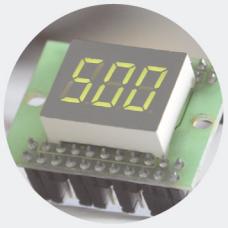
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES






# Perceção cooperativa para planeamento de trajetórias de um veículo terrestre

**ANDRÉ FILIPE COSTA PACHECO REIS**
novembro de 2019

POLITÉCNICO
DO PORTO

**ISEP**

Instituto Superior de Engenharia do Porto

# Perceção cooperativa para planeamento de trajetórias de um veículo terrestre

Tese de Mestrado

Para obter o grau de mestre no
Instituto Superior de Engenharia do Porto,

André Filipe Costa Pacheco Reis

Porto, Portugal.

Supervisor:

André Miguel Pinheiro Dias

Author email: 1131180@isep.ipp.pt

*À minha avó, que nos deixou saudade...*

# Agradecimentos

Quero agradecer a todos os meus amigos, colegas e familiares que me deram o apoio necessário e a motivação para a conclusão da dissertação. Em particular, agradeço aos meus pais por estarem sempre disponíveis para mim, aos meus colegas (amigos), pelo tempo que passamos juntos neste mestrado, pelos desafios que superamos em equipa, com esta turma fenomenal. Agradeço ainda aos meus colegas do trabalho por toda a ajuda que me ofereceram, e aos meus amigos que de tudo fizeram para que eu superasse este desafio difícil e importante na minha vida.

# Resumo

Esta dissertação resulta de um trabalho de investigação relativamente ao estudo de um cenário de cooperação multi-robótica. O trabalho tem um elevado foco na perceção robótica, onde através de um sensor LIDAR e outros sensores auxiliares, se tenta encontrar um caminho ótimo para um UGV através de um planeador baseado em RRT. O outro robô, o UAV, é responsável pela decisão da zona a mapear, onde se efetua uma decisão da área máxima a mapear por forma a reduzir a complexidade computacional. Com os dados recolhidos pelo UAV, efetua-se uma transformação de referencial para representar as coordenadas adquiridas no referencial global, onde se procede a uma filtragem de pontos, para eliminar ruído na amostra, seguindo-se uma segmentação à nuvem de pontos filtrada. Tenta-se encaixar o número máximo possível de planos nesta amostra por forma a percecionar o que se está a mapear, classificando então o que é espaço ocupado, que corresponde a zonas com elevado declive, altura ou fora do alcance do UGV, do espaço não ocupado. Com esta informação, alimenta-se o planeador de trajetória e produz-se um caminho ótimo até ao alvo definido inicialmente ou até ao local mais próximo que é conhecido no mapa, se o alvo não estiver incluído na informação percecionada. Todo o trabalho desenvolvido foi validado recorrendo ao simulador MORSE, emulando cenários e mensagens de sensores compatíveis com o *middleware* ROS, para uma simulação em tempo real.

# Abstract

This thesis works with LIDAR technology, focusing on processing this sensor data in a context of multirobotic cooperation, between two robots - One UAV and one UGV. The UAV is responsible for mapping the area where the UGV plans to cross and the goal is to apply path planning techniques based on an RRT planner in the data processed. With the data collected by the LIDAR sensor attached to the UAV, it's applied two transformations in order to visualize the collected points in the world frame. Then the data is filtered to remove noise points and to discard points outside the area considered for UGV motion. The next step is the segmentation that tries to fit as many planes as we can on the data with the objective to filter ground points and discard noisy points, and a set of points with high covariance in Z axis. With this processing, it should be possible to distinguish between occupied and free cells, and this information is fed in the path planner to find the optimal path to the mission or, if the goal is not within the mapped area, then the robot should move towards the end of that area, and wait for more environment information. The development is validated with MORSE simulator, performing tests within ROS middleware, with scenarios and sensor messages emulating real sensors.

# Conteúdo

# Lista de Figuras

# Lista de Tabelas

# Acrónimos

| | |
|---|---|
| **ASV** | Autonomous Surface Vehicle |
| **AUV** | Autonomous Underwater Vehicle |
| **BSC** | Binary Shape Context |
| **CGH** | Cumulative Gaussian Histogram |
| **CNN** | Convolutional Neural Network |
| **FALKO** | Fast Adaptive Laser Keypoint Orientation-invariant |
| **DARPA** | Defense Advanced Research Projects Agency |
| **GNSS** | Global Navigation Satellite System |
| **GPF** | Ground Plane Fitting |
| **GP-INSAC** | Gaussian Process Incremental Sample Consensus |
| **GPU** | Graphics Processing Unit |
| **HSV** | Hue Saturation Value |
| **IMU** | Inertial Measurement Unit |
| **IEC** | International Electrotechnical Commission |
| **ICP** | Iterative Closest Point |
| **LSA** | Laboratório de Sistemas Autónomos |
| **LiDAR** | Light Detection And Ranging |
| **LED** | Light-emitting Diode |
| **KF** | Kalman Filter |
| **OGM** | Occupancy Grid Map |
| **OC** | Orthogonal Corner |
| **PF** | Particle Filter |
| **RANSAC** | Random Sample Consensus |
| **RRT** | Rapidly-exploring Random Tree |
| **RGB** | Red Green Blue |

| | |
|---|---|
| **ROS** | Robot Operating System |
| **SLR** | Scan Line Run |
| **SLAM** | Simultaneous Localization and Mapping |
| **SVM** | Support Vector Machine |
| **TOF** | Time of flight |
| **UAV** | Unmanned Aerial Vehicle |
| **UGV** | Unmanned Ground Vehicle |
| **WLAN** | Wireless Local Area Network |

# 1 Introdução

Um sistema autónomo é um robô, terrestre, aquático ou aéreo que consegue operar sem que haja necessidade de controlo humano. Na atualidade, existe uma enorme procura por este tipo de sistemas, uma vez que robôs autónomos substituem as pessoas na realização de algumas tarefas diárias ou mesmo outras atividades, em que as mesmas possam colocar em causa a vida humana. Entre os cenários de aplicação deste tipo de robôs, existe o setor militar, na medida em que existem sistemas de reconhecimento (aéreos) de identificação de tropas inimigas, ou até veículos móveis (terrestres) que são equipados com armamento carregado [11]. Na agricultura, começam a aparecer alguns robôs que, recorrendo a técnicas de visão em conjunto com outros sensores, plantam sementes, ou tratam da terra removendo ervas daninhas [14], replicando o processo de um agricultor.

Um outro cenário de aplicação dos sistemas autónomos insere-se num contexto de operações de busca e salvamento. Nestas situações há que ter em consideração que quando ocorre algum desastre, seja devido a causas naturais ou por negligência humana, em alguns destes cenários é vantajoso ter um robô que consiga efetuar operações de salvamento, reduzindo os riscos associados aos operadores que conduzem este tipo de operações. Como requisitos neste segmento de robôs, são incluídos vários sensores, entre os quais sensores que detetam gases inflamáveis e fontes de calor. A maioria deste tipo de robôs são terrestres, como é o caso do projeto RAPOSA [26]. Como características principais neste robô destacam-se a sua dimensão, a sua mobilidade e o número de sensores que transporta. A sua construção foi pro-

jetada de modo a que, em caso de derrocadas de edifícios, fosse possível aceder ao interior desses edifícios através dos circuitos de saneamento. Com os sensores que este robô transporta, é possível detetar fugas de gás e encontrar pessoas no meio dos destroços, através de fontes de calor detetadas por uma câmara térmica. Para além da perceção de pessoas, onde pela imagem térmica a ser vista em tempo real na plataforma de controlo consegue-se identificar se a imagem recolhida se assemelha com a geometria de uma pessoa, também se podem detetar zonas onde estejam a deflagrar pequenos incêndios.

Em Portugal existe algum investimento neste tipo de robôs, em particular no segmento aquático. Alguns exemplos de robôs construídos têm como objetivo efetuar missões autónomas à superfície do mar / oceano, como é o caso do ROAZ [63]. Para além deste robô, outros foram construídos para efetuarem missões subaquáticas como o EVA [63] e também o conjunto multirobótico do projeto a nível europeu UNEXMIN [51]. No caso deste conjunto de robôs o seu objetivo é investigar de um modo autónomo minas abandonadas que se encontrem inundadas.

## 1.1 Âmbito da dissertação

Nesta dissertação pretende-se apresentar uma investigação focada na perceção robótica em ambiente não controlado e desconhecido. Melhorar a perceção sensorial de um robô autónomo terrestre é um problema bastante comum de *Simultaneous Localization And Mapping* (SLAM) e que é fundamental para garantir o melhor desempenho no planeamento de caminho e no planeamento de movimento, que se efetuam quando existe uma perceção do ambiente à volta do robô.

Navegação multirobótica é um tópico emergente na área da robótica móvel. Aperfeiçoar um sistema autónomo não é uma tarefa fácil e dependendo do contexto, usar mais do um robô para efetivar uma missão, pode ser mais vantajoso do que um só robô isolado. Cooperação entre robôs pode acontecer em diferentes meios, por exemplo entre um *Unmanned Aerial Vehicle* (UAV) e um *Unmanned Ground Vehicle* (UGV), ou até entre um *Autonomous Surface Vehicle* (ASV) e um veículo submarino: *Autonomous Underwater Vehicle (AUV).*

A base de um sistema autónomo assenta na sua capacidade de perceção ambiental, onde na maior parte dos robôs que foram construídos, efetuam esta acção através de câmaras, radares ou *lasers* que recorrem à tecnologia *Light Detection And Ranging* (LiDAR). Nuvens de pontos 3D obtidas com LIDAR são uma fonte rica em informação, conseguindo representar com elevado detalhe o que se encontra ao seu alcance, limitando-se por isso ao seu ângulo de visão vertical e horizontal no que

diz respeito aos pontos que são recolhidos, variando entre modelos. Estes sensores são muito utilizados, uma vez que as suas medições apresentam um erro de projeção muito reduzido. Por outro lado, a manipulação de nuvens de pontos em tempo real pode tornar-se bastante dispendiosa, tendo em conta que o processamento pode ser dividido por unidades de processamento gráfico, *Graphics Processing Unit* (GPU) , há que ter sempre em consideração o espaço que este *hardware* ocupa num robô, logo nem sempre se recorre a estes sistemas de processamento gráfico.

São muitos os casos implementados de cooperação multirobótica, como a exploração de minas subaquáticas, ou um cenário de busca e salvamento, que na sua generalidade é caracterizado como um terreno difícil de navegar. Um mapa global entre robôs que colaboram entre si, é uma característica fundamental de um sistema cooperativo, de modo que exista conhecimento da localização de cada robô de um dado conjunto. Isto é, para uma navegação conjunta, a localização de cada robô em relação a um referencial comum é um requisito fundamental. Com um sistema de *Global Navigation Satellite System*(GNSS) é possível obter as coordenadas de cada robô no referencial global. Outra solução baseia-se em técnicas de processamento de imagem e/ou de uma nuvem de pontos. Através de *features* em imagens do conjunto robótico, apesar de corresponderem a perspectivas diferentes, se estas alcançarem zonas iguais abre-se a possibilidade de detetar pontos em comum entre as imagens, e com essa informação é exequível uma localização de cada um dos robôs.

O âmbito desta dissertação incide na implementação de um mecanismo de cooperação robótica entre um UAV e um UGV. O veículo aéreo tem como objetivo auxiliar na navegação do robô terrestre, mapeando a área em redor do robô com um LiDAR Velodyne VLP-16 e partilhar essa informação com o outro veículo. A localização de ambos os robôs tem que ser conseguida em tempo real, e partindo de um cenário que seja propício à aquisição de posicionamento global: GPS/GNSS. Ambos os robôs serão equipados com sensores inerciais do tipo *Inertial measurement unit* (IMU) de modo a que consiga extrair a sua odometria. O robô aéreo tem como principal função auxiliar a navegação do robô móvel, através da disponibilização de um mapa de elevações de modo a que possa planear a sua trajetória.

Para o desenvolvimento será utilizado o *middleware Robot Operating System* (ROS) que corresponde ao *software* utlizado nos demais projetos realizados durante o mestrado em Engenharia Eletrotécnica e de computadores - Sistemas autónomos.

## 1.2 Objetivos

O objetivo desta dissertação é a validação de algoritmos de segmentação de nuvem de pontos, adquiridas através de um LIDAR, e com esses dados construir um mapa de elevações para alimentar o robô móvel. O UAV terá a responsabilidade do mapeamento, por forma a tirar partido do ponto de vista aéreo que ajuda a representar com maior detalhe o terreno. O robô aéreo deverá permanecer sempre próximo do robô móvel, de forma a que a representação efetuada apenas ocupe uma pequena área de cada vez que for efetuado um pedido de um mapa elevação. Efetua-se então o seguinte levanto de requisitos:

- Identificação das técnicas de partilha de informação (mapas de elevação) num cenário multi-robôs;
- Identificação de técnicas, baseadas em LiDAR em UAV para a construção de um mapa global de elevação;
- Exploração de técnicas de planeamento de trajetórias baseado em *constraints* (mapa de elevação);
- Desenvolvimento de um modelo de simulação que permita a validação de todos os algoritmos a serem desenvolvidos no âmbito da dissertação;
- Desenvolvimento de um algoritmo de estimação do mapa de elevações baseado em LiDAR;
- Desenvolvimento de um algoritmo de estimação da área a considerar para processamento, na vizinhança do robô móvel;
- Desenvolvimento de um conjunto de *waypoints* para alimentar o robô aéreo de forma a representar com elevado detalhe o terreno à volta do UGV;
- Validação de todos os conceitos aplicados em tempo real num simulador 3D;

## 1.3 Estrutura da dissertação

Após este capítulo introdutório é apresentado um estado da arte sobre técnicas de segmentação de nuvens pontos, onde se irá refletir sobre os diversos métodos propostos para a resolução deste problema. Ainda nessa secção são abordados os sensores que tipicamente são utilizados em navegação robótica, terrestre e aérea, bem como exemplos de sistemas que foram publicados em artigos científicos, em que nesses

sistemas são demonstrados diferentes tipos de navegação colaborativa entre robôs. O estado da arte termina com a apresentação de alguns métodos de mapeamento, localização e planeamento de caminhos e de movimentos. Segue-se então o capítulo dos fundamentos teóricos, que apresenta todo o estudo que serve como base à fundamentação de todos os métodos desenvolvidos no âmbito desta dissertação. De seguida, apresenta-se uma secção que explica todos os algoritmos implementados, bem como o raciocínio lógico da solução do problema proposto, começando com uma arquitectura de alto nível do sistema e evoluindo até pormenores mais específicos tais como algoritmos de segmentação e métodos de alocação dinâmica numa estrutura de dados de pontos. Por último, apresentam-se os resultados obtidos, comparando com o estado da arte e ainda uma conclusão que analisa se os objetivos foram alcançados e qual o trabalho futuro a desenvolver.

# 2

# Estado da arte

Este capítulo inicia com o estudo de trabalhos desenvolvidos na área da perceção cooperativa multi-robótica publicados e validados, onde se vai efetuar uma breve descrição desses sistemas, como os sensores utilizados e ainda os métodos/algoritmos implementados que se encontram no âmbito da dissertação.

Noutro sub-capítulo, sensores como LiDAR ou câmara RGB-D vão ser também apresentados, por constituírem os principais elementos de aquisição de informação percecional usado para navegação de sistemas autónomos. Para além destes tópicos, vão ser estudados alguns métodos de planeamento de caminho (*Path Planning*) e de movimentos (*Motion Planning*) aplicados em robôs terrestres.

A segmentação de informação sensorial, nomeadamente de LiDAR e de câmaras RGB-D, é outro item fundamental na perceção ambiental, pelo que foram estudadas diversas técnicas que se aplicam ao âmbito da dissertação. O capítulo termina com a apresentação de algumas *frameworks* de acesso livre (*open-source*) que possam ir ao encontro do contexto de navegação autónoma, seguido de uma reflexão de todos os métodos estudados.

## 2.1 Navegação colaborativa

A ideia de uma colaboração entre robôs tem como objetivo distribuir tarefas a cada robô de um dado conjunto, por forma a que se aproveite ao máximo as melhores capacidades de cada robô. A colaboração entre um robô terrestre e um robô aéreo

[50], no contexto deste *paper*, tem a finalidade de efetuar um mapeamento de um ponto de vista aéreo, onde é possível obter uma melhor perceção do terreno. Há que ter em consideração que um robô aéreo tem as suas limitações, sendo que não pode transportar cargas elevadas, por isso, para além do peso das baterias de um UAV autónomo, os sensores que são escolhidos para exercer este tipo de missão vão influenciar o tempo de voo. O sensor laser mais usado neste tipo de veículos é o velodyne VLP-16 [1], ou um sistema de visão, composto por uma ou mais câmaras *Red Green Blue*(RGB) (por exemplo, um par *stereo*), ou ainda câmaras *Red Green Blue and Depth* RGB-D, que representa a tecnologia implementada no kinect [2], que é o produto mais conhecido dotado desta tecnologia. No entanto este tipo de sensores apresenta limitações quando recolhe dados em *outdoor* [25], uma vez que o brilho subjacente a este tipo de cenários, não permite uma correta perceção, pelo que se perde bastante informação quando a exposição é elevada [25]. A escolha do LiDAR VLP-16 em específico deve-se à sua dimensão, ao seu peso de apenas 600 g, sendo por isso mais leve em relação a modelos de gamas semelhantes, e ainda o seu preço, que carece de uma revisão à data da leitura desta dissertação, que no entanto ronda os 4000 [3] à data da escrita. Por forma a concluir o levantamento das características deste sensor, destaca-se ainda a elevada resolução na representação de pontos até uma distância máxima de 100 m, que tendo em conta a gama em que se insere este sensor, é uma distância elevada.

Um dos exemplos publicados de cooperação robótica, entre um robô móvel, denominado StarETH, e um UAV com o nome Asctec Firefly Hexacopter são apresentados no seguinte *paper* citado [50]. A partilha de informação entre ambos, ocorre através de uma rede *Wireless Local Area Network* (WLAN), apenas acessível entre estes, pelo que trata-se de uma rede local encriptada. Os robôs estão equipados com *VI-Sensor* que consiste num par *stereo*, em que cada lente tem uma abertura de 120º. Para além das câmaras, o StarETH está equipado com um laser LiDAR Hokuyo. O robô aéreo é responsável pela criação do mapa de elevações, que corresponde ao ponto de partida para a localização do robô móvel, que recorrendo ao sensor que transporta, tenta obter correspondências no mapa que lhe foi disponibilizado. No que diz respeito à localização entre ambos, através do mapa construído em *backend*, são extraídas *keyframes* e *landmarks*, que se obtém com o método *visual-inertial odometry*. Este mapa de *landmarks* é denominado de *global localization map* e com base neste mapa construído, o StarETH vai varrer com o laser o terreno por forma a juntar esta informação, com a outra componente fornecida pelo UAV através do

[1] https://velodynelidar.com/vlp-16.html
[2] https://developer.microsoft.com/en-us/windows/kinect
[3] https://www.spar3d.com/news/lidar/velodyne-cuts-vlp-16-lidar-price-4k/

par *stereo*. Esta fusão permite localizar cada robô num referencial comum a ambos. Esta fusão de informação entre ambos vai ao encontro do tópico referido no capítulo introdutório, onde se abordou a importância de um mapa que fosse comum aos robôs da missão de colaboração.

Outro exemplo de cooperação, entre um UAV e um UGV, aborda um cenário semi-autónomo, no qual a navegação totalmente autónoma é alcançável, embora exista a possibilidade de uma comutação para uma condução tele-operada do UGV [1]. Neste cenário, ambos os veículos recorrem a processamento de imagem, para se localizarem mutuamente. O UAV disponibiliza imagens aéreas para efetuar o planeamento de caminhos do UGV que estará visível através da plataforma webGIS [4]. O veículo aéreo segue de um modo autónomo o UGV, pelo que tem que estar a aplicar em tempo real algoritmos de processamento de imagem para nunca o perder de vista. Outra funcionalidade deste conjunto robótico é a possibilidade de o UAV conseguir aterrar autonomamente no UGV, quando a missão termina, ou se precisar de carregar as suas baterias. Deste modo, as restrições do tempo de duração da missão podem ser expandidas, isto porque o tempo de voo de um UAV normalmente não é muito elevado e com a possibilidade do carregamento das baterias, resolve-se um problema que é comum quando se projetam UAV autónomos.

A localização entre robôs durante uma missão, adotando como exemplo os casos estudados referidos nos parágrafos anteriores, é um requisito fundamental, de modo a que se consiga navegar num contexto colaborativo. No segundo parágrafo, na publicação citada, [50], foi apresentada uma solução para o problema da localização de cada robô. Existem outros métodos, entre os quais, uma localização de um UAV, por parte de um outro robô, neste caso um UGV [37], representado na figura 2.1, que através de uma câmara deteta um conjunto de *Light-emitting Diode* (LED) montados no UAV, emitindo luz na direção do robô. A câmara deteta comprimentos de onda de luz infravermelha, que corresponde à gama de emissão dos LED, por isso através de um limiar aplicado à filtragem da luz detetada pela câmara, consegue-se detetar com precisão a fonte emissora, que está acoplada ao UAV. Desta forma, através de uma constante aquisição de dados desta câmara, em tempo real sabe-se a localização, se o UAV estiver dentro do campo de visão da câmara fixa do UGV.

Outro cenário de colaboração multirobótica [38], descreve um caso prático de uma missão de localização e neutralização de minas subaquáticas. Dois robôs, um *Autonomous Surface Vehicle* (ASV) e um *Autonomous Underwater Vehicle* (AUV), trabalham em conjunto na procura de minas. A deteção preliminar é efetuada pelo ASV, que através de um sonar, procura detetar algum objeto não identificável,

[4] http://www.webgis.com/

**Figura 2.1:** *UAV pousado numa plataforma do UGV [1]*

no fundo oceânico. Se estas condições forem atingidas, o ASV lança o AUV de modo a que este siga o objeto. A missão divide-se em várias fases: Começa com o ASV a lançar o AUV na procura de minas subaquáticas, que através de um sensor acústico, tenta localizar o local exato do objeto e rastreia-o. Depois do varrimento, o AUV volta ao local de origem e partilha os dados com o ASV, que durante a fase de recolha de informação permaneceu nas mesmas coordenadas. A partilha de informação ocorre através do *software* MOOS-IvP [5], sendo que esta pode ser vista em tempo real por um operador, que através de um meio de comunicação acústico, garante a transmissão de informação. A fase de neutralização da mina ocorre com uma explosão durante a colisão prevista de um AUV dispensável com a mina. Meios de desativação que substituam uma explosão controlada, como é o caso deste *paper* [38] publicado, estão ainda em fase de desenvolvimento.

## 2.2 Métodos de segmentação

*Simultaneous Localization and Mapping* (SLAM) é um problema recorrente na implementação de sistemas capazes de operar autonomamente. Através de algoritmos que aglomeram diversas componentes, entre as quais uma de mapeamento, outra de localização, e aplicando *kalman filter* (KF), ou *particle filters* (PF), atinge-se um estado em que, com o máximo rigor possível, consegue-se provar que este conceito de navegação autónoma pode ser alcançado.

Um dos primeiros exemplos de condução autónoma, veio do *Defense Advanced Research Projects Agency* (DARPA) [6], desenvolvido pela Stanford Racing Team,

[5] www.moos-ivp.org
[6] https://www.darpa.mil/

representado na figura 2.2,foi premiado com o primeiro lugar em 2006 completando a prova com o melhor tempo: 6h54m, tendo sido batizado de Stanley[7].


**Figura 2.2:** *Stanley [2]*

Algoritmos de SLAM aplicados com sucesso como por exemplo o PUT SLAM [48], foi implementado com base numa otimização de grafos e um mapa denso de *features*. A estimação da posição e da velocidade do robô é efetuada recorrendo a KF, que tem como função corrigir erros de dispersão de informação quando se atualiza o mapa.

Entre os algoritmos probabilísticos de segmentação, estão incluídos algoritmos de odometria visual [48], que consiste numa associação de *features* provenientes de um mapa de elevação denso. Apesar de neste *paper* o sensor que foi utilizado não ser um LiDAR, mas sim uma câmara RGB-D, os métodos de segmentação são aplicáveis em nuvens de pontos obtidas com o sensor LiDAR. Outros métodos que são utilizados para representar um ambiente desconhecido, são baseados numa representação de *voxels*, recorrendo a uma biblioteca que foi desenvolvida para facilitar a construção de um mapa - Octomap *library* [3]. Esta biblioteca é uma ferramenta de livre acesso (*open-source*), que constrói mapas 3D, recorrendo a uma estimação de ocupação de *octrees* através de informação sensorial. Uma *octrees* [16] é uma estrutura de dados hierárquica, cujos nós estão representados na figura 2.3. Os nós desta árvore são representados por *voxels*, que correspondem a uns cubos de tamanho regulável, que têm como finalidade agregar um conjunto de pontos no seu interior. A resolução de cada *voxel*, exemplificada na figura 2.4, é definida conforme o caso prático de navegação robótica. O tamanho de um *voxel* permite estimar com precisão se um robô pode atravessar uma determinada área [4]. Neste caso foi definido [4] que o tamanho de um *voxel* não podia ser superior à área de contacto de cada perna do robô, de modo que o planeamento de movimento de cada perna fosse efetuado com a máxima precisão alcançável. A vantagem de uma *framework* como a octomap [3] é que consegue efetuar uma representação do espaço de atuação de um robô em tempo

[7] https://cs.stanford.edu/group/roadrunner//old/index.html

real, sem apresentar elevados consumos de memória, disco e custo de processamento. Para além disso, para mapas que representem uma área elevada, requer um maior espaço em disco para guardar o mapa. No entanto os autores [3] comprovaram um método eficaz de compressão do mapa, por forma a reduzir o espaço que este ocupa. A compressão é efetuada através de uma análise das probabilidades (*log-odds*) de cada *voxel* atingir o limite inferior ($l_{min}$), ou o limite superior ($l_{max}$), que corresponde a um intervalo que denota uma estimação de ocupação de células. As células pertencentes a este intervalo são classificadas como estáveis, o que signica que estas podem convergir numa só, reduzindo assim o número de células totais. E deste modo, o acesso à informação guardada torna-se uma operação menos dispendiosa.

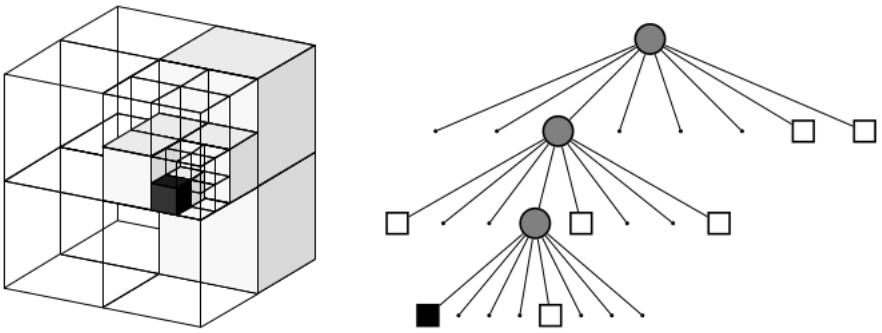


**Figura 2.3:** *Divisão hierárquica de uma octrees [3]*

Embora a biblioteca Octomap apresente informação bastante detalhada do ambiente que rodeia o robô, há que ter em consideração que pode não ser a melhor solução para planear os movimentos do robô. Outra solução que pode ser utilizada durante a navegação são os mapas de elevação [23]. Esta *framework* pretende servir de solução a uma navegação robótica à semelhança da Octomap, sendo que neste caso o mapa que se constrói não tem informação tão densa como na *framework* Octomap. Num mapa de elevação, a representação que é efetuada é numa perspetiva 2.5D, ao invés da representação 3D da Octomap. Isto significa que formas geométricas de objetos não são representadas na totalidade no mapa de elevação, sendo por isso a diferença visualmente mais evidente desta *framework* [23] em comparação com a octomap [3]. Um representação nesta perspetiva é equivalente a células 2D, no entanto contém informação sobre a elevação das células. Em alguns casos a perceção geométrica de objetos é importante para um posterior planeamento de trajetória e movimento, onde se quer evitar obstáculos. Há que ter em consideração que um mapa 3D é bastante mais denso quando comparado com um 2.5D. Esta situação tem um enorme peso quando se pretende projetar um sistema que funcione em tempo real.

Os mapas de elevação têm sido aplicados em robôs móveis [4, 22, 49, 50], de

modo a efetuar um mapeamento em tempo real eficiente, com acesso rápido a esse mapa [4]. Devido a limitações do campo de visão dos robôs móveis, surgiu uma ideia de cooperação multirobótica entre um robô móvel e um robô aéreo [50], ideia que motivou o desenvolvimento apresentado neste *paper*, que tem como objetivo fazer uso do campo de visão de um robô aéreo, por forma a que se consiga obter uma representação mais pormenorizada e mais precisa do ambiente de navegação. Através de um mapa com mais informação (octomap) [3], foi reduzido a um mapa de elevação [48] de modo a que em tempo real o acesso à informação do terreno pudesse ser efetuada o mais rapidamente possível, para alimentar algoritmos de planeamento de trajetórias a uma frequência mais elevada.

**Figura 2.4:** *Três resoluções diferentes de um octomap [3]*

Outro método de segmentação de uma nuvem de pontos, denomina-se *Random Sample Consensus* (RANSAC) [13, 48, 55]. Este método baseia-se na rejeição de *outliers*, por forma a tentar recursivamente encontrar um plano formado por um conjunto de pontos - *inliers*. Os *outliers* correspondem aos pontos de rejeição do modelo que se está a testar e os *inliers* são os pontos que pertencem ao modelo do teste. Douillard [55] publicou vários métodos validados para diferentes tipos de *point clouds*, as que contêm maior densidade de pontos e as mais dispersas. Entre os métodos de segmentação de nuvens de pontos mais densas, destaca-se o *Cluster-All* e o *Base-Of*. O primeiro algoritmo começa por efetuar uma classificação de pontos que correspondam a solo, pontos estes que estão inseridos em *voxels*. Os restantes pontos são agrupados noutro *cluster*[8]. Em relação ao método *Base-of*, os *voxels* são processados tendo em conta a variância vertical, com a finalidade de determinar se correspondem a *voxels* lisos. *Voxels* não similares, isto é com classificação diferente,

[8]Um Cluster corresponde a um conjunto de pontos agrupado num conjunto. Quando se refere este termo, está-se a considerar pontos que correspondam a um determinado objeto.

apenas são agrupados caso o *voxel* não liso se encontrar em baixo do outro *voxel* liso. Um exemplo deste agrupamento, abordando o exemplo prático referido no *paper* [55], em que o teto de um carro é segmentado com *voxels* lisos e para este método agrupar os *voxels* que reconstroiem o carro na *point cloud*, o método tem que agrupar esses *voxels* não lisos, que fazem parte da estrutura do veículo, que se encontram em baixo do *voxel* liso.

Em *point clouds* mais dispersas, os métodos propostos são o *Gaussian Process Incremental Sample Consensus* (GP-INSAC) e o *Mesh based*. Em relação ao GP-INSAC, foi desenhado para atuar de um modo probabilístico e incremental, estimando o que corresponda a solo. Este método corresponde a uma variação do método *Gaussian Process* (GP) [19], que tem um conjunto interessante de características - é probabilístico e contínuo. Porém, objetos são rejeitados pelo modelo testado, por existirem poucos *outliers*. Esta lacuna é corrigida pelo GP-INSAC, o que torna este método mais completo para uma segmentação baseada em GP. A segmentação *Mesh based* é dividida em três etapas: construção do terreno através de uma imagem de profundidade, extração de pontos correspondentes a solo e *clustering*[9] de pontos que não foram classificados de solo. A sua construção cria uma conexão entre pontos, de acordo com o método apresentado em [29]. Com este *output*, procede-se à extração de *ground points*. Todos os pontos *non-ground* são alimentados ao algoritmo *Cluster-All*, anteriormente explicado, para proceder ao *clustering* desses pontos.

Classificar um conjunto de pontos é outro método de segmentação, onde através de uma estimação da cor média de um plano, seja possível a distinção de tipos diferentes de superfícies. Uma das formas de classificação ativa de superfícies, é através de uma rede neuronal *Support Vector Machine* (SVM). Em [4], esta rede foi treinada por forma a classificar em tempo real o tipo de superfície de um conjunto de pontos semelhantes, representando uma qualquer superfície. Para o treino desta rede, que analisa múltiplos pixeis representados no esquema de cores *Hue Saturation Value* (HSV), provenientes de imagens da câmara RGB-D sem processamento (*raw image*). A classificação de cada pixel é efetuada de modo manual, sendo que após um treino de 1600 pixeis classificados com as cores corretas, a rede consegue classificar com sucesso em tempo real uma superfície conhecida, tais como asfalto, areia, relva, chão e madeira, como se encontra representada na figura 2.5.

Outro método de segmentação, que tem como objetivo aglomerar planos que correspondam a solo [5], recorre a uma *Convolutional Neural Network* (CNN). Esta rede é treinada manualmente, sendo definido num *dataset*, o ponto inicial de procura

[9]Corresponde ao método de agregação de pontos num *cluster*

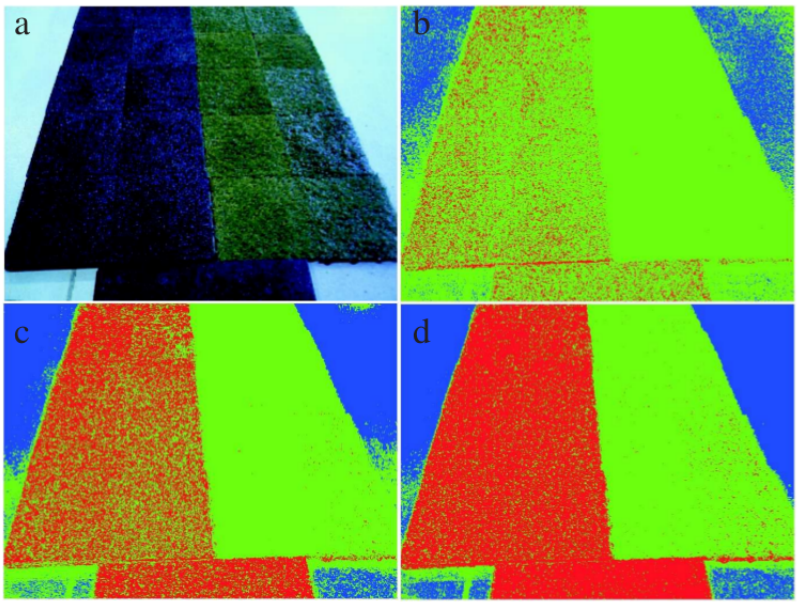



**Figura 2.5:** *Representação de três superfícies - chão (azul), relva (verde) e asfalto (vermelho). Na figura a) o treino foi efetuado com 200 amostras, na b) foram 400, c) e d) 1600 [4]*

e ainda a condição final de paragem - *breakpoint*. Esta condição é ativada quando algum ponto não incluído neste conjunto de pontos, for considerado muito diferente do anterior. Através de um limiar que analisa os últimos pontos do conjunto, caso o próximo ponto apresente uma altura superior ao limiar definido, em relação ao último ponto ou à semente inicial, então esse ponto será classificado de *breakpoint*, como se demonstra na figura 2.6.

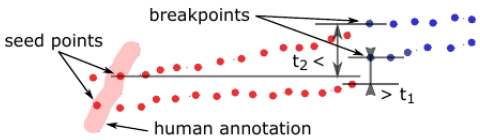



**Figura 2.6:** *Preenchimento da anotação humana até atingir o breakpoint [5]*

No que diz respeito à segmentação de pontos que correspondem ao solo, outra técnica com bons resultados desenvolvida pelo Zermas et al [33], denomina-se *Ground Plane Fitting* (GPF). Este algoritmo itera com várias sementes, com alturas pequenas, que serão posteriormente usadas para estimar o plano inicial. Cada ponto do plano estimado é comparado com o segmento da *point cloud*, onde as iterações começaram, o que resulta numa distância de um dado ponto, em relação à sua projeção no plano estimado. Avaliando esta distância em função de um limiar

definido a priori é efetuada a classificação de pontos que pertençam ao plano de extração de pontos que pertençam ao solo. Efetuada a extração desses pontos, em relação aos restantes, serão testados numa condição de aglomeração em *clusters* - corresponde a um conjunto de pontos que pertençam a um mesmo objeto. Para este passo foi apresentado o *Scan Line Run* (SLR) que analisa os pontos produzidos pelo mesmo anel do LiDAR que é designada de *scan-line*. Todos os pontos ao alcance de cada *scan-line* são organizados de forma contínua em vetores. O algoritmo SLR analisa os pontos da *point cloud* 3D como se de pixeis 2D se tratassem, com a ressalva de que no caso do LiDAR, entre camadas diferentes, também o número de pontos é diferente. Durante a primeira *scan-line*, a cada *run*, que corresponde a cada ponto da camada, é-lhe atribuída uma nova *label*, que é partilhada entre os restantes pontos. Uma *label* é propagada para uma nova *run* se a distância entre esse ponto e o seu equivalente na *scan-line* anterior, atingir um limiar definido. Repetindo este processo para toda a *point cloud*, pontos com alguma proximidade são agrupados no mesmo conjunto.

## 2.3 Mapeamento e localização

O conhecimento das coordenadas de cada robô aquando de uma missão, corresponde a um requisito fundamental para o sucesso da mesma. Para construir um mapa colaborativo é necessário saber localizar cada robô num único referencial global. Satisfazendo estas condições, a navegação autónoma é um cenário possível. Localizar um robô em tempo real, considerando um ambiente com condições não favoráveis ao uso de GPS [46], é um problema que pode ser resolvido através de uma odometria visual e inercial [50]. Se o mapa do conjunto robótico for representado no mesmo referencial, então existem condições para um ambiente de cooperação. Através de *features* em cada robô, ou descritores, procede-se a uma identificação de cada robô e assim consegue-se obter a posição de cada um deles num referencial comum ao conjunto de robôs. Aplicando uma transformação matricial com informação proveniente de um GPS seria uma das formas mais triviais de localização mútua, mas nem sempre é possível recorrer a esse sistema de localização.

A agregação de informação sensorial no contexto de navegação multirobótica tem vantagem em missões de busca e salvamento, cujo objetivo é a representação do terreno com elevado detalhe, nomeadamente informação através de sensores LiDAR [32]. Neste *paper* foi apresentado um sistema de localização e mapeamento em tempo real com informação de sensores LiDAR 3D. A arquitetura deste sistema, ilustrada na figura 2.7, baseia-se num grafo de posições (*Pose-Graph*)proveniente

de uma fusão multi-sensorial e uma respetiva otimização do mesmo. O grafo é alimentado diretamente com informação inercial (IMU) e com informação do laser, que é processada por cada robô e de seguida integrada no grafo incremental. A integração de diferentes informações neste grafo é efetuada com *factors*, em que para cada medida sensorial está associado um só *factor*, ou seja, corresponde à relação entre as variáveis de um grafo, definido através de uma função que neste contexto significa a inserção de uma sensorial entre duas variáveis. Cada robô processa a sua respetiva informação sensorial, que gera três *factors*, um de odometria, outro de correspondência de varrimento do LiDAR, e ainda um mapa segmentado. Juntando toda esta informação do conjunto robótico, o bloco de processamento central gera um *factor* que tem como base a junção de toda a informação que resulta numa verificação de segmentos parecidos. Este é o *factor* de reconhecimento, que avalia todos os mapas, e o resultado desta operação é inserida no grafo, procedendo-se a uma otimização do mesmo, devolvendo depois esse resultado aos robôs.

Com recurso a técnicas de visão computacional provou-se obter bons resultados na construção de um mapa global com informação de sensores de diferentes robôs, através de *keypoints* e descritores [6]. Os métodos desenvolvidos neste *paper* são aplicáveis em *point clouds*. No que diz respeito à deteção de *keypoints*, foram dois os métodos desenvolvidos: *Fast Adaptive Laser Keypoint Orientation-invariant* (FALKO) e *Orthogonal Corner* (OC). Em relação aos descritores, *Binary Shape Context* (BSC) e *Cumulative Gaussian Histogram* (CGH) foram os algoritmos propostos na publicação. Começando a análise pelos detetores de *keypoints*, o OC e o FALKO, foram implementados tendo em conta a deteção de pontos estáveis, por exemplo cantos, não se focando em pontos isolados cuja perceção depende do ponto de vista. O FALKO seleciona pontos de interesse implementando um sistema de pontuação que classifica pontos vizinhos dependendo da curvatura percetível de um conjunto de pontos. Por outro lado o OC foca-se na análise do alinhamento ortogonal. A aplicação destes algoritmos é em *datasets indoor*, sendo que a geometria do ambiente não se altera. Varrimentos de LiDAR nestes cenários contêm pontos alinhados em dois planos dominantes ortogonais, se o LiDAR apresentar capacidade de representação a apenas duas dimensões. Tendo como base as condições apresentadas, o algoritmo OC encontra *keypoints* estáveis que se localizam na interseção entre as duas linhas do varrimento. A direção dominante dos pontos do LiDAR é calculada com o algoritmo *Hough Spectrum* [31].

Sobre os descritores, BSC e CGH, o primeiro representa os pontos num histograma linear-polar, centrando-o num *keypoint*, que por sua vez conta os pontos que pertencem a esta região de interesse. Em relação ao outro descritor gaussiano, o

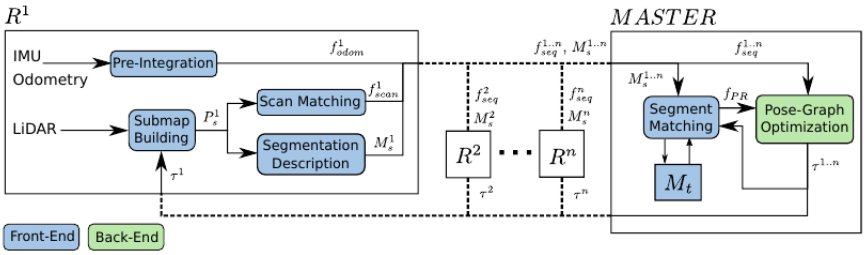



**Figura 2.7:** *Arquitetura do sistema de aquisição sensorial e processamento dos factors de odometria e o de correspondência de LiDAR. O bloco MASTER combina toda a informação adquirida pelo conjunto robótico [6]*

CGH, baseia-se na orientação entre pontos. Um conjunto de pontos é agregado num recipiente, sendo posteriormente analisados através do cálculo de um histograma de modo a obter uma orientação relativa quantizada. Um resultado de associação de *features* combinando o associador FALKO e o descritor BSC está exemplificado na figura 2.8, onde se visualiza a correspondência efetuada entre pontos semelhantes em diferentes leituras, uma delimitada a azul e outra a cor de rosa, e a verde é efetuada a correspondência entre pontos semelhantes, efetuada pelos algoritmos abordados.

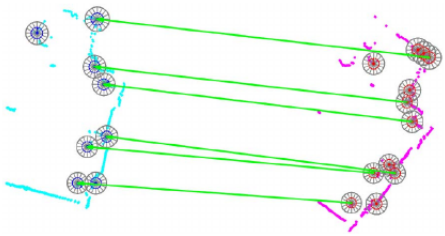


**Figura 2.8:** *Associação de features com o detetor FALKO e o descritor BSC [6]*

## 2.4 Path Planning e Motion Planning

No que diz respeito ao planeamento do caminho que se pretende que o robô efetue, são muito os algoritmos que se usam para resolver este problema. Um dos planeadores de movimento estudados foi o *Rapidly-exploring Random Tree* (RRT) [50], que tem como finalidade encontrar o caminho mais curto até à posição final que o robô tem que alcançar. Este método probabilístico pode não ser suficiente, sendo que em alguns casos [22], algumas áreas exploradas não correspondem aos melhores caminhos. Um algoritmo heurístico [22], A*, de planeamento de caminho resolve esse problema, encontrando o caminho ótimo num mapa com obstáculos já definidos. Em tempo real se o mapa for atualizado com frequência, o algoritmo A* tem essa

informação em consideração para o cálculo do caminho ótimo. Para tirar o máximo partido destes algoritmos [22, 50], a perceção do ambiente da missão tem que ser o melhor possível, sendo que tem estar visivelmente segmentado o que é considerado objeto [55] do que não o representa. Uma mapa rico em informação baseado em octrees, o octomap [3] representa com elevado detalhe e com uma elevada otimização um terreno, tendo como base informação sensorial de LiDAR ou de uma câmara RGB-D. O outro planeador estudado foi o algoritmo *Probabilistic roadmap* (PRM). O seu modo de atuação foca-se num conjunto de pontos aleatórios pertencentes ao espaço de configuração do robô, onde para esse conjunto de pontos, é testado o tipo de espaço em que estes ponto se encontram (livre ou ocupado). Através do planeamento dos movimentos possíveis que o robô pode efetuar ao longo do mapa, através do método abordado, obtém-se um conjunto de pontos vizinhos, em que cada ligação efetuada entre cada ponto próximo, corresponde a um movimento válido no espaço de configuração do robô. Deste modo, obtém-se um mapa com um conjunto de nós que interligam o ponto inicial e o alvo definido antes da execução do algoritmo. O último passo deste algoritmo é encontrar o caminho mais curto até ao alvo da missão com o mapa construído, onde para o efeito é executado o algoritmo de Dijkstra [10], para encontrar o caminho mais curto entre os dois pontos.

[10] https://www.sciencedirect.com/science/article/pii/S1877705812001208

# 3

# Fundamentos teóricos

Neste capítulo vão ser apresentadas todas as tecnologias que permitem suportar todo o desenvolvimento efetuado durante esta dissertação. Em relação ao sensor utilizado para efetuar a aquisição de dados, é da família LiDAR, cujo funcionamento desta tecnologia será abordado durante o primeiro sub-capítulo. Abordar-se-á ainda os fundamentos matemáticos que servem de base à reconstrução de uma nuvem de pontos. Depois vai ser apresentado um estudo relativo à organização de informação sensorial proveniente de um LiDAR recorrendo a estruturas de dados conhecidas. E por último vai-se explorar todos os fundamentos que serviram como base à implementação de algoritmos de segmentação de nuvens de pontos, tendo em conta o estudo de tecnologias do estado da arte, e ainda métodos de construção de mapas com informação sobre ocupâncias do espaço da missão, nomeadamente da *framework octomap*.

## 3.1 Funcionamento da tecnologia LiDAR e aplicações

Um LiDAR é um sensor que mede com elevada precisão a distância a que se encontram objetos, isto é, ecos que são refletidos por uma qualquer superfície, através de um conjunto de lasers, e calculando o intervalo de tempo que estes demoram a serem captados pelo sensor, obtém-se um conjunto de coordenadas representadas em três dimensões, numa relação direta com o referencial do sensor. Esta tecnologia é muito utilizada para mapeamentos densos, onde é possível representar objetos com

elevado detalhe, dada a precisão que estes sensores conseguem alcançar. De modo a reconstruir cenários, cujas aplicações variam desde o contexto da robótica (condução autónoma), em que a precisão de um mapa gerado com recurso a esta tecnologia ajuda na representação de cenários num contexto em tempo real, e ainda aplicações geológicas onde recorrendo à precisão referida no cenário anterior, também se pode aplicar. Muitos são os exemplos de descobertas geológicas com recurso a LiDAR, entre as quais encontra-se uma das maiores descobertas históricas dos últimos anos - Uma cidade da tribo Maia escondida pela alta vegetação na zona de Guatemala, onde através de sensores LiDAR, a recolher dados a partir de um avião, foi possível obter uma representação das estruturas desta antiga civilização. [1]

O modo de funcionamento de um sensor LiDAR tem como base a emissão de um conjunto de feixes de laser, que variam entre modelos. Milhões de feixes são emitidos por segundo, constituindo uma das propriedade destes tipo de sensores - a sua elevada frequência de funcionamento. Os lasers medem a distância da superfície que o refletiu, podendo ser recolhidas um conjunto de propriedades, tais como a refletividade do objeto. Calcula-se a distância em função do intervalo de tempo que cada feixe demora a ser novamente capturado, através da seguinte expressão:

$$Distância = \frac{c \times TOF}{2} \tag{3.1}$$

Em que *Time of flight* (TOF) corresponde ao intervalo de tempo que o laser demorou a ser captado e $c$ é a velocidade da luz no vazio - $3.0 \times 10^8 m/s$.

O laser emitido pelo LiDAR tem que cumprir os requisitos de segurança estipulados pelo *International Electrotechnical Commission* (IEC) 60825 *standard*, que tendo em conta cenários de longa exposição a estes lasers, é necessária a sua certificação, para garantir a segurança dos utilizadores expostos a estes sensores.

Conseguindo extrair a informação sobre distância e orientação de pontos que foram coletados pelo sensor, é possível construir nuvens de pontos com toda a informação agregada. Para a representação de uma nuvem de pontos de um local ser correta, é necessário complementar esta informação com outros sensores de modo a que se consiga extrair a informação sobre posição e orientação no instante em que os lasers são emitidos. Deste modo, para cada ponto aplica-se uma transformação translacional e rotacional para corrigir a localização destes pontos no referencial do sensor.

Para que seja possível a aquisição de uma nuvem de pontos que represente um qualquer local, num cenário não fechado, o esquema da missão deve ir ao encontro do que se encontra ilustrado na figura 3.1. Nesta figura é exemplificada a aquisição

[1] https://www.nationalgeographic.com/culture/2019/03/lasers-reveal-maya-war-ruins/

de dados num ponto de vista aéreo, onde se destaca a importância da utilização de um bom sensor inercial (IMU), pois neste caso podem existir rotações inesperadas em qualquer eixo rotacional: *Roll*, *Pitch* ou *Yaw*. A localização de cada ponto no referencial global, ou do mundo, numa missão em contexto *outdoor*, é obtida recorrendo a dados provenientes de um sistema de posicionamento global (GNSS).

Outros setores onde se aplicam este sensores variam desde o setor automóvel, onde se recorre a esta tecnologia para veículos de condução autónoma, até ao setor geológico e arqueológico, cujos exemplos foram abordados nos capítulos anteriores.

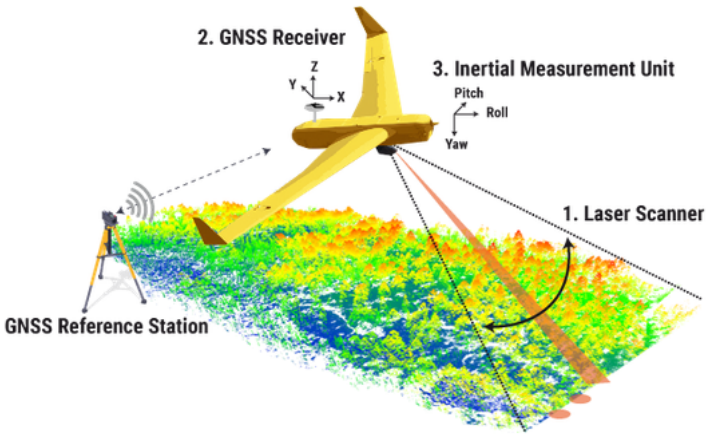



**Figura 3.1:** *Esquema de uma missão de mapeamento*

Outras aplicações têm que ver com manutenção de outras tecnologias, mais concretamente, painéis solares, em que um mapeamento da área em redor de uma casa ajuda a projetar cenários de instalação dos painéis, isto é, se alguma árvore ou arbustos estiverem a bloquear a projeção solar, é facilmente detetável com um mapeamento específico na zona pretendida. Outra razão que justifica o uso do LiDAR neste contexto de instalação tem que ver com a redução do risco de quedas acidentais que podem acontecer durante uma inspeção humana.

## 3.2 Velodyne VLP-16

No caso em específico do LiDAR no qual esta dissertação se baseia, o Velodyne VLP-16, é constituído por 16 feixes de laser que cobrem um campo vertical de 30º, como demonstra a figura 3.2 e horizontalmente cobrem um ângulo de 360º. Os 16 lasers rodam em torno do eixo vertical a uma frequência que pode variar entre 5 a 20 Hz, conseguindo representar um número máximo de 300000 pontos por segundo, num retorno simples, e o dobro dos pontos se for considerada a opção de retorno duplo, considerando que todos os feixes emitidos foram coletados pelo sensor. Estes

**Tabela 3.1:** *Parâmetros associados a um sensor LIDAR velodyne VLP-16*

| Parâmetro | Símbolo |
|---|---|
| Speed of light $(m/s)$ | c |
| Maximum range $(m)$ | Rmax |
| Range resolution $(m)$ | $\delta R$ |
| Scanning rate $(Hz)$ | fsc |
| Field of view $(^o)$ | FOV |
| Laser wavelength $(nm)$ | $\lambda$ |
| Pulse duration $(ns)$ | tp |
| Aperture of laser $(m)$ | D |
| Beam divergence $(rad)$ | $\gamma$ |
| Laser footprint $(cm^2)$ | AL |
| Pulse repetition rate $(kHz)$ | PRR |
| Azimuth angle $(^o)$ | $\alpha$ |
| Elevation angle $(^o)$ | $\beta$ |
| Swath Width $(m)$ | SW |

dados foram retirados do *datasheet* do sensor [10].

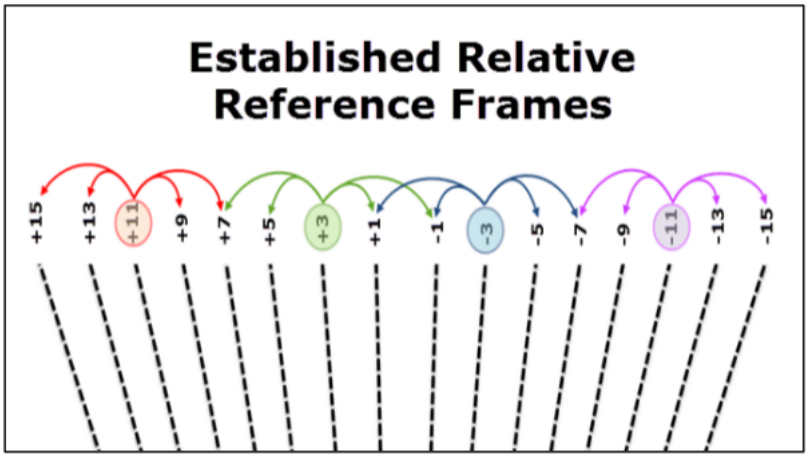



**Figura 3.2:** *Disposição vertical dos 16 lasers de um Velodyne VLP-16 [7]*

Para que se consiga efetuar uma representação do mundo com este sensor, o primeiro passo é a transformação dos feixes refletidos pelo laser, que foram captados pelo seu recetor. Para se efetuar este processo, é necessário um sensor que localize o conjunto LiDAR/robô, um GPS. Para além disto, pode ser necessário um sensor inercial (IMU), que deteta mudanças de posição e orientação.

A reconstrução é efetuada transformando coordenadas esféricas num referencial cartesiano, que advém dos parâmetros do LiDAR [7]$(R, \omega, \alpha)$, em que $\omega$ corresponde

ao ângulo vertical do laser emitido pelo lidar. $\alpha$ é o ângulo horizontal e $R$ é a distância do LiDAR aos objetos que refletiram os feixes de laser. Para converter em coordenadas cartesianas $(X, Y, Z)$ recorre-se às seguintes três fórmulas [61]:

$$X = R \times sin(\alpha) \times cos(\omega) \tag{3.2}$$

$$Y = R \times cos(\alpha) \times cos(\omega) \tag{3.3}$$

$$Z = R \times sin(\omega) \tag{3.4}$$

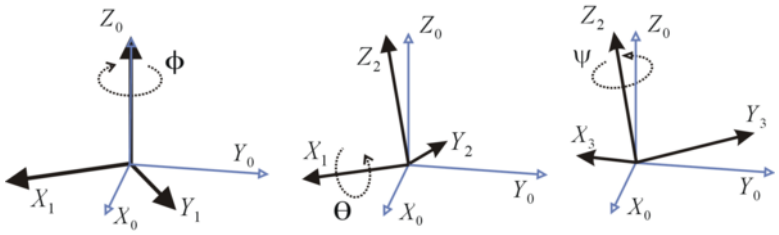



**Figura 3.3:** *Exemplificação dos ângulos de euler*

Estas equações representam o conceito de rotação espacial, que pode ocorrer através de matrizes de rotação com ângulos Euler,$\alpha, \beta, \gamma$, num sistema de coordenadas cartesianas $(X, Y, Z)$. A figura 3.3, exemplifica estes ângulos no sistema de coordenadas referido. A seguinte demonstração matemática, tem como objetivo validar o conceito de rotação espacial, dada uma matriz de transformação $R$, que é obtida pelo produto de três matrizes de transformação intermédias, mediante cada eixo de rotação. A rotação sobre o eixo $x$ é representado pelo ângulo , em $y$ é através de $\psi$ e $\phi$ em $z$. Consideramos ainda que $X$ representa o ponto sujeito à rotação, depois de aplicada a matriz de rotação, e $X'$ a coordenada sobre a qual a transformação é aplicada. As três matrizes de transformação seguem a seguinte notação: $\mathbf{R}_x(\psi)$, $\mathbf{R}_y(\theta)$, $\mathbf{R}_z(\phi)$, com $\mathbf{R} = \mathbf{R}_x(\psi)\ \mathbf{R}_y(\theta)\ \mathbf{R}_z(\phi)$.

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\theta) & -sin(\theta) \\ 0 & sin(\theta) & cos(\theta) \end{bmatrix} \tag{3.5}$$

$$R_y(\phi) = \begin{bmatrix} cos(\phi) & 0 & sin(\phi) \\ 0 & 1 & 0 \\ -sin(\phi) & 0 & cos(\phi) \end{bmatrix} \tag{3.6}$$

$$R_z(\psi) = \begin{bmatrix} cos(\psi) & -sin(\psi) & 0 \\ sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.7}$$

Multiplicando as três matrizes de rotação, obtém-se uma única matriz definida $\mathbf{R} = \mathbf{R}_x(\psi)\ \mathbf{R}_y(\theta)\ \mathbf{R}_z(\phi)$.

$$R = \begin{bmatrix} cos(\theta)cos(\phi) & sin(\psi)sin(\theta)cos(\theta) - cos(\psi)sin(\phi) & cos(\psi)sin(\theta)cos(\phi) + sin(\psi)sin(\phi) \\ cos(\theta)sin(\phi) & sin(\psi)sin(\theta)sin(\theta) - cos(\psi)cos(\phi) & cos(\psi)sin(\theta)sin(\phi) + sin(\psi)cos(\phi) \\ -sin(\theta) & sin(\psi)cos(\theta) & cos(\psi)cos(\theta) \end{bmatrix} \tag{3.8}$$

**A partir de $R$, matriz genérica que representa uma matriz de rotação de um qualquer ponto, que se queira representar noutro referencial. Um exemplo de tal aplicação, consiste na transformação de pontos que estão representados em relação ao LiDAR, no referencial global ou do mundo. Para efetuar esta transformação, recorre-se a uma notação que difere dos Ângulos de *Euler* - quaterniões. A grande vantagem desta notação tem que ver com um problema bastante conhecido da utilização da notação dos ângulos de Euler para efetuar transformação de referenciais, que é o *gimbal lock*, e consiste na perda de um grau de liberdade num espaço tri-dimensional.**

## 3.3 OctoMaps

**Quando se refere a passagem de dados de LiDAR para uma estrutura de dados, o processo denomina-se segmentação, que se refere à organização da informação numa estrutura específica, de forma a que algoritmos de processamento da informação guardada, sejam o mais otimizados possível. A representação de milhões de pontos adquiridos de um sensor LiDAR ocorre pelo meio de uma qualquer estrutura de dados, entre as quais, as que são mais utilizadas para este fim são as *octree* e as Kd trees.**

### 3.3.1 *octree*

**Uma *octree* é uma estrutura de dados hierárquica que é usada para representar um espaço tridimensional. Cada nó desta árvore é subdividido em 8 nós [62], [12], e cada nó é representado por um *voxel*. Um *voxel* é**

um volume cúbico de dimensão regulável tendo como função agregar um conjunto de pontos que encaixe no seu volume. Quanto menor é o tamanho dos *voxels* da árvore, a resolução do mapa resultante é maior, no entanto a árvore será mais densa, pois é necessário um maior número de *voxels*. Um exemplo de uma *octree* a representar espaço livre (a branco) e espaço ocupado (sombreado), está representada na figura 3.5. O primeiro nó da árvore, nó de raiz ou *root node*, é um apontador para a nuvem de pontos total e à medida que se vai progredindo no sentido descendente, os nós vão apontado para zonas mais específicas da *point cloud*. Uma visualização desta divisão está exemplificada na figura 3.4. Sumariando o que se acabou de explicar, uma *octree* é uma estrutura de dados que organiza informação em formatado de árvore. Essa organização é efetuada através de índices ou apontadores, para cada conjunto de pontos a ser representado através de *voxels*.

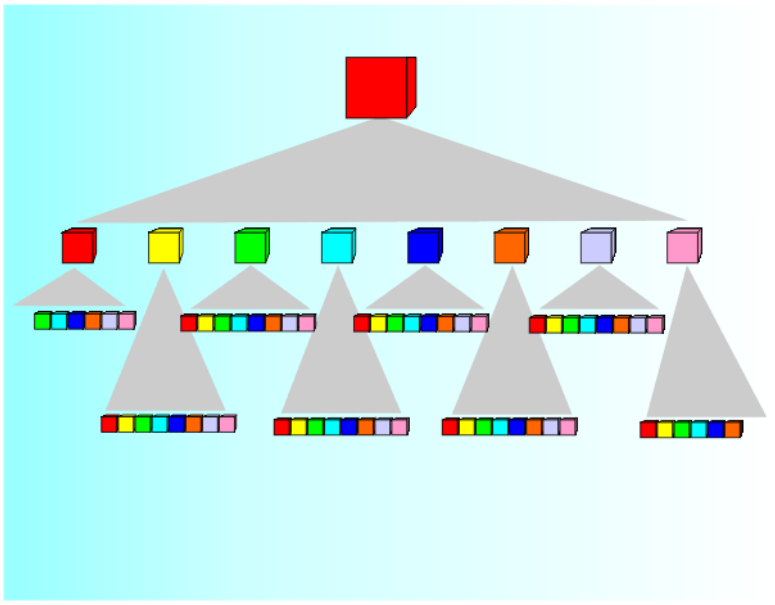


**Figura 3.4:** *Divisão de uma octree [8]*

Como referido no parágrafo anterior, diferentes volumes de *voxels* correspondem a resoluções diferentes, na medida em que a resolução aumenta com a redução do volume. Este efeito pode ser visualizado na figura ??, onde a representação da mesma nuvem de pontos é efetuada em diferentes resoluções, ficando mais percetível quanto maior for a resolução aplicada, que corresponde a *voxels* com 0,08 m de aresta, no exemplo representado. Porém a *octree* que representa este conjunto de

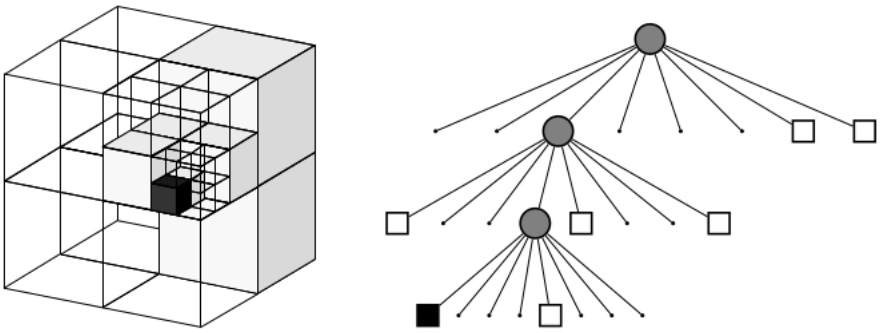


**Figura 3.5:** *Exemplo de pesquisa de um conjunto de pontos numa octree [3]*

**células com maior resolução será, das três representadas na figura ??, a mais densa, isto porque a árvore terá mais subdivisões de nós, o que resulta num aumento do consumo de memória. Esta situação corresponde a um problema recorrente na robótica, em que se necessita uma elevada resolução na perceção ambiental para tomadas de decisão. porém em tempo real, há que ter em conta todo o processamento que é necessário conjugar.**

**Uma *octree* não necessita de ser completamente instanciada [12], se na sua implementação forem usados apontadores para ligar nós aos seus respetivos nós filhos. Caso exista uma porção larga de nós contínuos no espaço, que representem o mesmo tipo de informação, seja referente a espaço ocupado ou livre, estes nós podem passar a ser representados por apenas um nó [12], truncando todos os nós filhos que tenham o mesmo valor. Deste modo, a representação em *octree* a ser acedida em tempo real garante eficiência na gestão de nuvem de pontos com elevadas densidades.**

### 3.3.2 K-d tree

**Este tipo de estrutura de dados assenta no princípio de construção baseada em árvores binárias. Estas árvores contêm um conjunto de nós que são sempre subdividos em dois nós filhos, e assim sucessivamente até não ser possível dividir nós. As *K-d tree* são construídas tendo como base este conceito em que cada nó tem uma dimensão $k$, daí a notação $k-d$ que significa *k-dimensional*. Num conjunto de pontos de dimensão $k$, este algoritmo procura pela mediana deste conjunto e fixa o primeiro ponto do conjunto. O primeiro nó corresponde a um hiperplano que divide o espaço em duas partes. Este processo repete-se até que o espaço esteja**

**todo dividido, estando representado na figura 3.6 um exemplo de uma divisão de um conjunto de pontos. A sua principal utilização é em situações de procura de vizinhos próximos, conhecido como algoritmos de *nearest neighbour search*, ou ainda em pesquisas num determinado intervalo de valores. A eficiência computacional desta estrutura não é superior à *octree*, caso se implemente esta árvore recorrendo a apontadores, isto porque, na construção de uma *K-d tree*, a sua complexidade é $O(n)$. Por outro lado, quando se efetua operações a *k-d trees*, tais como procura de vizinhos, inserção ou remoção de pontos na estrutura já construída, apresenta um complexidade menor: $O(log(n))$.**

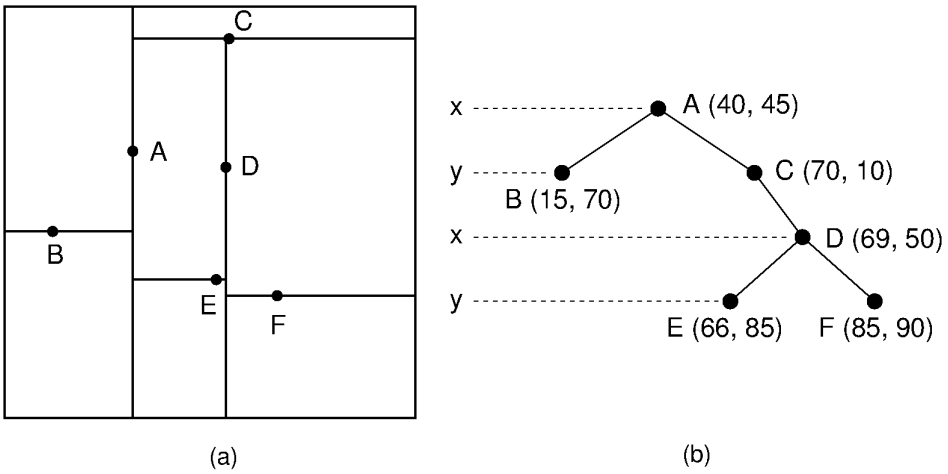



**Figura 3.6:** *Divisão de pontos em k-d tree 2D [9]*

## 3.4 Frameworks de mapeamento

**A Octomap *framework* está disponível em código livre (*open-source*) implementada em C++ e guarda mapas em modelo 3D de um ambiente qualquer, podendo corresponder a mapas aéreos, *underwater* ou ainda *outdoor*. A motivação da criação de uma *framework* com esta finalidade, pretende constituir uma ferramenta de auxílio na navegação robótica, criando modelos tridimensionais do ambiente de atuação de um robô, de um modo eficiente. Na generalidade das aplicações robóticas é necessário uma representação probabilística do ambiente, para determinar com rapidez quais os locais que correspondem a espaço livre ou ocupado. Para além disto, tem que existir uma eficiência nestes processos por forma a que a navegação robótica seja efetuada em tempo real.**

**Uma representação probabilística com medidas sensoriais, nomeadamente um LiDAR ou uma câmara RGB-D, sendo que no caso desta dissertação o estudo recai num sensor LiDAR Velodyne VLP-16. As medidas sensoriais têm uma incerteza associada, o que dificulta a tarefa de perceção do ambiente envolvente, uma vez que não existe uma certeza absoluta de que cada ponto representado corresponde à localização exata no mundo. Por isso a incerteza dos sensores usados tem que ser tida em consideração para minimizar este efeito, senão existe a possibilidade de iludir o robô com algum obstáculo. O ideal é existir uma fusão de incertezas de modo a convergir para um estado que seja o mais próximo possível do estado verdadeiro do ambiente.**

**O segundo aspeto fundamental para uma navegação robótica é a modelação de áreas não mapeadas que não podem ser exploradas porque não existe informação. Depois de se obter um mapa, a tarefa que se segue é a deteção de espaços que correspondem a caminho livre ou a caminho ocupado. Como estes processos podem ser exigentes em termos de processamento computacional em tempo real, a eficiência é um ponto importantíssimo. Para resolver este problema, tal como foi referido anteriormente, o uso de modelos probabilísticos para determinar a ocupância do mapa, correspondem às técnicas mais utilizadas para esta finalidade. O modelo segundo o qual esta *framework* constrói o mapa está representado na equação 3.9.**

**O consumo de memória é a terceira questão fundamental, sendo que esta gestão tem que ser o mais otimizada possível, não esquecendo que à medida que decorre a navegação vai existir um mapa cada vez com mais informação, o que o vai tornar cada vez mais denso com o aumento do tempo de mapeamento. Esta framework foi implementada com o intuito de servir como ferramenta para o desenvolvimento de sistemas robóticos autónomos. O mapeamento do ambiente de atuação do robô é efetuado através de uma fusão sensorial probabilística, sendo as leituras sensoriais integradas no mapa recorrendo a um *occupancy grid mapping*, onde a probabilidade de cada nó, $P(n|z_{1:t})$ [56], é descrita através da seguinte equação:**

$$P(n|z_{1:t}) = \left[1 + \frac{1 - P(n|z_t)}{P(n|z_t)} \frac{1 - P(n|z_{1:t-1})}{P(n|z_{1:t-1})} \frac{P(n)}{1 - P(n)}\right]^{-1} \tag{3.9}$$

**No que diz respeito aos parâmetros desta equação, $P(n)$ corresponde a**

**uma probabilidade anterior, $Z_t$ é a medida sensorial atual, $P(n|z_{1:t-1})$ é a medida sensorial anterior estimada e por último o parâmetro $P(n|z_t)$ denota a probabilidade de um determinado *voxel* $n$ estar ocupado, sabendo qual a medida atual. Resolvendo a equação 3.9 para uma distribuição de probabilidade uniforme , obtém-se $P(n) = 0.5$. Então, usando uma notação logarítmica, a equação de atualização do mapa, 3.9, pode ser reescrita como:**

$$L(n|Z_{1:t}) = L(n|Z_{1:t-1}) + L(n|Z_t) \tag{3.10}$$

$$L(n) = log\left[\frac{P(n)}{1 - P(n)}\right] \tag{3.11}$$

**A necessidade do uso de uma notação logarítmica para reescrever a equação 3.9, tem que ver com a complexidade do produto em termos computacionais. Por outro lado, substituindo esta operação por uma adição, como demonstrado na equação 3.10 resulta em atualizações mais frequentes do mapa. A curva de complexidade de funções do tipo logarítmicas, tendem a convergir para um valor máximo, ao invés de aumentarem gradualmente a sua complexidade.**

# 4

# Desenvolvimento

**Neste capítulo vai-se apresentar todo o desenvolvimento efetuado durante este trabalho de investigação. Uma arquitetura de alto nível, na figura 4.1, do sistema desenvolvido está presente na primeira secção deste capítulo, por forma a apresentar o sistema proposto com a finalidade de implementar um mecanismo de cooperação entre dois robôs. Segue-se o ambiente de simulação escolhido para efetuar a demonstração desta dissertação, o simulador MORSE [18]. A escolha recaiu neste simulador pela fácil integração dos robôs e respetivos sensores, nomeadamente o sensor LiDAR que foi usado para validar métodos de segmentação e de extração de mapas de elevação. Para além disso, a integração estável que este simulador possui com o *middleware* ROS corresponde a outra razão para esta escolha. Desta maneira é possível validar todos os métodos desenvolvidos num contexto em tempo real. Tendo em conta estes dados, a seção que se segue aborda como se efetuaram as transformações de referencial, por forma a que seja possível efetuar um tratamento posterior. Depois todos os algoritmos usados e desenvolvidos, no processo de filtragem de ruído e de segmentação são apresentados. No final do capítulo, todos os processos explicados até este tópico, serão aplicados numa simulação em tempo real no ambiente do simulador MORSE em conjunto com o *middleware* ROS.**

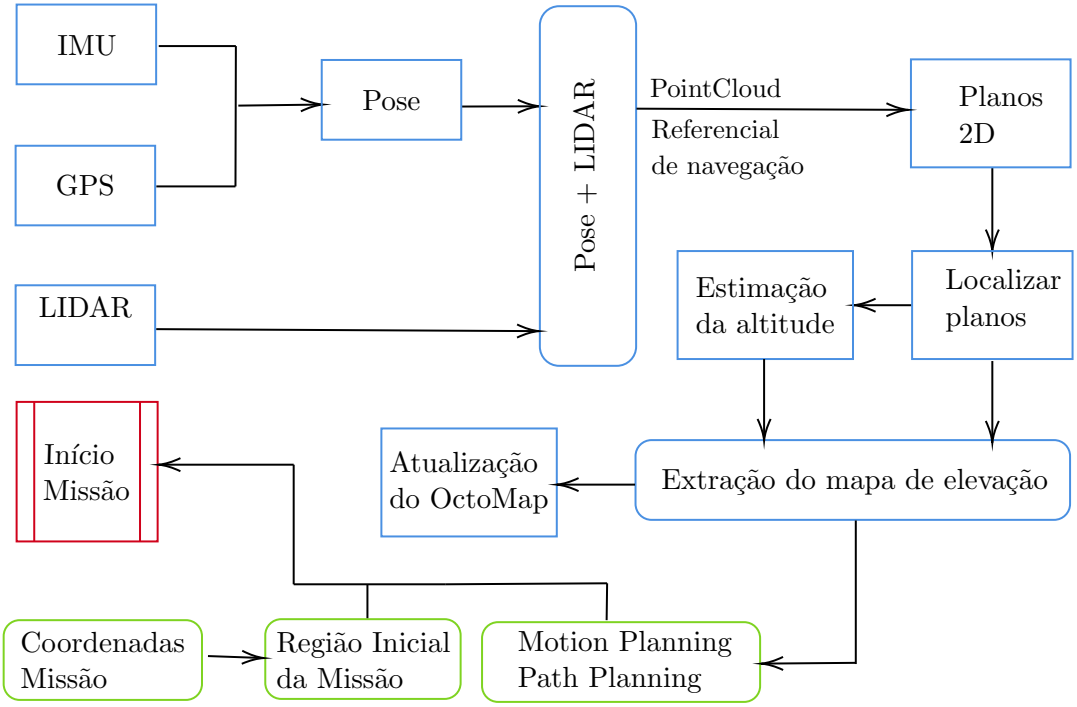



**Figura 4.1:** *Arquitetura de alto nível do sistema proposto*

## 4.1 Arquitetura de alto nível

**O objetivo deste sistema tem um elevado foco no processamento de nuvens de pontos obtidas através de um sensor LiDAR. Por forma a representar a informação adquirida no referencial do mundo, recorre-se à fusão de informação de mais duas fontes sensoriais: GPS e IMU. A informação deste dois sensores resultam numa posição (*Pose)*, que corresponde a uma coordenada no referencial do simulador MORSE. Após a fusão destas três fontes sensoriais, a nuvem de pontos resultante é filtrada e segmentada, de modo a que se consiga identificar e extrair todos os planos da nuvem. A filtragem até aqui referida, corresponde a um processo que inicia com uma redução de pontos, aplicada na *point cloud* recebida na *callback ROS*, que fica à escuta de mensagens do tipo "*sensor_msgs/PointCloud2*". A redução é efetuada recorrendo a *Voxels*, tal como foi referido no capítulo 3, em que se define um tamanho para o conjunto *Voxels* que iteram sobre o *dataset* completo, de modo a reduzir um conjunto de pontos no interior de cada *Voxel*, apenas ao seu centroide. Efetuada essa redução, procede-se à transformação de referencial da nuvem total, aplicando as equações de rotação abordadas nos**

**fundamentos teóricos 3.8, em que a primeira corresponde a uma transformação do referencial do sensor (LiDAR) para o referencial do robô, em que se aplica uma transformação rotacional do sensor em 90 º. Isto porque, no simulador, o LiDAR foi rodado em -90º, por forma a que fosse possível adquirir dados através do UAV utilizado no simulador. por forma a cobrir o cenário de simulação. A segunda transformação diz respeito à rotação do sensor para o mundo, onde através do *Pose* publicado pelo simulador MORSE, é possível obter os parâmetros que são substituídos na equação de transformação do capítulo 3, na equação 3.8. A segmentação corresponde ao processo de identificação de planos na nuvem que está a ser recebida na *callback*, em que existe uma organização dos pontos, através de uma passagem desta *cloud* numa *octree*. Os planos que se pretendem identificar são todos os que correspondem a solo. Deste modo, operações de iteração no *dataset* a ser analisado, são muito mais eficientes, uma vez que a organização subdividiu todos os pontos em formato de árvore, estabelecendo relações entre cada conjunto de pontos, o que num contexto *real time* confere uma maior rapidez no acesso a um determinado conjunto de pontos com coordenadas vizinhas. Com esta estrutura organizada são efetuadas operações de procura de planos 2D, horizontais, verticais e inclinados, em todo o *dataset* que foi recebido na *callback*. A identificação dos planos começa num ponto que corresponde à altura mínima, em relação ao ponto mais baixo conhecido no *dataset* organizado. A partir deste ponto começa a procura por pontos parecidos, o que significa o início da iteração nesse ponto incluindo pontos filhos, deste subconjunto. A operação repete até que no próximo subconjunto de 8 pontos, lembrando que no capítulo 3 se referiu que cada ponto terá exatamente 8 pontos (ou nós) filho, não seja possível encontrar pontos com uma altura em "Z"superior a 5 cm, valor que foi estudado nos cenários de simulação, no próximo capítulo. Este valor corresponde à diferença entre o conjunto de pontos que foi aceite, em relação ao próximo conjuntos de pontos vizinhos. Com esta condição de paragem, retiram-se os pontos que correspondem a solo. Esta operação de segmentação do solo, começa com 4 pontos de procura iniciais, correspondendo neste caso em concreto aos 4 pontos correspondentes às coordenadas limite da área que está a ser alvo de análise. Depois de se identificar tudo o que corresponde a solo, recorre-se ao algoritmo RANSAC, apenas em pontos muito próximos do robô móvel no referencial de navegação, por forma a garantir que os pon-**

tos extraídos correspondem de facto um plano. Definidos os *outliers* de cada plano, procede-se a uma análise da altura de cada plano. Se for muito superior à altura de referência do UGV (o solo), então todas as células contidas neste plano são identificadas como células proibidas. Por outro lado, se um plano horizontal tiver uma altura média ligeiramente superior ao solo (no máximo 5 cm), então todas essas células são atravessáveis. A localização dos planos acontece com outro sensor a bordo do UGV, outro sensor Pose, tal como foi equipado no UAV. Esta decisão tem como base a localização dos planos segmentados, no ponto de vista do UGV, isto porque para ser aplicado um algoritmo de *path planning*, tem que se ter conhecimento do ponto inicial da missão, que corresponde à posição atual do UGV, e ainda o ponto final da missão.

O OctoMap é então populado com a informação processada anteriormente, onde se identificou o que se trata de espaço livre e espaço ocupado, isto é. Por fim, tenta-se alimentar algoritmos de *path planning* com os dados processados, para validar todo o processamento efetuado, de modo a obter um caminho ótimo, com base na classificação do conjunto de células do mapa, no ponto inicial do planeamento, e ainda o alvo que se pretende alcançar. O algoritmo onde se irá efetuar este teste final com o mapa de navegação, ou de elevação, será o *Probabilistic roadmap* (PRM).

## 4.2 Simulação

Para validação do código desenvolvido, optou-se pela utilização do simulador 3D Morse [1], que tem como principal vantagens a sua fácil utilização e interação com diferentes *middlewares*, como é o caso do ROS, que da lista de compatibilidades, disponível para consulta, este é o que tem utilidade para o desenvolvimento. Com este simulador, é possível integrar com planeadores de caminho, e também com planeadores de movimentos, por forma a navegar autonomamente num dos cenários disponíveis. A integração de sensores, entre os quais sensores de visão, GPS, IMU, entre outros, constitui outra vantagem deste simulador, que devido à sua integração com o ROS, emula o envio e a receção de diferentes tipos de mensagens, tais como posição e orientação do robô, ou ainda nuvens de pontos produzidas por sensores LiDAR ou RGB-D. É também possível exportar estados das juntas de um qualquer robô, sendo que no caso dos

[1] https://www.openrobots.org/morse/doc/stable/what_is_morse.html

**robôs que são diretamente suportados pelo simulador, como é o caso do robô atrv [2], a exportação do estado de todas as juntas é efetuado de um modo automático através do módulo *robot state publisher* [3]. No entanto, também é possível fazê-lo para um qualquer robô desde que seja efetuada uma especificação dos estados que cada junta do robô consegue alcançar, e ainda um modelo do robô especificado em XML [4].**

**Com o Morse, implementou-se um cenário com um robô móvel e um UAV. Relativamente ao *drone*, agregou-se um sensor LiDAR, um IMU e um GPS, indo ao encontro do esquema de missão representado na figura 3.1. O outro robô foi equipado com um planeador de caminhos que recebe como parâmetro de entrada nuvens de pontos estruturadas e segmentadas já com a classificação efetuada. O objetivo é validar em que medida o processamento efetuado, da filtragem e segmentação, tem resolução suficiente para serem efetuadas decisões, com dados da perceção sensorial recolhida pelo LiDAR e posteriormente segmentada. E ainda se toda essa informação foi bem classificada. Com este simulador, integrou-se com o *middleware* ROS, por forma a comunicar com este sistema seguindo um modelo de *publisher/subscriber*. Através de tópicos implementados no morse, é possível publicar informação sensorial de diversos tipos, sendo as mensagens mais relevantes as que são do tipo *sensor_msgs/PointCloud2*, que representa o acumular de informação de uma nuvem de pontos adquirida através de um LiDAR. O outro tipo de mensagem com muita importância, corresponde à informação sobre a posição e orientação do UGV ou do UAV, através de sensores como o GPS e IMU, que são do tipo *geometry_msgs/PoseStamped*, que guardam informação de posição e orientação.**

**A simulação efetuada tem como base um tutorial apresentado na página oficial deste simulador, onde através de alguns *plugins* para o *middleware* ROS é possível simular um ambiente de navegação para um robô móvel [5]. Este exemplo tem como objetivo demonstrar a interação que este simulador consegue efetuar com o ambiente robótico anteriormente referido. Através de *scripts* programados em python, constrói-se o ambiente de simulação, desde o cenário onde vai ocorrer a recolha de dados,**

[2] https://www.openrobots.org/morse/doc/stable/user/robots/atrv.html
[3] http://wiki.ros.org/robot_state_publisher
[4] http://wiki.ros.org/robot_state_publisher/Tutorials/Using%20the%20robot%20state%20publisher%20on%20your%20own%20robot
[5] https://www.openrobots.org/morse/doc/stable/user/advanced_tutorials/ros_nav_tutorial.html

**até aos sensores que efetuam essa aquisição. Recorrendo a este exemplo, um dos sensores que foi utilizado, foi um sensor de odometria, que publica esta informação através do tópico "*/odom*". Este sensor publica as transformações de todas as juntas do robô que foi usado neste tutorial, através do módulo ROS denominado TF [6]. O módulo que possibilitou a publicação de todas as transformações das juntas do robô presente neste exemplo, atrv, é denominado *robot_state_publisher*[7]. No que diz respeito à criação do mapa, recorreu-se neste exemplo ao módulo, *gmapping*[8] que constrói mapas a duas dimensões de ocupâncias através de técnicas de SLAM. Elaborando o funcionamento deste módulo, este espera um *input* de um sensor do tipo laser e transformações do módulo *TF*, já referido anteriormente. As mensagens subscritas são do tipo *sensor_msgs/LaserScan* e *tf/tfMessage*. Relativamente aos tópicos publicados, o mais relevante é o tópico "/map"que publica uma mensagem *nav_msgs/OccupancyGrid*.**

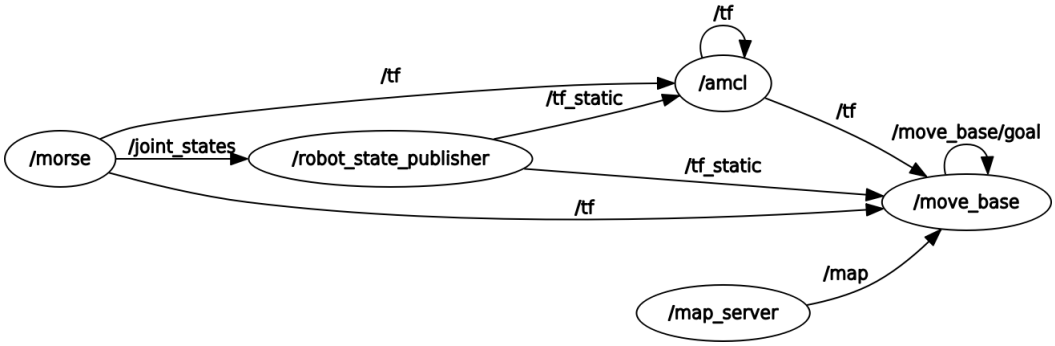



**Figura 4.2:** *Nós e tópicos do esquema de simulação*

## 4.3 Translação e rotação da nuvem de pontos

**Estas operações, rotação e translação, são necessárias de modo a que a perceção do mundo através de visão sensorial seja semelhante à que é perceptível visualmente. A rotação é necessária uma vez que na simulação, rodou-se o LiDAR em -90º por forma a que a aquisição de dados cobrisse uma área lateral maior, e por forma a aumentar a resolução dessa mesma área. Isto é, tendo em consideração a dispersão dos lasers do Velodyne VLP-16, representado na figura 3.2, bem como as propriedades deste sensor (tabela 3.1), este conjunto de lasers giram em torno do**

[6] http://wiki.ros.org/tf
[7] http://wiki.ros.org/robot_state_publisher
[8] http://wiki.ros.org/gmapping

**eixo vertical a uma frequência elevada, por isso rodando o eixo vertical em 90º, consegue-se garantir cobertura da área de interesse. Deste modo é esperado que alguns pontos adquiridos não obtenham retorno ou ainda que detetem o UAV. As transformações rotacionais aplicadas a cada ponto são efetuadas com o objetivo de projetá-los no referencial global. Para que a projeção seja possível é necessário saber a posição e orientação do robô no instante em que se adquire um conjuntos de pontos. Deste modo, aplicando as equações apresentadas na secção 3.2 e sincronizando informação sensorial de um IMU e um GPS que corresponde ao módulo *Pose*, referido no início do capítulo na figura 4.1, obtêm-se então os parâmetro necessários para substituir nas equações referidas no capítulo 3. A implementação destas transformações foi baseada nos mecanismos já implementados pela biblioteca PCL, que consiste numa rotação de referencial baseada em quaterniões.**


**Figura 4.3:** *Velodyne vlp-16*

**Na figura 4.4 é possível visualizar como foram implementadas as rotações de referencial aplicadas a cada nuvem de pontos subscrita, de modo a que se consiga uma representação no referencial do mundo. Na primeira rotação, a variável *stampedTF* é do tipo *geometry_msgs::TransformStamped* [9], e guarda a informação relativamente à transformação de referencial do LiDAR para o UAV. Apesar de não estar representado na figura 4.4, esta variável foi preenchida manualmente, uma vez que o LiDAR está acoplado ao UAV, o que significa que esta transformação de referencial tem valores fixos, pois não se consideram deslocamentos posicionais entre os dois objetos durante a simulação. No que diz respeito às rotações aplicadas, neste caso recorrendo à *tf::Quaternion* [10], cujo construtor utilizado correspondeu ao que inicializa os escalares da transformação rotacional**

[9] http://docs.ros.org/jade/api/geometry_msgs/html/msg/TransformStamped.html
[10] http://docs.ros.org/jade/api/tf/html/c++/classtf_1_1Quaternion.html

```
tf::Transform transform;
tf::Quaternion q;
Eigen::Matrix4f sensorToWorld, sensorTorobot;

pcl::PointCloud<pcl::PointXYZ> laserCloudIn;
pcl::fromROSMsg(*laserCloudMsg, laserCloudIn);

// cloudDownsampling(laserCloudIn,laserCloudIn);

transform.setOrigin(tf::Vector3(stampedTF.transform.translation.x,
                                stampedTF.transform.translation.y,
                                stampedTF.transform.translation.z));
q = tf::Quaternion(stampedTF.transform.rotation.x,
                   stampedTF.transform.rotation.y,
                   stampedTF.transform.rotation.z,
                   stampedTF.transform.rotation.w);
transform.setRotation(q);

pcl_ros::transformAsMatrix(transform, sensorTorobot);

std::vector<int> indices;
pcl::removeNaNFromPointCloud(laserCloudIn, laserCloudIn, indices);

transform.setOrigin(tf::Vector3(pose.pose.position.x,
                                pose.pose.position.y,
                                pose.pose.position.z));

q = tf::Quaternion(pose.pose.orientation.x,
                   pose.pose.orientation.y,
                   pose.pose.orientation.z,
                   pose.pose.orientation.w);
transform.setRotation(q);

pcl_ros::transformAsMatrix(transform, sensorToWorld);

pcl::transformPointCloud(laserCloudIn, laserCloudIn, sensorTorobot);
pcl::transformPointCloud(laserCloudIn, laserCloudIn, sensorToWorld);

pcl::removeNaNFromPointCloud(laserCloudIn, laserCloudIn, indices);
```

**Figura 4.4:** *Implementação das rotações matriciais do LiDAR para o robô e do robô para o mundo*

em $X, Y, Z$ e $W$. **O resultado desta transformação está exemplificado no capítulo 5, nas três figuras referentes aos cenários de validação, no simulador MORSE. A segunda rotação, é aplicada do robô para o mundo, e conta com um modo semelhante de raciocínio quando se compara com a transformação anterior. Neste caso a única diferença, tem que ver com os valores que estão a ser alimentados no *tf::Quaternion*, que correspondem à fusão representada na figura 4.1, em que se obtém um *Pose*. Os valores atualizados no *Pose*, posição e orientação, estão constantemente a ser atualizados, pois depende sempre da posição do UAV em relação ao mundo. Por isso, para efetuar esta projeção, foi necessário existir o cuidado de manter estes valores constantemente atualizados, e por essa**

**mesma razão, esta variável que guarda esta informação (variável *pose*) está a receber mensagens noutra *callback*, mantendo assim a variável atualizada com a informação mais recente. Por último, as instruções representadas por *pcl::transformPointCloud* aplicam as transformações matriciais construídas em *sensorTorobot* e *sensorToWorld*, na nuvem de pontos recebida nesta *callback*.**

## 4.4 Algoritmo desenvolvido para filtragem da pointcloud

**O primeiro passo no processamento que se considerou neste trabalho, numa mensagem que defina uma nuvem de pontos, tem que ver com pontos que não representem algo significante, ou seja, por outras palavras, estes pontos correspondem a um conjunto, sobre o qual se estime que correspondam a ruído na mensagem. Ruído pode corresponder a pontos, que foram submetidos a algum tipo de erro inesperado, por exemplo, devido a erros na leitura de sensores como o LiDAR, o IMU ou o GPS. Erros esses que podem ser de natureza física ou ainda devido a algum erro de sincronização entre as frequências de funcionamento destes sensores que, no ambiente de simulação, se definiu valores que correspondem a frequências reais de funcionamento. A frequência do LiDAR corresponde ao máximo suportado pelo *Velodyne*, cujo valor pode ser consultado na tabela 3.1, que corresponde a 20 Hz. Outro tipo de ruído pode ter que ver com um ambiente de mapeamento exposto a situações que dificultem a leitura sensorial, levando a que a perceção que se obtenha do local a mapear não seja a mais correta. Exemplos de tais distorções poderão ter que ver com mudanças de direção no *drone*, que transporta o LiDAR, provocadas pela direção e força do vento durante a missão, sendo que estas mudanças bruscas poderão não ser corrigidas pelos sensores a bordo, neste caso em concreto pelo IMU.**

**Para minorar este tipo de ruído na nuvem de pontos final, que representa o acumular de todas as nuvens processadas durante a missão, criou-se um algoritmo que aplica um filtro passa banda, para que de um modo eficiente fosse possível extrair esta informação que prejudica o processo seguinte, de identificação e classificação de planos. Para que este algoritmo funcione, é necessário conhecer as coordenadas que delimitam cada *cluster* que passa por este processo de filtragem, sendo que surgiu a necessidade de implementar um algoritmo para efetuar esta tarefa,**

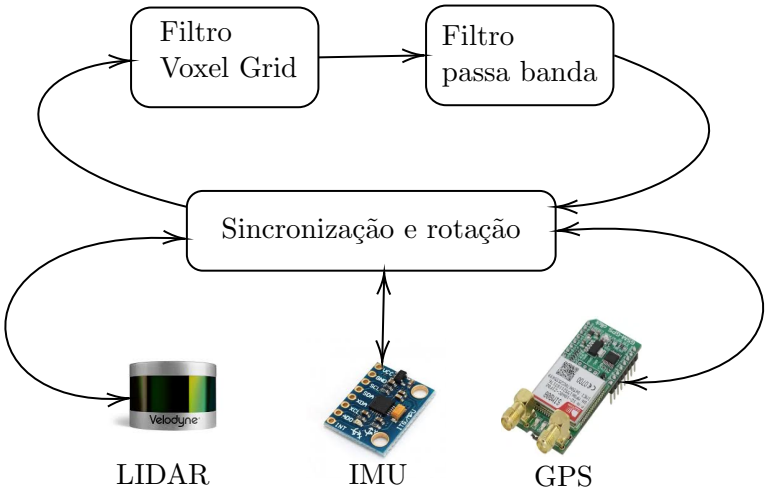



**Figura 4.5:** *Esquema de filtragem proposto*

**cuja implementação está presente na figura 4.7. Nesta implementação, ocultou-se metade do método, porque era demasiado extenso, pelo que é possível demonstrar toda a lógica implementada que está presente no algoritmo 1, apenas com o excerto apresentado na figura 4.7. Para comparar resultados em relação a este algoritmo, tentou-se encontrar soluções já implementadas, mais concretamente métodos disponibilizados pela biblioteca PCL. Em relação ao filtro que é usado para delimitar um intervalo de pontos da biblioteca PCL - *PassThrough filter* [11], a sua utilização está presente na figura 4.9. O modo de utilização deste filtro consiste numa avaliação de pontos dispersos em relação ao eixo dos "X"ou dos "Y", onde através das coordenadas obtidas pelo do algoritmo 1 define-se uma janela de ação para este filtro.**

**Na figura 4.5 está representado em ciclo o sistema de filtragem proposto. Através de três fontes sensoriais que adquirem dados a diferentes frequências, aplicou-se uma matriz rotacional baseada em quaterniões, como abordado na seção anterior, são submetidos a um filtro de redução de pontos- Filtro de Voxel. Aplicando uma janela de tamanho regulável, é possível definir a densidade de pontos sobre os quais se quer obter uma aproximação similar, em que todos os pontos no interior dessa janela, são resuzidos ao seu centroide. A inclusão de tal filtro tem como alvo a redução da densidade das nuvens de pontos que vão ser processadas posteriormente, acrescentando também que se a aproximação da janela**

[11] http://pointclouds.org/documentation/tutorials/passthrough.php#passthrough

**englobar um número de pontos elevado, corre-se o risco de perder resolução na representação da nuvem de pontos. De seguida este conjunto de pontos é processado por um outro filtro, passa banda, que corresponde a um delimitador de intervalos, rejeitando valores que não pertençam ao intervalo definido. A atribuição deste intervalo tem que ver diretamente com subconjuntos de pontos que representam uma elevada porção da *cloud*. Ou seja, do conjunto de todos os pontos, existe um subconjunto de pontos bastante denso e outros subconjuntos com elevada dispersão, sendo que para referência do filtro, os pontos a guardar são os que pertencem ao subconjunto mais denso, que se consideram com um conjunto que obtém informação mais detalhada, isto é com menor ruído. Este filtro *voxel grid*, cuja utilização pode ser consultada na figura 4.5, foi utilizado de modo a que os restantes processos de análise, fossem mais eficientes. Ao definir um tamanho para a janela de cada voxel, uma vantagem da aplicação deste filtro tem que ver com o modo como os planos ficam melhor definidos, isto porque o filtro é aplicado à nuvem completa, o que pode visto como se houvesse uma tentativa de encaixar cubos de igual tamanho, até que não haja mais pontos a englobar nos cubos. Para concluir esta ideia, este *output*, representa uma vantagem na descoberta de planos 2D, que se irá abordar no próximo capítulo.**

```
void passThroughFilterHand(pcl::PointCloud<pcl::PointXYZ> &cloud){

    double timeBefore, timeAfter;
    timeBefore = pcl::getTime();
    auto Size = cloud.points.size();

    for (int i = 0; i < Size; i++)
    {
        if (cloud.points[i].x > Xmin && cloud.points[i].x < Xmax){
            if(cloud.points[i].y > Ymin && cloud.points[i].y < Ymax)
            {
                cloudFiltered->points.push_back(cloud.points[i]);
            }
        }
    }
    timeAfter = pcl::getTime();

    std::cout << "Cloud Size: " << cloudFiltered->points.size() << "\n";
    std::cout << "Time passed passThroughFilterHand " << (timeAfter - timeBefore) * 1000 << "\n";
}
```

**Figura 4.6:** *Implementação do filtro que delimita intervalos de pontos a serem considerados para análise*

**Neste algoritmo é apresentado o raciocínio lógico que levou à implementação do método representado na figura 4.7. Na inicialização das**

```
// maximos e minimos de uma cloud x,y,z
std::vector<float> cloudCoordinates(pcl::PointCloud<pcl::PointXYZ> input)
{
    double timeBefore, timeAfter;
    timeBefore = pcl::getTime();

    float aux1_x, aux2_x, aux1_y, aux2_y, aux1_z, aux2_z;
    float x_max, y_max, x_min, y_min, z_min, z_max;
    x_max = y_max = z_max = initVar;
    x_min = y_min = z_min = -initVar;
    int SegmentIndice = 0;
    std::vector<float> return_array;
    int SegmentSize = input.points.size();

    for (SegmentIndice = 0; SegmentIndice < SegmentSize - 1; SegmentIndice++)
    {
        aux1_x = input.points[SegmentIndice].x;
        aux2_x = input.points[SegmentIndice + 1].x;
        aux1_y = input.points[SegmentIndice].y;
        aux2_y = input.points[SegmentIndice + 1].y;
        aux1_z = input.points[SegmentIndice].z;
        aux2_z = input.points[SegmentIndice + 1].z;
        // X máximo
        if (aux1_x > aux2_x && aux1_x > x_max)
        {
            x_max = aux1_x;
        }
        if (aux2_x > aux1_x && aux2_x > x_max)
        {
            x_max = aux2_x;
        }
        // X mínimo
        if (aux1_x < aux2_x && aux1_x < x_min)
        {
            x_min = aux1_x;
        }
        if (aux2_x < aux1_x && aux2_x < x_min)
        {
            x_min = aux2_x;
        }
```

**Figura 4.7:** *Implementação de um método para o cálculo das coordenadas máximas e mínimas de clusters*

**variáveis foi definido um valor alto para servir como referência durante o passo seguinte de comparação. Iterando de zero até um valor de N - 1, em que N é o tamanho do *cluster* que está a ser analisado, são criadas variáveis auxiliares que guardam o valor da iteração corrente, e da próxima iteração (linhas 8 e 9). A notação $i$ tem como objetivo representar cada eixo da coordenada em análise, por isso cada instrução que contenha este identificador, é sempre repetida para $X, Y$ e $Z$. As condições neste ciclo verificam ponto a ponto, de modo a guardar em variáveis separadas os valores máximo e mínimo que foram descobertos durante a iteração a todos os pontos do *cluster*. No final, retorna-se um vetor com o valor das coordenadas máxima e mínima em cada eixo, no**

**input :** *cluster* com $N$ pontos
**output:** vetor com as coordenadas máxima e mínima $[X_1, Y_1, Z_1; X_2, Y_2, Z_2]$
1 i representa X,Y e Z ;
2 Inicialização: ;
3 initVar = 100000 ;
4 vector = { } ;
5 imax = initVar ;
6 imin = −initVar ;
7 **for** $I \leftarrow 0$ **to** $N - 1$ **do**
8 $i_1aux = points[I]$ ;
9 $i_2aux = points[I + 1]$ ;
10 **if** $i_1\_aux > i_2\_aux$ ***and*** $i_1\_aux > imax$ **then**
11 $imax \longleftarrow i_1\_aux$;
12 **end**
13 **if** $i_2\_aux \geq i_1\_aux$ ***and*** $i_2\_aux \geq imax$ **then**
14 $imax \longleftarrow i_2\_aux$;
15 **end**
16 **if** $i_1\_aux \leq i_2\_aux$ ***and*** $i_1\_aux \leq i\_min$ **then**
17 $imin \longleftarrow i_1\_aux$;
18 **end**
19 **if** $i_2\_aux \leq i_1\_aux$ ***and*** $i_2\_aux \leq i\_min$ **then**
20 $imin \longleftarrow i_2\_aux$;
21 **end**
22 vector $\leftarrow [X_{min}, Y_{min}, Z_{min}; X_{max}, Y_{max}, Z_{max}]$
23 **end**

**Algorithm 1:** Coordenadas limite de um *cluster*

**formato representado na linha 22.**

## 4.5 Segmentação

**Em relação à segmentação, aplicou-se a conversão da nuvem de pontos numa estrutura de dados, de modo a que todos os pontos estivessem organizados para operações de procura de pontos com características similares pudessem ser otimizadas. A informação contida numa nuvem, cujos pontos estão bastante dispersos, é organizada numa árvore, em que cada nó da árvore contém exatamente oito nós filhos. Estes nós derivados têm propriedades semelhantes ao seu nó pai, isto significa que estes índices que organizam a nuvem de pontos, têm como principal objetivo otimizar operações de procura de pontos vizinhos com características semelhantes. A razão desta otimização tem que ver com a quantidade de**

```
void max_min_pcl(pcl::PointCloud<pcl::PointXYZ>::Ptr cloud)
{
    double timeBefore, timeAfter;
    timeBefore = pcl::getTime();
    // pcl::PointCloud<pcl::PointXYZ>::Ptr cloud;
    // cloud = pcl::PointCloud<pcl::PointXYZ>::Ptr (new pcl::PointCloud<pcl::PointXYZ>);
    pcl::PointXYZ minPt, maxPt;
    pcl::getMinMax3D (*cloud, minPt, maxPt);

    /*     std::cout << "Max x: " << maxPt.x << std::endl;
    std::cout << "Max y: " << maxPt.y << std::endl;
    std::cout << "Max z: " << maxPt.z << std::endl;
    std::cout << "Min x: " << minPt.x << std::endl;
    std::cout << "Min y: " << minPt.y << std::endl;
    std::cout << "Min z: " << minPt.z << std::endl; */

    timeAfter = pcl::getTime();

    std::cout << " (PCL)  Time passed: " << (timeAfter - timeBefore) * 1000 << "\n";
}
```

**Figura 4.8:** *Implementação do método da biblioteca PCL que retorna coordenadas limite de uma PointCloud*

```
pcl::PointCloud<pcl::PointXYZ> passThroughFilter(pcl::PointCloud<pcl::PointXYZ> &cloud, int x1, int x2, int y1, int y2)
{
  double timeBefore, timeAfter;
  // timeBefore = pcl::getTime();

  pcl::PointCloud<pcl::PointXYZ>::Ptr cloudNav(new pcl::PointCloud<pcl::PointXYZ>);
  pcl::IndicesPtr indicesX(new std::vector<int>);
  pcl::IndicesPtr indicesExtractedX(new std::vector<int>);
  pcl::IndicesPtr indicesXY(new std::vector<int>);
  pcl::PassThrough<pcl::PointXYZ> pass(true);

  pass.setInputCloud(cloud.makeShared());
  pass.setFilterFieldName("x");
  pass.setFilterLimits(x1, x2);
  //pass.setFilterLimitsNegative (true);
  pass.filter(*indicesX);

  pass.setIndices(indicesX);
  pass.setFilterFieldName("y");
  pass.setFilterLimits(y1, y2);
  pass.filter(*cloudNav);
  // timeAfter = pcl::getTime();

  // std::cout << "Cloud Size:" << cloudFiltered->points.size() << "\n";
  // std::cout << "Time passed passThroughFilter " << (timeAfter - timeBefore) * 1000 << "\n";
  return *cloudNav;
}
```

**Figura 4.9:** *Implementação do método PassThrough presente na biblioteca PCL*

**pontos que são excluídos da procura, porque pertencem a nós completamente distintos, o que significa que até se obter o conjunto de pontos desejado, o número de iterações será muito inferior quando comparada com operações de procura sequenciais.**

**Como se pode observar observar pelo algoritmo, o procedimento de**

```
void cloudDownsampling(pcl::PointCloud<pcl::PointXYZ> &cloud, pcl::PointCloud<pcl::PointXYZ> &cloudOut)
{
  pcl::toPCLPointCloud2(cloud, *cloudNoFilter);
  pcl::VoxelGrid<pcl::PCLPointCloud2> sor;
  sor.setInputCloud(cloudNoFilter);
  sor.setLeafSize(0.08, 0.08, 0.08);
  sor.filter(*cloudDownsampled);
  const std::vector<int> indices;
  pcl::fromPCLPointCloud2(*cloudDownsampled, cloudOut);
  // std::cout << "Size after Downsampling: " << cloud.size() << std::endl;
  // pcl::copyPointCloud(*cloudDownsampled, indices, cloud);
}
```

**Figura 4.10:** *Implementação do método VoxelGrid presente na biblioteca PCL*

**input :** *cluster* filtrado com $N$ pontos

**output:** *Cluster* organizado em estrutura de dados

1 /* Função da linha 2 é representada pelo algoritmo 1 */

2 $[X_1, Y_1, Z_1; X_2, Y_2, Z_2] \leftarrow$ `cloudCoordinates(`*cloud filtrada*`);`

3 **for** $I \leftarrow 0$ **to** $nSearchPointGenerator$ **do**

4 $\quad searchPoint.x \leftarrow RandomNumberGenerator(X_1, X_2)$ ;

5 $\quad searchPoint.Y \leftarrow RandomNumberGenerator(Y_1, Y_2)$ ;

6 $\quad$ /* Vetor com as sementes de procura de pontos vizinhos */

7 $\quad searchVector[I] = searchPoint;$

8 **end**

9 $newCloud \leftarrow$ `cloudToOctree(`$cloudfiltrada$`)`

10 **for** $J \leftarrow 0$ **to** $searchVector.size$ **do**

11 $\quad$ **if** $octreeNearestKSearch(searchVector[J] > 0$ **then**

12 $\quad\quad$ **for** $K \leftarrow 0$ **to** $SearchResul.size$ **do**

13 $\quad\quad\quad$ /* Alocam-se os índices na variável neighborsResult */

14 $\quad\quad\quad neighborsResult \leftarrow SearchResul[K]t;$

15 $\quad\quad$ **end**

16 $\quad$ **end**

17 **end**

**Algorithm 2:** Segmentação inicial aplicada à nuvem de pontos filtrada

**segmentação aplicado neste trabalho, tem início num método já referido no algoritmo 1. O resultado deste método são as coordenadas que delimitam a nuvem de pontos que corresponde ao *input* do algoritmo 2. De seguida efetua-se uma operação de procura de pontos vizinhos, que tem início num conjunto de sementes, cujas coordenadas são geradas aleatoriamente, através de uma distribuição normal, dentro dos limites retornados pelo método representado na segunda linha do algoritmo 2. Após a geração das coordenadas das sementes que vão servir de base para operações de procura de pontos vizinhos semelhantes, é efetuada a passagem da nuvem de pontos para uma estrutura de dados organizada - *octree*. Este método, na linha 7, retorna os índices dos pontos organizados, otimizando assim ainda mais os métodos que se seguem da segmentação, isto porque este método de organização utiliza apontadores para os índices da nuvem de pontos. De seguida avaliam-se que pontos vizinhos entre as sementes têm um conjunto de características parecidas, isto é em coordenadas semelhantes. Se existir uma diferença de 10 cm em relação inicial da semente, então esse ponto não pertencerá ao conjunto de pontos vizinhos.**

**O próximo procedimento na análise da nuvem de pontos depende do local onde se pretende mapear, local este que é definido em função da localização. O critério definido nesta situação tem que ver com a possibilidade de apresentar resultados de processamento em tempo real que possam ser considerados viáveis para uma missão num cenário real. Esta afirmação significa que o número de pontos a processar/segmentar tem que ser o mais reduzido possível, por forma a que seja possível tomar alguma decisão com o módulo de *Path Planning*. Tendo em conta estes requisitos, o que foi desenvolvido neste sentido é como se fosse uma janela de tamanho fixo, que englobe uma área máxima de 10 $m^2$ em torno do robô. A escolha desta área teve que ver com vários testes efetuados com as possíveis áreas a considerar, em função da complexidade do cenário em contraste com o número de pontos que se estimem serem necessários para traçar um caminho navegável. E ainda o tamanho do robô, que foi o último de atribuição desta área, pelo que este valor teria que ser ajustado no caso de este método se aplicar a um UGV de diferentes dimensões. A área inicial não está obrigatoriamente a incluir o alvo da missão, uma vez que o objetivo é validar a acumulação dos vários caminhos calculados, até se obter um caminho que válido até ao alvo, no mapa de navegação.**

**O raciocínio relativamente à gestão da área a considerar para tomada de decisões está representado no fluxograma da figura 4.11. Caso o ponto final esteja incluído nesta área, então o plano é tentar alcançar este ponto final, alimentado inicialmente ao sistema. Em sentido inverso, a ordem enviada ao UGV é a de planear um caminho para o espaço não ocupado mais distante da localização onde se encontra antes de aplicar o algoritmo de planeamento. Se for possível atingir essa localização, então através do módulo "move_base", abordado no início do capítulo, recebe os blocos que foram considerados para a escolha do caminho até ao ponto de referência de final da nuvem. Na área em redor robô analisa-se os conjuntos de pontos agregados como foi esclarecido no algoritmo 2, onde se avalia pontos com alturas semelhantes, por forma a testar se algum plano, vertical ou horizontal pode ser encaixado. Um conjunto de pontos é considerado coplanar se apresentar na sua vizinhança um declive médio que corresponda a um valor muito próximo de 0º, com uma margem de erro máximo de 2º, para não se excluírem planos por falta de resolução na deteção de algum local. São também considerados planos verticais com um declive próximo de 90 º, que corresponde a alguma parede, ou alguma árvore que fora reduzida a um conjunto de pontos menos dispersos, formando um plano vertical. Todos os planos considerados verticais são marcados para exclusão de área a considerar para o planeamento de caminho, ou seja, é considerado espaço não atravessável. No caso dos planos horizontais é necesário efetuar uma verificação extra, isto porque um plano considerado horizontal pode não estar a uma altura próxima da referência, que corresponde à altura base do UGV, por ser o ponto de contacto do UGV com o solo. Por isso sempre que um plano é horizontal é calculada a altura média do pontos que constituem esse plano, sendo que esse é excluído se a sua altura média for superior ao valor de inclusão abordado previamente - 10 cm. Valores que são superiores, em módulo, são também considerados caminho proibido. Nesta situação pretende-se considerar objetos, cuja representação foi interpretada como um plano horizontal no seu ponto mais elevado, por exemplo, o teto de um carro. Por último, foi considerado a análise de planos inclinados, ou oblíquos, que correspondem a todos os planos não classificados nos critérios anteriores. Esta análise é importante, pois num terreno irregular os critérios de análise referidos podem falhar a sua classificação, pelo que se não fosse efetuada esta averiguação, existia o risco de aceitação**

**ou rejeição equívoca. Pequenas elevações, tais como rampas, ou até terreno que mude de declive num curto espaço, são objeto de análise. Caso o seu declive ultrapasse os 10º entre planos próximos considera-se que o local é perigoso de atravessar, então é rejeitado. No sentido inverso, ocorre a sua aceitação, no entanto há que ter em consideração que estes planos podem corresponder a outros objetos que foram percecionados desse modo, como exemplo desta situação incluem-se os telhados. Neste caso dependendo da orientação do *drone* e da casa do cenário, existe a possibilidade de se adquirirem dois planos que se intersetam, logo estão juntos no espaço, e por esta razão são alvo desta verificação, e se forem considerados atravessáveis numa primeira fase, levaria à introdução de planos errados. Por esta razão introduziu-se uma verificação depois de no caso de planos oblíquos serem considerados com pouco declive entre planos, em que, tal como no caso abordado no parágrafo anterior, é verificada a altura média desses planos, em que se for superior no mesmo fator anterior, ou se essa altura média for muito semelhante, com um máximo de 2 cm de erro em módulo, então os planos marcados como não atravessáveis.**

**No entanto, é uma tarefa complexa, uma vez que como se referiu no parágrafo anterior, a perceção de um telhado vai depender de como o *scan* é efetuado. Ou seja, se apenas for considerada uma parte do telhado, essa zona é excluída, e se a zona do outro lado casa da casa, no lado que não foi atingido pelo mapeamento, pode induzir em erro porque não está a ser considerado para análise. Nestas situações, em que se nota falta de informação é logo considerado espaço proibido, embora nesta primeira análise existe a possibilidade de se excluírem zonas atravessáveis. Estas só serão alvo de mapeamento, se a falta de informação gerar algum bloqueio nessa zona em particular. Toda esta informação é inserida num *octomap* que vai acumulando toda a informação que se vai percionando no decorrer da missão.**

## 4.6 Navegação

**Neste capítulo, até esta secção, foram abordados diversos métodos de processamento de nuvem de pontos, relativamente à filtragem e à segmentação que se aplicou nesta dissertação. Nesta seção, o desenvolvimento teve mais que ver com a interação entre os dois robôs no contexto**

**da missão. Veja-se então o seguinte fluxograma, na figura 4.11, onde é possível constatar a solução proposta para coordenação entre os dois robôs que navegam de um modo colaborativo. Como já foi referido no decorrer deste capítulo, a zona que está a ser considerada para o desenvolvimento de algoritmos de segmentação é uma área que encontra na vizinhança do robô terrestre. Por forma a mapear esta zona com uma resolução elevada, o esquema de missão desenvolvido, começa com um método que calcula a zona onde se irá aplicar todos os algoritmos até agora referidos, de modo a distinguir o que se trata de espaço livre e ocupado. De seguida o UAV sobrevoa a zona, começando na mesma localização do robô terrestre. O procedimento que se segue corresponde ao voo do UAV, através de um conjunto de cinco *waypoints*, por forma a que toda a área previamente determinada, seja abrangida pelo sensor LiDAR que encontra acoplado ao UAV.**

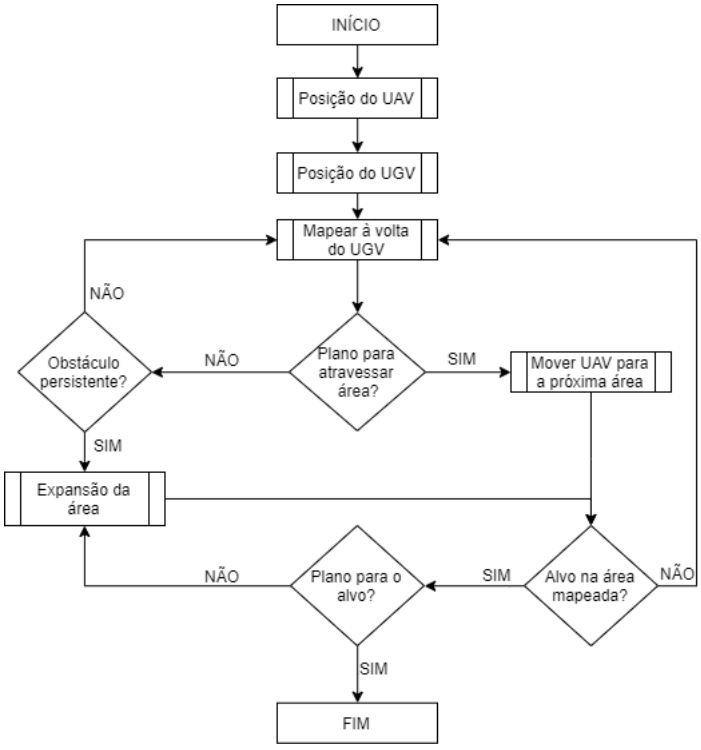



**Figura 4.11:** *Fluxograma da sincronização entre os dois robôs*

**As duas primeiras rotinas, a posição do UAV e do UGV, correspondem a dois subscritores, que atualizam as coordenadas de cada robô à medida que a missão decorre, em função da frequência de publicação das mensagens correspondentes aos dados referidos, que corresponde a**

**5 Hz. A rotina que se segue corresponde à ordem para o UAV começar a mapear com o LiDAR que tem a bordo, que apenas pára de mapear, se o resultado de todas as verificações que se seguem forem verdadeiras, atingindo então o bloco que representa o fim da missão.**

**De seguida, verifica-se se existe um caminho para atravessar a área que foi mapeada, em terreno próximo do UGV, que caso não se verifique vai-se ao encontro de uma outra condição que procura identificar se se trata de um obstáculo persistente, isto é, com a área definida para mapear, não foi possível obter conhecimento suficiente para o contornar. Neste caso é efetuado um novo cálculo da área a mapear onde se efetua uma expansão lateral e frontal da área a considerar para o processamento. O obstáculo é considerado persistente se não existir plano para ultrapassá-lo, durante a análise de pelo menos duas áreas, a tentar ultrapassar as mesmas células ocupadas. Em sentido oposto, se o obstáculo não for considerado persistente, significa que existe plano para ser ultrapassado depois de se expandir a área de aquisição apenas uma vez. Esta condição, por outras palavras, num primeiro momento considera que existe um bloqueio no cálculo do caminho por não existir detalhe suficiente para se efetuarem decisões sobre o caminho que se pode atravessar, até que após várias repetições, é considerado que a informação disponível não é suficiente para o contorno do obstáculo, ativando então a rotina que devolve ao UAV a nova área a considerar para o mapeamento. O UAV nesta condição recebe novos *waypoints* para mapear uma nova área, direções que aumentam a perceção tanto para a frente, como nas laterais do robô móvel.**

**Depois da expansão da área, atinge-se uma condição que avalia se esta contém as coordenadas do alvo da missão. Neste caso, o procedimento será o de tentar calcular o caminho até ao alvo, começando na posição atual do robô. Se não estiverem incluídas estas coordenadas após uma expansão, é efetuado o procedimento esclarecido no parágrafo anterior. Onde se vai tentar chegar ao local limite da área que está a mapear.**

# 5
# Resultados

**A validação dos algoritmos desenvolvidos neste contexto da dissertação foram efetuados recorrendo a cenários retirados do simulador MORSE, em que se gravaram tópicos compatíveis com o *middleware* ROS, para três cenários diferentes (figuras 5.1, 5.2 e 5.3. As mensagens $sensor\_msgs/PointCloud2$ foram gravadas recorrendo a um sensor velodyne integrado num UAV, na definição do cenário no simulador, a publicar a uma frequência de 20 Hz, e ainda o módulo *Pose* a publicar a uma frequência de 5 Hz através de mensagens $sensor\_msgs/PoseStamepd$. Durante a captura do cenário completo, foram definidos cinco *waypoints* para o UAV, por forma a representar da melhor forma possível os cenários que foi abordado nas figuras anteriormente referenciadas. Com esta informação gravada, foi possível executar de forma individual e retirar todos os resultados demonstrados ao longo do capítulo, na aplicação de todos os filtros, na segmentação e na criação do mapa de elevação.**

**Foram desenvolvidos três cenário para validar todos os algoritmos desenvolvidos nesta dissertação. O primeiro cenário, cuja representação está presente na figura 5.1, é o cenário de análise mais simples no que toca à recolha de dados, uma vez que o cenário foi populado com caixas, que são facilmente representáveis com LIDAR por apresentarem uma superfície plana, cuja altura em relação ao solo sabe-se a priori que será maior que o valor de referência mínimo do solo. O segundo**

**cenário apresenta um grau de complexidade maior, pois já foram colocadas árvores, na figura 5.2, que numa representação com LIDAR correspondem a condições difíceis de extrair ao efetuar um varrimento aéreo. A dificuldade deste cenário reside no facto de num varrimento aéreo os lasers do LIDAR terem um alta probabilidade de não serem retornados de volta ao sensor por baterem em folhas da árvore. A simulação efetuada no MORSE, como é baseada no *software* de modelação *blender* [1], permite conferir propriedades aos objetos, que se regem pelas leis da física. O último cenário de teste que pode ser consultado na figura 5.3 é o mais complexo dos três apresentados, isto porque contêm objetos cujo detalhe de representação é mais complicado de reconstruir. Como exemplo de tal complexidade, existem os telhados das casas da figura 5.3, em que se não forem contempladas todas as frentes de cada casa, então perder-se-à informação fulcral, de solo que não foi considerado para análise por não ter sido representado pelo LIDAR. Todo o cenário que não for contemplado na nuvem de pontos será automaticamente considerado caminho proibido.**

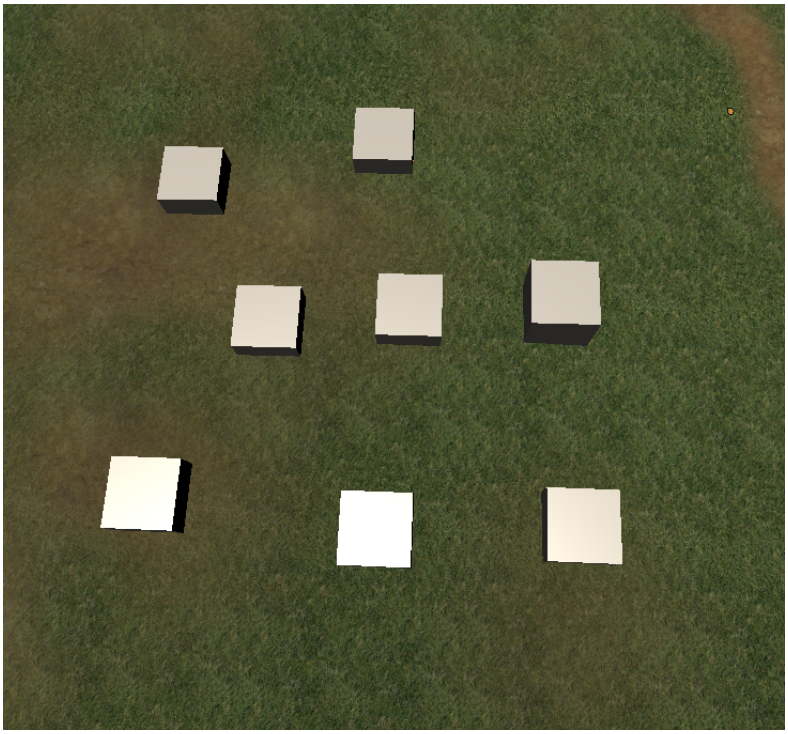


**Figura 5.1:** *Primeiro cenário de validação*

[1] https://www.blender.org/

| Cenários | Cenário 1 (caixas) | | Cenário 2 (árvores) | | Cenário 3 (casas) | |
|---|---|---|---|---|---|---|
| Parâmetros de teste | Número de pontos médio | Tempo de execução (ms) | Número de pontos médio | Tempo de execução (ms) | Número de pontos médio | Tempo de execução (ms) |
| Sem filtro | 7984 | 1.17 | 16281 | 1.04 | 16388 | 1.26 |
| Filtro de Voxel 2 cm | 2873 | 2.10 | 4920 | 2.68 | 6569 | 3.33 |
| Filtro de Voxel 3 cm | 2873 | 2.42 | 4919 | 2.72 | 6534 | 3.4 |
| Filtro de Voxel 5 cm | 2516 | 2.11 | 4384 | 2.85 | 5386 | 3.24 |
| Filtro de Voxel 10 cm | 1323 | 2.41 | 2306 | 3.13 | 2938 | 3.18 |

**Tabela 5.1:** *Comparação de performance do filtro de Voxel*

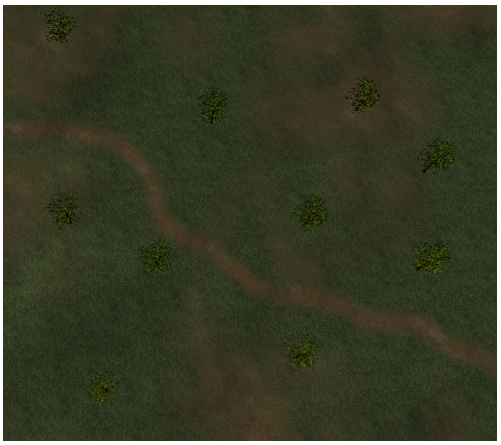


**Figura 5.2:** *Segundo cenário de validação*

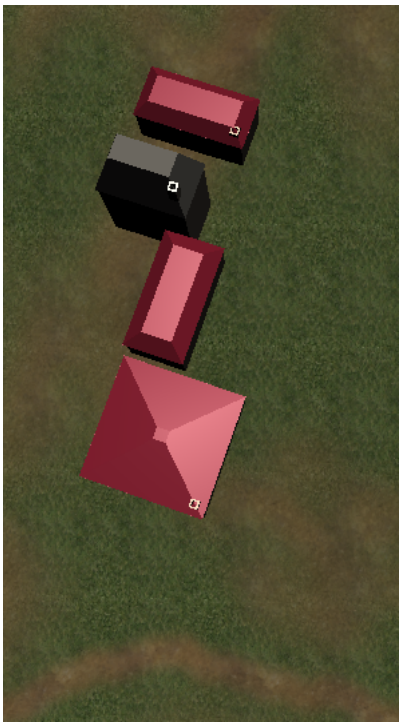


**Figura 5.3:** *Teceiro cenário de validação*

**A primeira demonstração de resultados corresponde à apresentação das imagens recolhidas antes de serem efetuadas transformações de referencial e após essas transformações. Nas figuras 5.4 e 5.6, podemos visualizar um acumular de informação sobreposta, sempre no mesmo espaço, pois as representações estão a ser retiradas em função do referencial do robô. Após as transformações de referencial, aplicando os conceitos abor-**

**dados teoricamente no capítulo 3 e cujo desenvolvimento foi explicado no capítulo 4, obteve-se os resultados visíveis nas figuras 5.1, 5.2 e 5.3, respetivamente. Os tempos de execução medidos na *callback*, nos instantes em que se efetua a conversão de referencial dos pontos para o mundo do simulador MORSE, podem ser consultados, na tabela 5.1, onde na linha referente ao processamento sem filtro, apenas foi considerado o cenário de aplicação das matrizes de transformação, alimentadas posteriormente com o módulo Pose, como abordado na arquitetura de alto nível, na figura 4.1. Através da análise destes tempos de execução, o processamento foi considerado rápido, uma vez que aplicou-se diretamente as matrizes de rotação, sem recorrer ao módulo TF ou TF2 do ROS, que iria introduzir outros problemas de *performance* na *callback*. Isto porque seria necessário escutar dentro de um ciclo *try...Catch*, por forma a atualizar as transformações de referencial em função do *timestamp* devolvido pelo MORSE. O que iria trazer outros problemas, nomeadamente o tempo de simulação sincronizado pelo MORSE, pelo módulo *Clock()*, levantou imensos problemas, uma vez que, o *buffer* do TF do ROS, não considerava a maior parte dos dados das transformações publicadas, o que resultou em cenários acumulados com informação em falta e na maior parte dos teste incorreta. Por isso, a aplicação direta das matrizes, com os dados recolhidos pela posição do UAV em relação ao mundo, gerou melhores resultados.**

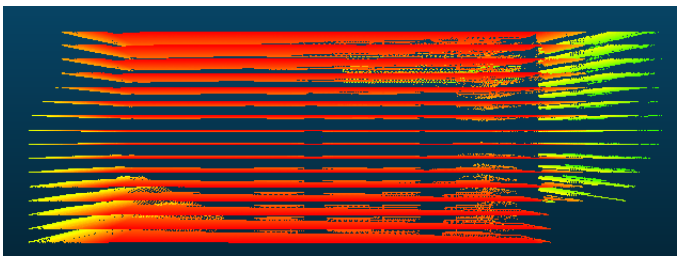


**Figura 5.4:** *Representação do cenário 1 no referencial do robô*

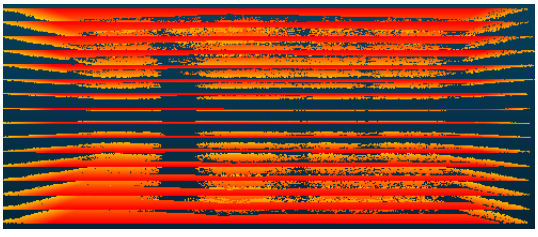


**Figura 5.5:** *Representação do cenário 2 no referencial do robô*

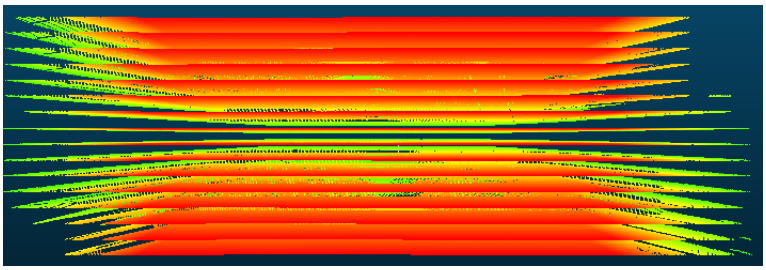


**Figura 5.6:** *Representação do cenário 3 no referencial do robô*

**O resultado desta transformação está presente nas figuras 5.7, 5.8 e 5.9 onde está bem definido o que corresponde a solo, ou caminho permitido, e o que são objetos, com as respetivas cores a corresponderem à sua altura em relação ao referencial do mundo. Analisando os tempos obtidos com a resolução dos cenários após a aplicação do filtro, nota-se uma grande semelhança nas *clouds* filtradas com um *leaf* de 2, 3, 5 e 10 cm, representados respetivamente nas figuras 5.10, 5.11, 5.12 e 5.13. Esta semelhança visual significa que o filtro aplicado não retirou resolução ao cenário original (figura 5.1), o que significa que a redução de pontos não tem influência negativa no próximo passo, a segmentação. E como os tempos de execução são bastante semelhantes entre as gamas escolhidas para teste de validação, então decidiu-se optar pelo uso do filtro com 5 cm de *leaf*, visto que o número de pontos médio por mensagem é inferior em relação ao de 2 e 3 cm. Em relação ao filtro de 10 cm, tem um número de pontos muito inferior aos restantes, no entanto, como é possível visualizar pela figura 5.13, a perda de resolução da representação foi substancial, por isso, nos *datasets* adquiridos com o simulador *MORSE* decidiu-se não utilizar *leafs* próximas dos 10 cm.**

**Na tabela 5.1 encontram-se os resultados obtidos com a aplicação do filtro de voxel, onde se analisa o tempo de execução da rotina de rotação de referencial, recorrendo a diferentes resoluções deste filtro, e sem a aplicação do mesmo. A receção de informação sensorial do LIDAR, através de mensagens $sensor\_msgs/PointCloud2$, numa simulação com um nó implemento no *middleware ROS*. Nesta análise, foi também incluído o método de filtragem implementado, cujo funcionamento foi abordado no capítulo 4 - *voxel grid*, em que se aplicou uma seleção de uma área de interesse a considerar para o seu processamento. Os dados que foram alvo de análise consistem no número de pontos médio por mensagem recebida na *callback*, depois da aplicação do método referido, efetua-se o**

processamento das mensagens publicadas pelo MORSE. Retirou-se ainda o tempo de execução desta *callback*, sem aplicação do filtro, e aplicando três tamanhos diferentes para o filtro - *leaf size*. Os resultados obtidos nos três cenários são semelhantes, visto que a fonte de recolha de dados para todos os cenários é igual (simulador MORSE) e o tratamento da densidade pontual tem características semelhantes também.

O tempo de execução sem aplicação do filtro foi o que obteve uma *performance* superior, como era expectável, uma vez que o tempo medido corresponde apenas às transformações matriciais aplicadas a cada nuvem que despoleta a execução desta *callback* de processamento de *pointclouds*. Por outro, neste teste sem filtro, o número de pontos médio acumulado por cada mensagem recebida é maior.

Por isso, tendo em conta os resultados dos três cenários conclui-se que uma *leaf* de 5 cm é o melhor parâmetro para ser aplicado ao tamanho do voxel de filtragem. Isto porque, tem um tempo de execução inferior às *leafs* de 2 e 3 cm, e apesar de ser menos eficiente que o filtro com 10 cm, confere uma maior resolução, reduzindo a probabilidade de classificar de um modo errático o espaço livre em cada cenário. Visualizando as figuras 5.12 e 5.13, que demonstram o resultado da aplicação do filtro com 2 e 10 cm respetivamente no cenário das caixas, suporta-se a ideia de que a perda de resolução não compensa em função da eficiência que se ganharia. Por outro lado, comparando a figura 5.12 com as figuras 5.10 e 5.11, não existem diferenças evidentes em relação à resolução apresentada, pelo que este é o segundo fundamento para a escolha de uma *leaf* de 5 cm.

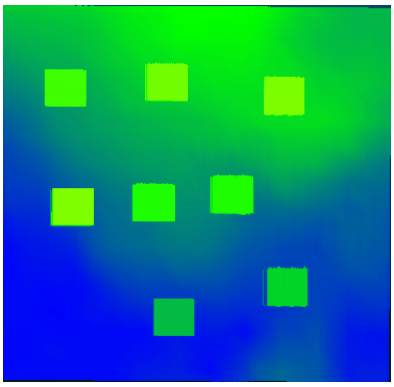


**Figura 5.7:** *Representação acumulada sem filtro voxel (cenário 1)*

O próximo teste tem que ver com a análise do segundo tópico da filtragem, a seleção de coordenadas a considerar para o processamento

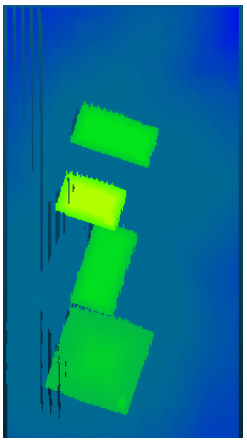


**Figura 5.8:** *Representação acumulada sem filtro voxel (cenário 3)*

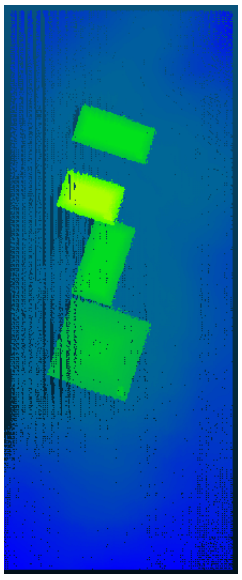


**Figura 5.9:** *Representação acumulada sem filtro voxel (cenário 3)*

**posterior. Tal como na figura 4.5, este é o passo que se segue no desenvolvimento efetuado. Novamente para os três cenários vão ser efetuados dois testes de aplicação de uma filtragem de coordenadas a serem consideradas. Este conjunto de coordenadas no contexto do cenário desenvolvido, corresponde a uma área que se definiu à volta do UGV, tal como foi abordado no capítulo do desenvolvimento, em que essa área deverá corresponder a uma zona intermédia que lhe permite alcançar o objetivo. Os resultado estão apresentados na tabela 5.2, conclui-se que o algoritmo implementado pela biblioteca PCL, apresenta melhores resultados nesta função, em relação ao algoritmo desenvolvido para este efeito. Os tem-**

| | PCL Máximos e Mínimos (ms) | Algoritmo implementado (ms) |
|---|---|---|
| Cenário 1 | 0.055 | 0.181 |
| Cenário 2 | 0.065 | 0.189 |
| Cenário 3 | 0.05 | 0.183 |

**Tabela 5.2:** *Comparação entre dois métodos de extração de coordenadas limites de uma cloud (bounding box)*

| | Filtro PassThrough (ms) | Algoritmo implementado (ms) |
|---|---|---|
| Cenário 1 | 0.61 | 0.23 |
| Cenário 2 | 0.84 | 0.33 |
| Cenário 3 | 0.62 | 0.34 |

**Tabela 5.3:** *Comparação de dois métodos para inclusão de pontos dentro de uma determinada área*

**pos de execução do algoritmo implementado, são muito semelhantes, no entanto são superiores ao método já implementado, pelo que decidiu-se recorrer a este algoritmo implementado pela biblioteca PCL no trabalho desenvolvido.**

**Depois de verificadas as coordenadas máximas e mínimas a considerar para cada *cloud*, a próxima etapa é criar um critério de aceitação para acumular apenas pontos contidos nesse intervalo. Os dois métodos comparados, cujos resultados estão presentes na tabela 5.3, compara o método desenvolvido para este efeito, e ainda um outro método disponível para utilização na biblioteca PCL, métodos que foram devidamente explicados no capítulo 4. Analisando os resultados, podemos constatar que o algoritmo implementado tem melhor *performance* neste tipo de cenários testados. Devido ao resultado obtido, optou-se por este algoritmo implementado, indo de encontro aos resultados abordados no parágrafo anterior, em que não se conseguiu implementar um algoritmo que fosse mais rápido neste análise em tempo real.**

**Os resultados que dizem respeito à segmentação da nuvens de pontos, cujo método foi explicado no capítulo 4, tem como objetivo filtrar tudo o que é pontos correspondentes a solo, através de uma comparação inicial com a altura de referência do UGV, e atualizando sempre a altura máxima a considerar, por forma a que pequenas elevações no cenário pos-**

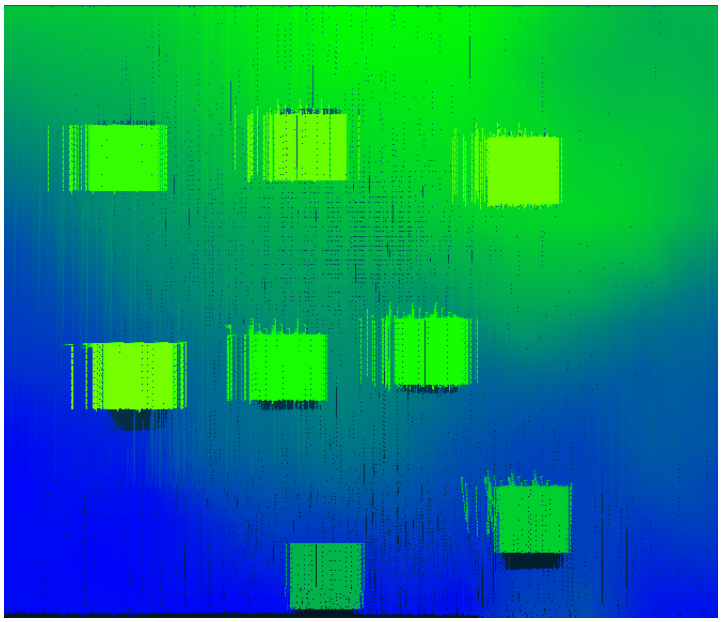


**Figura 5.10:** *Filtro de voxel com leaf de 2 cm (cenário 1)*

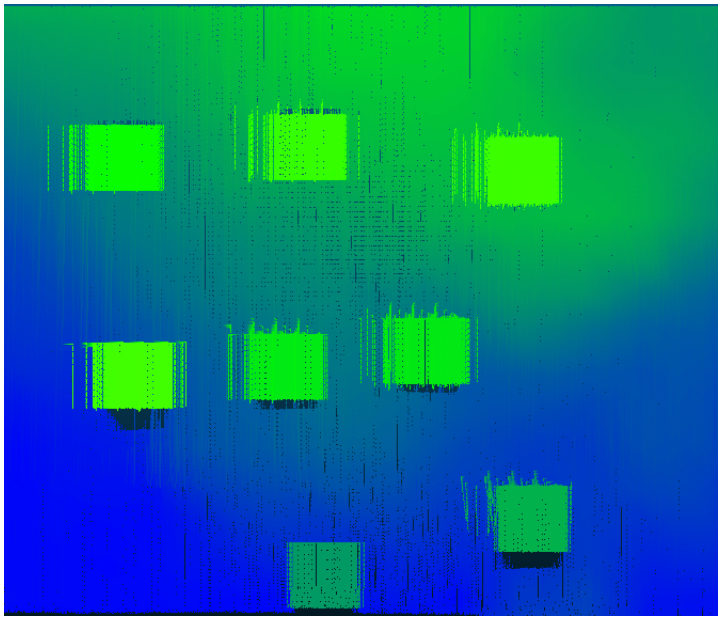


**Figura 5.11:** *Filtro de voxel com leaf de 3 cm ((cenário 1)*

**sam ser consideradas na zona de inclusão, e ao mesmo tempo excluindo pontos que correspondam a uma mudança abrupta de altura. Com este método, todos os pontos nesta zona de inclusão, cujo ponto máximo em altura a considerar é no máximo mais 10 cm em relação ao ponto máximo atualizado. Este método identifica tudo o que se trata de solo e aloca o**

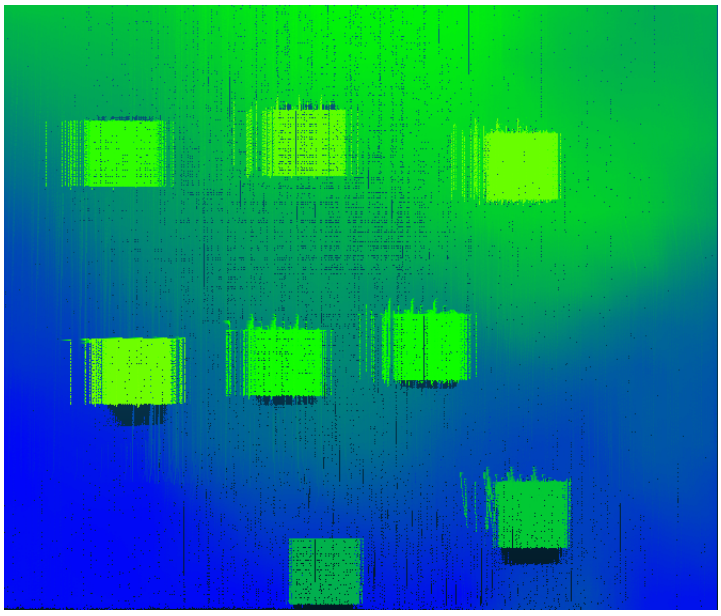


**Figura 5.12:** *Filtro de voxel com leaf de 5 cm (cenário 1)*

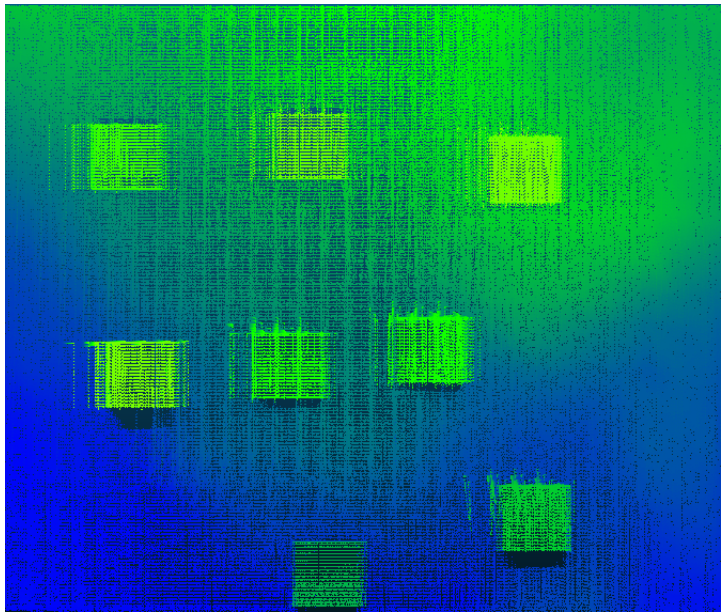


**Figura 5.13:** *Filtro de voxel com leaf de 10 cm (cenário 1)*

**resultado acumulado numa variável à parte. Os resultado obtidos, na figura 5.22, onde a vermelho está considerado o que é caminho proibido, e a verde está representado o caminho que pode ser atravessado. Através de uma análise a esta figura, conclui-se que algumas zonas neste cenário em específico, não foram corretamente consideradas, no entanto, uma**

**grande parte de espaços atravessáveis foram corretamente considerados.**

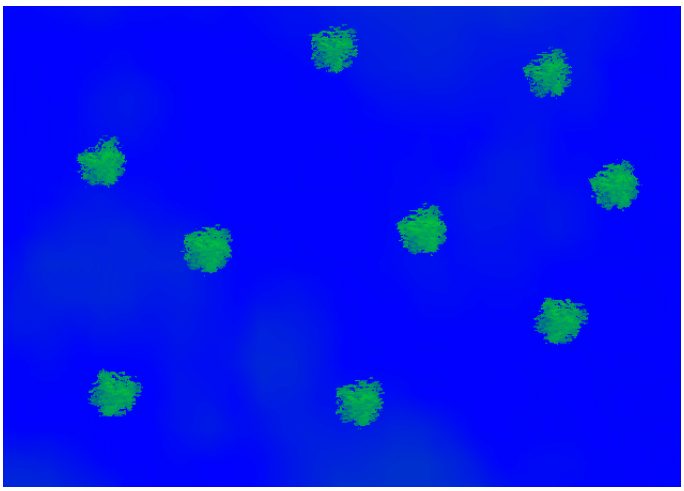


**Figura 5.14:** *Filtro de voxel com leaf de 2 cm (cenário 2)*

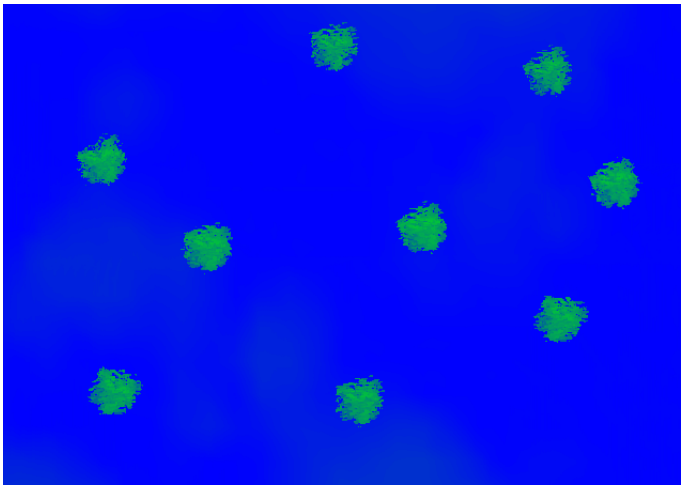


**Figura 5.15:** *Filtro de voxel com leaf de 3 cm (cenário 2)*

**Os mapas de elevações foram obtidos através da aplicação dos algoritmos de segmentação explicados no capítulo anterior, onde se alocou todos os pontos classificados como solo e preencheu-se uma mensagem que descreve um mapa de elevação:** *nav_msgs/OccupancyGrid* **em que guarda o acumular da informação sensorial numa grelha 2D. As probabilidades de ocupação variam entre 0 a 100, em que neste caso se preencheram com o valor 0 todos os pontos que foram incluídos pela segmentação, e com o valor 100, os restantes.**

**Apresenta-se de seguida o resultado do planeamento de caminhos ba-**

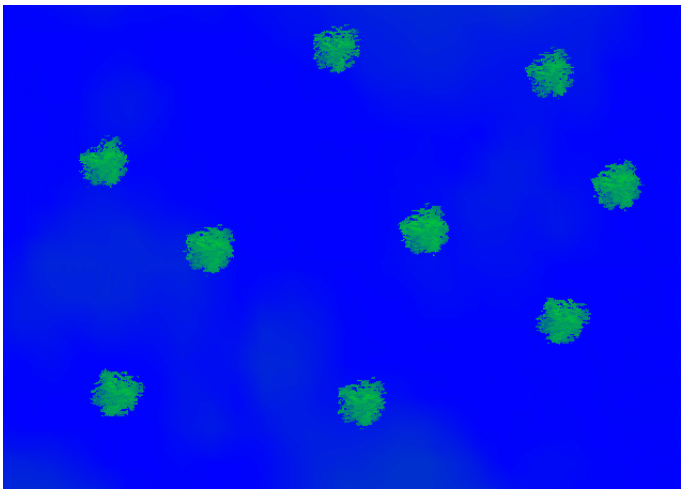


**Figura 5.16:** *Filtro de voxel com leaf de 5 cm (cenário 2)*

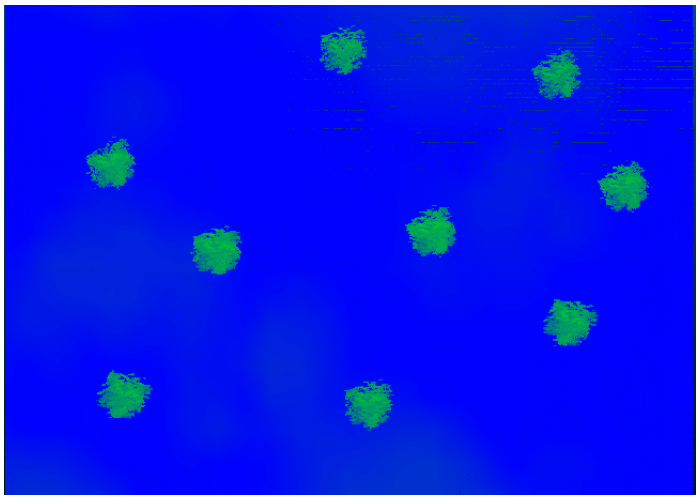


**Figura 5.17:** *Filtro de voxel com leaf de 10 cm (cenário 2)*

**seado num planeador PRM. Para validar a navegação nos cenários apresentados nesta dissertação, inseriu-se no *software* matlab, os ficheiros que correspondem ao acumular da informação segmentada e importaram-se esses dados no mesmo. Os dados correspondem a 4 colunas com informações dos pontos nos três eixos e na última coluna inseriu-se o resultado da ocupação que apenas terá o valor 0 ou 100, como explicado no parágrafo anterior. Com estes ficheiros carregados construiu-se um mapa binário com o espaço ocupado representado a cor preta, por forma a alimentar o planeador PRM, tal como se encontra explicado na docu-**

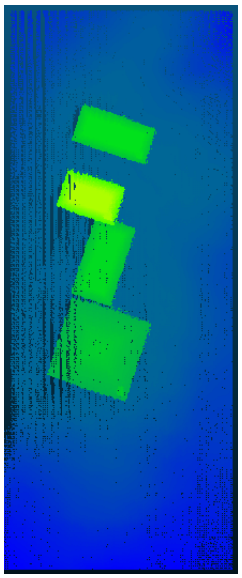


**Figura 5.18:** *Filtro de voxel com leaf de 2 cm (cenário 3)*

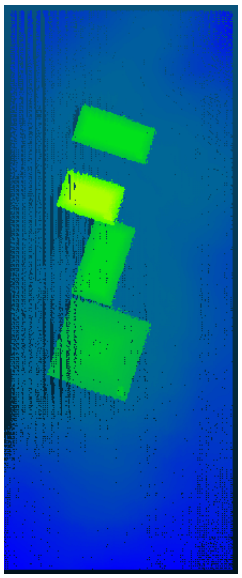


**Figura 5.19:** *Filtro de voxel com leaf de 3 cm (cenário 3)*

**mentação oficial do matlab [2]. Os resultados para os três cenários, com um planeamento de um caminho, cujos pontos de início e fim estão representados nos resultados obtidos, nas figuras 5.25, 5.26 e 5.27.**

[2]https://www.mathworks.com/help/robotics/ug/probabilistic-roadmaps-prm.html

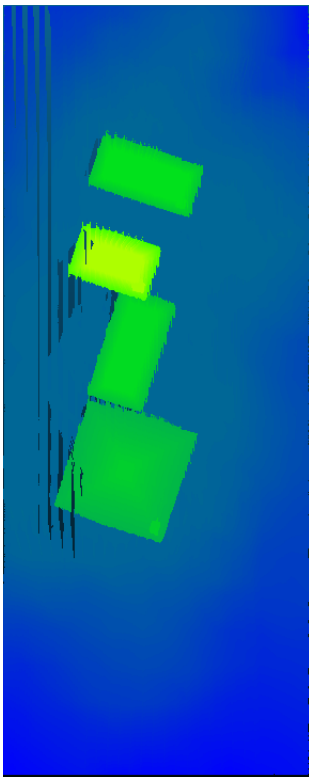


**Figura 5.20:** *Filtro de voxel com leaf de 5 cm (cenário 3)*

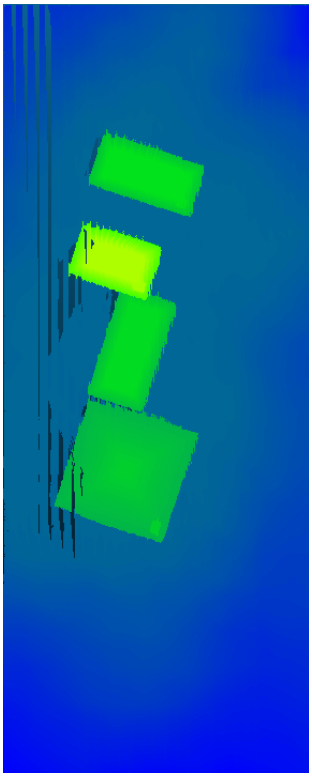


**Figura 5.21:** *Filtro de voxel com leaf de 10 cm (cenário 3)*

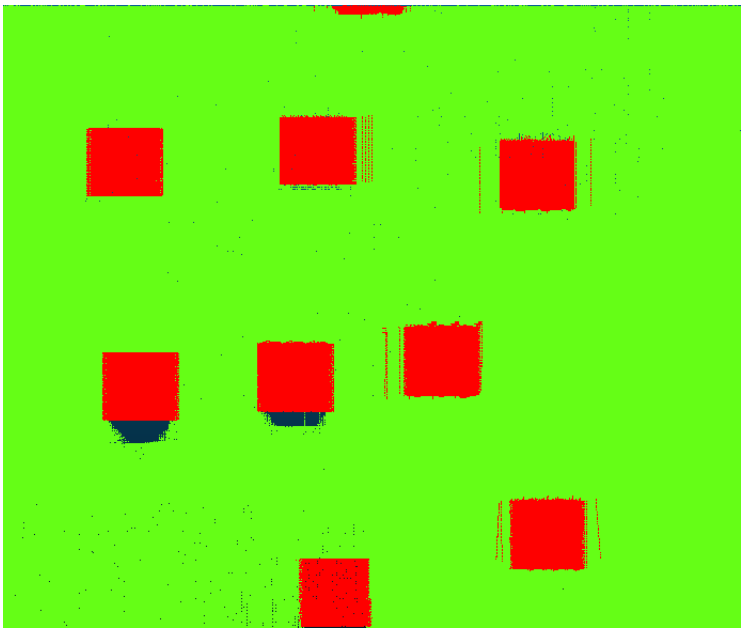


**Figura 5.22:** *Mapa de elevação obtido através da representação do cenário 1 (caixas)*

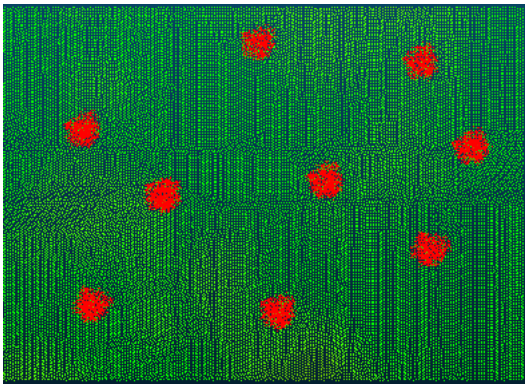


**Figura 5.23:** *Mapa de elevação obtido através da representação do cenário 2 (árvores)*

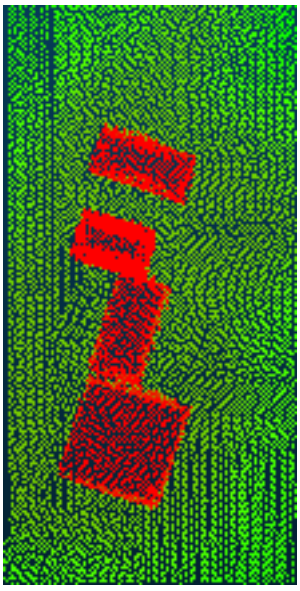


**Figura 5.24:** *Mapa de elevação obtido através da representação do cenário 3 (casas)*

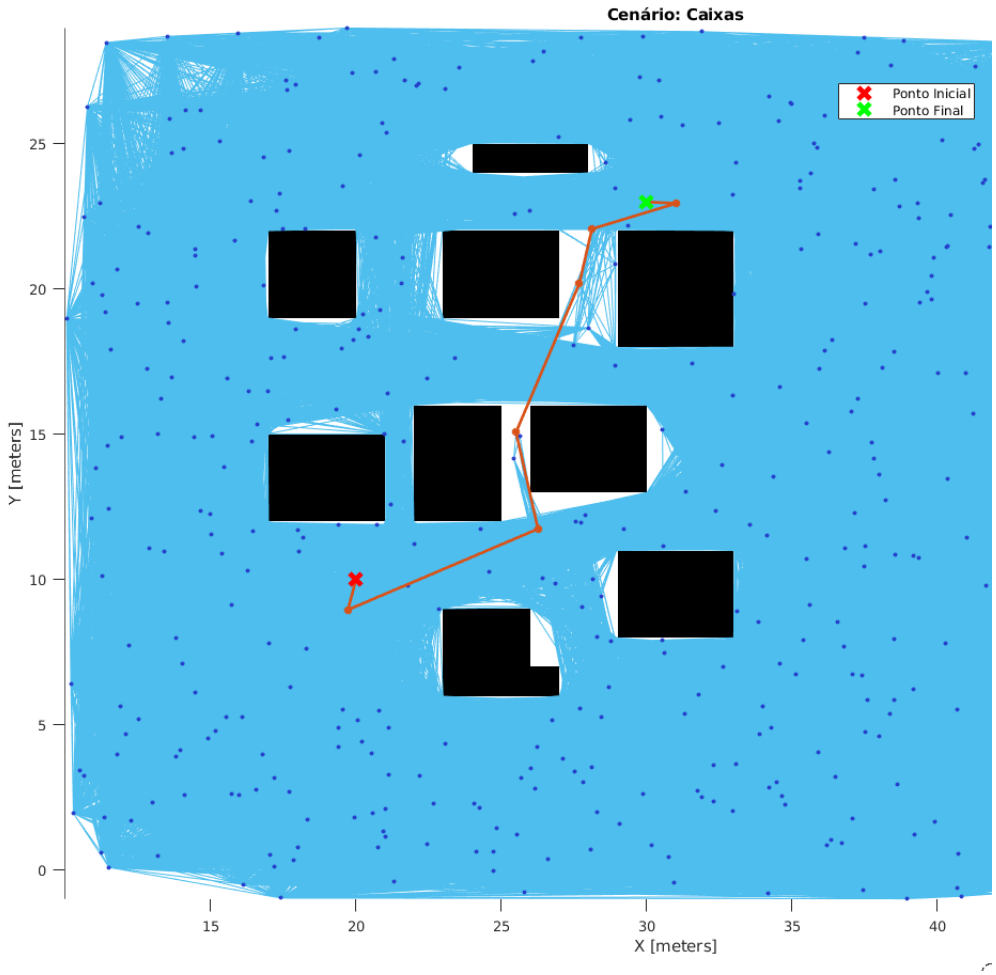



**Figura 5.25:** *Caminho calculado para o cenário 1 (caixas)*

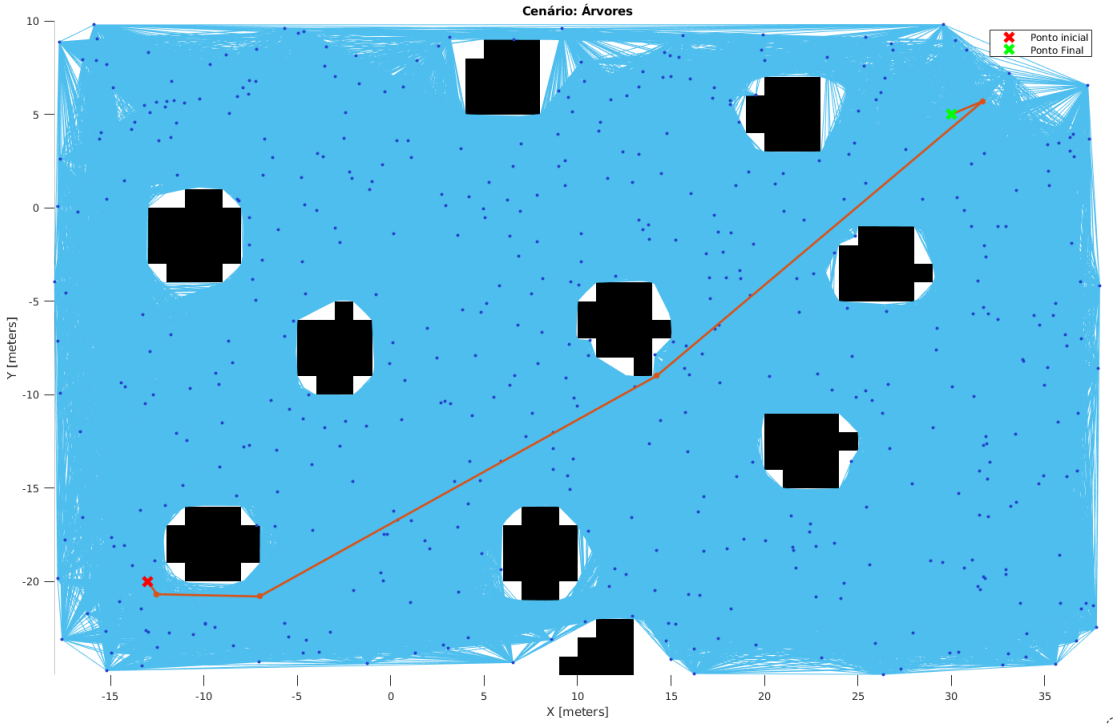



**Figura 5.26:** *Caminho calculado para o cenário 2 (árvores)*

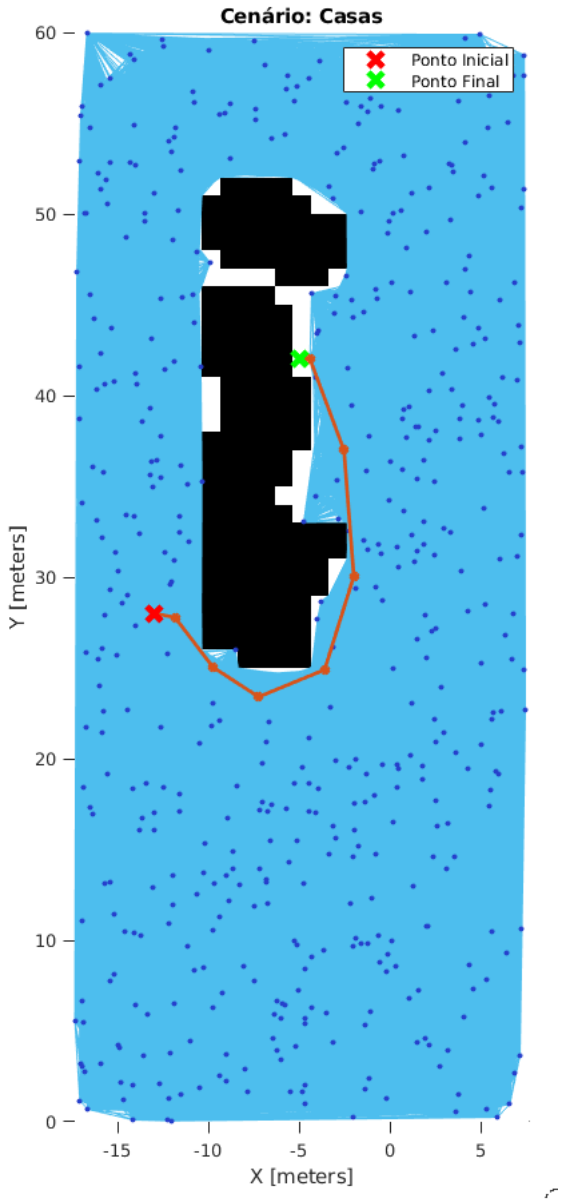



**Figura 5.27:** *Caminho calculado para o cenário 3 (casas)*

# 6

# Conclusões e trabalho futuro

**Relativamente aos objetivos propostos, os resultados finais vão ao encontro do que foi proposto inicialmente. Foram estudados métodos de análise de nuvens de pontos que tentassem melhorar a *performance* global de um conjunto multirobótico. Tal situação abriu a possibilidade de um vasto conjunto de abordagens diferentes, pelo que todo o desenvolvimento foi muito à base da exploração de várias opções diferentes, entre as quais, relativamente ao modo como os dois robôs se iriam sincronizar. Um sincronismo de posições, onde se procurou provar um método de *scan* de uma área específica à volta do UGV, de modo a que de diferentes ângulos se conseguisse representar com a melhor resolução possível todos os objetos e terreno livre nessa área de trabalho. A segmentação foi aplicada com sucesso, tendo-se conseguido marcar o espaço que correspondia a caminho livre com algum sucesso, pelo que a *framework octomap* apenas foi utilizada para acumular informação. A decisão sobre a marcação de espaço livre e ocupado foi efetuada com recurso à biblioteca PCL e também a *Pointclouds* da *framework octomap*, que representa uma estrutura diferente daquela que acumula a informação acumulada. Os tempos obtidos nos algoritmos analisados nos três cenários apresentaram resultados bons para um processamento em tempo real, tanto da biblioteca PCL como os que foram implementados. Para trabalho futuro, é necessário melhorar os mapas de elevação, de modo a que a sua integração em *real***

***time* seja possível.**

**O módulo de *path planning* foi validado através do acumular da informação processada, pelo que, apesar dos resultados obtidos no cálculo dos caminhos serem muito próximos do caminho em que visualmente se perspetiva ser o mais curto, um possível melhoramento neste módulo seria o cálculo de múltiplos caminhos até ao alvo.**

# Bibliografia


[1] L. Cantelli, M. Mangiameli, C. D. Melita, and G. Muscato, "UAV/UGV cooperation for surveying operations in humanitarian demining," in *2013 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, Oct. 2013, pp. 1–6.

[2] S. Behnke, M. Schwarz, T. Rodehutskors, D. Droeschel, M. Schreiber, A. Topelidou-Kyniazopoulou, D. Schwarz, C. Lenz, S. Schuller, J. Razlaw, I. Ivanov, N. Araslanov, and M. Beul, "Team NimbRo Rescue at DARPA Robotics Challenge Finals."

[3] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: an efficient probabilistic 3d mapping framework based on octrees," *Autonomous Robots*, vol. 34, no. 3, pp. 189–206, Apr. 2013.

[4] S. Bartoszyk, P. Kasprzak, and D. Belter, "Terrain-aware motion planning for a walking robot," in *2017 11th International Workshop on Robot Motion and Control (RoMoCo)*, Jul. 2017, pp. 29–34.

[5] M. Velas, M. Spanel, M. Hradis, and A. Herout, "CNN for very fast ground segmentation in velodyne LiDAR data," in *2018 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, Apr. 2018, pp. 97–103.

[6] F. Kallasi, D. L. Rizzini, and S. Caselli, "Fast Keypoint Features From Laser Scanner for Robot Localization and Mapping," *IEEE Robotics and Automation Letters*, vol. 1, no. 1, pp. 176–183, Jan. 2016.

[7] J. R. Kidd, "Performance Evaluation of the Velodyne VLP-16 System for Surface Feature Surveying," Ph.D. dissertation.

[8] "Documentation - Point Cloud Library (PCL)." [Online]. Available: http://www.pointclouds.org/documentation

[9] "K-d tree 2d representation." [Online]. Available: https://opendsa-server.cs.vt.edu/ODSA/Books/Everything/html/_images/KDtree.png

[10] "Puck™." [Online]. Available: https://velodynelidar.com/vlp-16.html

[11] **N. Sharkey, "Saying 'No!' to Lethal Autonomous Targeting," *Journal of Military Ethics*, vol. 9, no. 4, pp. 369–383, 2010.**

[12] **N. Fairfield, G. Kantor, and D. Wettergreen, "Real-Time SLAM with Octree Evidence Grids for Exploration in Underwater Tunnels," *Journal of Field Robotics*, vol. 24, no. 1-2, pp. 03–21, Jan. 2007.**

[13] **M. A. Fischler and R. C. Bolles, "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography," *Commun. ACM*, vol. 24, no. 6, pp. 381–395, Jun. 1981.**

[14] **U. Weiss and P. Biber, "Plant detection and mapping for agricultural robots using a 3d LIDAR sensor," *Robotics and Autonomous Systems*, vol. 59, no. 5, pp. 265 – 273, 2011.**

[15] **M. Popovic, G. Hitz, J. Nieto, I. Sa, R. Siegwart, and E. Galceran, "Online informative path planning for active classification using UAVs." IEEE, May 2017, pp. 5753–5758.**

[16] **J. Wilhelms and A. Van Gelder, "Octrees for Faster Isosurface Generation," *ACM Trans. Graph.*, vol. 11, no. 3, pp. 201–227, Jul. 1992.**

[17] **A.-V. Vo, L. Truong-Hong, D. F. Laefer, and M. Bertolotto, "Octree-based region growing for point cloud segmentation," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 104, pp. 88–100, Jun. 2015.**

[18] **G. Echeverria, N. Lassabe, A. Degroote, and S. Lemaignan, "Modular open robots simulation engine: MORSE," in *2011 IEEE International Conference on Robotics and Automation*. Shanghai, China: IEEE, May 2011, pp. 46–51.**

[19] **S. Vasudevan, F. Ramos, E. Nettleton, and H. Durrant-Whyte, "Gaussian process modeling of large-scale terrain," *Journal of Field Robotics*, vol. 26, no. 10, pp. 812–840, 2009.**

[20] **G. Forlani, C. Nardinocchi, M. Scaioni, and P. Zingaretti, "Complete classification of raw LIDAR data and 3d reconstruction of buildings," *Pattern Analysis and Applications*, vol. 8, no. 4, pp. 357–374, Feb. 2006.**

[21] **T. Andre and C. Bettstetter, "Collaboration in Multi-Robot Exploration: To Meet or not to Meet?" *Journal of Intelligent & Robotic Systems*, vol. 82, no. 2, pp. 325–337, May 2016.**

[22] **D. Belter, P. Łabecki, and P. Skrzypczyński, "Adaptive Motion Planning for Autonomous Rough Terrain Traversal with a Walking Robot," *Journal of Field Robotics*, vol. 33, no. 3, pp. 337–370.**

[23] **P. Fankhauser and M. Hutter, "A Universal Grid Map Library: Implementation and Use Case for Rough Terrain Navigation," in *Robot Operating System (ROS): The Complete Reference (Volume 1)*, ser. Studies in Computational Intelligence, A. Koubaa, Ed. Cham: Springer International Publishing, 2016, pp. 99–120.**

[24] **"2d Octree representation." [Online]. Available: https://www.i-programmer.info/images/stories/Core/Theory/QuadTrees/2DOct.png**

[25] **E. Lachat, H. Macher, T. Landes, and P. Grussenmeyer, "Assessment and Calibration of a RGB-D Camera (Kinect v2 Sensor) Towards a Potential Use for Close-Range 3d Modeling," *Remote Sensing*, vol. 7, no. 10, pp. 13 070–13 097, 2015. [Online]. Available: https://www.mdpi.com/2072-4292/7/10/13070**

[26] **C. Marques, J. Cristoao, P. Lima, J. Frazao, I. Ribeiro, and R. Ventura, "RAPOSA: Semi-Autonomous Robot for Rescue Operations," in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct. 2006, pp. 3988–3993.**

[27] **E. P. Baltsavias, "Airborne laser scanning: basic relations and formulas," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 54, no. 2, pp. 199 – 214, 1999. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0924271699000155**

[28] **J. Wojtanowski, M. Zygmunt, M. Kaszczuk, Z. Mierczyk, and M. Muzal, "Comparison of 905 nm and 1550 nm semiconductor laser rangefinders' performance deterioration due to adverse environmental conditions," *Opto-Electronics Review*, vol. 22, 2014.**

[29] **F. Moosmann, O. Pink, and C. Stiller, "Segmentation of 3d lidar data in non-flat urban environments using a local convexity criterion," in *2009 IEEE Intelligent Vehicles Symposium*, Jun. 2009, pp. 215–220.**

[30] **Z. Luo, M. V. Mohrenschildt, and S. Habibi, "A Probability Occupancy Grid Based Approach for Real-Time LiDAR Ground Segmentation," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–13, 2019.**

[31] **a. B. Jaggi and B. Palcic, "Hough spectrum and geometric texture feature analysis," in *Proceedings of the 12th IAPR International Conference on Pattern Recognition, Vol. 3 - Conference C: Signal Processing (Cat. No.94CH3440-5)*, vol. 2, Oct. 1994, pp. 583–585 vol.2.**

[32] **R. Dubé, A. Gawel, H. Sommer, J. Nieto, R. Siegwart, and C. Cadena, "An online multi-robot SLAM system for 3d LiDARs," in *2017***

***IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2017, pp. 1004–1011.**

**[33] D. Zermas, I. Izzat, and N. Papanikolopoulos, "Fast segmentation of 3d point clouds: A paradigm on LiDAR data for autonomous vehicle applications," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, May 2017, pp. 5067–5073.**

**[34] K. Miadlicki, M. Pajor, and M. Saków, "Ground plane estimation from sparse LIDAR data for loader crane sensor fusion system," in *2017 22nd International Conference on Methods and Models in Automation and Robotics (MMAR)*, Aug. 2017, pp. 717–722.**

**[35] M. Corah, C. O'Meadhra, K. Goel, and N. Michael, "Communication-Efficient Planning and Mapping for Multi-Robot Exploration in Large Environments," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1715–1721, Apr. 2019.**

**[36] A. Kelly, O. Amidi, M. Bode, M. Happold, H. Herman, T. Pilarski, P. Rander, A. Stentz, N. Vallidis, and R. Warner, "Toward Reliable Off Road Autonomous Vehicles Operating in Challenging Environments," in *International Journal of Robotics Research*, vol. 25, 2004, pp. 599–608.**

**[37] M. Faessler, E. Mueggler, K. Schwabe, and D. Scaramuzza, "A monocular pose estimation system based on infrared LEDs," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, May 2014, pp. 907–913.**

**[38] V. Djapic and D. Nad, "Using collaborative Autonomous Vehicles in Mine Countermeasures," in *OCEANS'10 IEEE SYDNEY*, May 2010, pp. 1–7.**

**[39] R. Käslin, P. Fankhauser, E. Stumm, Z. Taylor, E. Mueggler, J. Delmerico, D. Scaramuzza, R. Siegwart, and M. Hutter, "Collaborative localization of aerial and ground robots through elevation maps," in *2016 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, Oct. 2016, pp. 284–290.**

**[40] T. Whelan, H. Johannsson, M. Kaess, J. J. Leonard, and J. McDonald, "Robust real-time visual odometry for dense RGB-D mapping," in *2013 IEEE International Conference on Robotics and Automation*, May 2013, pp. 5724–5731.**

**[41] A. Sampath and J. Shan, "Segmentation and Reconstruction of Polyhedral Building Roofs From Aerial Lidar Point Clouds," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 48, no. 3, pp. 1554–1567, Mar. 2010.**

[42] **T. M. Bonanni, B. D. Corte, and G. Grisetti, "3-D Map Merging on Pose Graphs," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 1031–1038, Apr. 2017.**

[43] **C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, "Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age," *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1309–1332, Dec. 2016.**

[44] **F. M. Noori, D. Portugal, R. P. Rocha, and M. S. Couceiro, "On 3d simulators for multi-robot systems in ROS: MORSE or Gazebo?" in *2017 IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*, Oct. 2017, pp. 19–24.**

[45] **I. Arvanitakis and A. Tzes, "Collaborative mapping and navigation for a mobile robot swarm," in *2017 25th Mediterranean Conference on Control and Automation (MED)*, Jul. 2017, pp. 696–700.**

[46] **P. Fankhauser, M. Bloesch, and M. Hutter, "Probabilistic Terrain Mapping for Mobile Robots with Uncertain Localization," vol. 3, pp. 3019–3026, 2018.**

[47] **S. Lynen, T. Sattler, M. Bosse, J. A. Hesch, M. Pollefeys, and R. Siegwart, "Get Out of My Lab: Large-scale, Real-Time Visual-Inertial Localization," in *Robotics: Science and Systems*, 2015.**

[48] **D. Belter and P. Skrzypczyński, "Feature-based RGB-D SLAM with Dense Terrain Mapping for a Walking Robot."**

[49] **P. Fankhauser, M. Bjelonic, C. D. Bellicoso, T. Miki, and M. Hutter, "Robust Rough-Terrain Locomotion with a Quadrupedal Robot," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, May 2018, pp. 1–8.**

[50] **P. Fankhauser, M. Bloesch, P. Krüsi, R. Diethelm, M. Wermelinger, T. Schneider, M. Dymczyk, M. Hutter, and R. Siegwart, "Collaborative navigation for flying and walking robots," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2016, pp. 2859–2866.**

[51] **"UNEXMIN - underwater explorer for flooded mines." [Online]. Available: https://www.unexmin.eu/**

[52] **R. Lavrenov and A. Zakiev, "Tool for 3d Gazebo Map Construction from Arbitrary Images and Laser Scans," in *2017 10th International Conference on Developments in eSystems Engineering (DeSE)*, Jun. 2017, pp. 256–261.**

[53] **R. L. Klaser, F. S. Osório, and D. F. Wolf, "Simulation of an Autonomous Vehicle with a Vision-Based Navigation System in Unstructured Terrains Using OctoMap," in *2013 III Brazilian Symposium on Computing Systems Engineering*, Dec. 2013, pp. 177–178.**

[54] **J. Baert, A. Lagae, and P. Dutré, "Out-of-Core Construction of Sparse Voxel Octrees," p. 6.**

[55] **B. Douillard, J. Underwood, N. Kuntz, V. Vlaskine, A. Quadros, P. Morton, and A. Frenkel, "On the segmentation of 3d LIDAR point clouds," in *2011 IEEE International Conference on Robotics and Automation*, May 2011, pp. 2798–2805.**

[56] **H. Moravec and A. Elfes, "High resolution maps from wide angle sonar," in *1985 IEEE International Conference on Robotics and Automation Proceedings*, vol. 2, Mar. 1985, pp. 116–121.**

[57] **L. M. F. Christino and F. d. S. Osório, "GPU-Services: Real-Time Processing of 3d Point Clouds for Robotic Systems Using GPUs," in *2015 12th Latin American Robotics Symposium and 2015 3rd Brazilian Symposium on Robotics (LARS-SBR)*, Oct. 2015, pp. 151–156.**

[58] **R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "G2o: A general framework for graph optimization," in *2011 IEEE International Conference on Robotics and Automation*, May 2011, pp. 3607–3613.**

[59] **S. Laine and T. Karras, "Efcient Sparse Voxel Octrees – Analysis, Extensions, and Implementation," p. 30.**

[60] **J. Zhang, S. Liu, B. Gao, and C. Zhong, "An Improvement Algorithm for OctoMap Based on RGB-D SLAM," in *2018 Chinese Control And Decision Conference (CCDC)*, Jun. 2018, pp. 5006–5011.**

[61] **Y. C. Fan, L. J. Zheng, and Y. C. Liu, "3d Environment Measurement and Reconstruction Based on LiDAR," in *2018 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, pp. 1–4.**

[62] **G. K. Rambally and R. S. Rambally, "Octrees and their applications in image processing," in *IEEE Proceedings on Southeastcon*, Apr. 1990, pp. 1116–1120 vol.3.**

[63] **"LSA Home." [Online]. Available: http://www.lsa.isep.ipp.pt/**

[64] **T. Koolen, J. Smith, G. Thomas, S. Bertrand, J. Carff, N. Mertins, D. Stephen, P. Abeles, J. Englsberger, S. McCrory, J. v. Egmond, M. Griffioen, M. Floyd, S. Kobus, N. Manor, S. Alsheikh, D. Duran, L. Bunch, E. Morphis, L. Colasanto, K. L. H. Hoang, B. Layton,**

**P. Neuhaus, M. Johnson, and J. Pratt, "Summary of Team IHMC's virtual robotics challenge entry."**

**[65] J. James, Y. Weng, S. Hart, P. Beeson, and R. Burridge, "Prophetic goal-space planning for human-in-the-loop mobile manipulation."**

**[66] C. G. Atkeson, B. P. W. Babu, N. Banerjee, D. Berenson, C. P. Bove, X. Cui, and M. DeDonato, "What Happened at the DARPA Robotics Challenge, and Why?"**

**[67] ROS, "PCL Tutorials - ROS Wiki." [Online]. Available: http://wiki.ros.org/pcl/Tutorials**

**[68] S. A. R. Kutty and S. Ayyappan, "A novel technique for LiDAR data segmentation and three-dimensional space projection," in *2015 Annual IEEE India Conference (INDICON)*, Dec. 2015, pp. 1–5.**

**[69] D. Fontanelli, L. Ricciato, and S. Soatto, "A Fast RANSAC-Based Registration Algorithm for Accurate Localization in Unknown Environments using LIDAR Measurements," Sep. 2007, pp. 597–602.**

**[70] "Documentation - ROS Wiki." [Online]. Available: http://wiki.ros.org/**

**[71] H. Niigaki, J. Shimamura, and A. Kojima, "Segmentation of 3d Lidar Points Using Extruded Surface of Cross Section," in *2015 International Conference on 3D Vision*, Oct. 2015, pp. 109–117.**

**[72] Z. Chen, B. Xu, and B. Gao, "An Image-Segmentation-Based Urban DTM Generation Method Using Airborne Lidar Data," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 9, no. 1, pp. 496–506, Jan. 2016.**

**[73] S. A. Mumtaz and K. Mooney, "Fusion of high resolution lidar and aerial images for object extraction," in *2008 2nd International Conference on Advances in Space Technologies*, Nov. 2008, pp. 137–142.**

**[74] R. Rusu, "Semantic 3d Object Maps for Everyday Manipulation in Human Living Environments," *KI - Künstliche Intelligenz*, vol. 24, Nov. 2010.**