deg/m	0.000038deg/m	0.000083deg/m	**0.000023**deg/m
400.0	**0.000019**deg/m	0.000034deg/m	0.000073deg/m	**0.000019**deg/m
500.0	0.000017deg/m	0.000032deg/m	0.000066deg/m	**0.000016**deg/m
600.0	0.000015deg/m	0.000030deg/m	0.000062deg/m	**0.000014**deg/m
700.0	**0.000013**deg/m	0.000029deg/m	0.000058deg/m	**0.000013**deg/m
800.0	**0.000012**deg/m	0.000028deg/m	0.000055deg/m	**0.000012**deg/m

Tabela 2.12: Erro médio de rotação com base na distância, nas sequencias 11-21 segundo a *KITTI framework.*[22]

2.6.3 Erro de translação com base na velocidade

Velocidade(km/h)	LOAM[4]	ISC-LOAM[10]	F-LOAM[15]	V-LOAM[18]
4.0	**0.008380**%	0.010311%	0.019300%	0.008396%
6.0	**0.007736**%	0.008961%	0.016216%	0.007740%
8.0	0.005608%	0.007031%	0.013741%	**0.005541**%
10.0	0.004414%	0.006030%	0.012255%	**0.004305**%
12.0	0.005833%	0.007439%	0.012012%	**0.005711**%
14.0	0.006190%	0.006719%	0.011616%	**0.005976**%
16.0	0.005501%	0.006498%	0.013231%	**0.005481**%
18.0	0.002739%	0.006368%	0.159347%	**0.002629**%
20.0	0.004617%	0.007415%	0.037311%	**0.004420**%
22.0	0.004924%	0.007611%	0.020442%	**0.004730**%
24.0	0.006901%	0.007929%	0.033714%	**0.006793**%

Tabela 2.13: Erro médio de translação com base na velocidade, nas sequencias 11-21 segundo a *KITTI framework.*[22]

2.6.4 Erro de rotação com base na velocidade

Velocidade(km/h)	LOAM[4]	ISC-LOAM[10]	F-LOAM[15]	V-LOAM[18]
4.0	**0.000044**deg/m	0.000055deg/m	0.000189deg/m	0.000045deg/m
6.0	**0.000029**deg/m	0.000043deg/m	0.000102deg/m	**0.000029**deg/m
8.0	**0.000023**deg/m	0.000038deg/m	0.000081deg/m	**0.000023**deg/m
10.0	**0.000021**deg/m	0.000035deg/m	0.000072deg/m	**0.000021**deg/m
12.0	**0.000033**deg/m	0.000040deg/m	0.000083deg/m	**0.000033**deg/m
14.0	**0.000039**deg/m	0.000043deg/m	0.000087deg/m	0.000041deg/m
16.0	**0.000025**deg/m	0.000029deg/m	0.000077deg/m	**0.000025**deg/m
18.0	**0.000013**deg/m	0.000027deg/m	0.000147deg/m	**0.000013**deg/m
20.0	**0.000018**deg/m	0.000035deg/m	0.000091deg/m	0.000019deg/m
22.0	**0.000020**deg/m	0.000036deg/m	0.000085deg/m	**0.000020**deg/m
24.0	0.000030deg/m	0.000042deg/m	0.000110deg/m	**0.000028**deg/m

Tabela 2.14: Erro médio de rotação com base na velocidade, nas sequencias 11-21 segundo a *KITTI framework*.[22]

2.6.5 Conclusões

Tendo apenas em conta os dados das Tabelas 2.11, 2.12 2.13 e 2.14 e considerando que esta dissertação pretende obter a melhor implementação apenas com base nos dados do LiDAR, podemos afirmar que a implementação que apresentou melhores resultados foi a apresentada no trabalho [4]. No outro extremo encontramos a *ISC-LOAM*, com um erro significativamente mais elevado que as restantes.

2.7 Trabalhos similares

Neste capítulo pretende-se apresentar algumas aplicações que já foram realizadas com UAVs onde se pretende estimar a posição e atitude com base em dados do LiDAR.

2.7.1 ARIA: the Aerial Robotic Infrastructure Analyst

Este projeto [23] tem como objetivo responder à necessidade da inspeção de pontes de uma forma eficiente e económica, uma vez que os métodos anteriormente utilizados ainda apresentavam inúmeras deficiências como, serem feitas por diferentes técnicos, que registam os dados em tabelas acompanhadas de desenhos manuais dos danos encontrados. Apresentando assim um elevado custo monetário e temporal, assim como, uma elevada dificuldade em manter coerência entre relatórios elaborados por diferentes técnicos.

Outro dos métodos utilizados era com a recolha de imagens em vídeo, mas este apresentava um novo desafio, sendo que por vezes era difícil para o técnico identificar o contexto no qual a imagem teria sido captada.

Desta forma, este projeto pretendia que através do recurso a um LiDAR, fosse possível fazer a reconstrução de toda a estrutura, ficando assim com uma espécie de maqueta digital da ponte em causa, de forma segura e económica.

Para tal ser possível, foi necessário recorrer à implementação presentada no trabalho [4], uma vez que para além de recolher os dados, é necessário estruturar os mesmo de forma compreensível para o ser humano.

Obtendo assim o resultado apresentado na Figura 2.28.

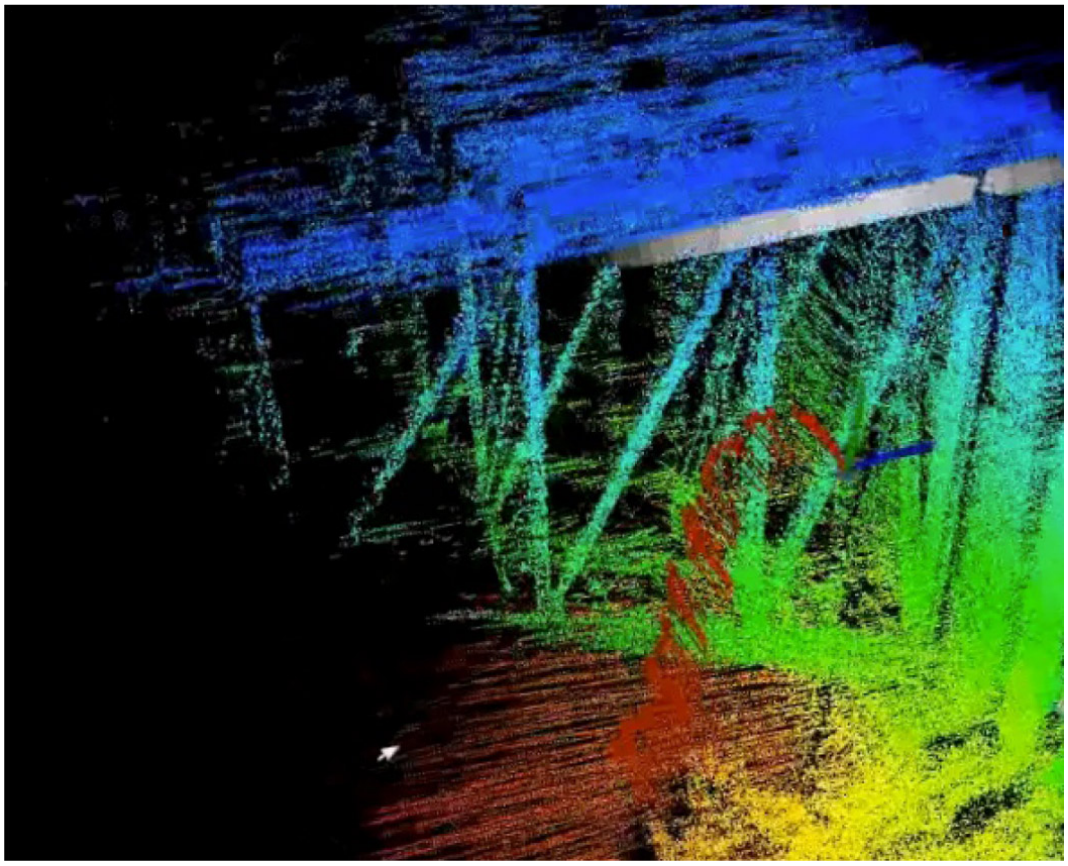


Figura 2.28: Parte da ponte Schenley, modelada pelo projeto ARIA com recurso ao trabalho [4].[23]

2.7.2 Bridge Inspection Using Unmanned Aerial Vehicle Based on HG-SLAM: Hierarchical GraphBased SLAM

Este projeto [24] tem como objetivo levar mais longe os progressos feitos pelo projeto apresentado no projeto [23], para tal uma nova metodologia de aquisição e tratamento da nuvem de pontos foi criado, sendo esta comparada com os resultados que seriam obtidos caso fosse utilizada a implementação apresentada no trabalho [4] com suporte do IMU.

Inicialmente foram executados testes numa ponte sobre terra, a qual apresenta maior ruído devido ao ambiente que a rodeia, sendo os resultados apresentados nas Figuras 2.29 a 2.31

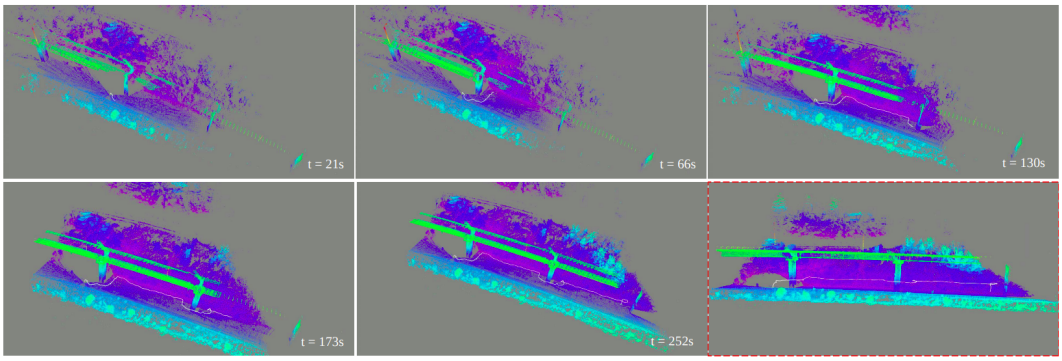


Figura 2.29: Ponte de Deungseon reconstruida com recurso à implementação [4], quando percorrida em linha reta.[24]

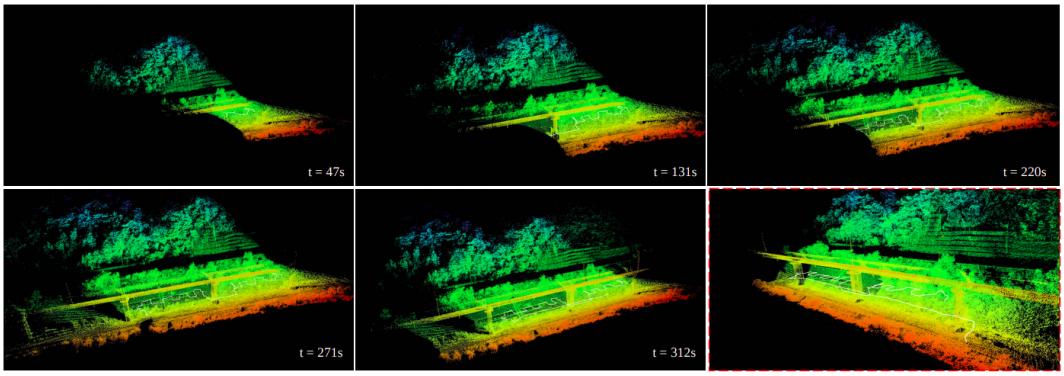


Figura 2.30: Ponte de Deungseon reconstruida com recurso à implementação [4], quando percorrida em zigue-zague.[24]

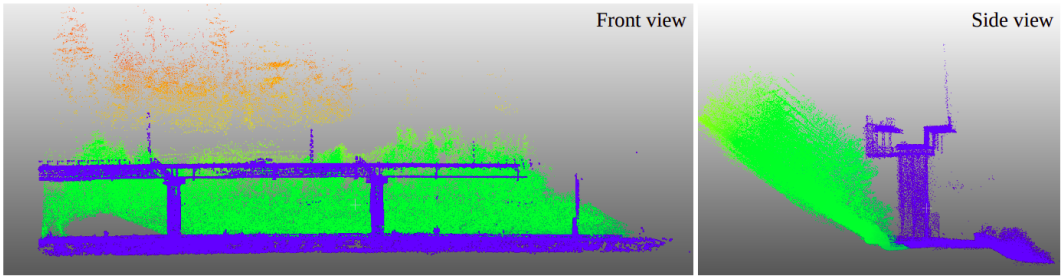


Figura 2.31: Vista lateral e frontal da reconstrução efetuada com o trabalho [4] da ponte de Deungseon.[24]

Foram ainda feitos testes numa ponto que atravessa um rio, sendo este apresentado na Figura 2.32.

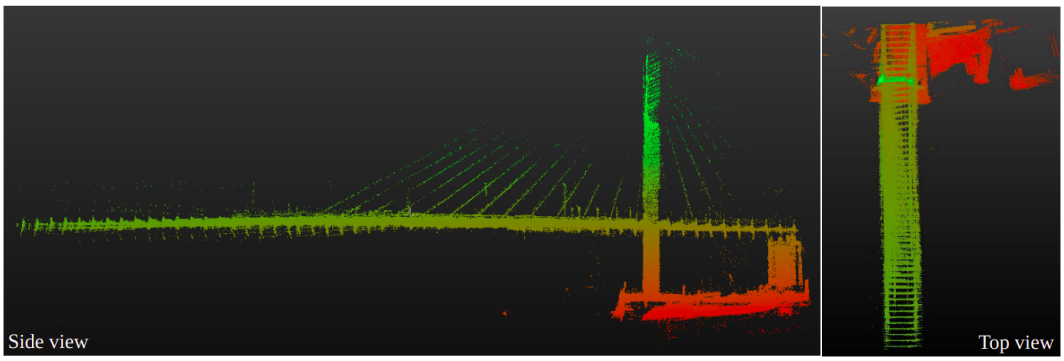


Figura 2.32: Vista lateral e frontal da reconstrução efetuada com o trabalho [4] da ponte de Geobukseon.[24]

Como conclusão do projeto apresentado [24], foi calculado o erro relativo a cada uma das implementações utilizadas, sendo que os valores relativos à reconstrução efetuado pelo trabalho [4], podem ser observados na Tabela 2.15.

De forma a facilitar a compreensão dos valores, podemos observar nas Figuras 2.33 e 2.34 as nomenclaturas utilizadas na elaboração da Tabela 2.15, em que w representa a largura de cada um dos tabuleiros das pontes, h a altura dos seus pilares e d a distância entre os mesmos.

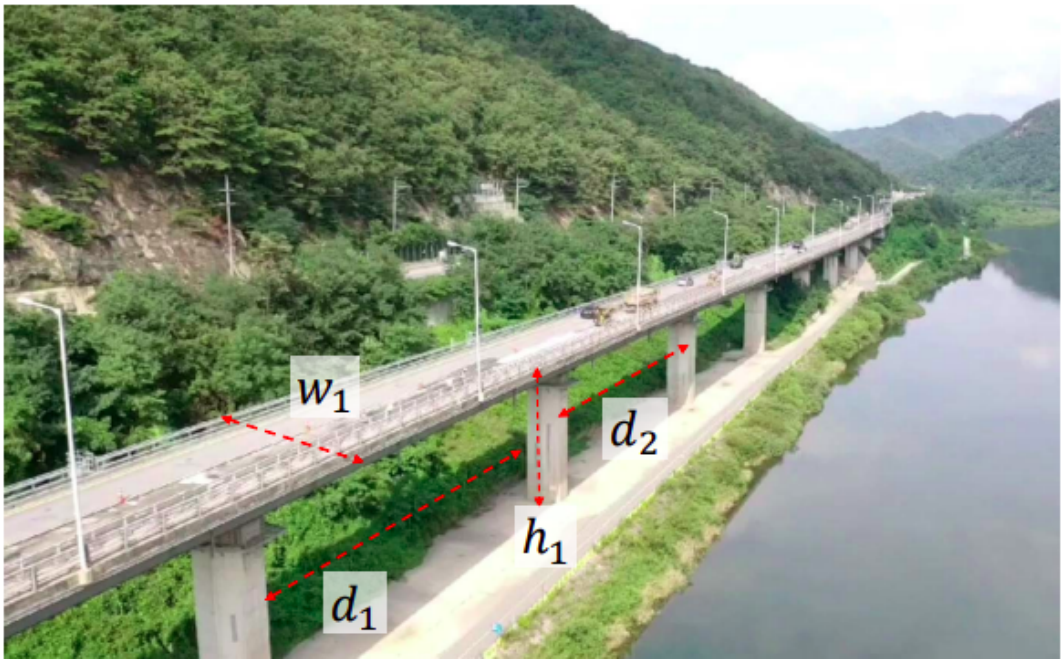


Figura 2.33: Ponte de Deungseon.[24]

Figura 2.34: Ponte de Geobukseon.[24]

Caso estimado	Distância real	Distância estimada	Erro
d1	50m	49.62m	0.76%
d2	50m	49.35m	1.30%
h1	14.5m	14.23m	1.86%
w1	11m	10.88m	1.09%
d3	230m	234.34m	1.88%
h2	23m	23.49m	2.13%
w2	20m	21.96m	9.80%

Tabela 2.15: Erro apresentado pela reconstrução com o trabalho [4].[24]

2.8 Métricas de avaliação do erro

Nesta secção iremos apresentar as várias metodologias de avaliação dos resultados obtidos, é necessário para tal compreender qual a melhor forma de avaliar os resultados tendo em conta o contexto da dissertação.

Os métodos que serão expostos nos próximos capítulos apresentam como principal diferença e variação entre si o peso atribuído a casos fora do comum.

Para além disso, cada método tem os seus pontos fracos e fortes, para casos com valores mais elevados, menores ou mais estáveis.

2.8.1 Erro médio absoluto (MAE)

O método conhecido como *Mean Absolute Error* (MAE) consiste na média do módulo do cálculo direto entre o valor calculado e o esperado. Esta utilização do módulo leva à perda da informação da direção do erro, sendo que apenas temos o valor absoluto do desvio.

$$MAE = \frac{1}{N} \sum |Y - \widehat{Y}| \tag{2.49}$$

Em que:

- N - Número de pontos;
- Y - Valor calculado;
- $\widehat{Y}$ - Valor estimado.

Como é possível perceber pela Equação 2.49, está a ser calculado um desvio médio, como tal, para um bom resultado do MAE, o objetivo passa por ter um valor o mais perto de zero possível.

Uma das vantagens desta implementação é a sua robustez a *outliers*, uma vez que ao efetuar a média estes valores são atenuados por todos os outros.

2.8.2 Erro médio absoluto percentual (MAPE)

No caso do *Mean Absolute Percentage Error* (MAPE), podemos considerar que se trata da mesma abordagem que no capítulo anterior, mas com o resultado final apresentado em forma de percentagem.

$$MAPE = \frac{1}{N} \sum |\frac{Y - \widehat{Y}}{Y}| \tag{2.50}$$

Em relação ao MAE, esta implementação (Equação 2.50) apresenta a grande vantagem de ser de mais fácil compreensão no momento de analise dos dados, no entanto para casos em que os valores sejam muito próximos de zero, ou até mesmo zero, esta implementação não é a indicada devido à divisão, que neste caso, seria por zero.

2.8.3 Erro médio quadrático (MSE)

Novamente, podemos considerar o *Mean Squared Error* (MSE) uma evolução do MAE. Com é possível compreender através da Equação 2.51, esta é em tudo semelhante à Equação 2.49, sendo que neste caso a diferença entre o valor estimado e o real é usado ao quadrado. Sendo que esta diferença leva a que este método seja mais sensível a *outliers*.

$$MSE = \frac{1}{N} \sum (Y - \widehat{Y})^2 \tag{2.51}$$

Esta diferença pode ser utilizada como termo de comparação com outros métodos, perceber a existência de *outliers*.

O único caso que pode ser considerado um defeito, acenta no facto de que, uma vez efetuado o quadrado do valor, a sua unidade também difere dos dados iniciais, ou seja, se estivermos a avaliar valores em metros, iremos obter um desvio em metros ao quadrado.

2.8.4 Raiz quadrada do erro médio (RMSE)

A implementação do *Root Mean Squared Error* (RMSE) pode ser considerada uma derivação do MSE, tal como é possível perceber através da comparação das Equações 2.52 e 2.51, respetivamente, nas quais a principal diferença é a raiz quadrada sobre o valor final.

$$MSE = \sqrt{\frac{1}{N} \sum (Y - \widehat{Y})^2} \tag{2.52}$$

Esta implementação apresenta todas as características da anterior, diferenciando-se apenas num ponto, pois o resultado do erro será apresentado na mesma unidade do caso em estudo, facilitando a interpretação dos valores.

2.8.5 R-quadrado (R2)

Assim como o MAPE, também o *R-Squared* (R2) é apresentado com um valor percentual. Sendo que no caso do R2, este cálculo (Equação 2.53) representa a proporção de uma dada variação de uma variável dependente, que pode ser explicada pela variação de uma variável independente.

Por exemplo, se o resultado de R2 for 50%, podemos então dizer que cerca de metade da variação detetada poderá ser explicada pela variável em estudo.

$$R^2 = 1 - \frac{SSe}{SSt} \tag{2.53}$$

$$MSE = SSe = \frac{1}{N}\sum(Y - \widehat{Y})^2 \tag{2.54}$$

$$SSt = \frac{1}{N}\sum(Y - \overline{Y})^2 \tag{2.55}$$

Em que:

- $\overline{Y}$ - Valor médio de Y.

.

Em comparação com o MSE, o R2 apresenta um valor padronizado, que se abstrai do tipo de valores apresentados, apresentando um valor percentual da variação.

2.8.6 Ferramenta de avaliação de erro da *KITTI Framework*

Por fim temos a ferramenta de avaliação de erro fornecida pela *KITTI Framework*, a qual foi usada para calcular os valores apresentados no Capítulo 2.6. Não existem dados relativos ao método utilizado para a avaliação do erro, mas através da interpretação do código foi possível perceber a arquitetura de alto nível da ferramenta, apresentada na Figura 2.35.

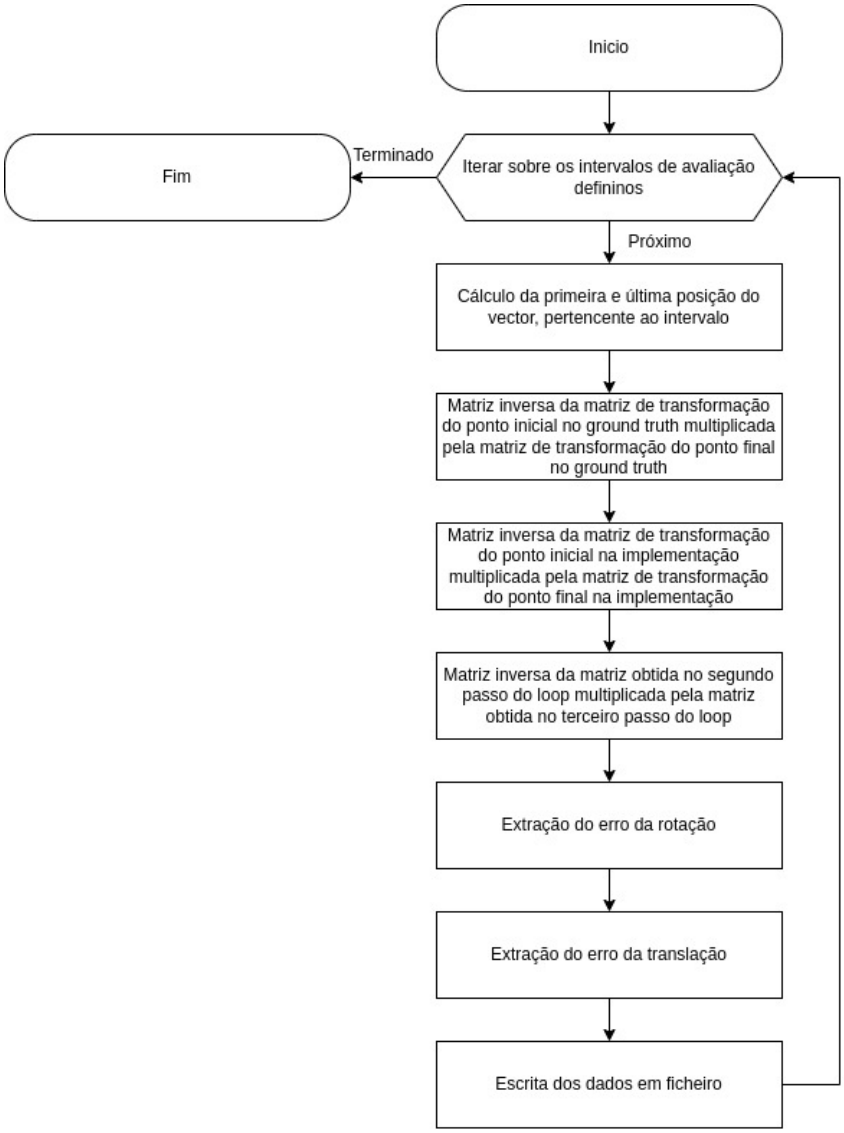


Figura 2.35: Arquitetura de alto nível da ferramenta de avaliação de erro fornecida pela KITTI Framework.

Esta ferramenta recebe como dados para avaliação, dois ficheiros de formato texto, com a informação da matriz de transformação de cada ponto em relação ao ponto inicial.

Uma matriz de transformação é composta por duas partes principais[25], como é possível observar na Figura 2.36, sendo estas:

- No canto superior esquerdo temos a matriz que representa a rotação, mas pode também incluir dados de escala;
- No canto superior direito temos por fim a matriz de translação.

$$\mathbf{M} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Figura 2.36: Estrutura de uma matriz de transformação.[25]

No caso desta ferramenta, a última linha da matriz é descartada, sendo apenas passados os valores pertencentes à matriz 3 por 4 da parte superior desta matriz.

2.8.7 Conclusões

Após a análise dos casos acima apresentados, é possível perceber que a melhor abordagem passaria pela utilização de múltiplos indicadores, preferencialmente um mais permeável a possíveis *outliers* e um que atribua maiores pesos a estes.

Como podemos ver nas Figuras 2.37 e 2.38, é possível confirmar a afirmação anterior. Quando temos erros menores que 1 (Figura 2.37), praticamente não temos diferença entre ambos os tipos de abordagem, mesmo em caso de erros até ≈5, podemos considerar que a diferença não é significativa, sendo que a partir deste, o seu peso torna-se mais significativo (Figura 2.39).

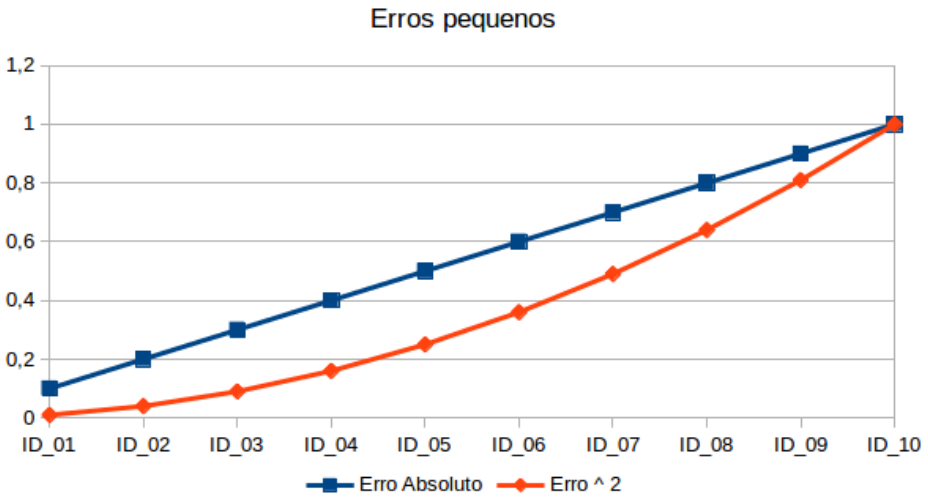


Figura 2.37: Performance em erros entre 0 e 1.[26]

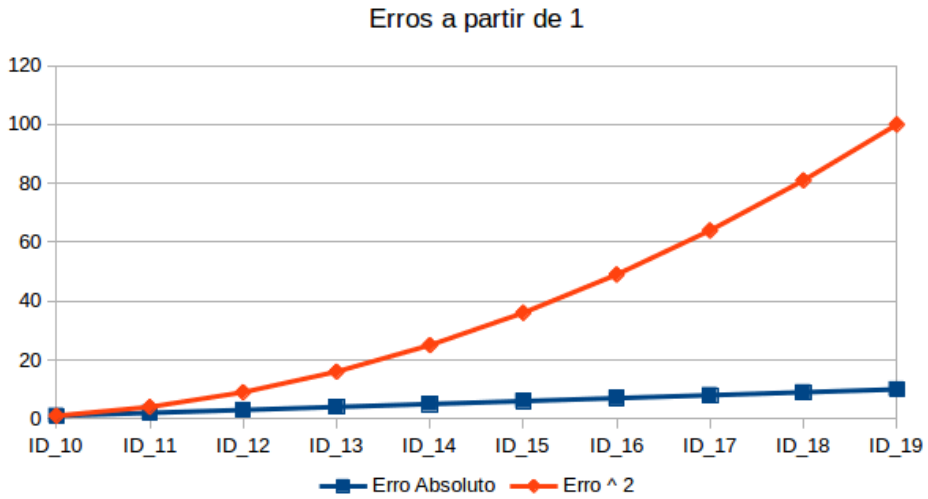


Figura 2.38: Performance em erros superiores a 1.[26]

CASE 1: Evenly distributed errors

ID	Error	|Error|	Error^2
1	2	2	4
2	2	2	4
3	2	2	4
4	2	2	4
5	2	2	4
6	2	2	4
7	2	2	4
8	2	2	4
9	2	2	4
10	2	2	4

MAE	RMSE
2.000	2.000

CASE 2: Small variance in errors

ID	Error	|Error|	Error^2
1	1	1	1
2	1	1	1
3	1	1	1
4	1	1	1
5	1	1	1
6	3	3	9
7	3	3	9
8	3	3	9
9	3	3	9
10	3	3	9

MAE	RMSE
2.000	2.236

CASE 3: Large error outlier

ID	Error	|Error|	Error^2
1	0	0	0
2	0	0	0
3	0	0	0
4	0	0	0
5	0	0	0
6	0	0	0
7	0	0	0
8	0	0	0
9	0	0	0
10	20	20	400

MAE	RMSE
2.000	6.325

Figura 2.39: Comparação entre o comportamento do RMSE e do MAE.[26]

Desta forma, com um método mais permeável a *outliers*, é nos possível ter uma visão global dos resultados, com uma visão geral da performance, desprezando assim a possibilidade de uma baixa quantidade de *outliers*. Enquanto que com um método que atribua diferentes pesos, podemos corroborar, ou pelo contrário, perceber que os primeiros valores não representam a realidade como um todo.

Capítulo 3

Implementação

Neste capítulo são apresentadas as ferramentas desenvolvidas e alteradas de forma a possibilitar e suportar a realização desta dissertação.

Como definido nos objetivos desta dissertação, um dos objetivos passa pela avaliação dos diferentes métodos identificados no estado de arte como promissores a serem utilizados no UAV. Desta forma, serão testados os resultados dos trabalhos [4], [10], [15] e [17].

De forma a analisar o resultado dessas implementações, será utilizado o UAV desenvolvido no ISEP, *STORK I*, apresentado na Figura 3.1.

Figura 3.1: UAV utilizado na recolha de dados.

3.1 Ferramentas desenvolvidas

Nesta secção serão descritas as ferramentas que foi necessário desenvolver, como forma de responder ás dificuldades apresentadas por esta dissertação, das quais a necessidade de adquirir dados da performance das diversas implementações, rotação da nuvem de pontos e cálculo do erro associado aos pontos estimados, entre outros.

3.1.1 Captação de dados de performance

De forma a obter uma conclusão mais informada e fiável, foram desenvolvidas duas ferramentas de aquisição de dados relativos ao processamento do *dataset*.

O foco passou por perceber que recursos seriam consumidos no *Central Processing Unit* (CPU), assim como na *Random Access Memory* (RAM).

De forma a avaliar a performance das diversas implementações e os seus consumos tanto de CPU como de RAM, foi criada um ferramenta para cada caso, que combina a utilização de um *script bash* (Anexo 6.3) com um *script Python* (Anexo 6.4).

Em ambos os casos a utilização dos *bash scripts* prendia-se pela necessidade de executar comandos *bash* de uma forma rápida e precisa, o mesmo poderia ser feito através de um *script Python*, mas iríamos assim estar a criar uma camada de abstração entre o comando e a execução do mesmo.

No segundo passo, a escolha recaiu sobre *Python* devido à sua reconhecida versatilidade na criação de ferramentas de suporte à execução de tarefas.

Por fim foi ainda criado um terceiro *bash scripts*, que permite despoletar e controlar de forma centralizada a duração do tempo de captação de dados, assim como a forma como pretendemos receber os resultados finais.

3.1.2 Utilização da posição estimada pelo UAV

Devido à utilização do GPS/*Real-Time Kinematic* (RTK) e à necessidade do mesmo de efetuar um pós processamento de forma a obter um resultado mais preciso, foi também desenvolvida uma ferramenta para recolher os dados de posição estimados no UAV, esta ferramenta tinha por objetivo servir como uma primeira iteração na comparação entre os resultados obtidos e o algo que se aproximasse de um *ground truth*.

Para esse fim, foi desenvolvida a ferramenta, sendo que o seu foco passava por gravar os dados de forma a serem consumidos por outras ferramentas que serão mencionadas de seguida, para avaliação da trajetória estimada pelas diversas implementações.

3.1.3 Sincronização de valores entre o ground truth e o resultados das implementações

Para a utilização da ferramenta fornecida pela *KITTI Framework* era necessário que os ficheiros de odometria recolhidos das implementações e o *ground truth* correlacionado tivessem o mesmo número de entradas e uma vez que estes apresentavam frequências de escrita diferentes, foi necessário sincroniza-los, para tal foi criado o *script* (Anexo 6.5) apresentado na Figura 3.2. Em que, como podemos observar, objetivo era, através do ficheiro com a frequência de leitura mais baixa, procurar no outro qual o valor que tinha sido captado no mesmo momento, ou proximo deste.

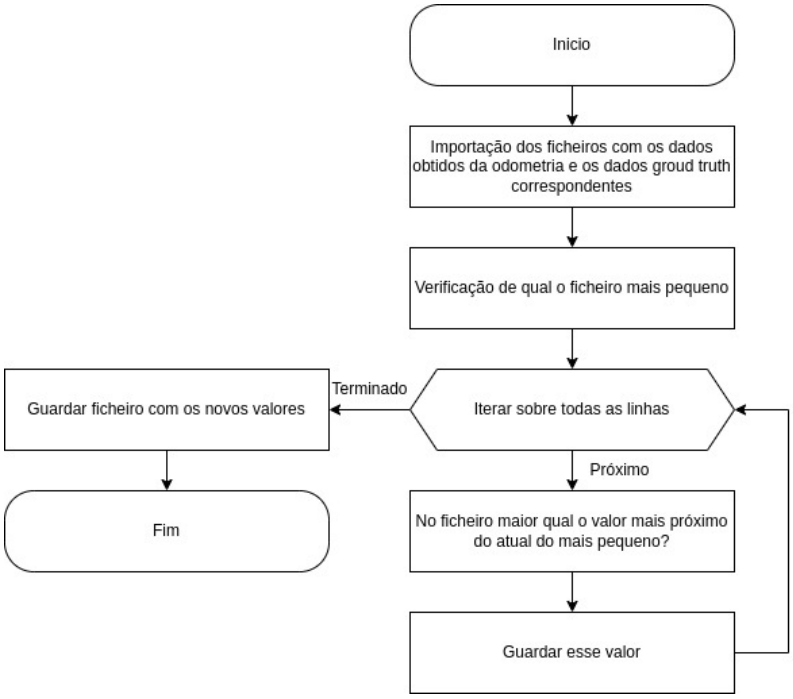


Figura 3.2: Arquitetura de alto nível do ficheiro de sincronização.

3.1.4 Cálculo do erro médio absoluto

De forma a complementar os dados obtidos através da ferramenta fornecida pela *KITTI Framework* foi necessário desenvolver uma ferramenta para calcular o erro médio absoluto (Anexo 6.6), para tal foi desenvolvida a ferramenta apresentada na Figura 3.3.

Esta ferramenta apresenta como resultados, o gráfico da evolução do erro associado a cada um dos eixos, e do erro da diferença entre dois pontos diretamente, assim como a média de cada um deles.

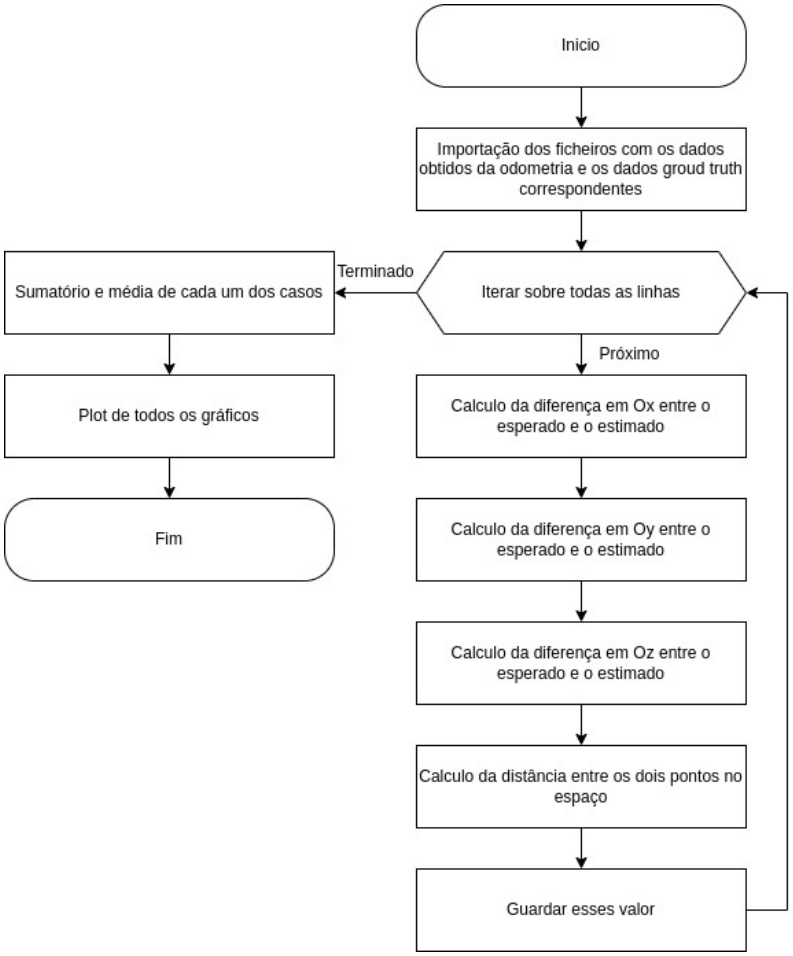


Figura 3.3: Arquitetura de alto nível do ficheiro de cálculo do erro médio absoluto.

3.1.5 Cálculo do erro médio quadrático

Como forma de complemento final, foi ainda criada a ferramenta apresentada (Anexo 6.7) na Figura 3.4, uma vez que a ferramenta apresentada no Capítulo 3.1.4 apresenta uma elevada permeabilidade a *outliers* e esta apresenta um maior peso nestes casos, como explicado no Capítulo 2.8.7.

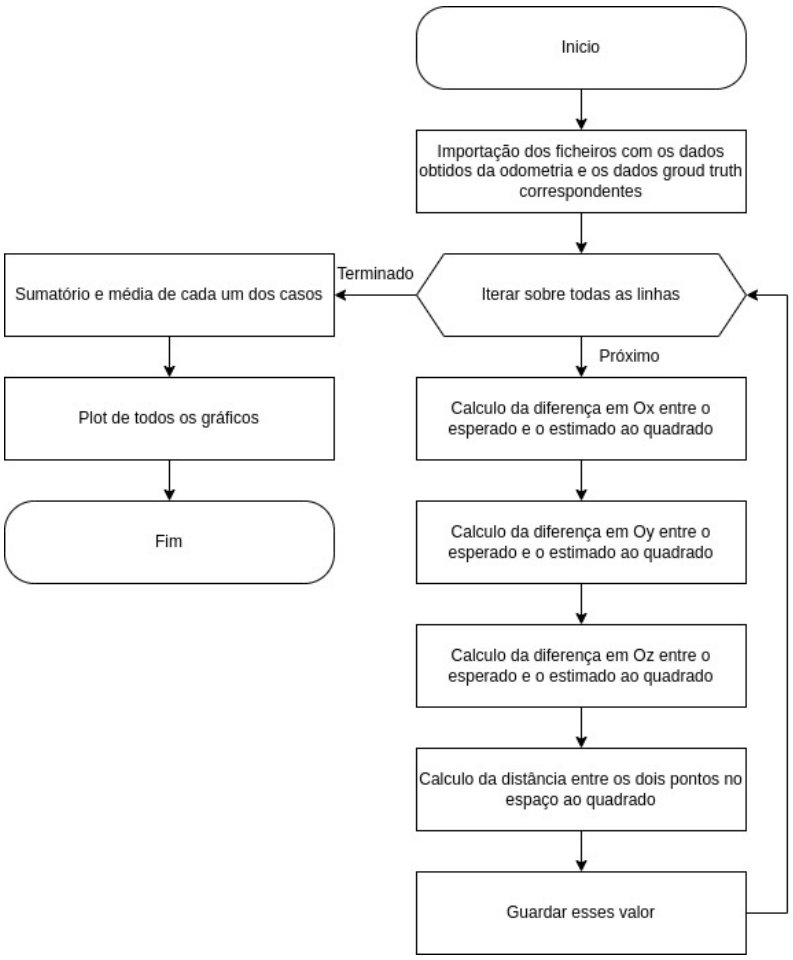


Figura 3.4: Arquitetura de alto nível do ficheiro de cálculo do erro médio quadrático.

3.1.6 Conversão das posições para matrizes transformação

Como mencionado no Capítulo 2.8.6 para a utilização da ferramenta de avaliação de erro da *KITTI Framework* foi necessário desenvolver uma ferramenta que possibilita a conversão dos pontos de posicionamento para matrizes de transformação.

Para este fim, foi desenvolvido o *script* apresentado na Figura 3.5.

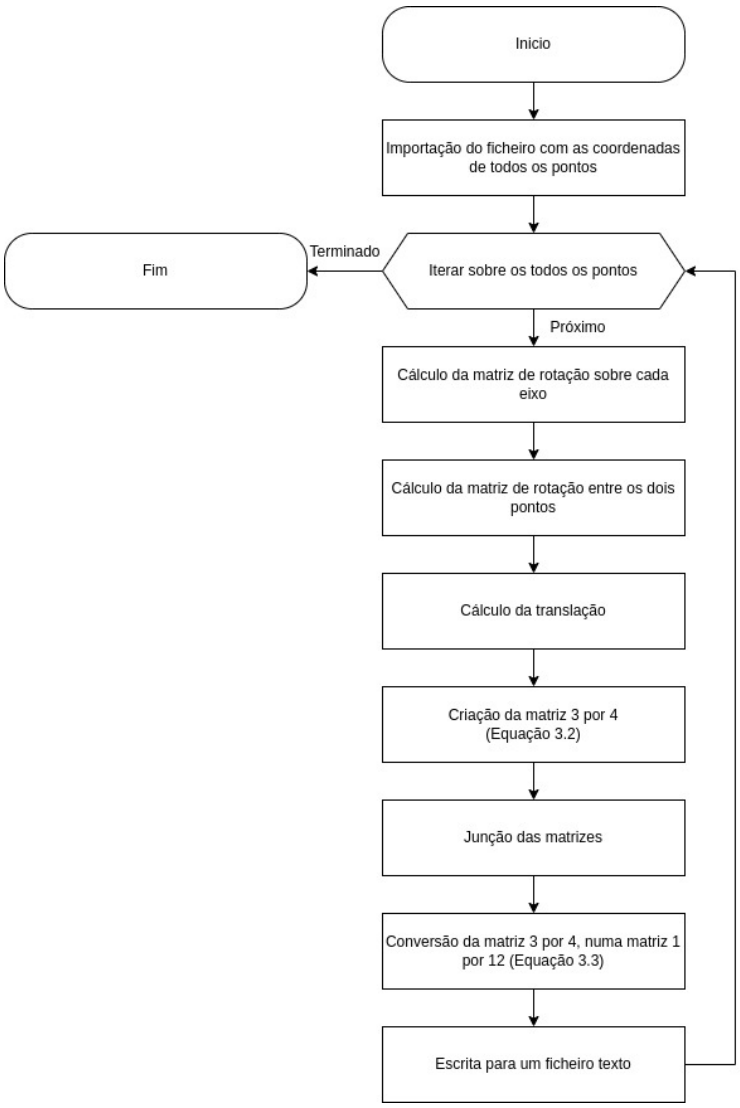


Figura 3.5: Arquitetura de alto nível da conversão dos pontos para matrizes transformação.

Como pontos principais desta ferramenta, temos a criação das matrizes de rotação sobre cada um dos eixos apresentadas na Figura 3.6, onde θ representa o ângulo entre os vetores que caracterizam cada um dos pontos em estudo. Temos a matriz de rotação no espaço, representada na Equação 3.1, onde nos é apresentada a matriz de rotação que que move o vetor que caracteriza o primeiro ponto no espaço, para a direção do vetor que representa o segundo ponto no espaço. E por fim, a

matriz de translação apresentada na Equação 3.2, que efetua a translação do novo ponto(vetor) no espaço, para a localização real do mesmo.

$$\mathbf{R}_X(\theta)=\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & \sin(\theta) & 0 \\ 0 & -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \mathbf{R}_Y(\theta)=\begin{pmatrix} \cos(\theta) & 0 & -\sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(\theta) & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \mathbf{R}_Z(\theta)=\begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 & 0 \\ \sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Figura 3.6: Matrizes de rotação sobre cada um dos eixos.

$$R = R_z(\theta) * R_y(\theta) * R_x(\theta) \tag{3.1}$$

$$T = (P_i - P_1) * R \tag{3.2}$$

$$\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \end{bmatrix} => \begin{bmatrix} a & b & c & d & e & f & g & h & i & j & k & l \end{bmatrix} \tag{3.3}$$

3.1.7 Rotação da nuvem de pontos

Tendo em atenção que as implementações foram criadas para ser utilizadas na recolha de dados em veículos terrestres e que através da análise feita no Capítulo 2 foi possível perceber que todas as implementações, com o código disponível, teriam sido testadas em veículos com o LiDAR posicionado de forma a respeitar o referencial do sensor. Foi assim necessário criar um *node* em *Robot Operating System* (ROS) que permita a rotação da nuvem de pontos para o referencial esperado pelas implementações.

Para tal, utilizando a biblioteca *transforms.h* disponibilizada pelo ROS foi então criado um node de rotação da nuvem de pontos através da rotação do seu referência, apresentado na Figura 3.7.

Este *node* tem por objetivo fazer de interface entre os dados recolhidos pelo LiDAR e as diversas implementações, como é possível observar na Figura 3.8 onde é apresenta a comunicação entre os diversos *nodes*.

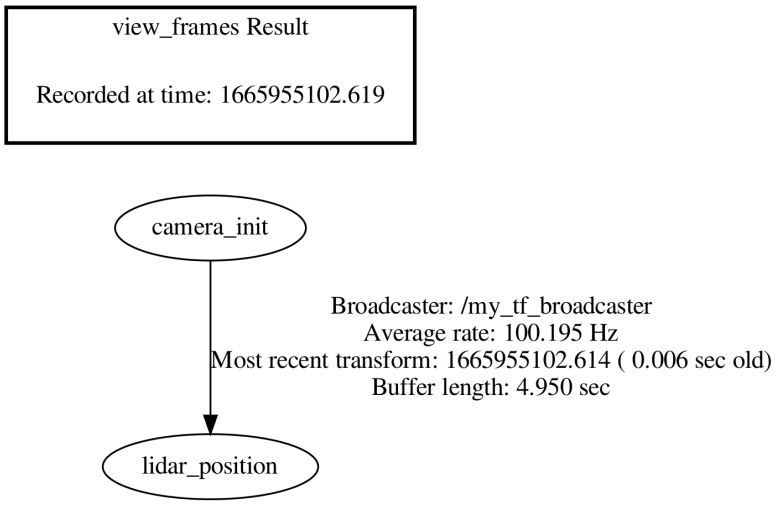


Figura 3.7: Representação da transformação de referenciais (Anexo 6.1).

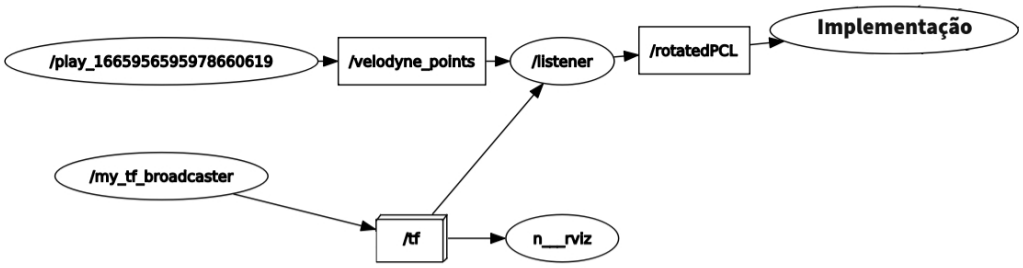


Figura 3.8: Arquitetura de comunicações entre os diversos *nodes* (Anexo 6.2).

3.1.8 Rotação da odometria obtida

Após os testes iniciais foi possível perceber que cada implementação apresentava uma dada rotação em relação aos dados recebidos provenientes do processamento executado. Desta forma foi necessário criar uma ferramenta que efetuasse a rotação correta, de forma a aproximar os resultados dos esperados.

Como trabalho futuro esta ferramenta deverá passar a um *node*, que dependendo do objetivo da utilização futura, deverá ser incorporado em localizações distintas. No entanto, para o objetivo desta dissertação e tendo em conta que estamos a tratar de dados já guardados e não em tempo real, uma ferramenta de pós processamento preenche as necessidades.

Esta ferramenta tem a sua arquitetura apresentada na Figura 3.9.

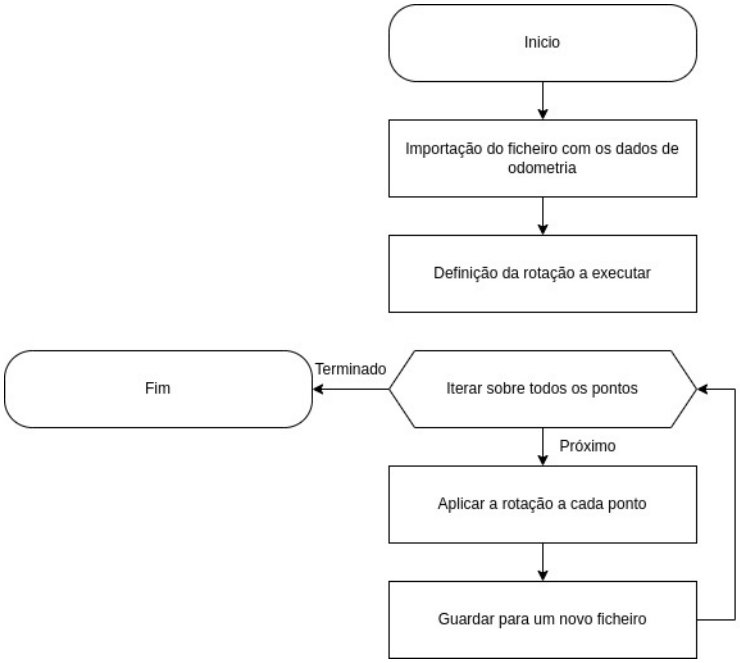


Figura 3.9: Arquitetura da ferramenta utilizada para rodar os pontos de odometria.

3.2 Ferramentas adaptadas

Neste capítulo será apresentada uma ferramenta que foi necessário utilizar durante o desenvolvimento desta dissertação e para tal foi necessário torna-la compatível com o caso em estudo.

3.2.1 Captação dos dados de odometria

As ferramentas utilizadas para a captação dos dados de odometria das diversas implementações foi baseada numa ferramenta já existente [25] e criada para a mesma função, mas apenas para a implementação do Capítulo 2.1.

Uma vez que esta dissertação se debruçava sobre o estudo de diferentes implementações e estas apresentam os seu resultados em formatos diferentes, foi necessário fazer uma adaptação à mesma, de forma a ser possível definir a implementação que se encontra a ser escutada.

Esta diferenciação pretendeu possibilitar a criação de ficheiros de resultados com diferentes nomes, assim como uma escolha mais direta do tipo de dados a ser escutado.

3.3 STORK I

De forma a recolher dados reais para a realização de testes, foi utilizado o UAV desenvolvido no ISEP, *STORK I* [27], apresentado na Figura 3.10. Desta forma,

esta secção tem por objetivo apresentar os componentes importantes do UAV para a realização desta dissertação.

Figura 3.10: STORK I[27].

3.3.1 Velodyne Puck (VLP-16)

O *STORK I* encontra-se equipado com um LiDAR *VLP-16* (Figura 3.11) que, como o seu nome indica, usa um conjunto de dezasseis feixes infravermelhos para recolher dados, sendo que cada um desses feixes é ainda disparado a uma frequência de 18.08 kHz [28].

Figura 3.11: VLP-16.[28]

Este LiDAR permite a configuração do tipo de retorno, sendo esta configuração disponibilizada tanto através do interface *web* como por linha de comandos.

As formas possíveis de configuração são:

- Modo de retorno simples:
  - *Strongest* - Neste modo, os feixes irão retornar o objeto que apresentar a maior reflexão, seja em área ou intensidade, devido à composição do

material, como é possível compreender através das Figuras 3.12, 3.13 e 3.14;

– *Last* - Neste modo, os feixes irão retornar o objeto mais distante, como é possível compreender através da Figura 3.12.

- Modo de retorno múltiplo:

  – Neste caso, se os objetos estiverem distanciados de pelo menos um metro, o sensor irá detetar ambos os objetos.

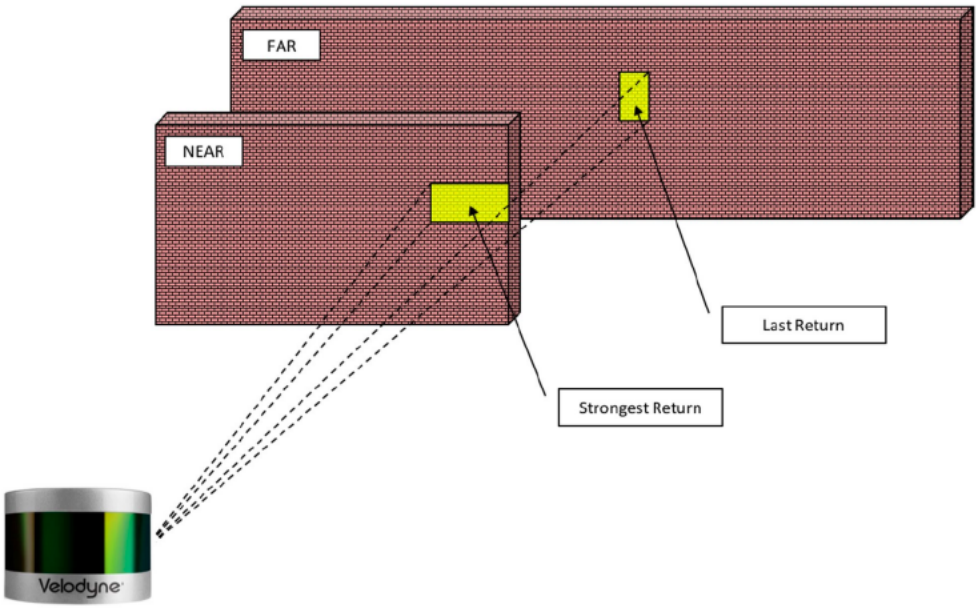


Figura 3.12: Modo de retorno simples, caso 1.[28]

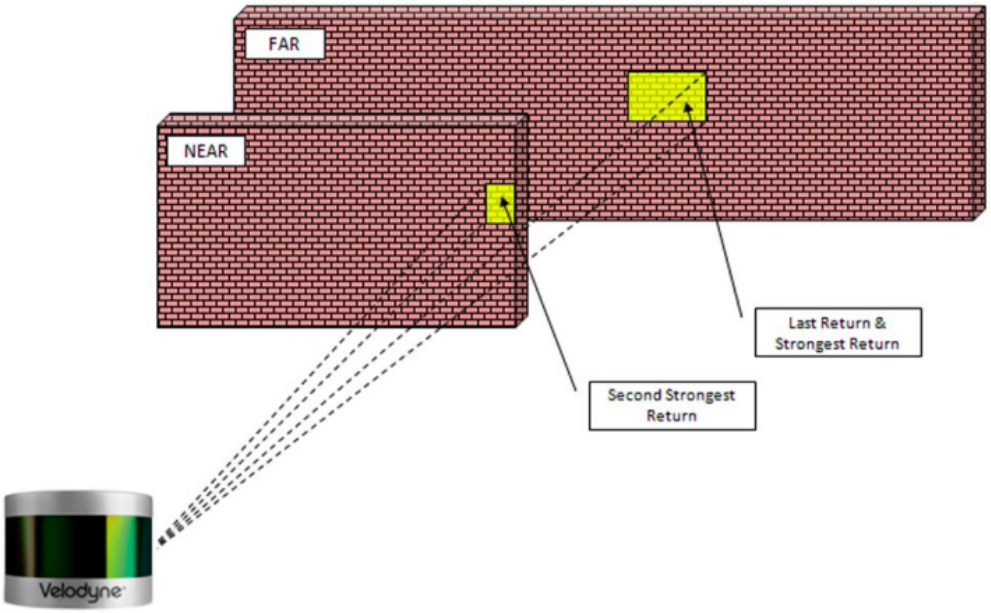


Figura 3.13: Modo de retorno simples, caso 2.[28]

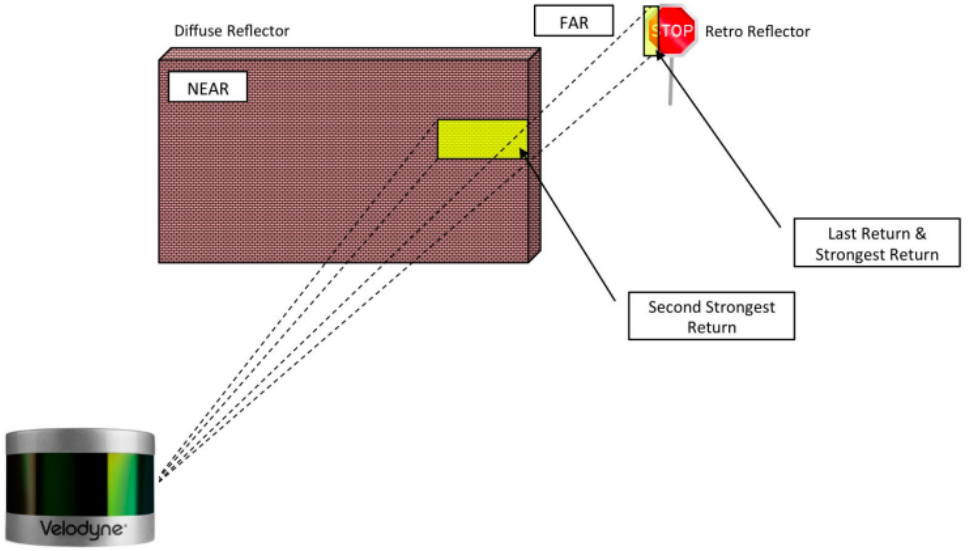


Figura 3.14: Modo de retorno simples, caso 3.[28]

3.3.2 Ublox Neo M8N

O *STORK I* encontra-se também equipado com um *Global Navigation Satellite System* (GNSS), *Ublox Neo M8N* [29], apresentado na Figura 3.15.

Figura 3.15: Ublox Neo M8N.[29]

Como tal, este sistema, permite não só a conecção ao sistema GPS, mas também aos sistemas *Galileo*, *GLObalnaya NAvigatsionnaya Sputnikovaya Sistema* (GLONASS) e *BeiDou Navigation Satellite System* (BDS). Esta possibilidade de conexão a diversas constelações de satélite permite assim, através do sistema RTK aumentar a possibilidade de um menor erro.

Capítulo 4

Resultados

De forma a atingir os objetivos apresentados no Capítulo 1.2.1, foi necessária uma análise dos dados de performance de cada uma das implementações e o erro associada às mesmas, finalizando com uma avaliação da possibilidade de implementação de, pelo menos, uma delas no contexto pretendido.

Desta forma, este capítulo tem por objetivo apresentar os diferentes tipos de testes realizados, assim como os resultados relativos com os mesmos. Apresentando as limitações e implicações da utilização dos mesmos. De notar que a implementação apresentada no Capítulo 2.4 não foi utilizada na obtenção de resultados, uma vez que um dos requisitos se prendia com que a implementação fosse permeável a condições de baixa luminosidade.

De notar também que de forma a tornar a apresentação de resultados mais concisa, no Capítulo 4.1 será apenas explicado o contexto e os resultados obtidos de forma superficial uma vez que que este teste se resumiu a uma prova de conceito com o objetivo de perceber quais as limitações que poderiam ser enfrentadas enquanto que no Capítulo 4.2 as implementações serão apresentadas de forma descriminada. Esta opção deve-se ao facto dos resultados obtidos, uma vez que para o primeiro caso os dados apresentados não apresentam valor comparativo entre a performance das diversas implementações.

De notar ainda que as implementações apresentadas no Capítulo 2.1 e Capítulo 2.5 apresentavam a possibilidade de utilização dos dados do IMU, no entanto, este não foi utilizado devido à incompatibilidade do tipo de mensagem em que se encontram e ao erro que a sua conversão iria apresentar.

Por fim, como forma de analisar de forma mais precisa as conclusões retiradas em relação a cada implementação, foi então calculado o erro associado à trajetória, para tal foram utilizados dois métodos, o Erro Médio Absoluto (Capítulo 2.8.1) e o Erro Médio Quadrático (Capítulo 2.8.3).

A escolha da combinação das duas, prendeu-se pela necessidade de uma melhor compreensão dos resultados obtidos, uma vez que através de uma primeira análise dos resultados obtidos através da ferramenta fornecida pela *KITTI Framework* (Capítulo 2.8.6), percebe-se ser necessário um pouco mais de enquadramento no contexto desta dissertação pois a avaliação do erro executada pela ferramenta foca-se unicamente no movimento no plano XY.

4.1 Primeiro teste realizado

Neste Capítulo serão apresentados os dados recolhidos durante o primeiro teste executado. Como é possível observar na Figura 4.1, o local era composto maioritariamente de pequenas árvores e pequenos bancos, uma vez que apenas o parque foi usado para a recolha dos dados, o único edifício que apresentava importância significativa para a reconstrução do mundo é possível ser observado na imagem do canto superior direito.

Figura 4.1: Local da recolha dos dados.

4.1.1 Preparação do *Hardware*

Para possibilitar a realização destes testes foi usado um UAV desenvolvido no Instituto de Engenharia de Sistemas e Computadores, Tecnologia e Ciência (INESCTEC) (Figura 4.2).

Figura 4.2: UAV utilizado na recolha de dados.

Do *hardware* que este dispõe, dois componentes apresentam maior preponderância para o contexto sobre o qual esta dissertação se debruça, sendo estes o LiDAR e o GPS/RTK.

O LiDAR equipado no UAV foi apresentado no Capítulo 3.3.1, com as configurações mais importantes apresentadas na Tabela 4.1.

Parâmetro	Configuração
Ângulo de leitura	360º
Raio de pontos descartados	1 metro

Tabela 4.1: Parametrização do LiDAR VLP-16.

No caso do GPS/RTK, este foi suportado pela estação estacionária T300 apresentada na Figura 4.3. Este tipo de GPS permite a recolha de dados com um erro menor, sendo que este depende do número de GPS's a que se conecta.

Figura 4.3: T300.

Estes dados foram pós processados com base nos dados posteriores recolhidos relativamente à posição dos satélites no momento da recolha de dados, reduzindo assim o erro dos dados para cerca de dois centímetros, podendo assim considerar estes dados como o percurso real do UAV, usando os desta forma como *ground truth* para o cálculo do erro.

Já no caso do LiDAR, como é possível observar na Figura 4.4, este encontrava-se rodado 90º sobre Ox, segundo o referencial do sensor.

Figura 4.4: Posicionamento do LiDAR.

Tendo em atenção esta rotação no sensor, é necessário começar por efetuar a rotação dos dados recolhidos. Para tal, foi criado um *node* em ROS que efetua a rotação dos referenciais. Esse *node*, como apresentado na Figura 4.5, irá receber a nuvem de pontos captada pelo sensor, efetuar a rotação e só depois efetuar a comunicação da mesma para as diversas implementações.

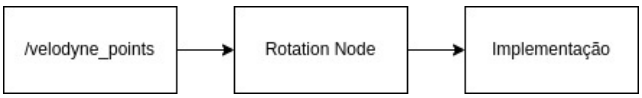


Figura 4.5: Arquitetura da incorporação do *node* em ROS no sistema.

Após este *node*, temos como resultado por exemplo, a rotação (Figuras 4.7 e 4.8) efetuada sobre o edifício (Figura 4.6), onde podemos ver a azul os dados *raw* enviados pelo sensor e a vermelho o resultado da rotação.

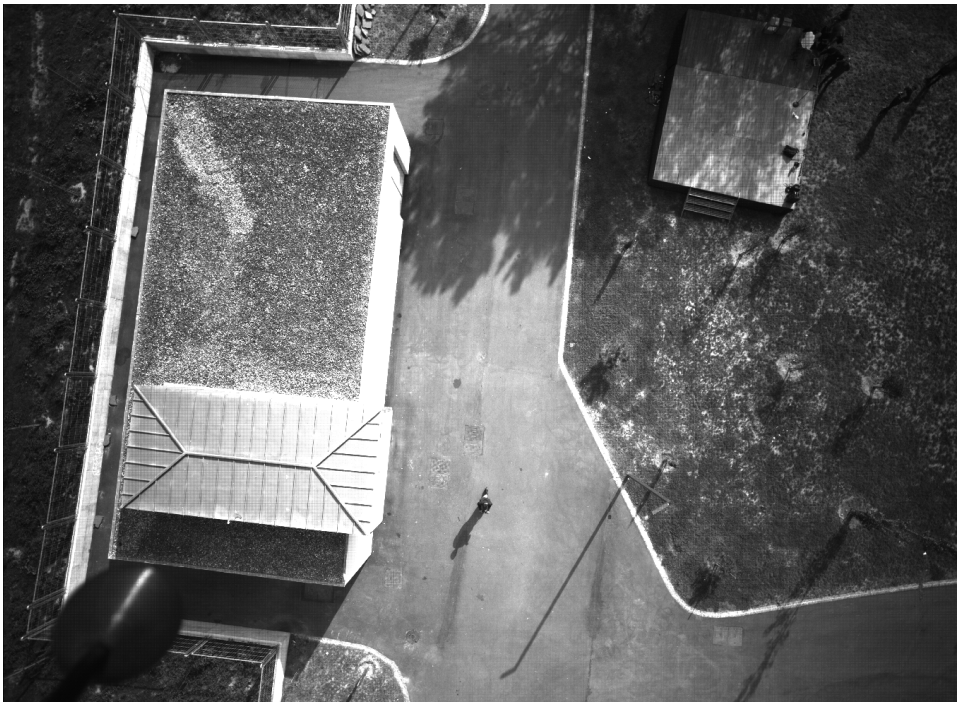


Figura 4.6: Edifício visto da camera do UAV.

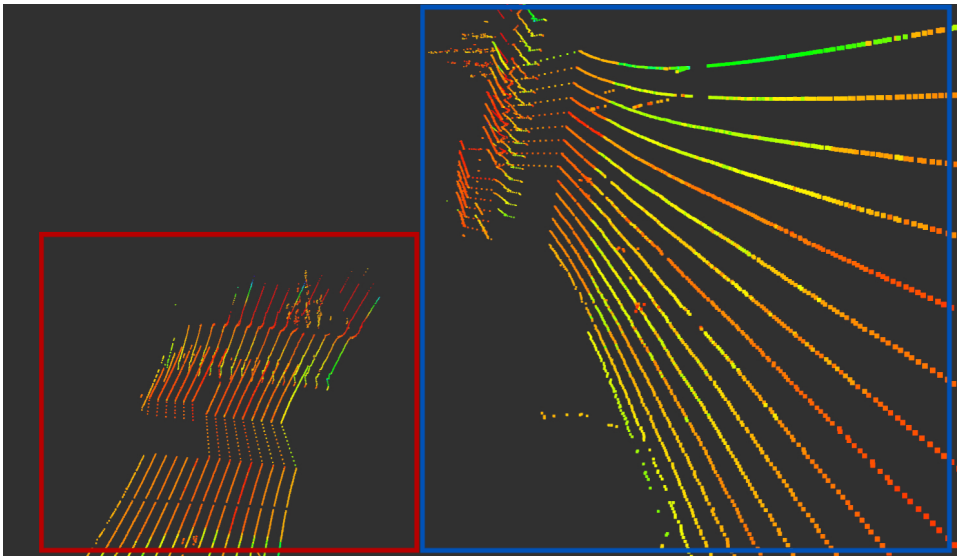


Figura 4.7: Efeitos da rotação no edifício, vista de lado.

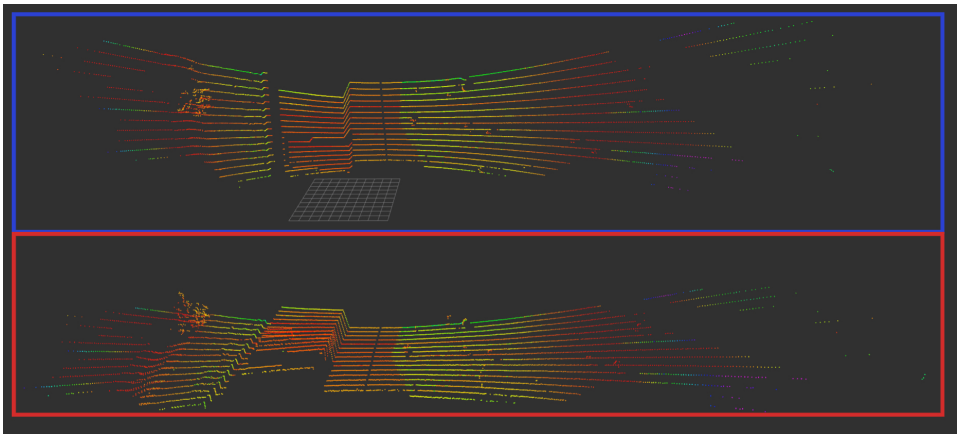


Figura 4.8: Efeitos da rotação no edifício, vista de frente.

4.1.2 Conclusões

Com recurso ás implementações foram então realizados testes iniciais com os dados recolhidos. Inicialmente todas implementações foram testadas com os dados *raw* e posteriormente com a execução da rotação da nuvem de pontos.

Através da análise do comportamento passo a passo das diversas implementações é possível perceber que para o correto funcionamento dos diversos trabalhos é importante ter em atenção:

- A quantidade de pontos de interesse no eixo U segundo o referencial *East-North-Up coordinate system* (ENU), comprovada através da analise de código e à preponderância que estes pontos apresentam na deteção de pontos de interesse;
- Quantidade de pontos de relevo no ambiente em que os dados foram recolhidos;
- Mudanças de direção abruptas [18].

O comportamento das implementações poderá ser melhorado através das seguintes melhorias:

- Ambiente com mais pontos de interesse e mais figuras geométricas;
- Diminuição no ângulo de captura do sensor;
- Mudanças de direção mais circulares.

4.2 Segundo teste realizado

Após as conclusões obtidas através do primeiro teste, foi então realizado um segundo teste com o objetivo de testar as conclusões retiradas no passo anterior. Das quais o foco prendeu-se pela utilização de um espaço com um maior número de formas geométricas e a alteração do ângulo de leitura do LiDAR.

Desta forma, como é possível observar na Figura 4.9, a escolha do local recaiu sobre o Edifício F (Figura 4.10) do ISEP uma vez que este apresenta, à sua frente, um espaço aberto onde era possível recolher dados de GPS/RTK para uma localização mais precisa, voar em segurança e uma vez que este espaço se encontra rodeado de infraestruturas e vegetação iria também ser possível obter os pontos de interesse necessários.

É ainda possível observar na Figura 4.9 identificado pela linha a vermelho, a zona na qual o UAV executou o seu voo e o local da sua aterragem, identificado pelo quadrado vermelho. O *dataset* foi começado a gravar, já com o UAV em voo e terminado apenas após este aterrar.

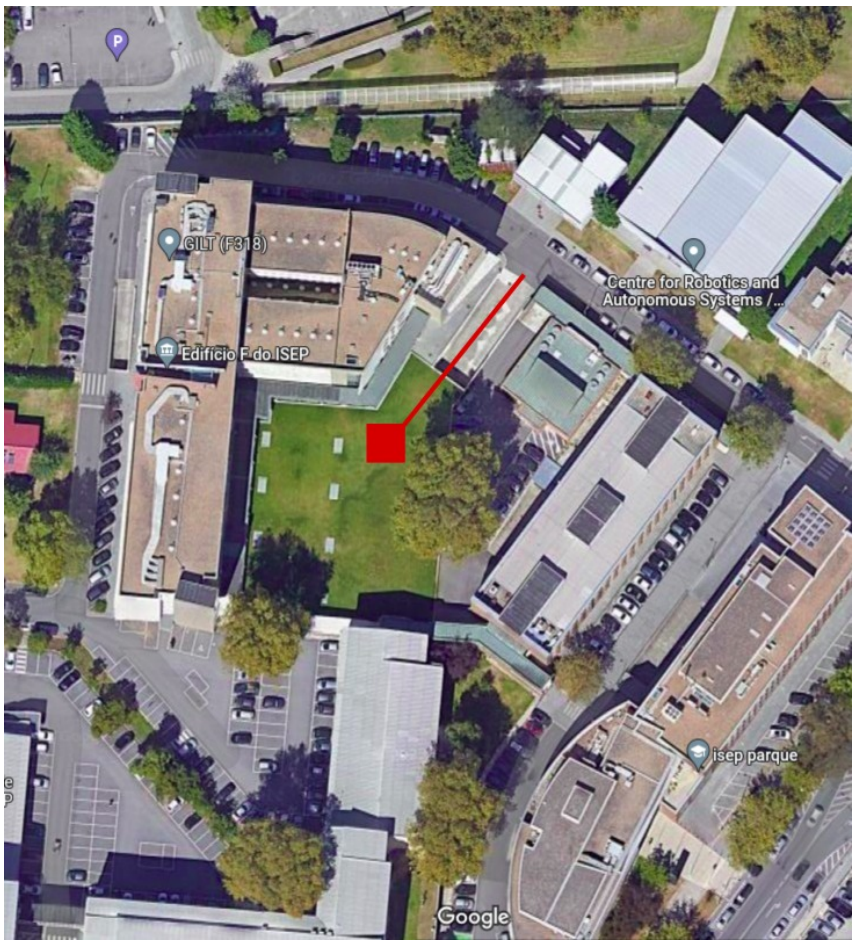


Figura 4.9: Localização e trajetória percorrida pelo UAV.

Figura 4.10: Edifício F do campus do ISEP.

Este segundo teste apresentou melhorias significativas nos resultados obtidos, como tal ao invés da forma resumida como foram apresentados os resultados do Capítulo 4.1.2, serão agora apresentados os resultados individuais de cada uma das implementações abordadas anteriormente.

4.2.1 Preparação do *Hardware*

Assim como apresentado no Capítulo 4.1.1, foi novamente utilizada a estação de GPS T300 apresentada na Figura 4.3 para uma maior precisão dos dados da localização e também o UAV apresentado na Figura 4.2. Sendo que neste segundo teste a única alteração no *hardware* utilizado recaiu sobre a variação no ângulo do LiDAR, sendo este rodado dos anteriores 90º (Figura 4.4) sobre Ox para os atuais 180º sobre Oy e 7.5º sobre Ox, que são possíveis de ser observados na Figura 4.11.

A escolha deste ângulo recaiu na necessidade de captar dados num ângulo diferente do anteriormente usado, que possibilitasse o aumento da densidade de pontos em Oy, sendo esta a única posição do LiDAR disponível para além da anteriormente apresentada.

Figura 4.11: Posicionamento do LiDAR no UAV.

4.2.2 Dados recolhidos do GPS

Como mencionado anteriormente, como *ground truth*, foram recolhido os dados do GPS/RTK. Após a receção dos dados, foi possível perceber que estes disponibilizavam a informação em diversos referenciais, sendo uns dos quais o referencial *Earth-centered, Earth-fixed coordinate system* (ECEF).

De forma a tornar dados mais fáceis de comparar e de mais fácil compreensão visual, estes foram então convertidos para o referencial ENU.

Após esta conversão foi então obtida a trajetória do UAV em ENU, sendo o resultado possível de ser observado na Figura 4.12.

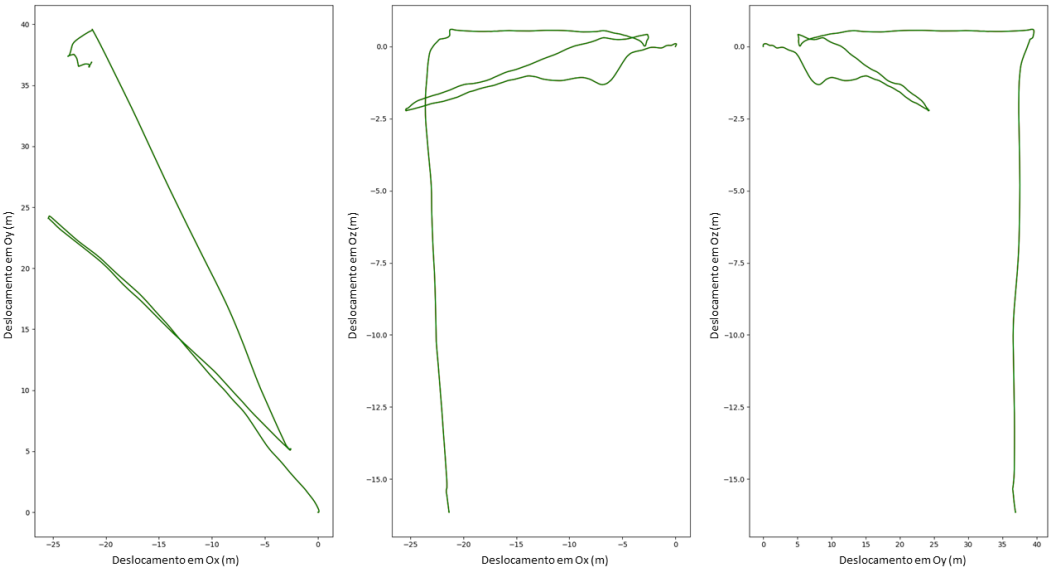


Figura 4.12: Dados do GPS após aplicada a rotação dos dados.

4.2.3 Resultados obtidos sem rotação prévia da nuvem de pontos

Esta secção pretende apresentar uma comparação entre a utilização dos diversos trabalhos [4], [10], [15] e [17], sem a rotação prévia da nuvem de pontos. Sendo possível desta forma comparar os diversos casos de uma forma simples e concisa.

As Figuras 4.13 a 4.16 apresentam cada uma das implementações, sendo que na primeira linha estas apresentam as vistas de cima e lateral do resultado final de cada uma das implementações, na segunda linha são apresentados os valores do erro ao longo do tempo para os diversos casos. Por fim, na terceira linha temos os valores de consumos de RAM e das frequências dos tópicos que entrada e saída dos diversos trabalhos.

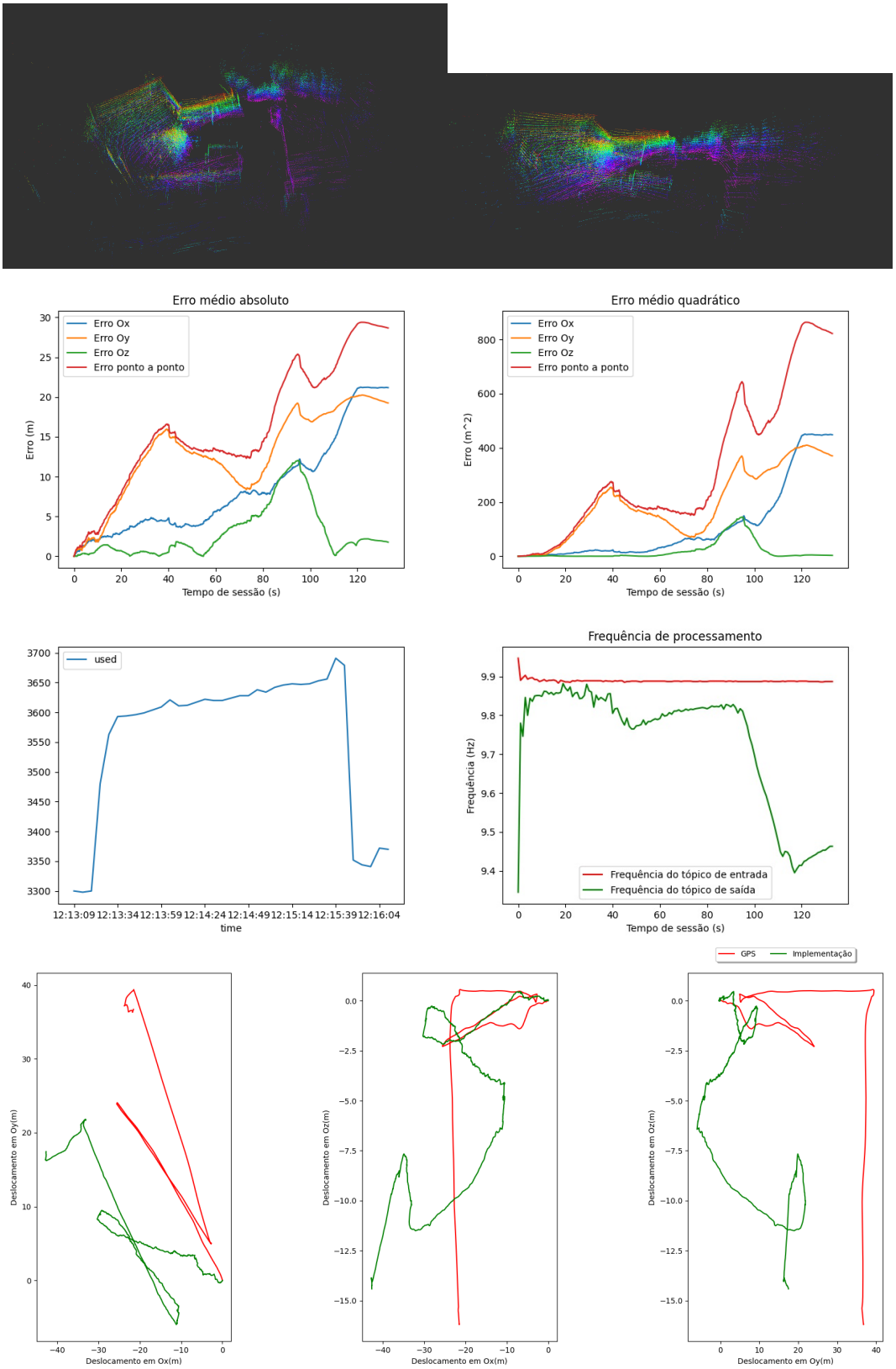


Figura 4.13: Resultados obtidos com recurso ao LOAM[4], onde temos a linha vermelha que representa o ground truth e a linha verde que representa a odometria estimada pela implementação.

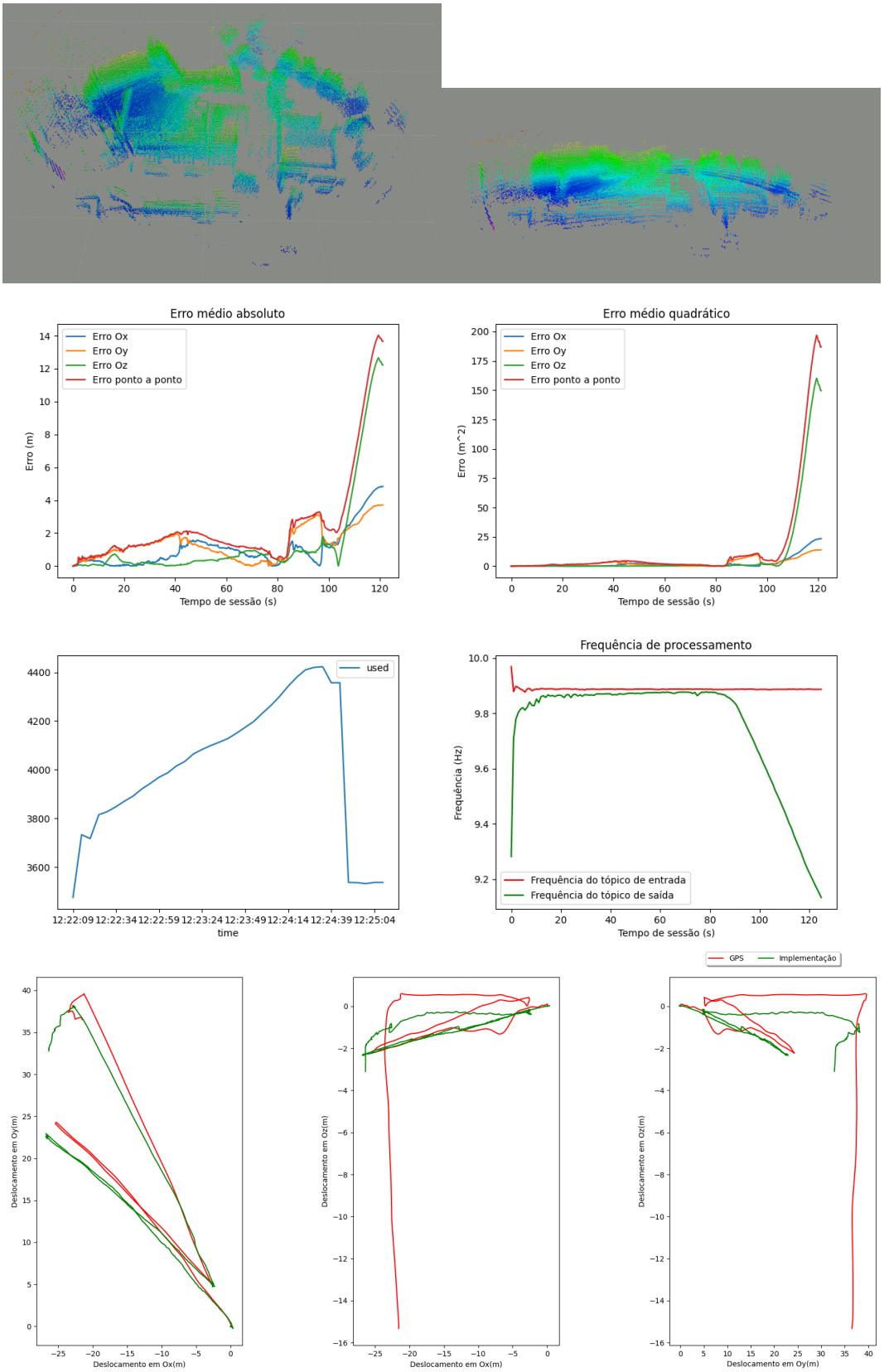


Figura 4.14: Resultados obtidos com recurso ao ISC-LOAM[10], onde temos a linha vermelha que representa o ground truth e a linha verde que representa a odometria estimada pela implementação.

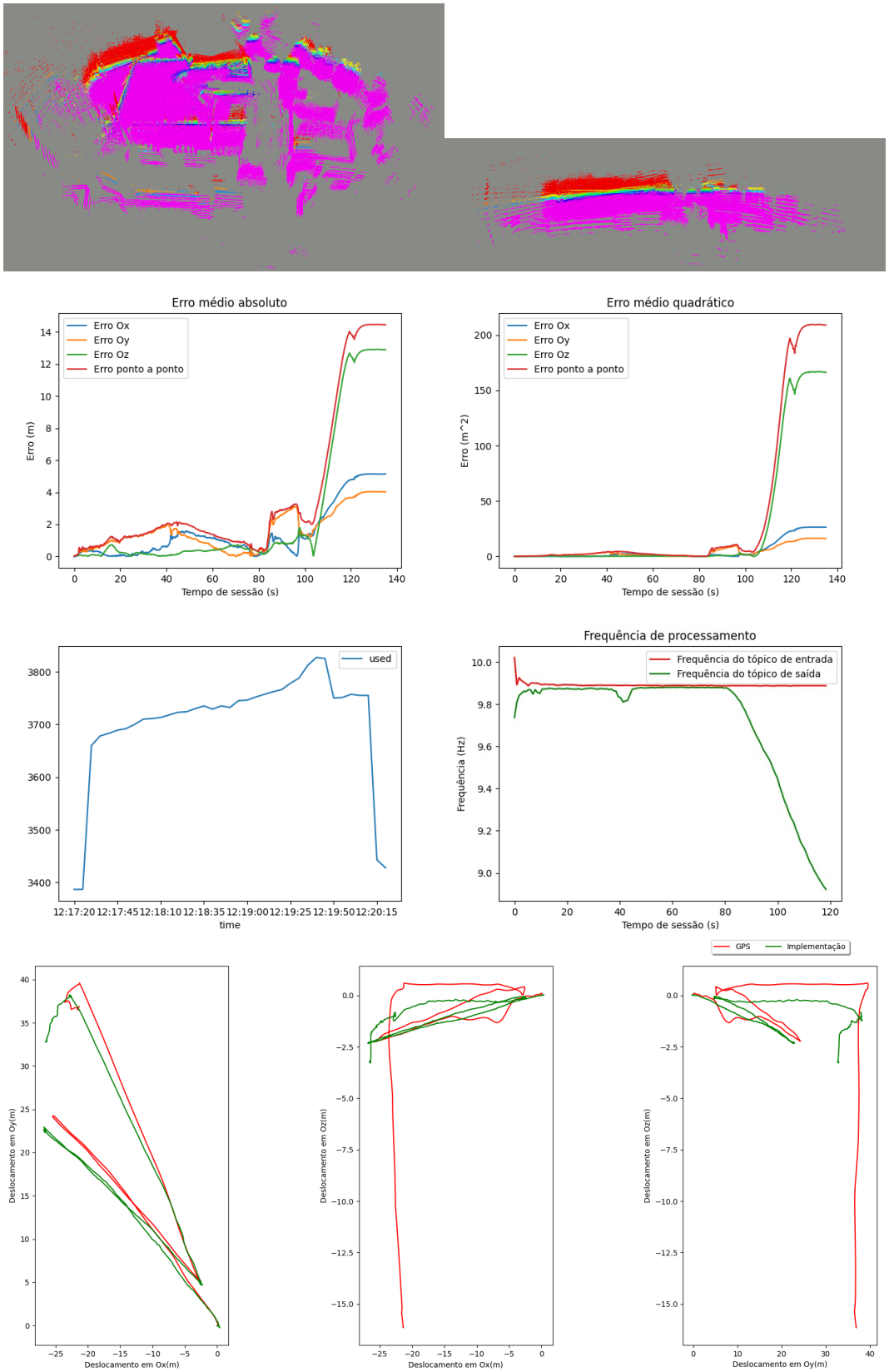


Figura 4.15: Resultados obtidos com recurso ao FLOAM[15], onde temos a linha vermelha que representa o ground truth e a linha verde que representa a odometria estimada pela implementação.

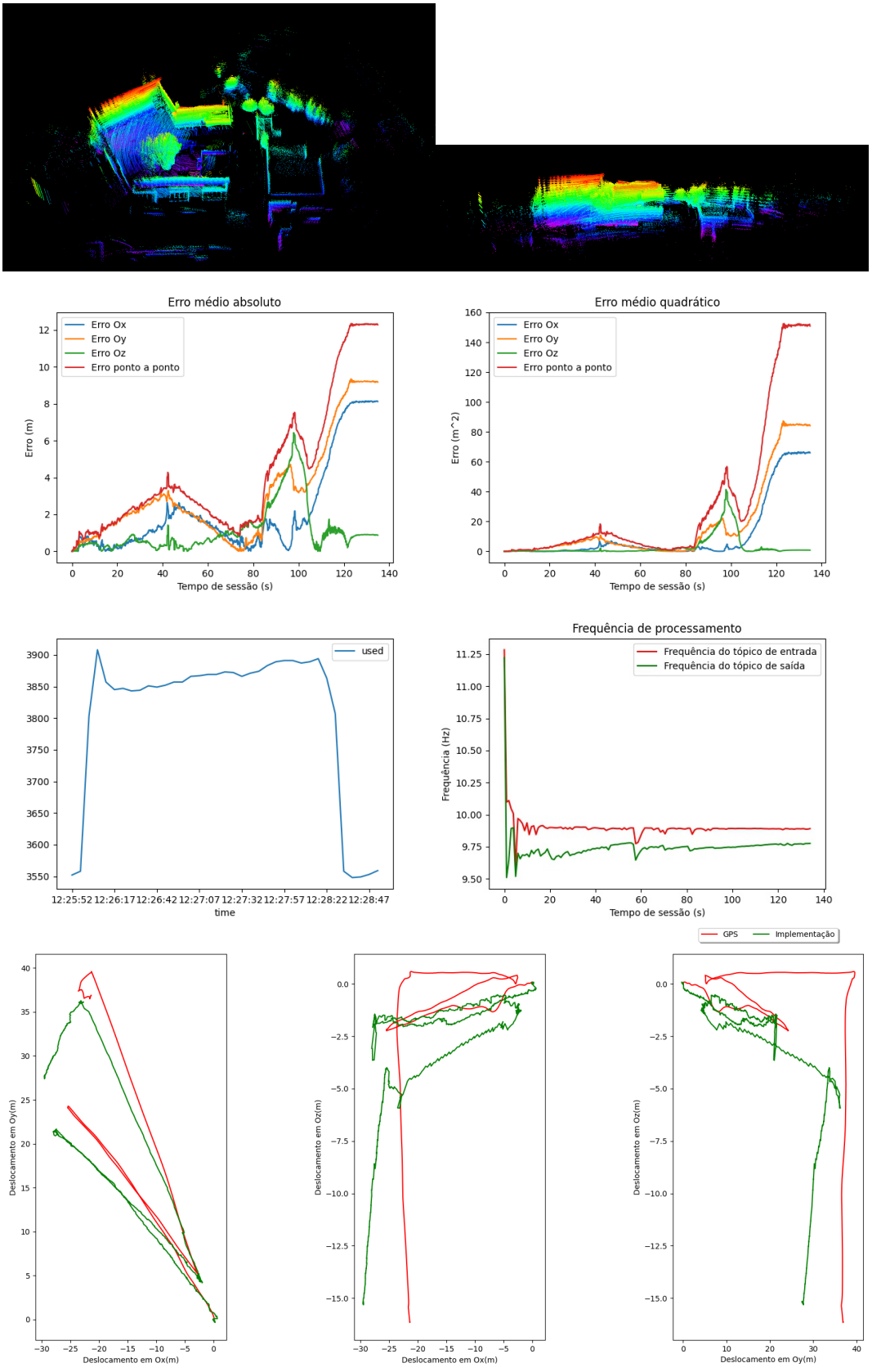


Figura 4.16: Resultados obtidos com recurso ao LEGO-LOAM[17], onde temos a linha vermelha que representa o ground truth e a linha verde que representa a odometria estimada pela implementação.

Através da analise dos gráficos de erro anteriormente apresentados, é possível perceber que os picos de erro nas diversas implementações acontece no mesmo momento, sendo que alguns métodos apresentam mais erro do que outros na resposta a esses momentos.

Através da análise dos dados é possível perceber que os dois primeiros picos apresentados correspondem aos momentos de inversão de sentido na trajetória do UAV e o último, corresponde ao caminho de volta para o local de aterragem assim como à aterragem.

Isto deve se ao facto deste último segmento da rota ter sido executado a uma velocidade mais eleva o que comprometeu a estabilidade do UAV, introduzindo assim ruído nas leituras.

Por fim, temos as Tabelas 4.2 e 4.3 que resumem os resultados do erro anteriormente discutidos. Como é possível observar nas mesmas, o trabalho que apresentou melhores resultados foi o trabalho [10], apresentando um erro médio absoluto de 2.53 metros entre a posição estimada e a posição real do UAV.

A única exceção no caso dos valores de erro apresentados, foi o valor ao longo de Oz, no qual a implementação [17] apresentou melhores resultados do que as demais.

Temos ainda os valores de performance de cada um dos trabalhos, através dos quais é possível concluir que em termos de consumos de RAM, apenas o trabalho [10] apresenta consumos diferentes dos demais, sendo que os seus consumos crescem com maior rapidez do que os demais, o que em tempo real pode-se tornar um problema em caso de voos mais longos.

Por outro lado, na comparação entre os valores da frequência dos tópicos, podemos perceber que excetuando o trabalho [17], todas as implementações apresentam valores muito próximo e estáveis de entrada e saída, o que nos indica que a informação está a ser processada em tempo real dentro das diversas implementações, sendo que todas elas apresentam uma baixa de performance no momento de pouso do UAV, o que coincide com o momento de maior erro de cada sessão. No caso do trabalho [17], este não apresenta qualquer perda de performance nos momentos finais da sessão.

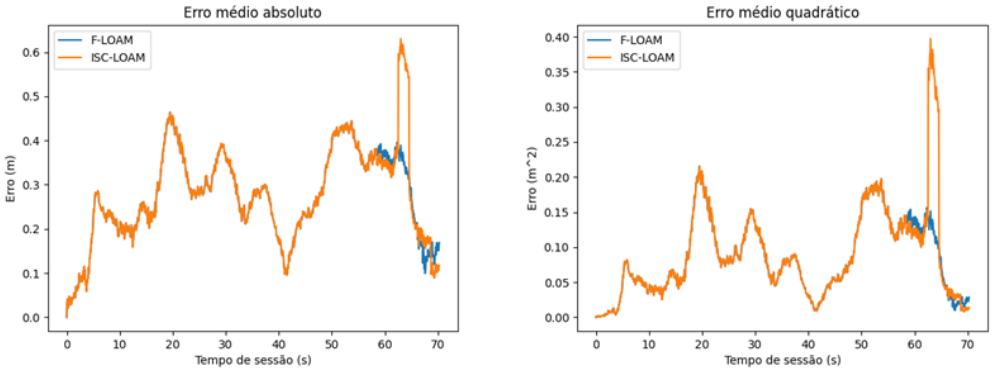


Figura 4.17: Gráfico resumo dos erros ponto a ponto.

	Ox	Oy	Oz	Erro entre a posição real e a posição estimada
LOAM [4]	8.22m	12.63m	2.89m	15.85m
ISC-LOAM[10]	**1.05m**	**1.35m**	1.43m	**2.53m**
FLOAM[15]	1.46m	1.61m	2.56m	3.71m
Lego-LOAM[17]	2.19m	3.16m	1.12m	4.32m

Tabela 4.2: Resultado do cálculo do MAE.

	Ox	Oy	Oz	Erro entre a posição real e a posição estimada
LOAM [4]	104.42m	189.44m	18.70m	312.57m
ISC-LOAM[10]	**2.41m**	**2.79m**	10.29m	**15.50m**
FLOAM[15]	4.81m	4.10m	26.06m	34.98m
Lego-LOAM[17]	11.21m	17.51m	2.97m	31.69m

Tabela 4.3: Resultado do cálculo do MSE.

4.2.4 Resultados obtidos com rotação prévia da nuvem de pontos

Esta secção pretende apresentar os resultados dos trabalhos [4], [10], [15] e [17] quando a rotação da nuvem de pontos é previamente executada.

Assim como na secção anterior, as Figuras 4.18 a 4.21 apresentam os dados obtidos com cada um dos trabalhos anteriormente mencionados.

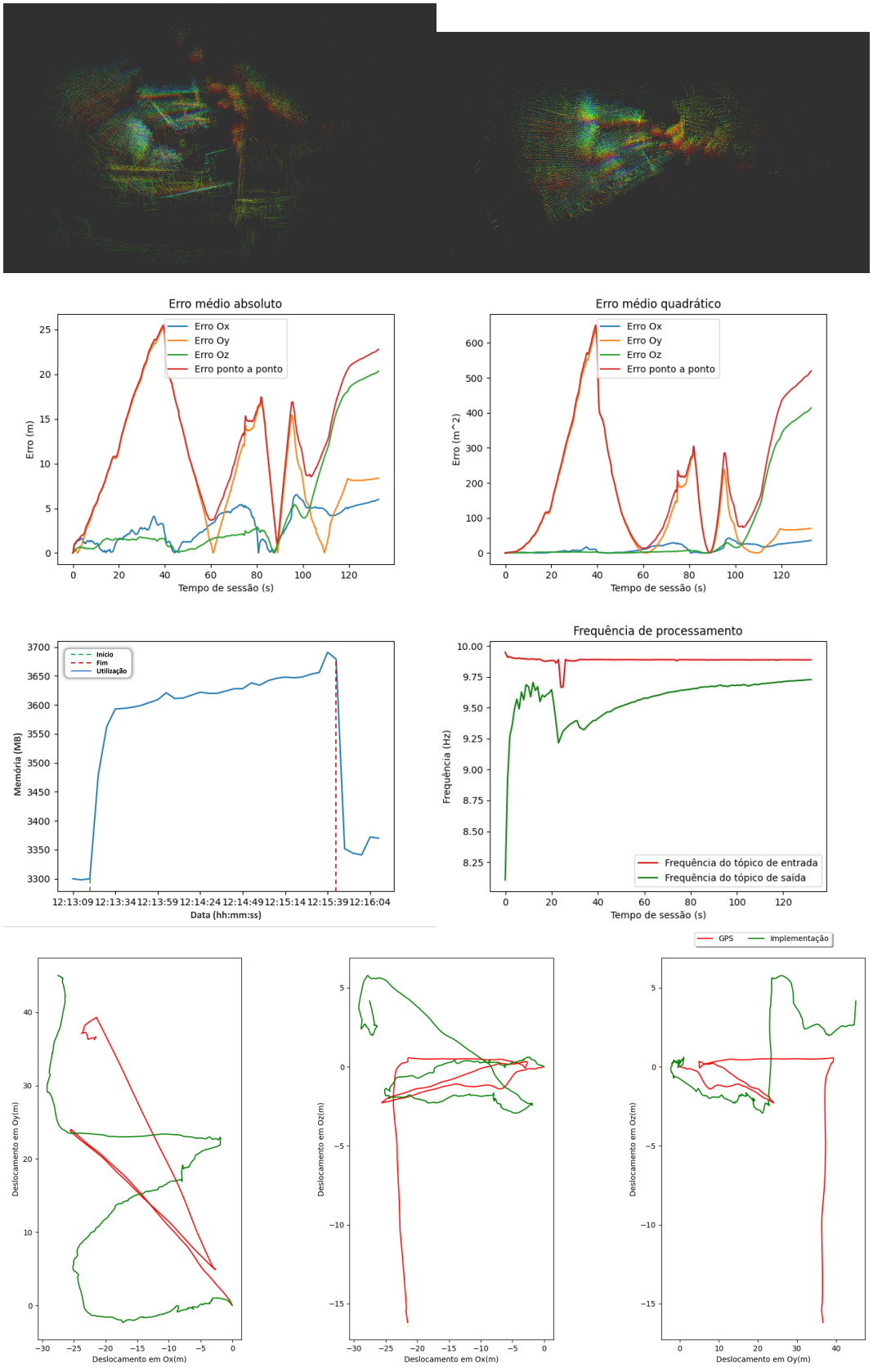


Figura 4.18: Resultados obtidos com recurso ao LOAM [4], após rotação da nuvem de pontos, onde temos a linha vermelha que representa o ground truth e a linha verde que representa a odometria estimada pela implementação.

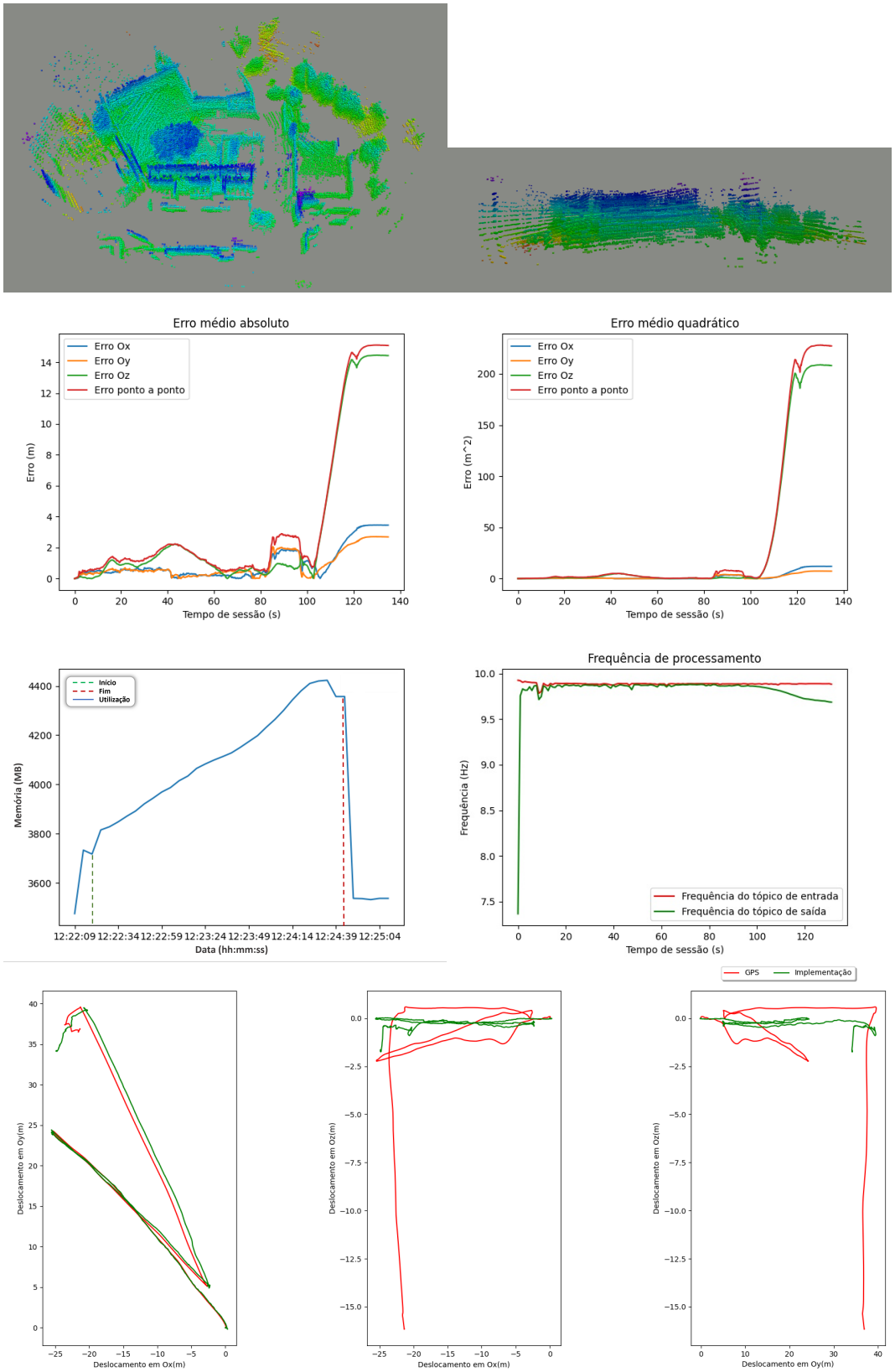


Figura 4.19: Resultados obtidos com recurso ao ISC-LOAM [10], onde temos a linha vermelha que representa o ground truth e a linha verde que representa a odometria estimada pela implementação

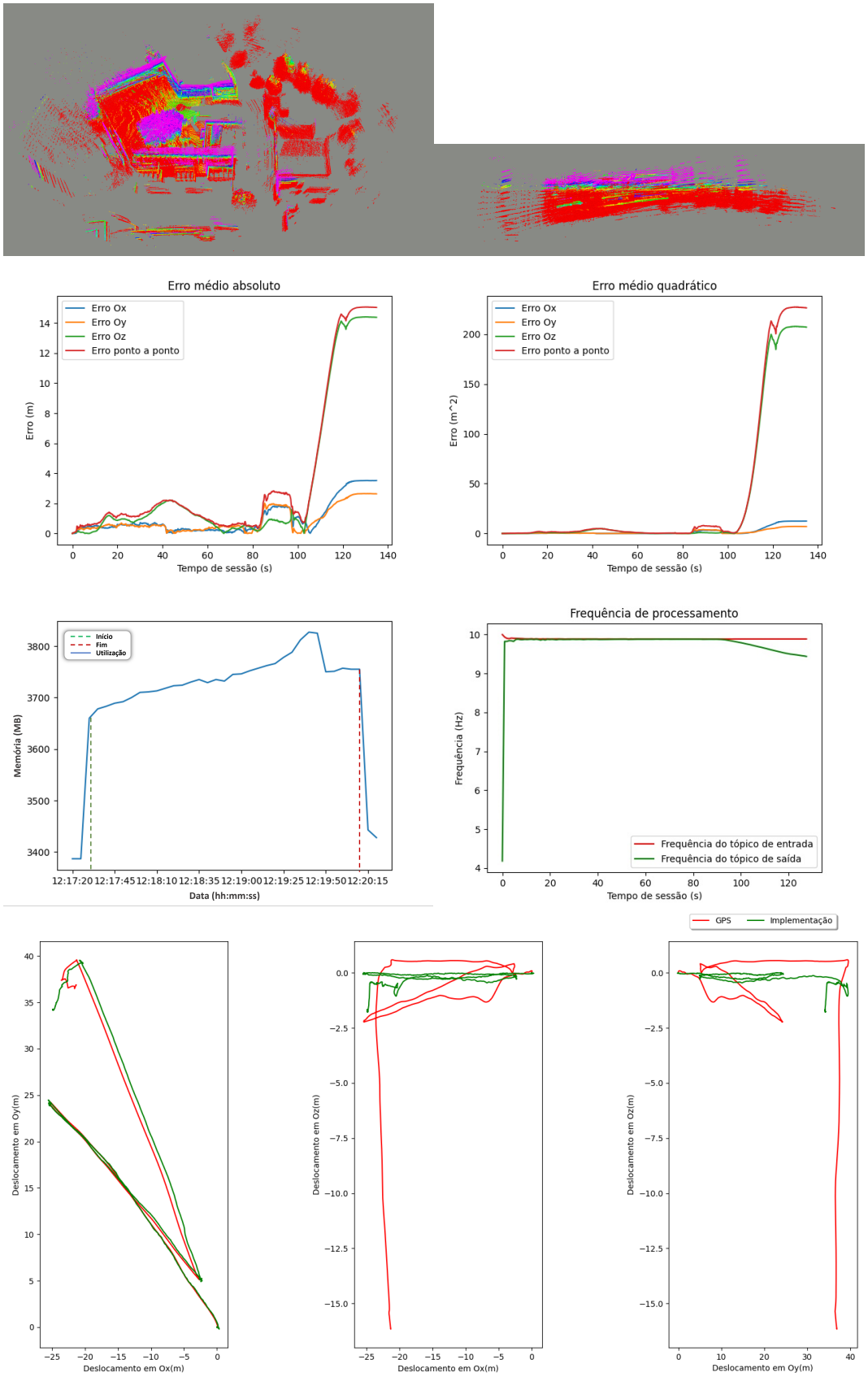


Figura 4.20: Resultados obtidos com recurso ao FLOAM [15], onde temos a linha vermelha que representa o ground truth e a linha verde que representa a odometria estimada pela implementação

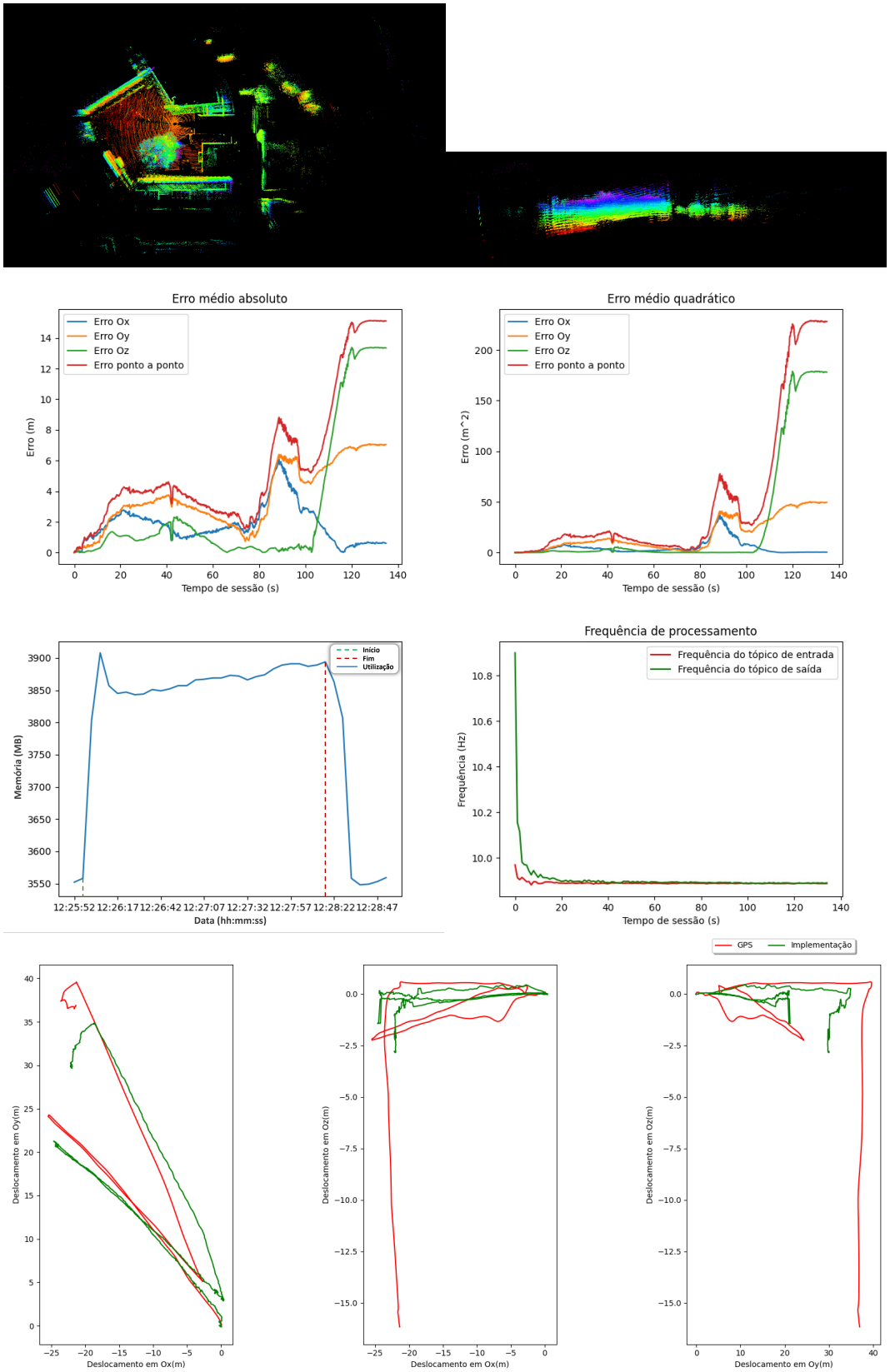


Figura 4.21: Resultados obtidos com recurso ao LEGO-LOAM [17], onde temos a linha vermelha que representa o ground truth e a linha verde que representa a odometria estimada pela implementação

Através da análise dos gráficos de erro do novo teste, é possível perceber que os dois casos onde as implementações acumulam mais erro voltaram a ser notórios. Sendo que estes se devem novamente aos motivos apresentados na secção anterior.

De seguida temos as Tabelas 4.4 e 4.5 onde, através da analise dos dados apresentados é possível perceber que os trabalhos que apresentaram melhores resultados foram [15] e novamente [10] com um erro médio absoluto de 3.65 metros para o primeiro caso e 3.68 metros para o segundo.

De notar que, novamente, como é possível observar no erro médio quadrático o erro Oz da aterragem apresenta um peso elevado no resultado final do erro.

No caso dos dados de processamento, é agora possível notar que todos os trabalhos à exceção do trabalho [10] apresentaram um maior consumo de RAM (cerca de 30%), este comportamento é justificado pela necessidade de efetuar a rotação da nuvem de pontos em tempo real, o que leva a uma maior necessidade de processamento. Já no caso do trabalho [10], este comportamento pode ser explicado pela a mais fácil identificação de pontos de interesse, uma vez que o mundo se encontra com a direção esperado.

Para o caso das variações nas frequências de comunicação dos tópicos, podemos novamente observar que uma degradação das mesmas ocorreu novamente no momento em que erros de leitura maiores foram observados.

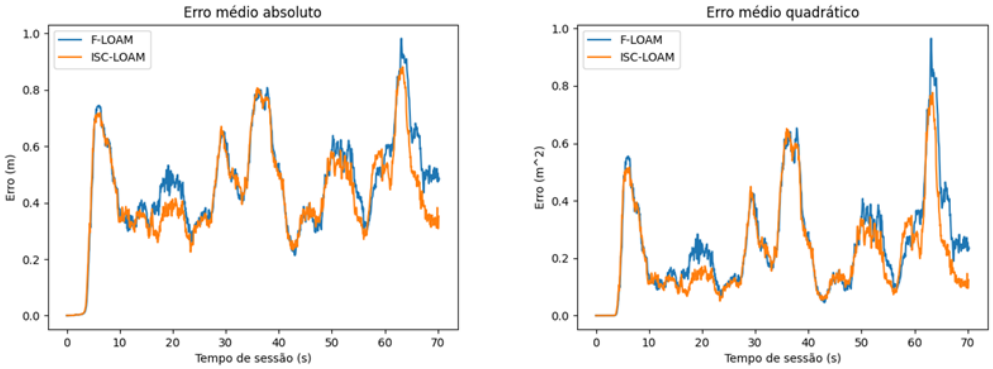


Figura 4.22: Gráfico resumo dos erros ponto a ponto.

	Ox	Oy	Oz	Erro entre a posição real e a posição estimada
LOAM [4]	3.08m	9.64m	4.79m	12.92m
ISC-LOAM[10]	**0.97m**	0.87m	3.24m	3.68m
FLOAM [15]	0.98m	**0.85m**	**3.22m**	**3.65m**
Lego-LOAM[17]	1.82m	3.66m	2.93m	5.72m

Tabela 4.4: Resultado do cálculo do MAE.

	Ox	Oy	Oz	Erro entre a posição real e a posição estimada
LOAM [4]	13.07m	133.67m	62.53m	209.27m
ISCLOAM	**2.05m**	1.47m	33.82m	37.35m
FLOAM[15]	2.09m	**1.41m**	**33.58m**	**37.08m**
Lego-LOAM[17]	4.87m	17.82m	28.78m	51.47m

Tabela 4.5: Resultado do cálculo do MSE.

4.2.5 Conclusões

Através da comparação dos valores de erro apresentados nas Tabelas 4.2 e 4.3, com os valores apresentados nas Tabelas 4.4 e 4.5, é possível observar que com a rotação da nuvem de pontos, os valores de erro diminuíram à exceção dos valores de Oz. No contexto desta dissertação este valor não apresenta uma preponderância tão significativa como os demais, uma vez que o UAV utiliza um barómetro para avaliar a altitude a que se encontra.

De notar que as implementações [4] e [17] apresentam a possibilidade de utilizar o IMU como forma de melhorar os dados provenientes do LiDAR, o que poderia ter ajudado a melhores resultados no que diz respeito a estas implementações.

De notar ainda que com a possibilidade de executar mais testes, as rotações poderiam ser melhor calibradas, o que poderia levar a uma ligeira variação do erro.

Através da análise dos diversos gráficos do erro médio absoluto é possível perceber que em momentos de rotação mais rápidas ou velocidades mais elevadas as diversas implementações acumulam erro, mas este diminui assim que as condições ideias são novamente satisfeitas.

Por outro lado, comparando os gráficos de performance acima apresentados, é possível notar um incremento de cerca de 30% nos consumos de RAM das diversas implementações do primeiro para o segundo teste, sendo este o dado mais importante a retirar, uma vez que devido ás possíveis limitações de performance do UAV o primeiro caso pode se apresentar como o mais indicado, apesar de apresentar um erro ligeiramente maior.

4.3 Teste realizado com *dataset open source*

Com o objetivo de testar as conclusões anteriormente retiradas, foi então utilizado um *dataset* disponibilizado em [30]. Aqui são fornecidos diversos tipos de dados com o objetivo de possibilitar o desenvolvimento na área da visão, dos quais, um *dataset* recolhido por um UAV equipado com um *Velodyne Puck 16* (VLP-16) numa sala (Figura 4.23).

Figura 4.23: Espaço no qual foi gravado o *dataset* fornecido em em [30].

4.3.1 *Hardware* utilizado

Para a recolha destes dados foi utilizado o UAV apresentado na Figura 4.24.

Figura 4.24: UAV utilizado na recolha de dados em [30].

Não são apresentados os dados em relação à orientação deste, apenas a esquemática apresentada na Figura 4.25. Apesar disto, através de alguns testes com os dados obtidos foi possível perceber que o LiDAR se encontra rodado sobre Oy cerca de 30º.

4.3.2 Resultados obtidos sem rotação prévia da nuvem de pontos

Esta secção pretende apresentar os resultados obtidos com as implementações [10] e [15] sem proceder à rotação prévia da nuvem de pontos. Esta secção irá apenas se focar nestes dois trabalhos pois como é possível observar na Figura 4.26, os trabalhos [4] e [17] apresentaram um erro tão elevado que é até possível observar no mapeamento.

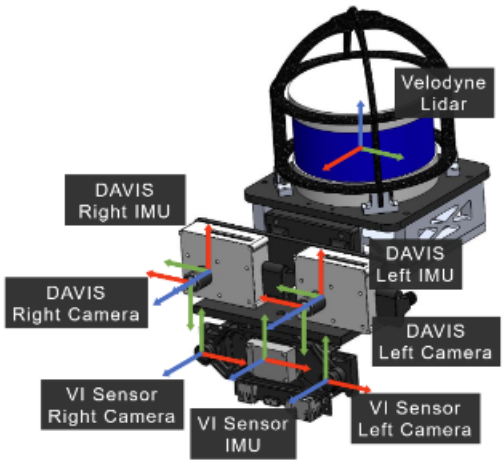


Figura 4.25: Posicionamento do *Hardware* em [30].

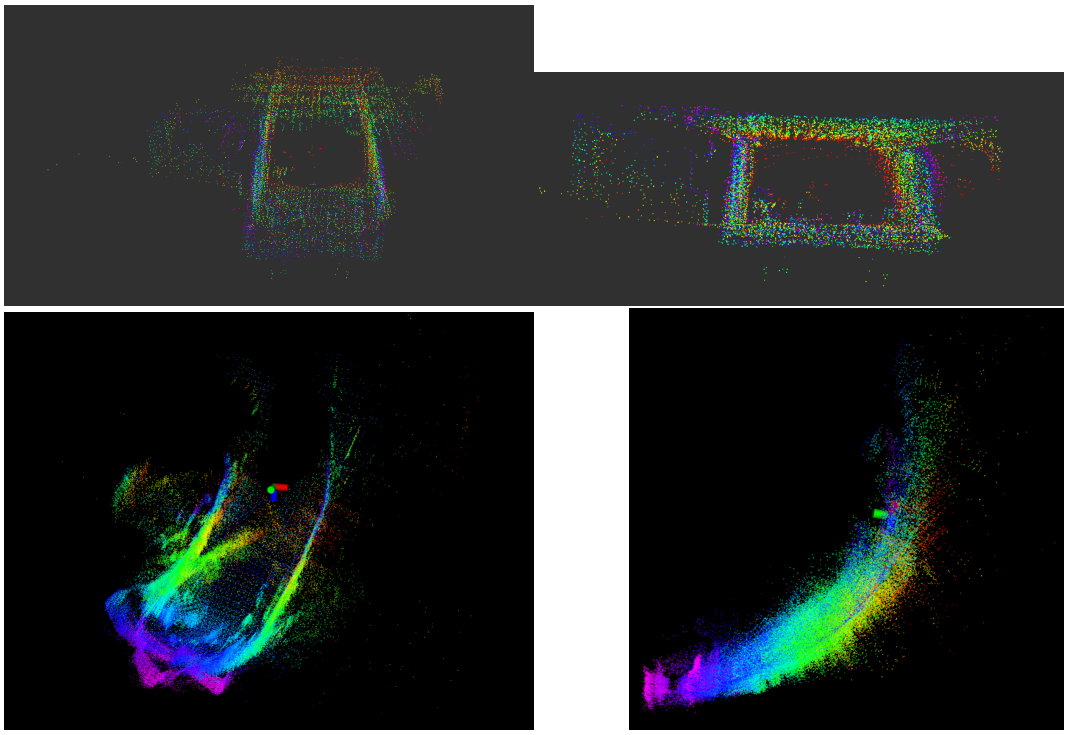


Figura 4.26: Resultados obtidos com recurso aos trabalhos LOAM[4] e LEGO-LOAM[17].

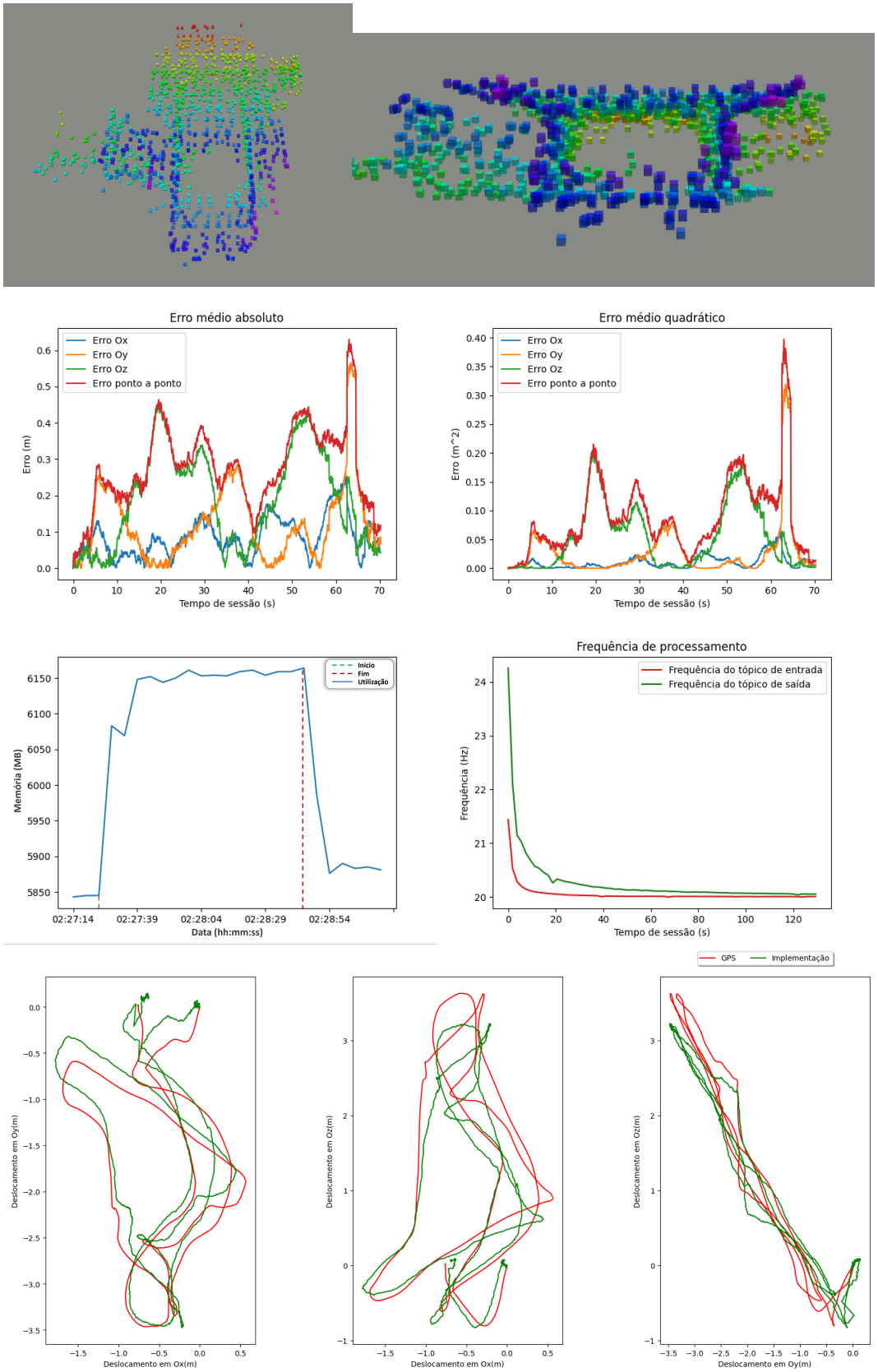


Figura 4.27: Resultados obtidos com recurso ao trabalho ISC-LOAM[10], onde temos a linha vermelha que representa o ground truth e a linha verde que representa a odometria estimada pela implementação.

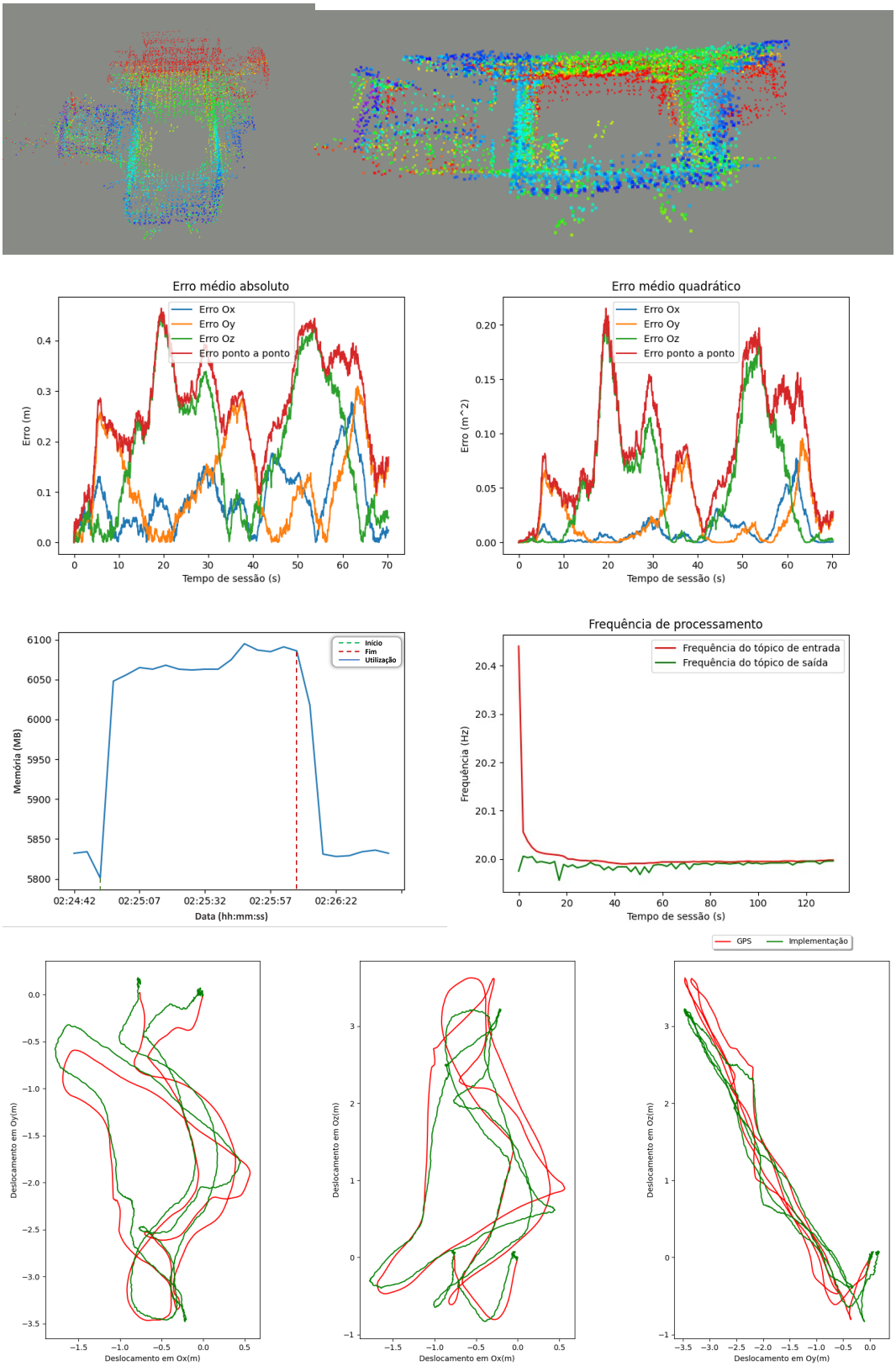


Figura 4.28: Resultados obtidos com recurso ao trabalho FLOAM[15], onde temos a linha vermelha que representa o ground truth e a linha verde que representa a odometria estimada pela implementação.

Apesar do espaço reduzido onde os dados foram recolhidos, este permite ainda assim perceber que, mais uma vez, os picos de erro advém das mudanças de direção, pois através de uma análise dos gráficos e da secção foi possível verificar que estes ocorriam ao mesmo tempo que as ligeiras mudanças de direção.

Permite também confirmar novamente que uma grande percentagem do erro da implementação corresponde aos dados do eixo Oz, sendo que estes dados são apresentados de forma mais percetível nas Tabelas 4.6 e 4.7.

No que diz respeito aos dados da performance, os diversos trabalhos apresentaram um consumo semelhante entre eles, repetindo o comportamento apresentado na secção 4.2.3.

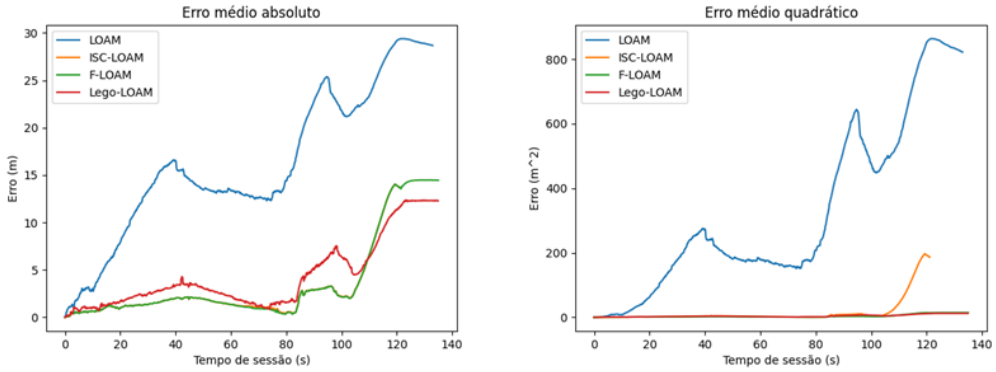


Figura 4.29: Gráfico resumo dos erros ponto a ponto.

	Ox	Oy	Oz	Erro entre a posição real e a posição estimada
ISC-LOAM[10]	**0.085m**	0.123m	0.188m	0.277m
FLOAM[15]	**0.085m**	**0.116m**	**0.186m**	**0.272m**

Tabela 4.6: Resultado do cálculo do MAE.

	Ox	Oy	Oz	Erro entre a posição real e a posição estimada
ISC-LOAM[10]	**0.010m**	0.027m	0.052m	0.090m
FLOAM[15]	0.011m	**0.020**	**0.053m**	**0.084m**

Tabela 4.7: Resultado do cálculo do MSE.

4.3.3 Resultados obtidos com rotação prévia da nuvem de pontos

Assim como na secção anterior, também nesta serão apresentados os dados com os resultados obtidos com as implementações [10] e [15], mas com a rotação prévia da

nuvem de pontos. Novamente podemos observar os dados obtidos pelos trabalhos [4] e [17], na Figura 4.30, os quais novamente acumularam demasiado erro.

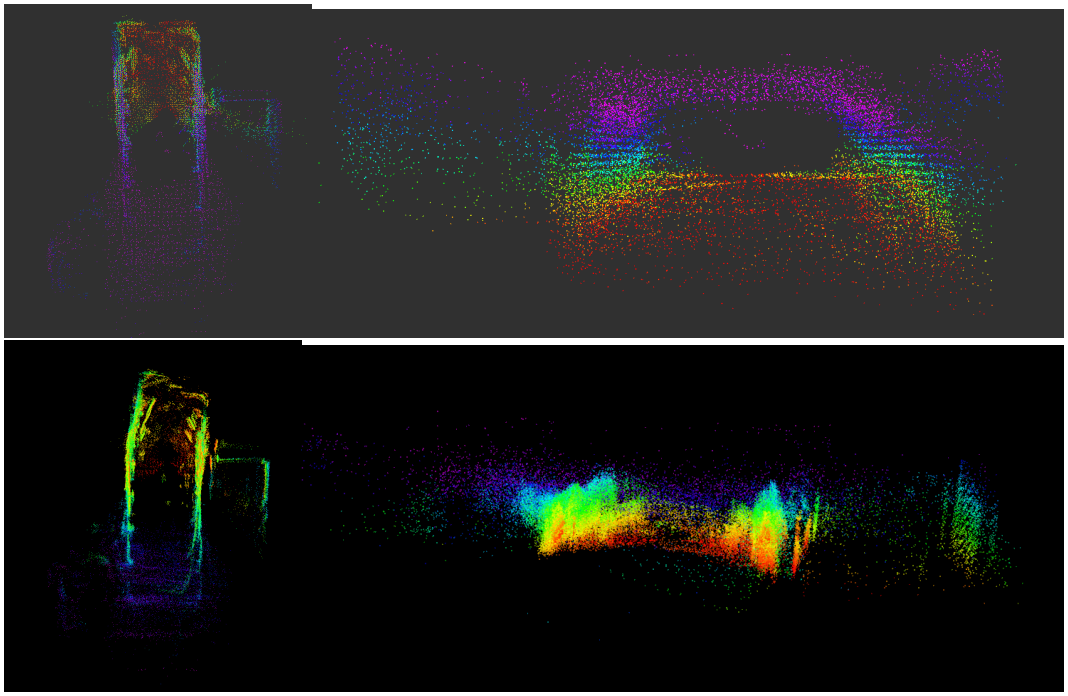


Figura 4.30: Resultados obtidos com recurso aos trabalhos LOAM[4] e LEGO-LOAM[17].

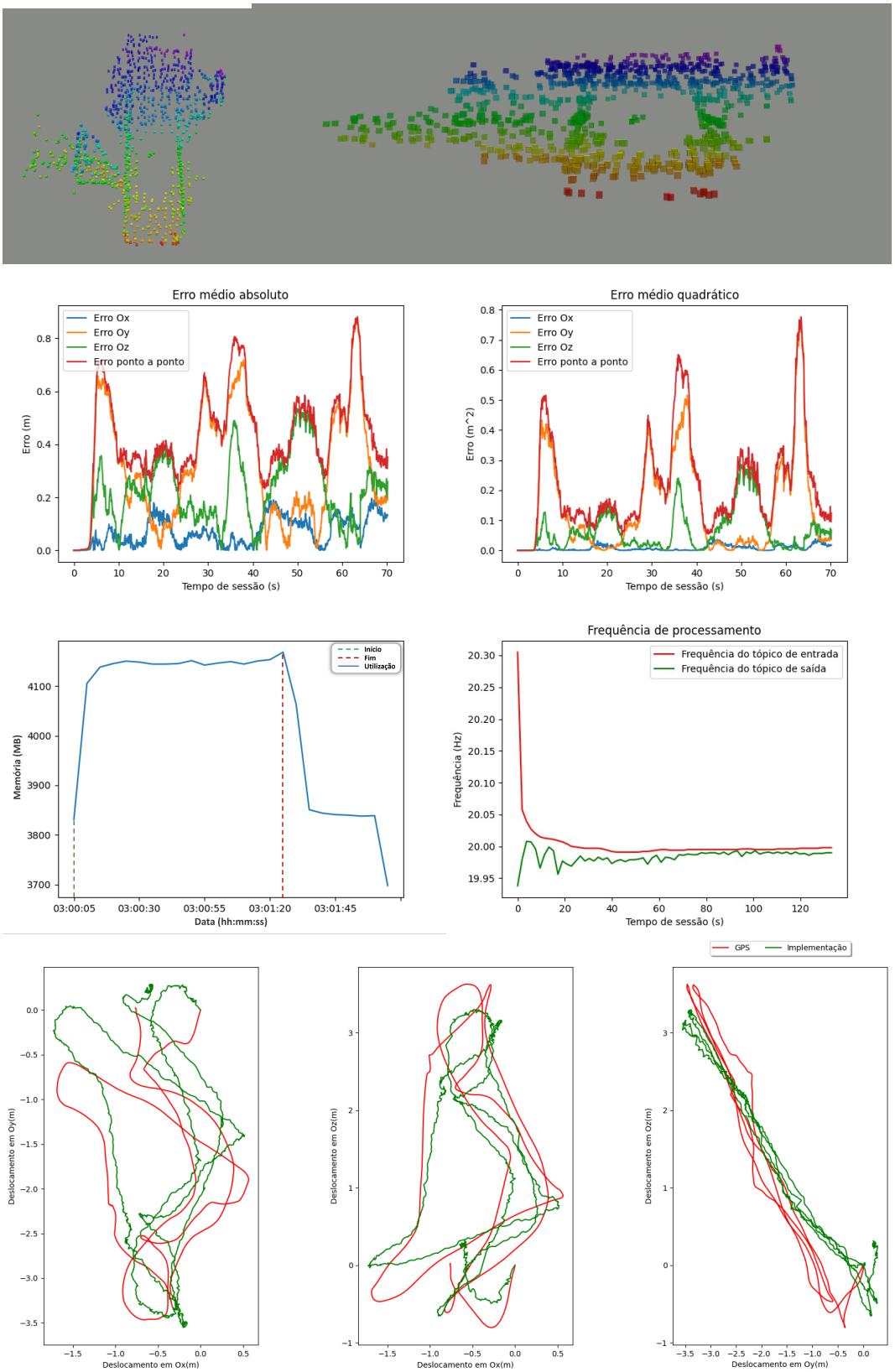


Figura 4.31: Resultados obtidos com recurso ao trabalho ISC-LOAM[10], onde temos a linha vermelha que representa o ground truth e a linha verde que representa a odometria estimada pela implementação.

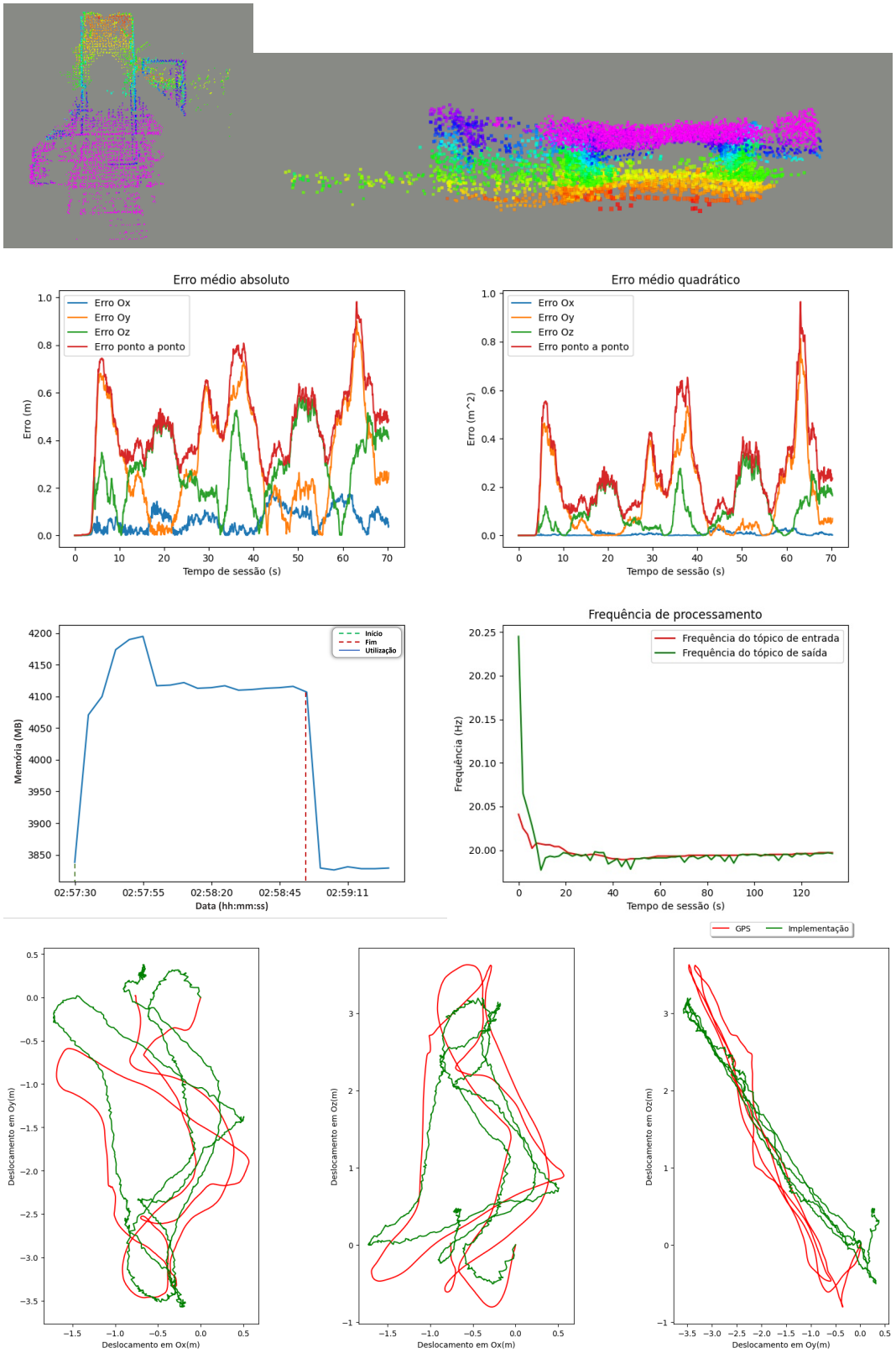


Figura 4.32: Resultados obtidos com recurso ao trabalho FLOAM[15], onde temos a linha vermelha que representa o ground truth e a linha verde que representa a odometria estimada pela implementação.

Através da análise dos diversos gráficos de erro é possível perceber que mais uma vez os valores confirmam que devido ás mudanças de direção as diversas implementações acumulam erro, o qual se decipa assim que as condições são repostas.

É possível ainda observar que neste caso, o erro aumentou após a rotação da nuvem de pontos, o que se pode dever ao facto da ausência de dados suficientes para rodar a nuvem de pontos para a posição exata como base no posicionamento do LiDAR.

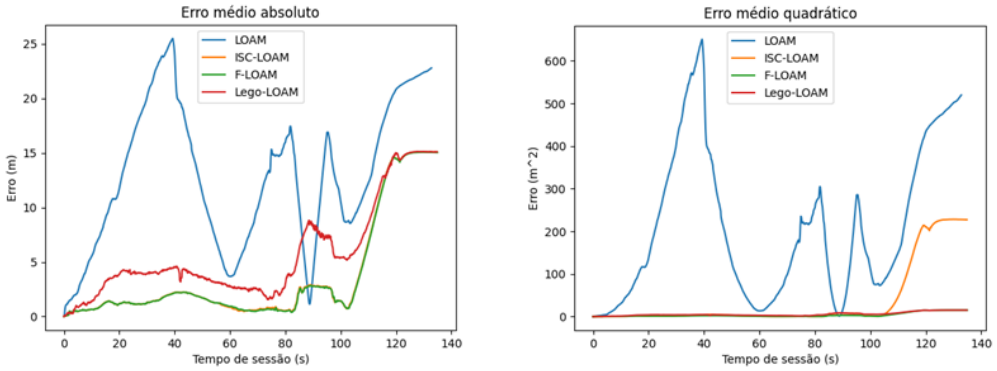


Figura 4.33: Gráfico resumo dos erros ponto a ponto.

	Ox	Oy	Oz	Erro entre a posição real e a posição estimada
ISC-LOAM[10]	**0.07m**	**0.316m**	**0.218m**	**0.434m**
FLOAM[15]	0.063m	0.309m	0.272m	0.464m

Tabela 4.8: Resultado do cálculo do MAE.

	Ox	Oy	Oz	Erro entre a posição real e a posição estimada
ISC-LOAM[10]	**0.008m**	**0.147m**	**0.065m**	**0.220m**
FLOAM[15]	0.006m	0.146m	0.098m	0.250m

Tabela 4.9: Resultado do cálculo do MSE.

Capítulo 5

Conclusões

Após os diversos testes executados e a análise dos mesmos, podemos por fim concluir que, de entre os trabalhos em estudo, LOAM[4], Isc-LOAM[10], FLOAM[15] e Lego-LOAM[17], os trabalhos Isc-LOAM[10] e FLOAM[15] foram aqueles que apresentaram os resultados mais promissores no contexto desta dissertação. Estes dois trabalhos apresentaram resultados consistente ao longo dos diversos testes, ao contrário dos dois trabalhos restantes, que em alguns testes apresentaram erros de odometria e mapeamento tão elevados que impossibilitou o reconhecimento do mundo.

Dado o exposto podemos concluir que o *node* de rotação da nuvem de pontos deve ser incorporado na implementação, uma vez que este ajuda a minimizar o erro apresentado pelos diversos trabalhos. Foi ainda observado que missões em áreas com baixa quantidade de objetos geométricos devem ser evitada a utilização dos diversos trabalhos estudados uma vez que, a ausência de pontos de interesse impossibilita o correto funcionamento do ICP.

Tendo em vista os gráficos dos erros, podemos ainda retirar alguns cuidados a ter com as trajetórias de forma a minimizar o erro na odometria, das quais:

- Velocidades mais elevadas devem ser evitadas, uma vez que estas tornam as nuvens de pontos menos densas;
- Rotações repentinas do UAV devem ser evitadas, uma vez que levam a distorções na deteção dos pontos;

- Movimentos a velocidades mais elevadas do UAV, como mudanças de direção abruptas, devem também ser evitadas uma vez que comprometem a estabilidade do UAV;
- Movimentos em Oz devem ser baseados noutro sensor, pois como apresentado, este eixo apresenta um erro elevado.

Tendo em vista os argumentos apresentados, podemos por fim afirmar que as duas implementações mencionadas seriam uma boa adição a um UAV, numa primeira fase como sistema de segurança de forma a responder a possíveis casos de *jamming* do sinal. E após alguns testes no *hardware* específico e calibração dos dados tendo em vista a diminuição do erro, criar a possibilidade de executar a navegação apenas com base no mapeamento previamente executado.

5.1 Trabalho Futuro

Como trabalho futuro, seria interessante numa primeira fase executar mais testes, possibilitando assim uma calibração mais precisa da rotação da odometria executada pelas implementações baseadas no trabalho [4]. Dos quais, alguns testes com o objetivo de apenas calibrar as rotações de forma mais precisa.

Numa segunda fase, passaria por testar novamente os resultados das duas implementações descartadas (LOAM[4] e Lego-LOAM[17]), com auxilio dos dados do IMU, uma vez que como apresentado no estado da arte, este melhora significativamente os resultados apresentados. Podendo ainda, devido à natureza do IMU, ajudar a combater o erro adicionado às leituras pelas vibrações do UAV.

Numa terceira fase, seria interessante executar testes no *hardware* do UAV de forma a perceber se a performance se mantém ou até se a inclusão de uma das implementações provoca a degradação de algum dos outros sistemas.

Por fim, o último passo passaria pela possibilidade de incorporar um sensor que possibilitasse medidas precisas em Oz, integrando o mesmo na implementação, tornando assim mais robusta aos movimentos verticais.

Capítulo 6

Código desenvolvido

6.1 frame_tf_broadcaster.cpp

```
#include <ros/ros.h>
#include <tf/transform_broadcaster.h>

int main(int argc, char **argv) {
    ros::init(argc, argv, "my_tf_broadcaster");
    ros::NodeHandle n;

    ros::Rate r(100);

    tf::TransformBroadcaster broadcaster;

    while (n.ok()) {
        broadcaster.sendTransform(
                tf::StampedTransform(
                        tf::Transform(tf::Quaternion(0, 0.9978589,
                        0, 0.0654031), tf::Vector3(0.0, 0.0, 0.0)),
                        ros::Time::now(),  "camera_init",
                        "lidar_position"));
        r.sleep();
```

```
    }
}
```

6.2 rotate_pcl.cpp

```
#include "ros/ros.h"
#include "std_msgs/String.h"
#include <pcl_ros/transforms.h>

#include <sensor_msgs/PointCloud2.h>
#include <pcl_conversions/pcl_conversions.h>

ros::Subscriber sub;
ros::Publisher pub;
tf::StampedTransform transform_ref;

void callbackVelodyne(const sensor_msgs::PointCloud2ConstPtr &msg_1)
{
    pcl::PointCloud<pcl::PointXYZI> laserCloudIn_1;
    pcl::PointCloud<pcl::PointXYZI> laserCloudIn_2;
    pcl::fromROSMsg(*msg_1, laserCloudIn_1);

    pcl_ros::transformPointCloud(laserCloudIn_1, laserCloudIn_2,
                        transform_ref);

    sensor_msgs::PointCloud2 msg_2;
    pcl::toROSMsg(laserCloudIn_2, msg_2);

    pub.publish(msg_2);
}

int main(int argc, char **argv)
{
    ros::init(argc, argv, "listener");
    ros::NodeHandle a, b, c;

    pub = b.advertise<sensor_msgs::PointCloud2>("/rotatedPCL", 10000);
    sub = c.subscribe<sensor_msgs::PointCloud2>("/velodyne_points",
```

```
                        100, callbackVelodyne);

    tf::TransformListener listener;

    ros::Rate rate(100);
    while (ros::ok()) {
        ros::spinOnce();

        try {
            listener.lookupTransform("/camera_init", "/lidar_position",
                        ros::Time(), transform_ref);
        }
        catch (tf::TransformException ex) {
            ROS_ERROR("%s", ex.what());
        }

        rate.sleep();
    }
    ros::spin();

    return 0;
}
```

6.3 recordAndConvertFileWithRamData.sh

```
#!/bin/bash

# Recording interval
interval=$1
# Recording duration
duration=$2
# Output file name
outputFile=$3

endtime=$(date -ud "$duration" +%s)

echo "      date     time $(free -m | grep total |
sed -E 's/^    (.*)/\1/g')" >> $outputFile
```

```
while [[ $(date -u +%s) -le $endtime ]]; do
    echo "$(date '+%Y-%m-%d %H:%M:%S') $(free -m |
    grep Mem: | sed 's/Mem://g')" >> $outputFile

    sleep $interval
done

# Remove multiple spaces and keep only one
sed -i 's/[[:space:]]\{1,\}/ /g' $outputFile

mv $outputFile ./dummy.txt
sed 's/^ //' dummy.txt >> $outputFile
rm dummy.txt

python3 ./recordRamUsage/plotRamConsumption.py
        --input-path ./$outputFile

exit
```

6.4 plotRamConsumption.py

```
#!/usr/bin/python

import argparse
import pandas as pd
import pathlib
import matplotlib.pyplot as plt

PARQUET_NAME = 'ramData.parquet'
IMAGE_NAME = 'ramData.png'

class PlotRamConsumption:
    def __init__(self, input_path):
        self.input_path = pathlib.Path(input_path)

    def main(self):
        data = pd.read_csv(self.input_path, sep=' ')
```

```
        self.plotGraph(data)
        data.to_parquet(PARQUET_NAME)

    def plotGraph(self, data):
        data.plot(x='time', y='used', kind='line')
        plt.savefig(IMAGE_NAME)

if __name__ == '__main__':
    parse = argparse.ArgumentParser()
    parse.add_argument("--input-path", required=True,
                       help="Path to the ram recorded file")

    args = parse.parse_args()

    plot_ram_consumption = PlotRamConsumption(
        input_path=args.input_path,
    )

    plot_ram_consumption.main()
```

6.5 compare_and_sync.py

```
#!/usr/bin/python

import argparse
import pandas as pd
import pathlib


class CompareAndReduce:
    def __init__(self, input_path_1, input_path_2):
        self.input_path_1 = pathlib.Path(input_path_1)
        self.input_path_2 = pathlib.Path(input_path_2)

    def main(self):
        data_1 = pd.read_csv(self.input_path_1, sep=' ')
                    .sort_values(by=['UTCTime'])
        data_1["X-ECEF"] = data_1["X-ECEF"] - data_1["X-ECEF"].iloc[0]
```

```
        data_1["Y-ECEF"] = data_1["Y-ECEF"] - data_1["Y-ECEF"].iloc[0]
        data_1["Z-ECEF"] = data_1["Z-ECEF"] - data_1["Z-ECEF"].iloc[0]

        data_2 = pd.read_csv(self.input_path_2, sep=' ', header=None,
                             names=['time', 'x', 'y', 'z'])
                             .sort_values(by=['time'])
        data_2.x = data_2.x - data_2.x.iloc[0]
        data_2.y = data_2.y - data_2.y.iloc[0]
        data_2.z = data_2.z - data_2.z.iloc[0]

        if data_1.shape[0] < data_2.shape[0]:
            print("First is smaller <- To keep")
            self.syncSize(data_1, data_2)
        elif data_1.shape[0] > data_2.shape[0]:
            print("Second is smaller <- To keep")
            self.syncSize(data_2, data_1)
        else:
            quit("They are already equal")

    def syncSize(self, smaller, bigger):
        pd.set_option('display.float_format', lambda x: '%0.6f' % x)
        output = pd.DataFrame()
        for index, row in smaller.iterrows():
            output = output.append(bigger.iloc[(bigger['UTCTime']
                        - row.time).abs().argsort()[:1]])

        tfile_1 = open('GT.txt', 'a')
        output['UTCTime'] = output['UTCTime'].astype(float)
        tfile_1.write(output.to_string(header=False, index=False))
        tfile_1.close()

        tfile_2 = open('01.txt', 'a')
        smaller.time = smaller.time.astype(float)
        tfile_2.write(smaller.to_string(header=False, index=False))
        tfile_2.close()

if __name__ == '__main__':
    parse = argparse.ArgumentParser()
    parse.add_argument("--input-path-1", required=True, help="Path
                        for the first file")
```

```
    parse.add_argument("--input-path-2", required=True, help="Path
                        for the second file")

    args = parse.parse_args()

    compare_and_reduce = CompareAndReduce(
        input_path_1=args.input_path_1,
        input_path_2=args.input_path_2,
    )

    compare_and_reduce.main()
```

6.6 mean_absolute_error.py

```
#!/usr/bin/python

import argparse
import math
import pandas as pd
import pathlib
import matplotlib.pyplot as plt


class MeanAbsoluteError:
    def __init__(self, input_path_gt, input_path_imp):
        self.input_path_gt = pathlib.Path(input_path_gt)
        self.input_path_imp = pathlib.Path(input_path_imp)

    def main(self):
        data_gt = pd.read_csv(self.input_path_gt, sep=' ', header=None,
            names=['time', 'x', 'y', 'z'])
        data_imp = pd.read_csv(self.input_path_imp, sep=' ', header=None,
            names=['time', 'x', 'y', 'z']).sort_values(by=['time'])

        result = pd.DataFrame()
        for i, row in data_gt.iterrows():
            time = data_imp.iloc[i].time - data_imp.iloc[0].time
            distX = abs(data_gt.iloc[i].x - data_imp.iloc[i].x)
```

```
            distY = abs(data_gt.iloc[i].y - data_imp.iloc[i].y)
            distZ = abs(data_gt.iloc[i].z - data_imp.iloc[i].z)
            distXYZ = self.calcule_dist(data_gt, data_imp, i)
            result = result.append({"time": time, "distX": distX,
                "distY": distY, "distZ": distZ, "distXYZ":
                distXYZ}, ignore_index=True)

        resume = pd.DataFrame()
        result = result.sort_values(by=['time'])
        meanX = result['distX'].sum() / result.shape[0]
        meanY = result['distY'].sum() / result.shape[0]
        meanZ = result['distZ'].sum() / result.shape[0]
        meanXYZ = result['distXYZ'].sum() / result.shape[0]
        resume = resume.append({"meanX": meanX, "meanY": meanY,
                "meanZ": meanZ, "meanXYZ": meanXYZ}, ignore_index=True)

        result = result[["time", "distX", "distY", "distZ", "distXYZ"]]
        tfile_1 = open('./result_mae.txt', 'a')
        result.time = result.time.astype(float)
        tfile_1.write(result.to_string(header=False, index=False))
        tfile_1.close()

        resume = resume[["meanX", "meanY", "meanZ", "meanXYZ"]]
        tfile_2 = open('./resume_mae.txt', 'a')
        tfile_2.write(resume.to_string(header=False, index=False))
        tfile_2.close()

        plt.plot(result['time'], result['distX'], label='Erro Ox')
        plt.plot(result['time'], result['distY'], label='Erro Oy')
        plt.plot(result['time'], result['distZ'], label='Erro Oz')
        plt.plot(result['time'], result['distXYZ'],
                label='Erro ponto a ponto')
        plt.xlabel("Tempo de sessão (s)")
        plt.ylabel("Erro (m)")
        plt.title('Erro médio absoluto')
        plt.legend()
        plt.savefig('Evolution_Error_mae.png')

    def calcule_dist(self, data_gt, data_imp, i):
        return abs(math.sqrt(
```

```
            math.pow((data_imp.iloc[i].x - data_gt.iloc[i].x), 2)
                    + math.pow((data_imp.iloc[i].y -
                    data_gt.iloc[i].y), 2) + math.pow(
                (data_imp.iloc[i].z - data_gt.iloc[i].z), 2)))


if __name__ == '__main__':
    parse = argparse.ArgumentParser()
    parse.add_argument("--input-path-gt", required=True, help="Path
            for the ground truth file")
    parse.add_argument("--input-path-imp", required=True, help="Path
            for the implementation output file")

    args = parse.parse_args()

    mean_absolute_error = MeanAbsoluteError(
        input_path_gt=args.input_path_gt,
        input_path_imp=args.input_path_imp,
    )

    mean_absolute_error.main()
```

6.7 mean_squared_error.py

```
#!/usr/bin/python

import argparse
import math
import pandas as pd
import pathlib
import matplotlib.pyplot as plt


class MeanSquaredError:
    def __init__(self, input_path_gt, input_path_imp):
        self.input_path_gt = pathlib.Path(input_path_gt)
        self.input_path_imp = pathlib.Path(input_path_imp)
```

```
    def main(self):
        data_gt = pd.read_csv(self.input_path_gt, sep=' ',
                header=None, names=['time', 'x', 'y', 'z'])
        data_imp = pd.read_csv(self.input_path_imp, sep=' ',
                header=None, names=['time', 'x', 'y', 'z'])
                .sort_values(by=['time'])

        result = pd.DataFrame()
        for i, row in data_gt.iterrows():
            time = data_imp.iloc[i].time - data_imp.iloc[0].time
            distX = pow((data_gt.iloc[i].x - data_imp.iloc[i].x), 2)
            distY = pow((data_gt.iloc[i].y - data_imp.iloc[i].y), 2)
            distZ = pow((data_gt.iloc[i].z - data_imp.iloc[i].z), 2)
            distXYZ = pow(self.calcule_dist(data_gt, data_imp, i), 2)
            result = result.append({"time": time, "distX": distX,
                    "distY": distY, "distZ": distZ, "distXYZ":
                    distXYZ}, ignore_index=True)

        resume = pd.DataFrame()
        meanX = result['distX'].sum() / result.shape[0]
        meanY = result['distY'].sum() / result.shape[0]
        meanZ = result['distZ'].sum() / result.shape[0]
        meanXYZ = result['distXYZ'].sum() / result.shape[0]
        resume = resume.append({"meanX": meanX, "meanY": meanY,
                "meanZ": meanZ, "meanXYZ": meanXYZ}, ignore_index=True)

        result = result[["time", "distX", "distY", "distZ", "distXYZ"]]
        tfile_1 = open('./result_mse.txt', 'a')
        result.time = result.time.astype(float)
        tfile_1.write(result.to_string(header=False, index=False))
        tfile_1.close()

        resume = resume[["meanX", "meanY", "meanZ", "meanXYZ"]]
        tfile_2 = open('./resume_mse.txt', 'a')
        tfile_2.write(resume.to_string(header=False, index=False))
        tfile_2.close()

        plt.plot(result['time'], result['distX'], label='Erro Ox')
        plt.plot(result['time'], result['distY'], label='Erro Oy')
        plt.plot(result['time'], result['distZ'], label='Erro Oz')
```

```
        plt.plot(result['time'], result['distXYZ'], label='Erro
                ponto a ponto')
        plt.xlabel("Tempo de sessão (s)")
        plt.ylabel("Erro (m^2)")
        plt.title('Erro médio quadrático')
        plt.legend()
        plt.savefig('Evolution_Error_mse.png')

    def calcule_dist(self, data_gt, data_imp, i):
        return abs(math.sqrt(
            math.pow((data_imp.iloc[i].x - data_gt.iloc[i].x), 2) +
                    math.pow((data_imp.iloc[i].y - data_gt.iloc[i].y),
                    2) + math.pow((data_imp.iloc[i].z -
                    data_gt.iloc[i].z), 2)))


if __name__ == '__main__':
    parse = argparse.ArgumentParser()
    parse.add_argument("--input-path-gt", required=True, help="Path
            for the ground truth file")
    parse.add_argument("--input-path-imp", required=True, help="Path
            for the implementation output file")

    args = parse.parse_args()

    mean_squared_error = MeanSquaredError(
        input_path_gt=args.input_path_gt,
        input_path_imp=args.input_path_imp,
    )

    mean_squared_error.main()
```

Referências

[1] R. Murray and S. Sastry, *A mathematical introduction to robotic manipulation*. 1994. [Citado nas páginas xvii e xviii]

[2] S. Liu, M. M. Atia, T. Karamat, S. Givigi, and A. Noureldin, "A dual-rate multi-filter algorithm for lidar-aided indoor navigation systems," pp. 1014–1019, 2014. [Citado na página 3]

[3] N. Hazards, "Terrestrial lidar to map flooding from hurricane isaac," Aug. 2012. [Citado nas páginas ix e 6]

[4] J. Zhang and S. Singh, "Low-drift and real-time lidar odometry and mapping," *Autonomous Robots*, vol. 41, pp. 401–416, 02 2017. [Citado nas páginas ix, x, xi, xii, xiii, 6, 7, 8, 11, 12, 13, 14, 19, 20, 21, 22, 27, 29, 30, 31, 33, 34, 35, 36, 37, 38, 39, 40, 41, 43, 49, 72, 73, 78, 79, 80, 84, 85, 86, 87, 91, 95 e 96]

[5] Y. Li and E. B. Olson, "Structure tensors for general purpose lidar feature extraction," pp. 1869–1874, 2011. [Citado na página 7]

[6] Z. J. Yew and G. H. Lee, "3dfeat-net: Weakly supervised local 3d features for point cloud registration," in *Computer Vision – ECCV 2018*, pp. 630–646, Springer International Publishing, 2018. [Citado nas páginas ix e 8]

[7] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. 2004. [Citado na página 10]

[8] M. O. M. de Berg, M. van Kreveld and O. Schwarzkopf, *Computation Geometry: Algorithms and Applications*. Springer, 2008. [Citado na página 11]

[9] A. G. Kashani, M. J. Olsen, C. E. Parrish, and N. Wilson, "A review of lidar radiometric processing: From ad hoc intensity correction to rigorous radiometric calibration," *Sensors*, vol. 15, no. 11, pp. 28099–28128, 2015. [Citado na página 15]

[10] H. Wang, C. Wang, and L. Xie, "Intensity scan context: Coding intensity and geometry relations for loop closure detection," pp. 2095–2101, 2020. [Citado nas páginas ix, xi, xii, 15, 18, 36, 37, 38, 49, 72, 74, 77, 78, 79, 81, 84, 86, 88, 90, 92, 94 e 95]

[11] G. Kim and A. Kim, "Scan context: Egocentric spatial descriptor for place recognition within 3d point cloud map," pp. 4802–4809, 2018. [Citado na página 15]

[12] G. Kim, B. Park, and A. Kim, "1-day learning, 1-year localization: Long-term lidar localization using scan context image," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1948–1955, 2019. [Citado nas páginas 15 e 17]

[13] J. Guo, P. V. K. Borges, C. Park, and A. Gawel, "Local descriptor for robust place recognition using lidar intensity," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1470–1477, 2019. [Citado na página 18]

[14] P. Besl and N. D. McKay, "A method for registration of 3-d shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 2, pp. 239–256, 1992. [Citado na página 18]

[15] H. Wang, C. Wang, C.-L. Chen, and L. Xie, "F-loam : Fast lidar odometry and mapping," pp. 4390–4396, 2021. [Citado nas páginas ix, xi, xii, xiii, 18, 19, 21, 22, 36, 37, 38, 49, 72, 75, 78, 79, 82, 84, 85, 86, 89, 90, 93, 94 e 95]

[16] R. B. Rusu and S. Cousins, "3d is here: Point cloud library (pcl)," pp. 1–4, 2011. [Citado na página 21]

[17] T. Shan and B. Englot, "Lego-loam: Lightweight and ground-optimized lidar odometry and mapping on variable terrain," pp. 4758–4765, 2018. [Citado nas páginas ix, x, xi, xii, xiii, 21, 27, 28, 29, 30, 31, 32, 33, 34, 35, 49, 72, 76, 77, 78, 79, 83, 84, 85, 86, 87, 91, 95 e 96]

[18] J. Zhang and S. Singh, "Visual-lidar odometry and mapping: low-drift, robust, and fast," pp. 2174–2181, 2015. [Citado nas páginas ix, xiii, 22, 23, 24, 26, 27, 36, 37, 38 e 69]

[19] S. S. . N. Michael, *The 13th International Symposium onExperimental Robotics*. Springer, 2013. [Citado na página 22]

[20] I. Bogoslavskyi and C. Stachniss, "Fast range image-based segmentation of sparse 3d laser scans for online operation," pp. 163–169, 2016. [Citado na página 28]

[21] M. Himmelsbach, F. v. Hundelshausen, and H.-J. Wuensche, "Fast segmentation of 3d point clouds for ground vehicles," pp. 560–565, 2010. [Citado na página 28]

[22] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012. [Citado nas páginas xiii, 36, 37 e 38]

[23] R. Caballero, J. Parra, M. Trujillo, F. J. Pérez-Grau, A. Viguria, and A. Ollero, "Aerial robotic solution for detailed inspection of viaducts," *Applied Sciences*, vol. 11, no. 18, 2021. [Citado nas páginas x, 38, 39 e 40]

[24] S. Jung, D. Choi, S. Song, and H. Myung, "Bridge inspection using unmanned aerial vehicle based on hg-slam: Hierarchical graph-based slam," *Remote Sensing*, vol. 12, no. 18, 2020. [Citado nas páginas x, xiii, 40, 41, 42 e 43]

[25] R. Goebel, "Spatial transformation matrices," 2020. [Citado nas páginas x, 46, 47 e 57]

[26] T. Rezende, "Rmse ou mae? como avaliar meu modelo de machine learning?," Nov. 2018. [Citado nas páginas x, 47 e 48]

[27] "Stork i - unmanned aerial vehicle (uav)." [Citado nas páginas x, 57 e 58]

[28] "Vlp-16 user manual." Feb. 2019. [Citado nas páginas x, xi, 58, 59 e 60]

[29] "Neo-m8 series." [Citado nas páginas xi e 61]

[30] A. Zhu, "The mvsec dataset." [Citado nas páginas xii, 85, 86 e 87]