FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Desenvolvimento de um robô híbrido: Caso de uso florestal

Daniel Filipe Queirós da Silva



Mestrado em Engenharia Eletrotécnica e de Computadores

Orientador: Prof. Vítor Hugo Machado Oliveira Pinto

Orientador: Prof. José Luís Sousa de Magalhães Lima

8 de outubro de 2026

© Daniel Silva, 2026

Desenvolvimento de um robô híbrido: Caso de uso florestal

Daniel Filipe Queirós da Silva

Mestrado em Engenharia Eletrotécnica e de Computadores

Aprovado em provas públicas pelo Júri:

Presidente: Filipe Baptista Neves dos Santos
Arguente: Agostinho Gil Teixeira Lopes
Vogal: Vítor Hugo Machado Oliveira Pinto

8 de outubro de 2026

Resumo

A navegação autónoma em ambientes florestais apresenta desafios complexos decorrentes da natureza não estruturada do meio, da degradação de sinais de posicionamento por satélite (*Global Navigation Satellite System* (GNSS)) sob coberto arbóreo, das variações extremas de iluminação e da presença de vegetação penetrável e de obstáculos rígidos. Os robôs terrestres enfrentam limitações físicas perante desníveis e vegetação densa, enquanto as plataformas aéreas apresentam um consumo energético elevado quando comparadas com a locomoção de solo. As plataformas híbridas terrestre-aéreas surgem como uma solução promissora, combinando a eficiência energética da deslocação em solo com a capacidade de contornar obstáculos por via aérea quando o terreno se torna intransponível.

Esta dissertação apresenta o desenvolvimento e validação experimental em simulação de um sistema de navegação, perceção e decisão para um robô híbrido terrestre-aéreo aplicado ao contexto florestal. A arquitetura proposta integra um pipeline de perceção multimodal (*Light Detection and Ranging* (LiDAR) e câmara) orientado ao mapeamento geométrico e semântico, distinguindo troncos, rochas, arbustos e troncos caídos da vegetação penetrável. Para garantir a estimativa contínua de pose, desenvolveu-se um sistema de *Simultaneous Localization and Mapping* (SLAM) baseado em grafos de fatores com extração dessas quatro classes como *landmarks*, consolidadas ao longo do tempo (*Tree-SLAM*), complementado por módulos de relocalização e fecho de laço baseados em geometria relacional entre árvores.

A representação do ambiente assenta numa malha de terreno 2.5D com estrutura persistente em *tiles* e num modelo de custo de traversabilidade contínuo. A camada de planeamento e decisão híbrida articula uma procura incremental (D^* Lite) para rotas globais, um único controlador (*Regulated Pure Pursuit*) que acopla o seguimento de trajetória ao evitamento reativo de obstáculos em solo, um piloto automático dedicado (*ArduPilot*) para o controlo de voo, e um módulo de decisão responsável pela comutação entre o modo de locomoção terrestre e aéreo. Para a validação, desenvolveu-se um gerador procedimental de mundos florestais (*ForestGen*) com distribuição natural de povoamentos, operado em Gazebo através de uma ferramenta *Command Line Interface* (CLI) dedicada (*forest*). Os resultados obtidos mostram, entre outros, uma redução do erro de trajetória entre três e quatro vezes com a integração do *Tree-SLAM* face a um estimador contínuo isolado, e confirmam a fiabilidade mecânica da transição entre modos de locomoção. A validação conjunta de todas as camadas foi também realizada numa missão completa em simulação, com sucesso em três cenários distintos, uma missão em ciclo fechado, uma travessia direta de ponta a ponta e um salto aéreo desencadeado por um obstáculo real.

Palavras-Chave: Robótica Florestal, Robótica Híbrida Terrestre-Aérea, Perceção Multimodal, SLAM por *Landmarks*, Traversabilidade, Planeamento e Decisão Híbrida.

Abstract

Autonomous navigation in forest environments poses severe challenges due to the unstructured nature of the terrain, GNSS signal degradation under dense canopy, extreme illumination variations, and the complex distinction between penetrable vegetation and rigid obstacles. Ground robots often encounter impassable physical barriers, whereas aerial platforms suffer from high energy consumption compared to ground locomotion. Hybrid ground-aerial platforms offer a compelling solution by combining ground-based energy efficiency with aerial maneuverability to bypass untraversable terrain.

This dissertation presents the design, integration, and experimental validation in simulation of an autonomous navigation, perception, and decision framework for a hybrid ground-aerial robot tailored to forestry applications. The proposed architecture incorporates a multimodal perception pipeline combining LiDAR and camera for geometric and semantic mapping, distinguishing trunks, rocks, bushes and fallen logs from penetrable vegetation. To achieve robust state estimation without reliable GNSS, a factor-graph SLAM framework (*Tree-SLAM*) is implemented, extracting these four classes as *landmarks* consolidated over time, with data association, relocalization and loop-closure based on relational tree geometry.

Environment representation relies on a 2.5D elevation grid with tiled persistent memory and a continuous traversability cost model. The planning and hybrid decision layer combines incremental search (D^* Lite) for global routing, a single controller (*Regulated Pure Pursuit*) that couples trajectory tracking with reactive ground obstacle avoidance, a dedicated autopilot (*ArduPilot*) for flight control, and a decision module responsible for switching between ground and aerial locomotion. Validation was supported by a custom procedural world generator (*ForestGen*) with natural stand distribution, operated in Gazebo through a dedicated CLI tool (*forest*). Results show, among others, a three- to four-fold reduction in trajectory error with *Tree-SLAM* relative to an isolated continuous estimator, and confirm the mechanical reliability of the locomotion-mode transition. Joint validation of all layers was also carried out in a complete simulated mission, succeeding across three distinct scenarios, a closed-loop mission, a direct end-to-end traversal, and an aerial hop triggered by a real obstacle.

Keywords: Forestry Robotics, Hybrid Ground-Aerial Robotics, Multimodal Perception, Landmark SLAM, Traversability, Hybrid Planning and Decision-Making.

Objetivos de Desenvolvimento Sustentável das Nações Unidas

O trabalho desenvolvido nesta dissertação insere-se no quadro dos Objetivos de Desenvolvimento Sustentável (ODS) das Nações Unidas, contribuindo para a transição digital e tecnológica na gestão e preservação dos recursos florestais. A Tabela seguinte identifica os ODS e as metas específicas a que o trabalho se associa, a contribuição concreta do sistema desenvolvido para cada uma e os indicadores de desempenho medidos ao longo desta dissertação que sustentam essa contribuição.

ODS	Meta	Contribuição	Indicadores de desempenho e métricas
9	9.5	Desenvolvimento de uma arquitetura de perceção multimodal, localização e mapeamento por *landmarks* e decisão híbrida terrestre-aérea para robótica de campo, um domínio de aplicação pouco coberto pela robótica autónoma convencional.	Redução do erro de trajetória entre 3,8 e 4,2 vezes com o *Tree-SLAM* face a um estimador contínuo isolado; fiabilidade de 100% (30/30 ciclos) na transição mecânica entre os modos de locomoção terrestre e aéreo.
15	15.2	Deteção, classificação e mapeamento não destrutivo de elementos estruturais da floresta (troncos, rochas, arbustos, troncos caídos), com aplicação direta ao inventário florestal e à deteção de sinais de gestão insustentável ou de desflorestação.	Classificação por LiDAR de árvore e rocha com cerca de 85% de acerto em banco de ensaio offline; *mAP@50* de 0,79 no detetor de câmara; cobertura de 58,6% do mapa de *landmarks* face ao *ground-truth* do mundo simulado.

Agradecimentos

Gostaria de começar por expressar o meu sincero agradecimento ao meu orientador, Professor Vítor Hugo Pinto, pela oportunidade de desenvolver esta dissertação e pela proposta do tema, que me permitiu explorar uma área de grande interesse e relevância tecnológica. Agradeço também a sua constante disponibilidade, simpatia e proximidade ao longo deste percurso, bem como todos os conselhos e incentivo que sempre me transmitiu.

À minha família, o meu porto seguro, deixo um agradecimento muito especial. Obrigado pelo apoio incondicional, pelos valores que me transmitiram, pela educação que me proporcionaram e por acreditarem sempre em mim. Sem o vosso incentivo e confiança, este percurso não teria sido possível.

Aos meus amigos mais próximos, obrigado pela amizade e pela presença constante ao longo destes anos. Apesar de, nos últimos meses, a vida nos ter levado por caminhos mais ocupados e de nem sempre ter sido possível estarmos juntos, cada momento partilhado continuou a ser sinónimo de boa disposição, companheirismo e força para enfrentar os desafios desta etapa.

Quero também agradecer a todos os amigos que fiz ao longo da faculdade, pelas conversas e por todas as experiências que contribuíram para tornar este percurso académico tão enriquecedor. Um agradecimento muito especial ao Pedro Lopes e ao Afonso Mateus, por todo o caminho que percorremos juntos. As competições em que participámos, as noites mal dormidas, as aventuras que vivemos, os desafios que ultrapassámos e o espírito de equipa que sempre nos uniu marcaram profundamente a minha experiência universitária e contribuíram para o meu crescimento, tanto a nível pessoal como profissional.

Agradeço ainda aos Professores António Paulo Moreira e Paulo Costa pela oportunidade de integrar a equipa de robótica e por todo o conhecimento, experiência e ensinamentos que me transmitiram ao longo destes anos.

À minha namorada, deixo o meu mais profundo e sentido agradecimento. Obrigado por seres a pessoa que és e por todo o apoio, carinho e compreensão que me tens dado ao longo destes últimos tempos. Obrigado por acreditares sempre em mim, por me ajudares a crescer enquanto pessoa e por continuares a inspirar-me a ser melhor a cada dia. Nesta fase particularmente exigente, a tua paciência e compreensão foram essenciais, especialmente durante as muitas horas e dias em que o trabalho nos manteve longe um do outro. Obrigado por nunca deixares que isso abalasse aquilo que construímos e por transformares cada momento que passámos juntos numa oportunidade para recuperar energias, espairecer a mente e recordar-me da importância de manter o equilíbrio. A tua presença tornou este percurso muito mais leve e esta conquista também é, em grande parte, tua.

Por fim, agradeço a todas as pessoas que, de alguma forma, contribuíram para este percurso e para a concretização desta dissertação. Ainda que não sejam aqui mencionadas individualmente, o vosso apoio, incentivo e confiança tiveram um papel importante nesta etapa da minha vida.

A todos, o meu mais sincero obrigado.

DanielSilva

"The ones who are crazy enough to think that they can change the world, are the ones who do."

Steve Jobs

x

Conteúdo

1 Introdução **1**
- 1.1 Contexto, Motivação e Problema . . . 1
- 1.2 Objetivos . . . 2
- 1.3 Contribuições . . . 3
- 1.4 Estrutura do Documento . . . 3

2 Fundamentos e Revisão Bibliográfica **5**
- 2.1 Robótica em Ambientes Florestais e Off-Road . . . 5
- 2.2 Robótica Híbrida Terrestre-Aérea . . . 6
  - 2.2.1 Propulsão separada . . . 6
  - 2.2.2 Propulsão partilhada . . . 7
- 2.3 Perceção . . . 7
  - 2.3.1 Sensores e Modalidades . . . 8
  - 2.3.2 Perceção Geométrica . . . 8
  - 2.3.3 Perceção Semântica . . . 10
  - 2.3.4 Fusão Multimodal . . . 11
- 2.4 Localização e Mapeamento Simultâneos . . . 12
  - 2.4.1 Filtros e Grafos de Fatores . . . 13
  - 2.4.2 Do Mapa Denso ao Mapa de Landmarks . . . 14
  - 2.4.3 SLAM por *landmarks* em ambiente florestal . . . 15
  - 2.4.4 Relocalização e fecho de loop em floresta . . . 17
- 2.5 Representações do Ambiente e Traversabilidade . . . 18
  - 2.5.1 De ocupação binária a mapas de risco . . . 18
  - 2.5.2 Traversabilidade aparente e relativa . . . 20
  - 2.5.3 Representações 2.5D e volumétricas . . . 21
- 2.6 Planeamento de Trajetória e Decisão de Locomoção . . . 22
  - 2.6.1 Planeamento global em ambientes parcialmente conhecidos . . . 23
  - 2.6.2 Planeamento local e evitamento . . . 24
  - 2.6.3 Decisão híbrida . . . 26
- 2.7 Controlo e Seguimento de Trajetória . . . 27
  - 2.7.1 Controlo de seguimento em solo . . . 27
  - 2.7.2 Controlo aéreo . . . 29
- 2.8 Simulação, Geração Procedimental e Distribuição Natural . . . 30
  - 2.8.1 Simulação em Robótica de Campo . . . 30
  - 2.8.2 Geração Procedimental de Terreno . . . 31
  - 2.8.3 Distribuição Natural de Obstáculos em Floresta . . . 33

3 Desenvolvimento Experimental **35**
3.1 Idealização do Robô Híbrido . . . 37
3.1.1 Máquina de estados da transição de modo . . . 39
3.2 Perceção . . . 40
3.2.1 LiDAR . . . 41
3.2.2 Câmara . . . 46
3.2.3 Fusão . . . 52
3.3 Localização e Mapeamento Simultâneos . . . 53
3.3.1 Estimação de estado contínua . . . 54
3.3.2 Associação de dados . . . 54
3.3.3 Nascimento e consolidação dos *landmarks* ao longo do tempo . . . 56
3.3.4 O grafo de fatores e a otimização . . . 58
3.3.5 Relocalização e autoridade de modo . . . 60
3.4 Representação do Ambiente e Traversabilidade . . . 60
3.4.1 Arquitetura da camada . . . 61
3.4.2 A malha de terreno 2.5D . . . 62
3.4.3 Terreno persistente em *tiles* . . . 63
3.4.4 O modelo de custo . . . 65
3.4.5 Integração com o planeamento . . . 69
3.5 Planeamento e Decisão . . . 70
3.5.1 Planeamento global em ambientes parcialmente conhecidos . . . 71
3.5.2 Planeamento local e evitamento de obstáculos . . . 72
3.5.3 Decisão híbrida . . . 73
3.6 Controlo . . . 75
3.6.1 Controlo de voo por autopiloto dedicado . . . 75
3.7 Simulação, Geração Procedimental e Distribuição Natural . . . 76
3.7.1 Escolha do Gazebo . . . 76
3.7.2 O CLI `forest` . . . 77
3.7.3 Evolução da geração procedimental de mundos . . . 78
3.7.4 Distribuição natural dos obstáculos . . . 82

4 Resultados Experimentais **85**
4.1 Robótica Híbrida Terrestre-Aérea . . . 85
4.1.1 Transição observada em simulação . . . 86
4.1.2 Fiabilidade da transição . . . 86
4.1.3 Análise crítica . . . 87
4.2 Perceção . . . 88
4.2.1 LiDAR . . . 88
4.2.2 Câmara . . . 91
4.2.3 Deteção do LiDAR ao vivo, nos três mundos . . . 93
4.2.4 Análise crítica . . . 99
4.3 Localização e Mapeamento Simultâneos . . . 100
4.3.1 Erro de trajetória, Tree-SLAM contra estimador contínuo isolado . . . 101
4.3.2 Fidelidade do mapa de *landmarks* . . . 103
4.3.3 Contributo do alinhamento local e do fecho de laço . . . 105
4.3.4 Parametrização do raio do alinhamento local . . . 105
4.3.5 Generalização a outros cenários . . . 107
4.3.6 Análise crítica . . . 108
4.4 Representação do Ambiente e Traversabilidade . . . 109

4.4.1 Validação do modelo de custo . . . 109
4.4.2 Consistência sob deriva, ruído e escolha de fórmula de custo . . . 110
4.5 Planeamento e Decisão . . . 113
4.5.1 Planeamento global em ambientes parcialmente conhecidos . . . 113
4.5.2 Planeamento local e evitamento de obstáculos . . . 116
4.5.3 Decisão híbrida . . . 117
4.5.4 Análise crítica . . . 118
4.6 Controlo . . . 118
4.6.1 Precisão de seguimento em solo . . . 118
4.6.2 Recuperação de perturbação . . . 120
4.6.3 Estabilidade em *hover* sob autopiloto dedicado . . . 121
4.6.4 Precisão de aterragem . . . 122
4.6.5 Seguimento de trajetória em voo GUIDED . . . 123
4.6.6 Análise crítica . . . 124
4.7 Simulação de Missão Completa em Pilha Integrada . . . 125
4.7.1 Missão em ciclo fechado . . . 125
4.7.2 Travessia direta ponta a ponta . . . 128
4.7.3 Salto sobre obstáculo real durante uma missão . . . 130
4.7.4 Análise crítica . . . 132

5 Conclusão e Trabalhos Futuros **135**
5.1 Síntese do Trabalho Realizado . . . 135
5.2 Limitações . . . 137
5.3 Trabalhos Futuros . . . 138

Referências **141**

Lista de Figuras

2.1 Exemplos das duas famílias de robôs híbridos terrestre-aéreos. (a) HUUVER [12], de propulsão separada, com lagartas para o solo e hélices dedicadas para o voo. (b) HyTAQ [13], de propulsão partilhada, em que os mesmos rotores impulsionam o voo e o rolamento de uma gaiola cilíndrica. (c) ATMO [14], de propulsão partilhada com transição morfológica em pleno voo, aqui em configuração de voo. 6

2.2 *Pipeline* típico da perceção geométrica em ambiente florestal, da nuvem de pontos bruta aos parâmetros físicos de cada objeto. 9

2.3 As três vias da perceção semântica sobre a mesma cena. Adaptado do conjunto de dados RELLIS-3D [21]. 11

2.4 Taxonomia geral de fusão sensorial câmara-LiDAR, fusão forte vs. fusão fraca. Fonte: Huang *et al.* [22]. 12

2.5 As duas famílias de SLAM, baseada em filtro (à esquerda) e baseada em grafo de fatores (à direita). 13

2.6 Fases típicas de um *pipeline* de SLAM por *landmarks* de troncos em ambiente florestal, a partir das deteções já fornecidas pela perceção. 15

2.7 As duas fases do reconhecimento de local em floresta, da observação atual até à pose no mapa já existente. 17

2.8 (a) Cena real. (b) Grelha de ocupação clássica, $\mathbb{P}(\mathrm{occ})$. (c) Lambda Field, intensidade de colisão Λ. Adaptado de Laconte *et al.* [36]. 19

2.9 Árvore de famílias da estimação de traversabilidade. O nível de topo segue a distinção aparente/relativa de Yoo *et al.* [38], os ramos sob "Aparente"seguem a divisão geometria/aparência/híbrida de Papadakis [39]. 20

2.10 Duas formas de organizar espacialmente o ambiente. (a) adaptada de Fankhauser *et al.* [43]. (b) figura retirada de Hornung *et al.* [44]. 22

2.11 Procura incremental perante o mesmo obstáculo. Em cima o Lifelong Planning A*, medido a partir do início. Em baixo o D* Lite, medido a partir do objetivo. Ilustração própria dos conceitos descritos em Koenig et al. [52] e Koenig e Likhachev [53]. 24

2.12 Como o mesmo obstáculo entra na decisão em cada família de planeamento local. Ilustração própria dos comportamentos descritos em Fox et al. [54], Khatib [55], Williams et al. [56] e Mayne [57]. 26

2.13 Os três níveis de rigor no controlo de seguimento em solo. Diagrama próprio, ilustrativo dos princípios descritos em Coulter [58], Kanayama et al. [60] e na formulação geral de NMPC de seguimento. 28

2.14 A singularidade da representação por ângulos de Euler (bloqueio de cardan). Diagrama próprio. 29

2.15 Cena de floresta gerada pela ferramenta de código aberto de Newlands & Zauner, com árvores individuais por L-systems distribuídas por simulação de ecossistema. Figura retirada do artigo original [75]. 33

3.1 Arquitetura do sistema e fluxo de dados entre as sete camadas. 36
3.2 Robô híbrido idealizado, em modo terrestre (esquerda) e em modo aéreo (direita), com as lagartas rodadas e os pares de hélices coaxiais expostos. 37
3.3 Efeito da inclinação α do LiDAR. A parte da frente do sensor lê o solo à frente do robô, enquanto a parte de trás aponta para as copas e o céu. 38
3.4 Evolução do desenho da lagarta, com (a) o esboço à mão e (b) a modelação tridimensional da mesma. 39
3.5 Diagrama de estados da máquina de transição de modo. 40
3.6 *Pipeline* de perceção geométrica implementado sobre o LiDAR, do frame bruto à deteção. 41
3.7 Interface web de afinação em tempo real, com cada parâmetro do *pipeline* exposto como um controlo deslizante, agrupado por secção. 42
3.8 Efeito imediato, na nuvem classificada em simulação, de mexer num controlo deslizante da interface da Figura 3.7, antes e depois da alteração. 43
3.9 Os quatro objetos estruturais classificados numa mesma cena, tronco, rocha, arbusto e tronco caído. Vista da simulação (esquerda) e vista da nuvem classificada a partir do RViz (direita). 44
3.10 Mecanismo que separa rocha de arbusto no LiDAR. 45
3.11 Ambiguidade entre rocha e tronco caído no LiDAR e a sua resolução pela altura. 46
3.12 Fluxo do rotulador automático. Cada objeto conhecido do simulador é projetado para a imagem como caixa candidata, que passa por um teste de oclusão pelo buffer de profundidade e por um *gate* de movimento antes de entrar no conjunto de dados. 47
3.13 Teste de oclusão pelo buffer de profundidade. A verde estão as caixas aceites e a vermelho as rejeitadas. 47
3.14 Sequência de frames capturados ao longo de um percurso, com as caixas de referência geradas automaticamente para cada objeto visível. 48
3.15 Vista da câmara em cada tipo de mundo de captura. Aparecem pela seguinte ordem: tronco, rocha, arbusto, tronco caído e mundo misto. 49
3.16 Exemplos de deteção do detetor treinado sobre imagens de validação, com a classe e a confiança de cada caixa. 50
3.17 Recalibração offline de um envelope de rotulagem sobre o mesmo frame, antes (esquerda) e depois (direita) de apertar o critério de tamanho, sem reabrir o simulador. 51
3.18 Arquitetura de fusão assimétrica desenhada. 53
3.19 Arquitetura da localização, com o estimador contínuo e o Tree-SLAM. 54
3.20 Ciclo que o Tree-SLAM corre por cada mensagem de *landmarks*. 55
3.21 Porque a associação por constelação substitui o *gate* absoluto. 56
3.22 Vida de um *landmark* cuja classe verdadeira é tronco, tomado aqui como exemplo. 57
3.23 Estimação do diâmetro por acumulação multi-vista. (a) Um só frame vê apenas um arco parcial do tronco, e o ajuste do círculo a esse arco é sensível ao ruído. (b) Poses sucessivas trazem arcos diferentes que, acumulados, cobrem uma fração cada vez maior da circunferência. (c) Quando a cobertura angular acumulada é suficiente, o ajuste do círculo de Taubin estabiliza e o diâmetro deixa de variar entre frames. 58
3.24 Grafo de fatores do Tree-SLAM, com as poses do robô e os *landmarks* como variáveis. 59

3.25 Fluxo de dados da camada de traversabilidade. As entradas são saídas de outras camadas e não sensores em bruto. O produto final é um mapa de custo contínuo que alimenta o planeamento e o controlo. 61
3.26 Malha de terreno 2.5D construída em simulação. A cor indica a altura estimada do solo em cada célula. Os marcadores sobrepostos assinalam *landmarks* já promovidos pelo SLAM, mostrados aqui apenas como referência, pois pertencem a uma camada distinta da mesma grelha e não a esta camada de elevação. 63
3.27 Terreno persistente em *tiles*. (a) A grelha fixa de blocos em coordenadas do mundo, com blocos ativos e inativos, cada um a guardar a sua elevação e variância e ancorado à keyframe ativa no momento em que nasceu. (b) O efeito de um fecho de laço, em que o bloco acompanha a correção da sua keyframe pela mesma transformação rígida, sem reprocessar nenhuma medição. 64
3.28 Grelha de *tiles* em simulação, com os *landmarks* promovidos pelo SLAM e as ligações de vizinhança entre eles sobrepostas à grelha de blocos, para mostrar como o terreno e os marcos partilham a mesma partição do mundo. 65
3.29 Porque a combinação substitui o máximo. Duas células estão à mesma distância de uma árvore, logo com o mesmo custo de objeto, mas uma está em chão plano e a outra numa encosta (a). Com o máximo, ambas ficam com o mesmo custo e o declive desaparece na vizinhança do obstáculo, enquanto a combinação $1-(1-c_t)(1-c_o)$ faz a encosta custar mais (b). 67
3.30 Funções de custo implementadas. À esquerda, o custo de terreno em função da inclinação, que satura no declive máximo admissível. À direita, o perfil radial do custo de um objeto, com o sólido a saturar no valor letal e a vegetação mole a saturar abaixo dele, ambos a decair na zona de inflação. 68
3.31 Mapa de custo final construído em simulação, combinando o custo de terreno, o custo de objetos e a margem de risco. 68
3.32 O que cada via de entrega ao Nav2 preserva, a camada de custo contínua ou a marcação binária. 69
3.33 Arquitetura de planeamento e decisão, entre o gestor de missão e os dois ramos de execução. 70
3.34 Efeito de como o planeador interpreta as células ainda não observadas. Tratadas como letais, todo o espaço por observar fica bloqueado e o planeamento falha logo no arranque da missão, mesmo em terreno aberto (a). Tratadas como livres, o robô planeia para dentro do desconhecido e corrige o caminho à medida que o vai observando (b). 72
3.35 Regra de decisão entre os dois modos de locomoção. 74
3.36 Fronteira de delegação do controlo de voo. O nó desta dissertação envia apenas comandos de alto nível por MAVLink, enquanto o ArduPilot corre as malhas internas de atitude e de taxa angular a cerca de mil ciclos por segundo e calcula a velocidade de cada rotor. 76
3.37 `Tree1`, repositório `forest_gen` da ETH Zurique [79]: (a) a malha visual original, usada tal e qual no ForestGen; (b) o invólucro convexo simplificado sobreposto à árvore, que substitui a malha de colisão original. 79
3.38 Progressão do terreno gerado por `solo_gen.py`: só ruído macro, macro com meso, e macro com meso e micro, para ilustrar o efeito de cada camada isoladamente. 80
3.39 Dois terrenos gerados pelo `solo_gen.py` a partir da mesma configuração de camadas, só com a amplitude do relevo alterada, para mostrar a variação que a mesma técnica consegue produzir. 80

3.40 Distinção de texturas do solo em simulação: verde para relva, castanho para terra solta, cada uma com um coeficiente de atrito próprio. 81
3.41 Comparação do modelo de arbusto antes e depois do refinamento visual, na mesma posição do mundo, mantendo a mesma forma de colisão em ambas as versões. . . 81
3.42 Catálogo dos modelos base usados na geração de mundos, cada categoria isolada sobre fundo neutro, sem sobreposição de copas ou de folhagem: (a) as seis espécies de árvore, (b) as cinco instâncias do modelo de arbusto atual, (c) as três variantes de rocha, (d) o tronco caído. 82
3.43 Comparação de três modelos de distribuição espacial com a mesma densidade total de pontos: Poisson uniforme, Poisson disco isolado (testado e descartado por regularidade excessiva) e o modelo final de agregação com distância mínima. . . 83
3.44 Vista aérea de um mundo gerado com as clareiras assinaladas (centro e raio de cada uma sobrepostos à imagem), mostrando as zonas de densidade de árvores reduzida a zero. 84

4.1 Transição de modo observada em simulação, do modo terrestre ao voo 86
4.2 Distribuição do tempo de permanência em cada estado da máquina, ao longo de trinta ciclos terra-ar-ar-terra. 87
4.3 Isolação de pontos de tronco (fuste) em relação a copa e ramos, uma vista representativa por árvore. 89
4.4 Matriz de confusão do classificador LiDAR, árvore e rocha (fração da classe real em cada classe prevista). 91
4.5 Matriz de confusão normalizada do detetor de câmara, incluindo fundo. 92
4.6 Deteção e classificação por distância à superfície, por classe. 94
4.7 Acerto de classificação em função da densidade de pontos prevista pela geometria, argumento central para a acumulação multi-vista no SLAM. 95
4.8 Classificação da árvore por modelo, evidenciando a bimodalidade escondida pelo agregado. 96
4.9 Complementaridade entre LiDAR e câmara, classificação em função da distância. 98
4.10 Erro médio de posição por volta ao longo de doze voltas, para o Tree-SLAM e para o estimador contínuo isolado. Média de cinco corridas independentes, com a banda a indicar o desvio padrão entre corridas. 102
4.11 Corrida isolada e mais longa, sobreposição das trajetórias (esquerda) e erro de posição ao longo do tempo (direita), Tree-SLAM contra estimador contínuo isolado. 103
4.12 Mapa final de *landmarks* do Tree-SLAM sobreposto à referência do mundo simulado, com a trajetória estimada e a trajetória verdadeira. 104
4.13 Erro mediano de posição em função do raio do alinhamento local. Os pontos escuros indicam as corridas individuais. 106
4.14 Mapa de erro de elevação alinhado, ancoragem a *keyframes* contra grelha ingénua, mundo `forest_realistic_v2_trees_rocks`. 111
4.15 Exemplo de mapa gerado com densidade média de obstáculos ao nível do robô e o caminho encontrado pelo D* Lite entre o início e o objetivo. 114
4.16 Trajetória percorrida num ensaio de replaneamento progressivo, densidade média. 116
4.17 Caminho de referência contra a trajetória real do robô, medida pela pose de *ground-truth*, para as quatro rotas testadas. Desvio máximo indicado no título de cada painel. 119
4.18 Erro lateral ao longo do tempo, a partir de um desvio inicial de 1.0 m, para três valores de *lookahead* do *Regulated Pure Pursuit* (RPP). 120
4.19 Ponto de aterragem comandado contra o ponto de aterragem real, para os sete saltos. 122

4.20 Trajetórias de voo reais (linha cheia) contra a linha reta ideal para cada alvo (tracejado), cinco dos sete saltos. 123
4.21 Trajetória fundida sobreposta ao *ground-truth* e à rota de referência, dez voltas ao mundo de 83 obstáculos. 126
4.22 Erro de localização ao longo da missão, com os quatro episódios de obstáculo fantasma assinalados a roxo. 127
4.23 Vista do RViz no final da décima volta. 128
4.24 Mapa de landmarks, *costmap* de traversabilidade e mapa de incerteza no mesmo instante final da Figura 4.23. 128
4.25 Percurso fundido e *ground-truth* sobre os 350 obstáculos do mundo gerado, spawn à esquerda e objetivo à direita. 129
4.26 Mapa de landmarks no RViz no final da travessia. 130
4.27 Grelha de elevação 2.5D no RViz no mesmo instante da Figura 4.26. 130
4.28 Corrida 1, percurso fundido e *ground-truth* com o ponto exato de descolagem e de aterragem assinalados, lidos diretamente da fase do salto na mesma amostra de telemetria. 131
4.29 Vista do RViz depois da aterragem, com o robô já relocalizado do outro lado da parede de rochas. 132

Lista de Tabelas

2.1 Famílias de sensores consideradas para a perceção em ambiente florestal. 8
2.2 Comparação entre SLAM baseado em filtros e SLAM baseado em grafos de fatores. 14
2.3 Grelha de ocupação binária vs. mapa de risco contínuo. 19
2.4 Comparação entre o PX4 e o ArduPilot, os dois autopilotos *open source* mais usados, com base nos dados de Aliane [69]. 30

3.1 Tecnologia central escolhida em cada camada do sistema. 35
3.2 Assinatura geométrica de cada classe usada pelo classificador. 44
3.3 Mundos de simulação usados na captura do conjunto de dados de câmara. 49
3.4 Modos da máquina de estados e autoridade sobre a transformação `map`→`odom` em cada um. 60
3.5 Comandos principais do CLI `forest` e o que cada um faz. 78

4.1 Fiabilidade da máquina de estados ao longo de trinta ciclos terra-ar-ar-terra. . . . 86
4.2 Tempo de permanência em cada estado da máquina, ao longo de trinta ciclos. . . 87
4.3 Pontos médios selecionados como fuste por árvore e distância 90
4.4 Resultados de deteção por classe (conjunto de validação). 92
4.5 Execução das três corridas de avaliação ao vivo, robô conduzido pela pose *ground-truth*. 93
4.6 Deteção/classificação por distância à superfície, corrida `moderate`. 94
4.7 Classificação em função do número de pontos previsto pela geometria do sensor . 95
4.8 Classificação offline da árvore, discriminada por modelo (12 amostras cada). . . . 96
4.9 Efeito da análise de componentes principais restrita ao núcleo de fuste, mesmas nuvens offline, antes/depois. 97
4.10 Classificação por classe, LiDAR contra câmara, mesma passagem. 98
4.11 Erro de posição ao longo de doze voltas, Tree-SLAM contra estimador contínuo isolado. 101
4.12 Fidelidade do mapa de *landmarks* ao fim de doze voltas, média de cinco corridas independentes. 103
4.13 Contributo do alinhamento local e do fecho de laço, quatro combinações com três corridas cada. Mediana do erro médio de posição sobre seis voltas. 105
4.14 Varrimento do raio do alinhamento local, três corridas por valor, seis voltas cada, com o fecho de laço sempre ligado. 106
4.15 Desempenho em quatro mundos distintos, três corridas por mundo, seis voltas cada, com a mesma rota em todos. 107
4.16 Modelo de custo, versão inicial contra versão corrigida, mesmo cenário. 109
4.17 *Root Mean Square Error* (RMSE) de elevação alinhado, ancoragem a *keyframes* contra grelha ingénua. 110

4.18 RMSE de elevação por configuração de filtragem, regimes de *outliers* severo e moderado. 111
4.19 Custo de terreno, custo de objeto e custo resultante, três células reais dentro do raio duro do tronco. 112
4.20 Custo de terreno, custo de objeto e custo resultante por fórmula, três células reais na zona de inflação com maior divergência entre fórmulas. 113
4.21 Planeamento global em 30 mapas (50 m x 20 m, obstáculos ao nível do robô/DBH), 10 sementes por densidade. 115
4.22 Replaneamento progressivo com revelação parcial do mapa, D* Lite incremental contra resolver do zero. 116
4.23 Política de custo do `hop_decision_node` (réplica exata da fórmula real), 5 cenários offline construídos de propósito. 117
4.24 Erro de seguimento do RPP, agregado pelas 12 corridas e separado por tipo de troço. 119
4.25 Tempo e distância até o erro lateral convergir para menos de 10 cm. 120
4.26 Estabilidade em *hover* sob ArduPilot, altitude alvo de 5 m. 121
4.27 Erro de aterragem por salto. 122
4.28 Desvio da trajetória voada face à linha reta ideal, por salto. 124
4.29 Missão em ciclo fechado, 10 voltas, mundo de referência (83 obstáculos). 126
4.30 Travessia direta, 90 m em linha reta, mundo novo gerado aleatoriamente (350 obstáculos). 129
4.31 Salto sobre parede real de rochas durante uma missão, 3 repetições válidas, mesmo mundo e parâmetros. 131

Lista de Acrónimos

APF	*Artificial Potential Fields*
CLI	*Command Line Interface*
EKF	*Extended Kalman Filter*
FSM	*Finite State Machine*
GNSS	*Global Navigation Satellite System*
GPS	*Global Positioning System*
HITL	*Hardware In The Loop*
IMU	*Inertial Measurement Unit*
LB-NMPC	*Learning-Based Nonlinear Model Predictive Control*
LiDAR	*Light Detection and Ranging*
MPC	*Model Predictive Control*
NMPC	*Nonlinear Model Predictive Control*
ODS	Objetivos de Desenvolvimento Sustentável
PCA	*Principal Component Analysis*
PRM	*Probabilistic Roadmap Method*
RANSAC	*Random Sample Consensus*
RGB	*Red, Green, Blue*
RMSE	*Root Mean Square Error*
ROS 2	*Robot Operating System 2*
RPP	*Regulated Pure Pursuit*
SLAM	*Simultaneous Localization and Mapping*
TF	*Transform* (frames de coordenadas)
UGV	*Unmanned Ground Vehicle*
UAV	*Unmanned Aerial Vehicle*
CSF	*Cloth Simulation Filter*
DWA	*Dynamic Window Approach*
ESDF	*Euclidean Signed Distance Field*
MPPI	*Model Predictive Path Integral*
RRT	*Rapidly-exploring Random Tree*
SITL	*Software In The Loop*

TSDF *Truncated Signed Distance Field*

Capítulo 1

Introdução

1.1 Contexto, Motivação e Problema

A monitorização, gestão e intervenção em ambientes florestais são tarefas cruciais para a sustentabilidade económica e ambiental das áreas rurais e florestais. No entanto, a execução manual destas atividades é frequentemente morosa, dispendiosa e fisicamente exigente, decorrendo em terrenos acidentados e sob condições climatéricas adversas. A introdução da robótica de campo e da autonomia neste domínio surge como uma oportunidade para aumentar a frequência de monitorização, melhorar a precisão do inventário florestal e reduzir os riscos humanos associados a operações em zonas de difícil acesso.

Apesar deste potencial, a navegação autónoma em floresta enfrenta condicionalismos severos que a distinguem claramente de ambientes urbanos ou industriais:

1. **Degradação de Sinais de Posicionamento:** A densidade da copa das árvores atenua o sinal e causa reflexões multipercurso (*multipath*) nos sinais *Global Navigation Satellite System* (GNSS), além de reduzir o número de satélites com potência suficiente para uso, o que inviabiliza o *Global Positioning System* (GPS) convencional para localização fiável e contínua [1, 2, 3].

2. **Complexidade Geométrica e Semântica:** O ambiente combina elementos rígidos (troncos, rochas) com vegetação penetrável e deformável (erva alta, arbustos), pelo que modelos tradicionais de ocupação binária tendem a falhar ao tratar vegetação atravessável como obstáculo intransponível.

3. **Repetitividade Visual e Geométrica:** A semelhança morfológica entre troncos da mesma espécie introduz ambiguidades graves na associação de dados e no fecho de laço em algoritmos de *Simultaneous Localization and Mapping* (SLAM).

4. **Limitações de Locomoção Terrestre vs. Aérea:** Veículos terrestres (*Unmanned Ground Vehicle*s (UGVs)) possuem elevada autonomia energética, mas ficam bloqueados por obstáculos simples como troncos caídos, ou por valas, vegetação densa e declives acentuados.

Por outro lado, veículos aéreos não-tripulados (*Unmanned Aerial Vehicle*s (UAVs)) superam facilmente obstáculos físicos, mas apresentam um consumo energético cinco a sete vezes superior ao da locomoção de solo devido à necessidade constante de sustentação ativa [4, 5].

Para superar esta dicotomia, surge a oportunidade de desenvolver plataformas **híbridas terrestre-aéreas**, capazes de deslocação eficiente no solo e de efetuar saltos ou voos pontuais apenas quando o terreno terrestre se torna intransponível. Contudo, a literatura que resolve a decisão de transição de modo fá-lo sobretudo isolada sobre planeamento e controlo [5, 6, 7], pelo que a integração dessa decisão com uma perceção florestal real, uma localização por *landmarks* e um mapa de custo de traversabilidade a funcionar em conjunto continua pouco explorada.

1.2 Objetivos

O objetivo principal desta dissertação consiste no projeto, implementação e avaliação de uma arquitetura modular de navegação e decisão autónoma para um robô híbrido terrestre-aéreo aplicado a cenários florestais. Para atingir este objetivo global, definiram-se os seguintes objetivos específicos:

- **Conceção e Modelação do Robô Híbrido:** Idealizar conceptualmente a plataforma híbrida terrestre-aérea, incluindo a disposição dos seus atuadores e sensores, modelar tridimensionalmente os seus elementos de locomoção e implementar a máquina de estados que rege a transição mecânica entre os modos de solo e de voo.
- **Desenvolvimento do Pipeline de Perceção Multimodal:** Combinar dados de *Light Detection and Ranging* (LiDAR) 3D e câmara *Red, Green, Blue* (RGB) para deteção e classificação geométrica e semântica dos elementos estruturais do ambiente (troncos, rochas, arbustos e troncos caídos).
- **Implementação de SLAM por *Landmarks* (*Tree-SLAM*):** Formular um estimador baseado em grafos de fatores no *backend* (via GTSAM/iSAM2) com rotinas dedicadas de associação de dados e fecho de laço relacional.
- **Representação do Ambiente e Custo de Traversabilidade:** Criar uma camada de mapa 2.5D persistente organizando o terreno em *tiles* e integrar um modelo de custo contínuo que distingue vegetação penetrável de obstáculos rígidos.
- **Planeamento de Trajetória e Decisão de Modo:** Implementar um planeador global incremental (D^* Lite) articulado com um controlador de seguimento local reativo e um módulo de decisão responsável pela comutação do modo de locomoção.
- **Ambiente de Simulação e Validação Experimental:** Gerar mundos florestais virtuais no Gazebo, com distribuição espacial natural de árvores, para testar e validar quantitativa e qualitativamente os algoritmos desenvolvidos.

1.3 Contribuições

As principais contribuições científicas e de engenharia resultantes deste trabalho incluem:

- Uma arquitetura modular e integrada de autonomia para robôs híbridos terrestre-aéreos operando em cenários de floresta e todo-o-terreno.
- Uma arquitetura de classificação semântica multi-classe por acumulação de evidência em log-odds ao longo do tempo, com fusão assimétrica entre LiDAR e câmara em que a classe de cada *landmark* nunca é decidida a partir de um único ponto de vista.
- Um conjunto de dados sintético e um auto-*labeler* associado, que gera automaticamente as caixas delimitadoras de treino a partir do *ground-truth* do Gazebo, usados para treinar o detetor de câmara sem anotação manual.
- Um algoritmo de *Tree-SLAM* baseado em grafos de fatores otimizados incrementalmente, com capacidade de relocalização e fecho de laço apoiados na geometria relacional de triângulos formados por troncos.
- Um modelo de traversabilidade 2.5D com mapas de risco contínuos e memória persistente em *tiles*, otimizado para planeamento global em grande escala.
- Um gerador procedimental de mundos florestais (*ForestGen*) para o Gazebo, com distribuição espacial natural de árvores e rochas por agregação com inibição por distância mínima.
- Uma ferramenta *Command Line Interface* (CLI) (*forest*) que integra e automatiza todo o ciclo de vida de uma sessão de simulação em *Robot Operating System 2* (ROS 2), do arranque do Gazebo e dos lançamentos até à limpeza final, eliminando a gestão manual de processos.
- Uma avaliação experimental exaustiva em simulação cobrindo os módulos de perceção, SLAM e a eficácia das decisões de transição de modo de locomoção.

1.4 Estrutura do Documento

A presente dissertação encontra-se organizada em cinco capítulos principais, estruturados da seguinte forma:

- **Capítulo 1 – Introdução:** Apresenta o enquadramento, a motivação, a formulação do problema, os objetivos específicos, as contribuições do trabalho e a organização do documento.
- **Capítulo 2 – Fundamentos e Revisão Bibliográfica:** Efetua uma análise do estado da arte na robótica florestal e na robótica híbrida terrestre-aérea, abordando sensores e perceção, algoritmos de SLAM baseados em filtros e grafos, representações de traversabilidade, planeamento de trajetória e decisão de locomoção híbrida, controlo de seguimento em solo e em voo, e simulação com geração procedimental e distribuição natural de mundos florestais.

- **Capítulo 3 – Desenvolvimento Experimental:** Descreve detalhadamente a arquitetura concetual do robô híbrido, o pipeline de perceção, o sistema de localização e mapeamento (*Tree-SLAM*), o modelo de traversabilidade em *tiles*, a camada de planeamento e decisão híbrida, o controlo de seguimento em solo e em voo, e a geração procedimental de mundos florestais (*ForestGen*) gerida pelo CLI *forest*.
- **Capítulo 4 – Resultados Experimentais:** Apresenta e discute os resultados obtidos nos ensaios de simulação, avaliando o desempenho da perceção, a precisão da trajetória estimada pelo SLAM, o modelo de traversabilidade, o comportamento do planeador global e local, o controlo de seguimento em solo e em voo, a execução das transições de modo de locomoção e o teste de missão completa sobre mundos florestais gerados proceduralmente.
- **Capítulo 5 – Conclusão e Trabalhos Futuros:** Sintetiza as principais conclusões retiradas do trabalho, analisa as limitações da solução atual e aponta perspetivas de evolução para investigação futura.

Capítulo 2

Fundamentos e Revisão Bibliográfica

2.1 Robótica em Ambientes Florestais e Off-Road

A navegação autónoma em ambiente florestal enfrenta um conjunto de adversidades que a distinguem claramente de contextos urbanos ou industriais estruturados, e que decorrem sobretudo da própria natureza física, ótica e ecológica do meio. A luz que atravessa as copas raramente é difusa e uniforme, porque é filtrada de forma irregular pela folhagem. Por isso forma manchas de sol e sombra que se deslocam ao longo do dia e mudam de posição em poucos metros. O contraste entre zonas iluminadas e sombreadas pode ser extremo, mesmo à escala de um só tronco [8]. À superfície, o sinal de posicionamento por satélite (GNSS) é igualmente perturbado pela mesma vegetação que filtra a luz. O sinal percorre um caminho mais longo do que o direto ao refletir-se em troncos e ramos antes de atingir o recetor, um efeito conhecido como *multipath*, e essa diferença extra é interpretada como distância adicional ao satélite, o que introduz um erro sistemático na posição calculada. Além disso, as copas reduzem o número de satélites com potência suficiente para serem usados [1].

O terreno em si constitui outra fonte de dificuldade, desta vez mecânica, porque raízes salientes criam ressaltos abruptos e imprevisíveis, a inclinação varia de forma irregular em distâncias curtas e a vegetação rasteira esconde frequentemente falhas como valas, buracos ou desníveis. A isto acresce a alta variabilidade dos solos, que podem ser, por exemplo, mais compactos, soltos ou lamacentos, com propriedades de aderência e deformação muito distintas entre si. A literatura recente de traversabilidade em contexto florestal identifica precisamente este conjunto de fatores como algo que a análise tradicional, pensada para terreno mais regular, tem dificuldade em capturar de forma unificada [9]. Além disso, ao contrário de um ambiente urbano onde um mapa se mantém válido durante meses, a floresta é um ambiente não estruturado e em constante mudança, porque a vegetação cresce, os ramos caem e a folhagem varia com a estação do ano. Por isso um mapa construído numa só passagem envelhece com rapidez, e sistemas de navegação de campo em floresta reportam a mudança do ambiente como um dos desafios centrais além da falta de sinal GNSS e da variabilidade de iluminação [2].

A repetição geométrica da própria vegetação introduz uma dificuldade adicional, de natureza

distinta das anteriores. Troncos de árvores da mesma espécie e idade têm frequentemente diâmetros e formas muito semelhantes entre si, e essa aparência repetitiva pode confundir um sistema de perceção, levando a associações erradas entre o que está a ser observado e o que já foi mapeado anteriormente [10]. Por fim, nem todo o volume detetado como ocupado corresponde de facto a um obstáculo intransponível, por exemplo, erva alta, arbustos e ramos finos ocupam espaço físico e são detetados como tal, mas cedem ou deformam-se com a passagem do robô, ao contrário de uma rocha ou tronco rígido de dimensão equivalente [11].

2.2 Robótica Híbrida Terrestre-Aérea

Face a um terreno onde nem tudo é transponível pelo solo, seja por obstáculos rígidos, por desníveis ou por falhas que o robô não consegue atravessar, a literatura converge para plataformas que combinam locomoção terrestre e aérea num único robô e reservam o voo para os casos em que contornar pelo ar é a única alternativa. Esta preferência pelo solo não é arbitrária e reflete o custo energético claramente assimétrico entre os dois modos. No solo, o peso do robô é suportado passivamente pelo terreno e a energia gasta serve apenas para vencer o atrito e o declive. Já em voo não existe qualquer suporte passivo, pelo que o robô tem de gerar sustentação ativa e contínua contra a gravidade, um gasto permanente que se traduz, experimentalmente, em consumos cinco a sete vezes superiores aos da locomoção terrestre, consoante a plataforma medida [4, 5]. A comunidade diverge, no entanto, na forma como resolve este problema. Os mecanismos de locomoção terrestre e aérea podem usar conjuntos de atuadores dedicados e distintos, ou podem partilhar e reconfigurar os mesmos atuadores para os dois modos, como ilustrado na Figura 2.1.



Figura 2.1: Exemplos das duas famílias de robôs híbridos terrestre-aéreos. (a) HUUVER [12], de propulsão separada, com lagartas para o solo e hélices dedicadas para o voo. (b) HyTAQ [13], de propulsão partilhada, em que os mesmos rotores impulsionam o voo e o rolamento de uma gaiola cilíndrica. (c) ATMO [14], de propulsão partilhada com transição morfológica em pleno voo, aqui em configuração de voo.

2.2.1 Propulsão separada

Na propulsão separada, o robô transporta dois sistemas de atuação distintos, tipicamente rodas ou lagartas para o solo e rotores dedicados para o voo, que operam de forma desacoplada. Um exemplo deste tipo é o HUUVER [12], um protótipo europeu com um par de lagartas para se deslocar no solo e um conjunto próprio de oito hélices, alinhadas com as lagartas quando não estão

em uso, para voar. A vantagem principal é o desacoplamento dinâmico, porque cada subsistema pode ser modelado e controlado de forma praticamente independente do outro e o controlo de um não depende do estado do outro. A desvantagem é o peso morto estrutural, uma vez que os atuadores do modo inativo, como os motores e lagartas durante a locomoção aérea, são massa e volume adicionais que não contribuem para a locomoção em curso e penalizam a autonomia energética do robô.

2.2.2 Propulsão partilhada

Na propulsão partilhada, o mesmo conjunto de atuadores serve os dois modos de locomoção através de reconfiguração mecânica. Um exemplo criativo é o HyTAQ [13], um quadrotor articulado no centro de uma gaiola cilíndrica em que o mesmo conjunto de rotores impulsiona tanto o voo como o rolamento da gaiola sobre o solo, sem que a locomoção terrestre exija qualquer atuador ou controlador adicional face ao voo. A vantagem é evidente, porque sem duplicação de atuadores elimina-se o peso morto característico da propulsão separada. A desvantagem do HyTAQ está na locomoção terrestre em si, que fica subatuada. Os mesmos rotores que sustentam o voo são a única fonte de movimento no solo, e por isso o robô só avança inclinando-se para a frente, sem capacidade de manobra fina nem de vencer terreno acidentado. Uma variante mais recente desloca a dificuldade para outro ponto ainda. No ATMO, Mandralis *et al.* [14] usam também propulsão partilhada mas com transformação morfológica entre modos, e mostram que essa transformação é o passo crítico. Transformar a geometria do robô junto ao solo implica interações de contacto solo-veículo complexas que limitam a aplicabilidade em terreno irregular, enquanto transformar em pleno voo exige operar os atuadores perto ou além dos seus limites nominais e gerir ao mesmo tempo efeitos aerodinâmicos complexos. A propulsão partilhada resolve assim o problema do peso morto, mas em troca ou restringe a locomoção no solo ou torna a transição entre modos muito mais difícil de conceber.

2.3 Perceção

Um robô a operar numa floresta precisa de responder a duas perguntas distintas sobre aquilo que o rodeia. A primeira é geométrica e responde a onde está algo e que forma tem, sem que isso implique ainda saber o que esse algo representa. A segunda é semântica e responde ao que é essa forma, distinguindo por exemplo um tronco de uma rocha. Esta separação não é arbitrária, decorre de duas necessidades de informação diferentes e é reconhecida como um problema particularmente difícil em ambiente florestal, porque a estrutura do meio combina obstáculos rígidos como troncos e rochas com elementos deformáveis como erva alta ou ramos finos, e porque as condições de iluminação, poeira e vibração do terreno degradam significativamente alguns sensores de perceção [15]. As subsecções seguintes tratam primeiro os sensores que captam a informação bruta, depois o que se pode extrair dela ao nível geométrico e semântico, e por fim como as duas respostas podem ser combinadas numa só decisão.

2.3.1 Sensores e Modalidades

A obtenção de informação sobre o ambiente pode seguir dois princípios distintos. Nos sensores ativos, o próprio sensor emite energia, seja laser, seja onda rádio, seja som, e mede o tempo ou a fase até essa energia regressar depois de refletida por um objeto. Nos sensores passivos, o sensor não emite energia própria e a informação é inferida a partir da energia ambiente que naturalmente atinge o sensor, tipicamente luz visível refletida pelos objetos. No caso de câmaras, a profundidade não é medida diretamente e tem de ser inferida, tipicamente por triangulação entre duas vistas ou por aprendizagem a partir de uma única imagem. Importa notar que a mesma família de sensores serve frequentemente tanto à perceção geométrica como à perceção semântica, por exemplo uma câmara fornece cor para classificação semântica e, no caso de um par estéreo, também profundidade, e um LiDAR fornece geometria diretamente mas os seus pontos também podem ser classificados semanticamente. A Tabela 2.1 resume as famílias mais relevantes para a perceção em robótica de campo e as suas limitações principais em ambiente florestal, agrupadas consoante o sensor seja ativo, passivo, ou híbrido dos dois princípios.

Tabela 2.1: Famílias de sensores consideradas para a perceção em ambiente florestal.

Sensor	**Vantagens**	**Limitações em floresta**
SENSORES ATIVOS		
LiDAR 3D	Nuvem 3D densa e robusto à luz variável	Peso e consumo elevados, oclusão e retornos parciais em vegetação densa
Radar	Robusto a chuva, poeira e neblina	Resolução angular insuficiente para separar troncos próximos
SENSORES PASSIVOS		
Câmara estéreo	Leve e barata, dá cor e textura e estima profundidade por triangulação	A profundidade falha em superfícies sem textura, e como qualquer câmara degrada-se com pouca luz, poeira ou chuva
Câmara monocular	Muito leve e barata, dá cor e textura	Não mede profundidade, apenas a infere e sem escala métrica garantida, e partilha a fragilidade das câmaras a más condições
SENSOR ATIVO COM COMPONENTE PASSIVA		
RGB-D	Profundidade densa a curto alcance com cor associada	Satura com luz solar direta e tem alcance curto, o que a inviabiliza no exterior

2.3.2 Perceção Geométrica

O objetivo da perceção geométrica em floresta é extrair automaticamente de uma nuvem de pontos a posição e as dimensões de cada objeto do ambiente, sejam troncos, rochas ou outros obstáculos. No caso dos troncos, essas medidas incluem também o diâmetro, uma grandeza que a medição manual em campo recolhe de forma lenta e dispendiosa [16]. A via clássica para lá chegar,

historicamente desenvolvida para o inventário florestal, é um *pipeline* de vários passos, resumido na Figura 2.2, que vai isolando progressivamente a informação até restar a superfície do objeto a medir.

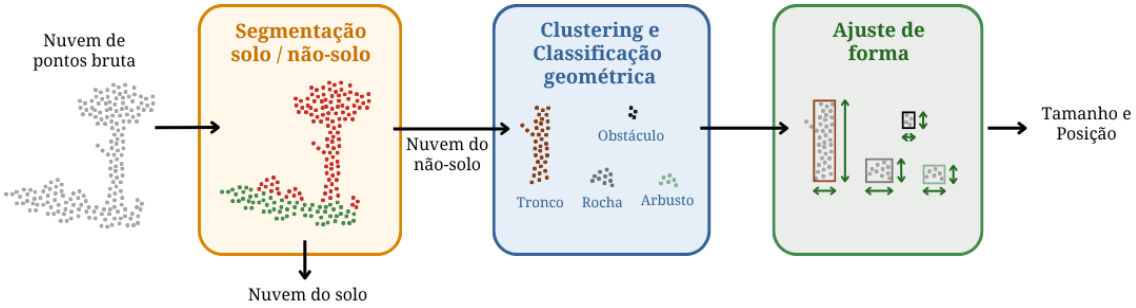


Figura 2.2: *Pipeline* típico da perceção geométrica em ambiente florestal, da nuvem de pontos bruta aos parâmetros físicos de cada objeto.

- **Segmentação solo/não-solo**: separa os pontos que pertencem ao terreno dos que pertencem a algo acima dele. O método clássico de referência é o filtro morfológico progressivo, que aplica à superfície uma janela de dimensão crescente e, em cada passo, classifica como não-solo os pontos cuja elevação excede a da superfície filtrada mais do que um limiar. Como a janela começa pequena e vai crescendo, o filtro remove primeiro os objetos pequenos e só depois os maiores, o que lhe permite eliminar estruturas de dimensão progressivamente maior sem confundir uma encosta longa e contínua com um objeto pousado sobre o terreno, preservando assim o relevo [17]. A sua principal fraqueza prática é o número de parâmetros a afinar, como a dimensão inicial e final da janela e o declive assumido para o terreno, cujo efeito não é intuitivo e cujo valor ótimo muda de cena para cena.

  Uma alternativa mais recente inverte a lógica do problema em vez de a refinar. O *Cloth Simulation Filter* (CSF) vira a nuvem de pontos ao contrário e deixa cair sobre ela, de cima, um tecido virtual rígido, simulado como uma malha de massas ligadas por molas sujeitas à gravidade. O tecido pousa sobre o que era originalmente o terreno, sem conseguir descer para dentro das reentrâncias que os objetos ocupam, e no fim os pontos que ficam a menos de um dado limiar da superfície do tecido são classificados como solo e os restantes como não-solo [18]. A vantagem é que os parâmetros passam a ter significado físico direto, nomeadamente a rigidez do tecido, a resolução da malha e a distância de classificação, o que torna a afinação muito mais intuitiva do que a de um filtro morfológico. Em troca, o método assume que o terreno é uma superfície contínua sobre a qual um tecido pode assentar, e por isso tende a absorver como solo os objetos baixos e achatados, como uma rocha rasteira ou um tronco caído, que o tecido cobre por cima sem os distinguir do relevo. É uma limitação estrutural do modelo e não um defeito de afinação, o que obriga qualquer sistema que o adote a prever um mecanismo de recuperação desses objetos.

- ***Clustering* e classificação geométrica**: agrupa os pontos não-solo em objetos individuais e caracteriza a sua forma. Uma ferramenta comum para essa caracterização é a análise dos

valores próprios da matriz de covariância de uma vizinhança local, que é uma análise em componentes principais aplicada localmente e distingue três categorias [19]. Há dispersão quando os três valores próprios são comparáveis entre si, o que é próprio de volumes porosos como erva ou copas de árvores. Há linearidade quando um valor próprio domina os outros dois, o que caracteriza estruturas finas e alongadas como ramos. E há planaridade quando dois valores próprios dominam o terceiro, o que caracteriza superfícies como o solo ou a face de uma rocha.

Esta atribuição depende, no entanto, do raio da vizinhança escolhida e não apenas do objeto observado. A superfície de um tronco aparece como planar quando a vizinhança é pequena face ao seu diâmetro, porque a essa escala a casca é aproximadamente uma superfície. Mas o mesmo tronco aparece como linear quando a vizinhança é grande o suficiente para abranger a sua extensão vertical, ou quando a densidade de pontos sobre ele é baixa. A escala de análise é por isso um parâmetro de projeto e não um detalhe de implementação.

- **Ajuste de forma**: é o passo que distingue um *pipeline* florestal de um *pipeline* genérico de robótica móvel. Enquanto num ambiente urbano basta saber que um *cluster* é um obstáculo, numa floresta interessa saber que dimensão tem o tronco, porque é essa dimensão que permite identificar a mesma árvore mais tarde e que constitui, por si só, informação de inventário. O passo extrai então parâmetros físicos concretos, ajustando uma forma geométrica simples, um círculo ou um cilindro, aos pontos do *cluster*. O método de referência é o *Random Sample Consensus* (RANSAC), que ajusta o modelo a subconjuntos aleatórios pequenos dos dados e escolhe o ajuste que reúne mais pontos consistentes, descartando implicitamente ramos ou folhagem sobreposta que não pertencem à superfície do tronco [20]. Ferramentas de medição automática de povoamentos atingem erros médios da ordem de poucos centímetros no diâmetro à altura do peito quando validadas contra milhares de medições manuais em campo [16].

2.3.3 Perceção Semântica

Enquanto a perceção geométrica responde a onde e que forma, a perceção semântica responde ao que é cada elemento observado. Essa resposta pode vir por uma de três vias.

- **Segmentação por pixel** (Figura 2.3a): atribui uma classe a cada pixel de uma imagem, produzindo um mapa denso da cena inteira. É a via mais completa em cobertura espacial, mas também a mais exigente em computação, porque decide uma classe para cada um dos milhões de pixels da imagem.

- **Deteção por caixas delimitadoras** (Figura 2.3b): isola apenas os objetos que interessam, cada um com uma caixa, uma classe e uma confiança. É mais leve do que a segmentação por pixel, mas a caixa inclui sempre alguma área à volta do objeto que não lhe pertence, perdendo precisão nos contornos.

- **Segmentação em nuvem de pontos** (Figura 2.3c): aplica a mesma lógica da segmentação por pixel, mas diretamente aos pontos 3D captados pelo LiDAR, sem passar por uma imagem. Divide-se em métodos baseados em regras geométricas explícitas e métodos baseados em aprendizagem ponto a ponto.



(a) Pixel (b) Caixas delimitadoras (c) Nuvem de pontos

Figura 2.3: As três vias da perceção semântica sobre a mesma cena. Adaptado do conjunto de dados RELLIS-3D [21].

Esta divisão entre regras explícitas e aprendizagem não é exclusiva da nuvem de pontos, e atravessa as três vias. Um classificador clássico, como um classificador linear ou uma máquina de vetores de suporte sobre características calculadas manualmente, é mais leve computacionalmente e mais fácil de interpretar, mas exige engenharia manual de características para cada ambiente novo. Um classificador de aprendizagem profunda generaliza melhor entre ambientes diferentes, à custa de precisar de muitos mais dados anotados e de mais capacidade de computação embarcada.

Os classificadores de aprendizagem profunda têm um problema adicional, o *domain gap*, que consiste em perder desempenho quando avaliados num ambiente diferente daquele em que foram treinados. Grande parte dos modelos existentes foi treinada em ambiente urbano, onde as classes que interessam estão em primeiro plano, como pessoas, carros e sinais, enquanto o fundo da imagem é tratado como pouco relevante. Numa floresta essa lógica inverte-se, porque o fundo, ou seja, o solo, a vegetação e o relevo, é precisamente o que a perceção precisa de analisar bem para saber por onde o robô pode passar. Um modelo treinado para ignorar o fundo urbano chega mal preparado a um problema onde o fundo é o próprio alvo. Um conjunto de dados de referência recolhido em ambiente todo-o-terreno expõe bem este problema, nomeadamente o desequilíbrio entre classes e a topografia irregular do solo que a cidade não coloca [21].

2.3.4 Fusão Multimodal

Depois de obtidas as respostas geométrica e semântica, resta decidir como combiná-las. A literatura de fusão sensorial distingue esta decisão sobretudo pela fase do processamento em que ocorre. Um levantamento recente sobre fusão de câmara e LiDAR em condução autónoma propõe uma taxonomia com duas classes principais, nomeadamente a fusão forte e a fusão fraca, e subdivide a fusão forte em quatro variantes, a precoce, a intermédia, a tardia e a assimétrica, como resume a Figura 2.4 [22].

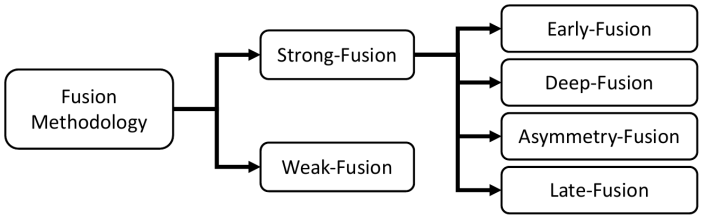


Figura 2.4: Taxonomia geral de fusão sensorial câmara-LiDAR, fusão forte vs. fusão fraca. Fonte: Huang *et al.* [22].

O critério que distingue as quatro variantes da fusão forte é o nível do *pipeline* em que os dois ramos, o da câmara e o do LiDAR, se encontram. Cada ramo processa a sua modalidade em três níveis sucessivos, nomeadamente os dados brutos, as características extraídas desses dados e as propostas de objeto. Na fusão precoce os dois ramos juntam-se logo ao nível dos dados, o que preserva toda a informação disponível mas exige uma calibração espacial e temporal muito apertada entre sensores. Na fusão intermédia cada ramo produz primeiro uma representação latente, e são essas representações que se combinam, num compromisso entre expressividade e robustez a pequenos desalinhamentos.

Na fusão tardia cada ramo chega sozinho a uma proposta de objeto quase final, e só essas propostas são combinadas no fim, por votação, por média ponderada ou por acumulação bayesiana de log-odds. É a variante mais modular e mais fácil de depurar, porque cada ramo pode ser desenvolvido e validado isoladamente, mas perde a informação fina que só existe na combinação dos dados brutos. A fusão assimétrica quebra a simetria entre ramos em vez de escolher um único nível de junção para ambos. Um dos ramos, tipicamente o LiDAR, avança até um nível de processamento mais tardio, e o outro serve apenas de apoio, sem chegar a gerar uma proposta independente por si só.

A fusão fraca segue uma lógica diferente das restantes quatro, porque em vez de combinar representações internas usa a saída já decidida de uma modalidade para orientar o processamento da outra, por regras explícitas em vez de aprendizagem conjunta.

2.4 Localização e Mapeamento Simultâneos

Para um robô se localizar e para construir um mapa do que o rodeia, há normalmente duas situações mais simples do que aquela que se enfrenta numa floresta.

Na primeira situação já existe um mapa fiável do ambiente, construído antes ou fornecido de outra forma. O robô só precisa então de resolver uma incógnita, a sua própria pose, comparando o que observa a cada instante com esse mapa já conhecido. É o problema de localização pura.

Na segunda situação o robô conhece a sua pose com precisão a cada instante, por exemplo através de GNSS de alta precisão em espaço aberto. Falta então resolver só a outra incógnita, a estrutura do ambiente, registando o que é observado em cada pose conhecida até formar um mapa. É o problema de mapeamento puro.

Nenhuma destas duas situações se verifica numa floresta. Não existe mapa prévio do ambiente florestal, e não existe uma fonte de pose fiável e contínua, porque o terreno é não estruturado e o sinal de GNSS degrada-se de forma significativa sob coberto arbóreo denso. Sem mapa e sem pose fiável, as duas incógnitas têm de ser resolvidas ao mesmo tempo, apoiando-se uma na outra à medida que o robô se desloca. A este problema dá-se o nome de SLAM.

2.4.1 Filtros e Grafos de Fatores

A resolução do problema de SLAM divide-se historicamente em duas famílias de abordagens, que diferem sobretudo na forma como tratam a incerteza acumulada ao longo do tempo. Um levantamento recente sobre SLAM com LiDAR propõe exatamente esta dicotomia como esquema organizador do campo, separando os sistemas existentes entre os baseados em filtro e os baseados em otimização de grafo [23]. A Figura 2.5 contrasta as duas famílias antes de cada uma ser descrita em detalhe.

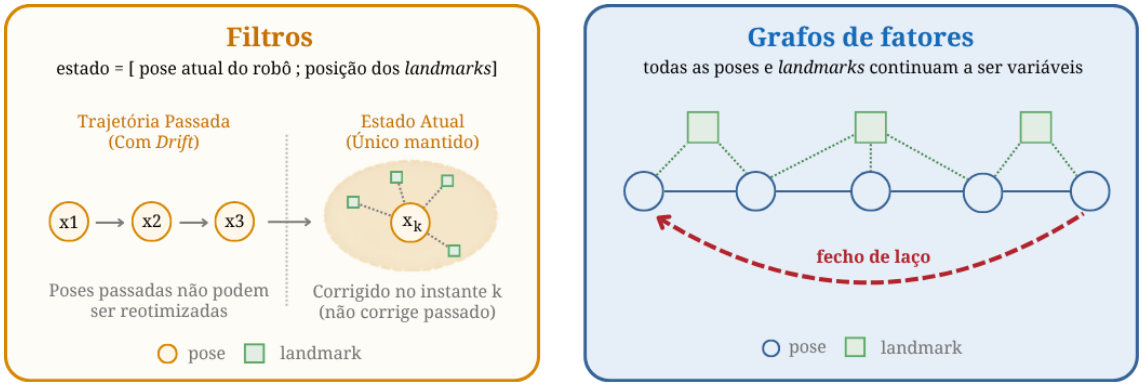


Figura 2.5: As duas famílias de SLAM, baseada em filtro (à esquerda) e baseada em grafo de fatores (à direita).

Na família dos filtros, o estado do mundo, composto pela pose do robô e pelas posições dos *landmarks* conhecidos, é representado por uma distribuição de probabilidade, tipicamente uma gaussiana com uma média e uma matriz de covariância. A cada novo movimento do robô, essa distribuição é propagada para a frente segundo o modelo de movimento, e a cada nova observação do ambiente é corrigida segundo o modelo de sensor, de forma recursiva. Um exemplo contemporâneo desta família aplicado a odometria LiDAR-inercial é o FAST-LIO2 (*Fast Direct LiDAR-Inertial Odometry*), que usa um filtro de Kalman iterado fortemente acoplado para registar diretamente os pontos brutos do LiDAR contra o mapa em construção, sem passar por uma etapa explícita de extração de *features*, atingindo taxas de atualização até 100 Hz [24]. A limitação estrutural desta família é que o estado passado nunca é revisitado de forma explícita. Um erro cometido numa observação antiga fica implicitamente contido na estimativa atual e só pode ser compensado pelas observações seguintes, nunca corrigido diretamente com efeito retroativo sobre poses ou *landmarks* anteriores.

Na família dos grafos de fatores, o problema é formulado como um grafo em que os nós, como as poses do robô ao longo do tempo e posições de *landmarks*, são variáveis a estimar e as arestas são fatores, restrições probabilísticas entre essas variáveis derivadas da odometria e das observações do ambiente. Resolver o SLAM passa a ser um problema de otimização não linear, pois é necessário encontrar a configuração de todas as variáveis que melhor satisfaz o conjunto de restrições. A vantagem central desta formulação é que, quando surge uma restrição nova entre variáveis distantes no tempo, por exemplo ao reconhecer que o robô voltou a um local já visitado, essa restrição pode propagar-se e corrigir toda a trajetória e o mapa construídos entre essas duas visitas, não apenas a pose mais recente. O LIO-SAM (*Lidar Inertial Odometry via Smoothing and Mapping*) ilustra bem esta arquitetura, combinando fatores de odometria LiDAR, fatores de *Inertial Measurement Unit* (IMU) pré-integrado e fatores opcionais de GPS e de fecho de *loop*, todos resolvidos de forma incremental através de suavização e mapeamento com árvore de Bayes [25].

A Tabela 2.2 resume esta comparação.

Tabela 2.2: Comparação entre SLAM baseado em filtros e SLAM baseado em grafos de fatores.

Critério	**Filtros**	**Grafos de fatores**
Representação do estado	Média e covariância de todo o estado atual	Grafo de variáveis (poses, *landmarks*) ligadas por fatores
Corrige o passado	Não, só ajusta a estimativa presente	Sim, reotimiza o histórico quando surge nova evidência
Custo computacional	Cresce com o número de *landmarks* no estado	Cresce com o tamanho do grafo, mas com atualização incremental
Robustez a erros de associação	Um erro antigo fica implícito na estimativa atual	Um fator novo pode corrigir associações e poses passadas
Exemplos	*Extended Kalman Filter* (EKF)-SLAM clássico, FAST-LIO2	LIO-SAM, SLAM por *landmarks* de troncos [26]

Esta diferença entre ajustar só o presente e corrigir o passado ganha particular relevância num ambiente onde a associação de dados, ou seja, decidir a que objeto do mundo corresponde cada nova observação, é difícil e sujeita a erro, como acontece tipicamente numa floresta densa com muitos troncos de aparência semelhante.

2.4.2 Do Mapa Denso ao Mapa de Landmarks

Além da distinção entre filtros e grafos, existe uma segunda escolha independente sobre o que compõe o mapa construído. No SLAM denso, o mapa é sempre alguma forma de geometria acumulada do ambiente, mas o grau de processamento dessa geometria varia de sistema para sistema. Nalguns casos o mapa é literalmente a nuvem de pontos bruta captada pelo sensor, sem qualquer extração intermédia, como acontece no FAST-LIO2, que regista diretamente os pontos brutos do LiDAR contra o mapa em construção [24], e no KISS-ICP (*Iterative Closest Point*, no

espírito do princípio *Keep It Simple, Stupid* que lhe dá o nome), que alinha nuvens de pontos consecutivas por correspondência ponto a ponto, com muito poucos parâmetros a afinar e sem exigir sequer dados de IMU [27]. Noutros casos o mapa é construído a partir de *features* extraídas da nuvem de pontos, como arestas e superfícies planas, em vez da nuvem completa, o que é o caso do LeGO-LOAM (*Lightweight and Ground-Optimized Lidar Odometry and Mapping*), otimizado especificamente para operar sobre terreno variável e inclinado sem pressupor um solo globalmente plano [28], e do já referido LIO-SAM. Em qualquer uma destas variantes, o mapa continua a não ter significado semântico próprio, porque um ponto ou uma *feature* não distingue por si só um tronco de uma rocha.

No SLAM esparso por *landmarks*, o mapa não é a geometria bruta mas sim uma lista de objetos discretos identificados no ambiente, com a sua posição e eventualmente características descritivas próprias, como no caso de uma árvore o diâmetro do seu tronco. É esta a via seguida pelos sistemas de SLAM por árvores discutidos na subsecção seguinte.

A escolha entre estas duas vias envolve um compromisso relevante para o contexto florestal. Os sistemas densos 3D mais recentes já não pressupõem um solo plano e lidam bem com relevo irregular, mas mantêm mapas cuja dimensão cresce com a escala do ambiente percorrido, exigindo mais memória e mais computação, e cujo conteúdo não tem significado próprio, porque um ponto do mapa não distingue por si só um tronco de uma rocha ou de um ramo. Os sistemas 2D mais antigos de *scan-matching*, como o Hector SLAM e o slam_toolbox, acrescentam ainda a limitação de restringirem o mapa a um único plano horizontal, sem representar altitude, o que é particularmente problemático para um veículo capaz de se deslocar também no ar. Um SLAM esparso por *landmarks* reduz drasticamente a dimensão do mapa e acrescenta-lhe significado semântico, à custa de depender inteiramente da qualidade da deteção prévia desses objetos, um problema de perceção que antecede o SLAM em si.

2.4.3 SLAM por *landmarks* em ambiente florestal

O SLAM por *landmarks* recebe as deteções já feitas pela perceção e organiza-se a partir daí em três fases, ilustradas na Figura 2.6.



Figura 2.6: Fases típicas de um *pipeline* de SLAM por *landmarks* de troncos em ambiente florestal, a partir das deteções já fornecidas pela perceção.

- **Associação de dados e *tracking*.** Cada deteção nova tem de ser associada a um *landmark* já existente no mapa ou reconhecida como um *landmark* novo. Este passo é tipicamente formulado como um problema de atribuição entre o conjunto de deteções atuais e o conjunto de *landmarks* conhecidos, resolvido por otimização combinatória. A dificuldade desta associação está ligada sobretudo à repetitividade da disposição espacial dos troncos vizinhos, e não apenas à densidade da floresta. Quanto mais parecidas entre si forem as configurações locais de árvores vizinhas, mais fácil é confundir um trio de troncos com outro semelhante situado noutra zona do mapa. Essa repetitividade pode surgir tanto de um espaçamento regular, como o de muitas plantações, como de uma zona de floresta natural densa e geometricamente homogénea. Uma disposição mais irregular tende a produzir configurações locais mais distintas e por isso mais fáceis de associar corretamente, ainda que traga consigo outras dificuldades, como maior oclusão entre troncos.

- **Construção e otimização do grafo de fatores.** As poses do robô ao longo do tempo e as posições dos *landmarks* entram como variáveis num grafo de fatores, ligadas por fatores de odometria entre poses consecutivas e por fatores de observação entre cada pose e os *landmarks* nela vistos. Resolver este grafo em tempo real, à medida que o robô avança e o grafo cresce, exige software de otimização preparado para isso, e existem várias bibliotecas com este propósito. O g2o e o Ceres Solver são duas bibliotecas de otimização não linear de uso geral, amplamente adotadas em SLAM baseado em grafos, que recalculam a solução do grafo inteiro a cada nova otimização. O GTSAM segue uma filosofia diferente, mais direcionada à robótica, fornecendo os blocos para descrever poses, *landmarks* e os vários tipos de fatores entre eles, e é tipicamente combinado com o algoritmo iSAM2, que otimiza o grafo de forma incremental em vez de o resolver do zero a cada novo fator. O iSAM2 mantém uma estrutura de dados chamada árvore de Bayes, que regista quais as variáveis afetadas por cada nova restrição, e recalcula apenas essa parte do grafo, o que torna viável otimizar continuamente um grafo que cresce a cada instante e a cada novo *landmark*, sem acumular um atraso crescente [29]. É esta combinação de GTSAM com iSAM2 que aparece com mais frequência no SLAM por *landmarks* de árvores. A arquitetura conceptualmente mais próxima de um SLAM por *landmarks* de troncos combina *landmarks* cilíndricos com descritores de triângulo formados entre trios de troncos para apoiar a associação e o fecho de *loop*, tendo sido validada num modelo simulado de um veículo comercial a atravessar um ambiente florestal [26]. Um sistema mais recente aplicado a pomares funde GPS, odometria e observações de tronco exatamente neste tipo de grafo, otimizado de forma incremental com GTSAM e iSAM2, reportando um erro de geolocalização tão baixo quanto 18 cm no seu melhor caso [10], o que valida esta combinação de *backend* para o problema de SLAM por árvores.

- **Mapa de *landmarks* e fecho de *loop*.** O resultado final desta fase é um mapa esparso de *landmarks*, sujeito a correção adicional sempre que o robô reconhece ter voltado a uma zona já mapeada. Este reconhecimento, o fecho de *loop*, é tratado em detalhe na subsecção

seguinte. Um sistema recente sem dependência de IMU ou GNSS demonstra que este tipo de SLAM por *landmarks* se mantém viável mesmo sem nenhuma das fontes de pose absoluta habituais, exatamente a condição identificada no início desta secção como característica do ambiente florestal [3].

2.4.4 Relocalização e fecho de loop em floresta

Reconhecer que o robô voltou a uma zona já mapeada é mais difícil numa floresta do que na maioria dos outros ambientes, porque um tronco isolado dificilmente se distingue de outro por aparência, e o que realmente identifica um local é o padrão relativo formado por vários troncos próximos entre si, não qualquer tronco individual.

O reconhecimento de local resolve-se por isso em duas fases encadeadas, um desenho que o levantamento de SLAM com LiDAR já identifica como estrutura habitual deste problema, primeiro uma pesquisa por descritor que procura na base de dados os locais parecidos com o atual, e só depois uma verificação que confirma a correspondência e estima a pose [23]. A Figura 2.7 ilustra as duas fases. A primeira é grosseira e barata, e resume o padrão local num descritor compacto, como um histograma das distâncias e dos diâmetros entre *landmarks* vizinhos, o que reduz rapidamente centenas de locais possíveis a um punhado de candidatos. A segunda é fina e cara, e por isso só corre sobre esses poucos candidatos, verificando a geometria exata de triângulos formados por trios de *landmarks* próximos e rejeitando as correspondências que não resistem a esse teste.

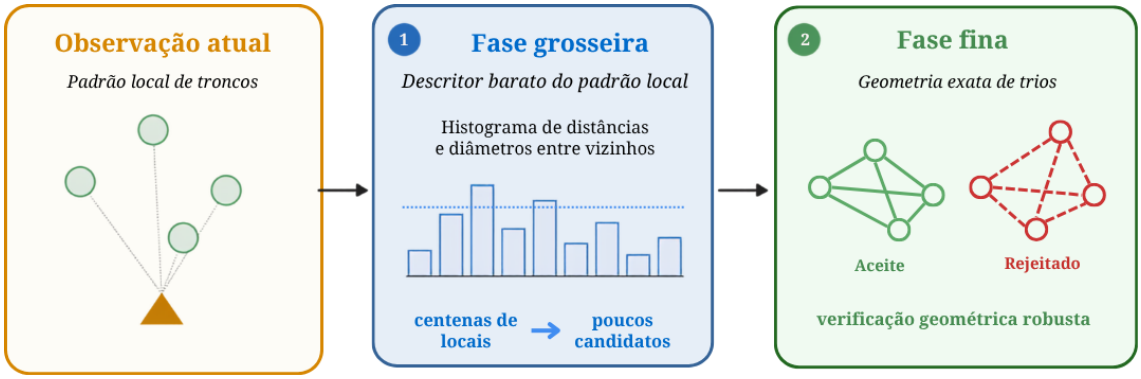


Figura 2.7: As duas fases do reconhecimento de local em floresta, da observação atual até à pose no mapa já existente.

O TreeLoc segue exatamente este desenho, combinando um histograma de distribuição de árvores como filtro grosseiro com um descritor de triângulo como verificação fina, sem qualquer etapa de aprendizagem, correndo em menos de 50 ms em CPU, com uma taxa de reconhecimento correto à primeira tentativa de 0,91 no melhor conjunto de dados testado, ainda que caia para 0,77 noutros mais exigentes [30].

Um problema que este tipo de relocalizador deixa em aberto surge precisamente quando a floresta é densa o suficiente para que vários trios de árvores formem triângulos parecidos entre si, gerando falsas correspondências que a verificação geométrica isolada nem sempre rejeita.

Uma extensão deste método resolve este caso ao acrescentar filtragem pelo diâmetro dos troncos envolvidos e votação de orientação entre candidatos, reduzindo especificamente os falsos positivos que surgem de constelações de árvores ambíguas e repetidas [31].

Falta ainda o caso em que o robô perde por completo a continuidade da sua localização e tem de se reencontrar num mapa construído antes, eventualmente a partir de um ponto de vista bastante diferente daquele em que esse mapa foi feito. De Lima *et al.* tratam este problema em floresta, usando a classe semântica dos objetos observados para orientar a procura do local certo e resolvendo depois a pose num grafo de fatores que liga entre si vistas diferentes do mesmo sítio [32]. O que torna este trabalho relevante para esta dissertação é não exigir uma trajetória contínua entre o momento em que o mapa foi construído e o momento em que o robô se tenta relocalizar, que é exatamente a condição de um veículo cuja localização é interrompida.

2.5 Representações do Ambiente e Traversabilidade

Um robô a operar numa floresta pode ter a sua localização e o seu mapa do ambiente perfeitamente resolvidos e mesmo assim não saber se pode avançar. Saber onde se está e o que existe à volta não responde à pergunta que importa para a navegação, que é decidir se um determinado pedaço de terreno pode ser atravessado e a que custo. É a esta pergunta que a representação do ambiente para navegação tenta responder, e ela pode ser decomposta em três perguntas mais pequenas e ortogonais entre si. A primeira pergunta é o que se guarda por célula do mapa, um valor discreto de livre ou ocupado, ou um valor contínuo que exprime risco. A segunda é para quem esse valor vale, se é uma propriedade do terreno em si ou se já depende do robô concreto que o vai atravessar. A terceira é como essa informação é organizada no espaço, se apenas como uma superfície ou como um volume completo. As três subsecções seguintes tratam estas três perguntas por esta ordem.

2.5.1 De ocupação binária a mapas de risco

A forma mais antiga de representar o ambiente para navegação é a grelha de ocupação, proposta por Moravec e Elfes em 1985 para mapear o que rodeia um robô a partir de leituras de sonar [33]. Cada célula guarda uma estimativa da probabilidade de estar ocupada, que vai sendo atualizada por estimação bayesiana à medida que chegam novas leituras [34], e que na prática é cortada num limiar para produzir uma decisão binária de livre ou ocupado. É ainda sobre grelhas deste tipo que a literatura recente formula o risco de traversabilidade [35]. O modelo funciona bem em ambientes estruturados, como interiores ou estradas, onde um objeto ou está mesmo lá ou não está.

Em ambiente natural este modelo binário deixa de ser suficiente. Um tufo de relva alta não é um obstáculo sólido, mas também não é uma zona claramente livre, é algo intermédio que o robô pode eventualmente atravessar assumindo algum risco. Um sistema que só sabe responder livre ou ocupado é obrigado a tratar essa relva como obstáculo e a planear sempre um caminho à volta dela, mesmo quando atravessá-la seria mais eficiente e perfeitamente seguro. Além disso, a grelha de ocupação clássica converge sempre para os extremos zero ou um à medida que acumula medições, o que apaga a incerteza real da estimativa em vez de a preservar.

A alternativa proposta na literatura recente é substituir a probabilidade de ocupação por um valor contínuo de risco por célula. Laconte *et al.* propõem exatamente esta mudança com o chamado Lambda Field, uma representação em que cada célula guarda uma intensidade de colisão em vez de uma simples probabilidade de ocupação, e que não converge para os extremos zero ou um mas se mantém no valor real de ocupação da célula [36]. Os autores definem o risco em termos físicos concretos, como a força de colisão esperada ao longo de um caminho. A Figura 2.8 mostra o efeito desta mudança numa cena com vegetação penetrável. O painel (b) é a grelha clássica, onde cada célula guarda uma probabilidade de ocupação, e o painel (c) é o Lambda Field, onde cada célula guarda uma intensidade de colisão contínua que separa a vegetação penetrável, de valor baixo, dos obstáculos sólidos, de valor alto. Enquanto a grelha binária trata essa vegetação como obstáculo e obriga o robô a contorná-la, o Lambda Field deixa-o atravessá-la mantendo noção do risco assumido.

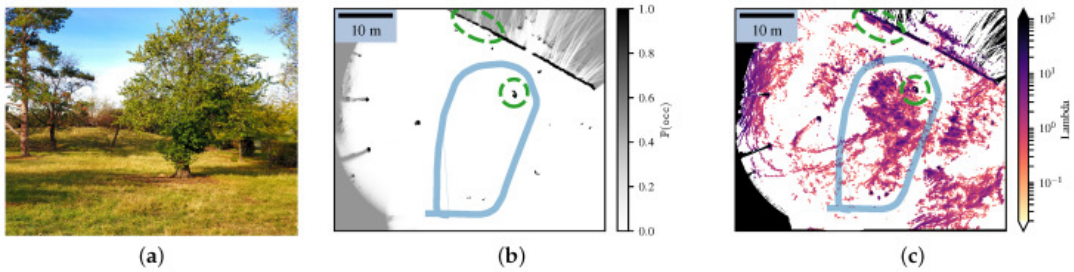


Figura 2.8: (a) Cena real. (b) Grelha de ocupação clássica, $\mathbb{P}(\text{occ})$. (c) Lambda Field, intensidade de colisão Λ. Adaptado de Laconte *et al.* [36].

Mesmo com esta mudança para um valor contínuo, guardar apenas a sua média esconde o quão confiante a estimativa é, e há trabalho que mostra valer a pena pesar também os piores casos e não só a média [37]. Uma revisão recente sistematiza este panorama, classificando os algoritmos de navegação consciente do risco pela forma como caracterizam esse risco, revendo como ele é avaliado ao longo de um caminho sobre grelhas de traversabilidade, e discutindo até que ponto as métricas hoje usadas na robótica são consistentes entre si [35].

A Tabela 2.3 resume o contraste central desta subsecção.

Tabela 2.3: Grelha de ocupação binária vs. mapa de risco contínuo.

	Ocupação binária	**Risco contínuo**
Valor guardado por célula	Probabilidade de ocupação	Intensidade de risco ou custo
Comportamento a longo prazo	Converge para 0 ou 1	Mantém-se no valor real de ocupação
Célula ambígua (ex. relva)	Tratada como obstáculo sólido	Fica com risco intermédio, pode ser atravessada
Decisão do planeador	Contornar tudo o que está ocupado	Escolher caminhos de menor risco acumulado

2.5.2 Traversabilidade aparente e relativa

Saber que uma célula tem um determinado valor de risco não chega, porque esse valor pode ou não depender de quem vai lá passar. Yoo *et al.* tornam esta distinção explícita ao separar a traversabilidade em duas naturezas diferentes, a traversabilidade aparente e a traversabilidade relativa, e ao incorporar essa separação diretamente no planeamento e no controlo do robô [38]. A traversabilidade aparente é estimada só a partir de sensores exterocetivos e descreve uma propriedade do terreno em si, independente de quem o vai atravessar. A traversabilidade relativa integra o estado interno do próprio robô, como a velocidade ou o binário nos motores, e a interação entre o robô e o terreno, o que faz da mesma célula de terreno ter um veredito diferente consoante o robô que a avalia.

Esta distinção de alto nível cruza-se com uma segunda forma clássica de organizar a literatura, que separa os métodos pela fonte de informação usada para estimar a traversabilidade aparente. Um método pode ser geométrico, olhando só à forma do terreno, como declive ou rugosidade, pode ser baseado em aparência, olhando à textura ou à cor captada por uma câmara, ou pode ser híbrido, combinando as duas fontes [39]. A Figura 2.9 organiza estas duas distinções numa única árvore de famílias.

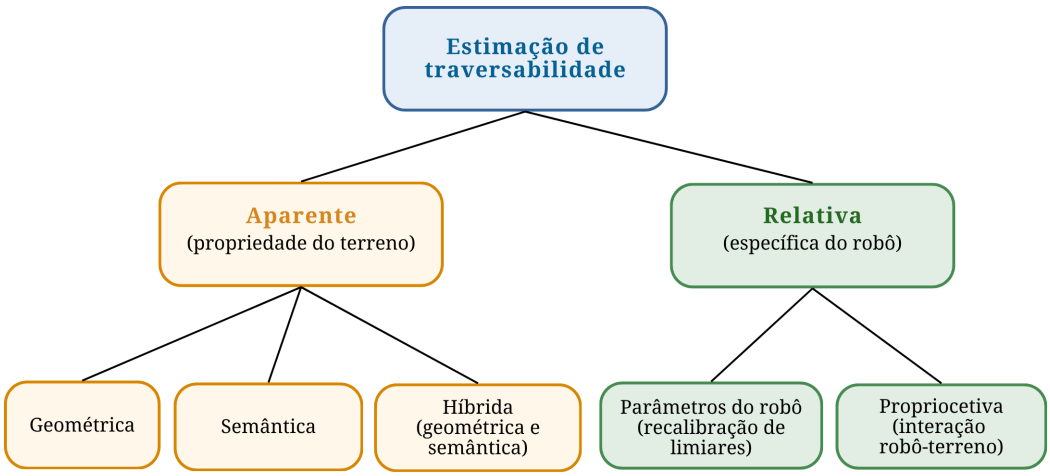


Figura 2.9: Árvore de famílias da estimação de traversabilidade. O nível de topo segue a distinção aparente/relativa de Yoo *et al.* [38], os ramos sob "Aparente"seguem a divisão geometria/aparência/híbrida de Papadakis [39].

Cada ramo distingue-se pela fonte que usa. A geométrica deriva o risco da forma do terreno, como o declive, a rugosidade ou a altura de degrau. A baseada em aparência deriva-o do aspeto visual, como a cor e a textura, o que permite separar superfícies de geometria parecida mas comportamento diferente. A híbrida combina ambas. Do lado relativo, os métodos por parâmetros do robô reaproveitam essas mesmas fontes mas recalibram os limiares à pegada, à distância ao chão e ao declive máximo do veículo concreto, enquanto os propriocetivos derivam o risco da interação já sentida entre o robô e o terreno, por vibração, deslizamento ou binário, e por isso só existem depois de o terreno ter sido experimentado.

O domínio florestal é onde esta distinção se torna mais visível na literatura recente, com trabalhos relevantes em ambos os lados da árvore. Do lado aparente, o ForestTrav estima traversabilidade só a partir de LiDAR em floresta densa, combinando uma representação voxel probabilística com redes convolucionais esparsas, e aprende a separar vegetação penetrável de obstáculo sólido apenas pela forma da nuvem de pontos, sem depender de nenhum parâmetro do robô, obtendo um coeficiente de correlação de Matthews de 0,62 contra 0,41 dos métodos volumétricos e 2.5D anteriores à resolução de voxel de 0,1 m [40]. Do lado relativo, há quem aprenda o custo a partir de pistas visuais e geométricas mas rotule o treino com informação propriocetiva, como a rugosidade e os solavancos que o próprio robô sentiu ao atravessar cada zona [41], e quem dispense por completo a anotação humana, combinando modelos de fundação visuais com mapeamento geométrico para produzir mapas de custo, de velocidade e de incerteza a partir de cerca de quarenta minutos de condução de um perito [42]. Estes dois últimos mostram que a fronteira entre aparente e relativo é sobretudo uma questão de onde vêm os rótulos, e não de que sensor observa a cena.

2.5.3 Representações 2.5D e volumétricas

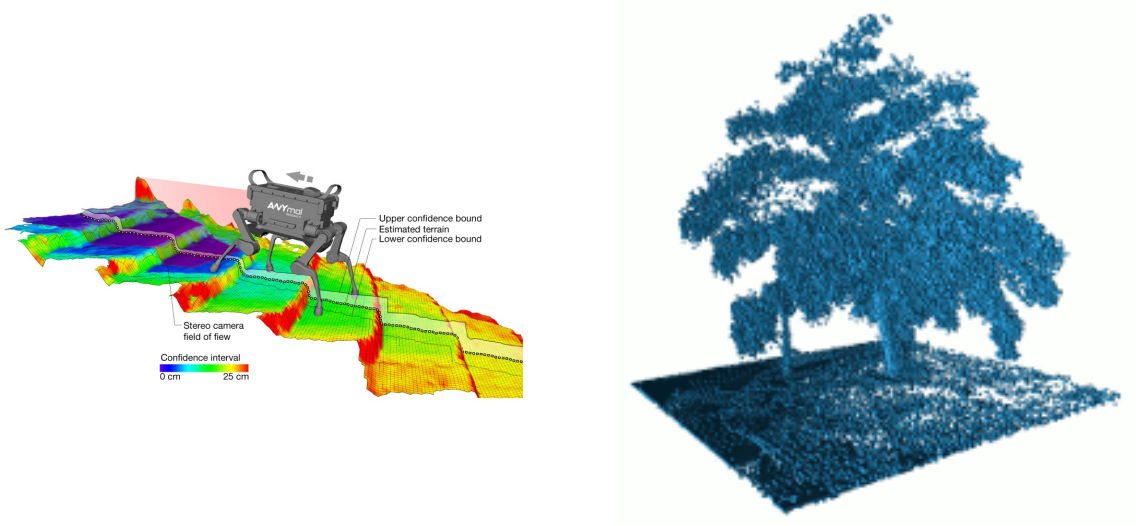
Depois de decidido que valor guardar por célula e para quem esse valor vale, falta decidir como organizar essa informação no espaço. Duas famílias dominam a literatura de mapeamento para navegação em robótica de campo.

A primeira família é a representação 2.5D, tipicamente uma grelha de elevação em que cada posição (x, y) guarda um único valor de altura. Fankhauser *et al.* propõem um dos exemplos mais influentes desta família, uma grelha de elevação probabilística que funde medições de um sensor de distância com a incerteza da própria localização do robô, produzindo por célula não só uma altura estimada mas também um limite de confiança superior e inferior [43]. A Figura 2.10a ilustra o resultado deste tipo de grelha, com a banda de confiança visível como uma faixa colorida à volta da superfície estimada. Esta representação é leve e rápida de atualizar, e chega perfeitamente quando o terreno é essencialmente uma única superfície contínua.

A segunda família é a representação volumétrica, que guarda ocupação ou distância em três dimensões completas em vez de um único valor de altura por posição (x, y). Esta família é necessária quando duas estruturas relevantes ocupam a mesma posição (x, y) mas alturas diferentes, por exemplo a copa de uma árvore e uma rocha abaixo dela, um caso que uma grelha 2.5D não consegue representar por só reservar lugar para uma altura em cada posição.

O que distingue as sub-famílias dentro desta é a forma como o volume é organizado e o que cada célula guarda. Umas guardam ocupação e poupam memória variando a resolução, mantendo células pequenas apenas junto das superfícies ocupadas e células grandes nas regiões vazias afastadas delas, que é o princípio da octree popularizado pelo OctoMap [44] e o que a Figura 2.10b ilustra. Outras guardam antes, em cada célula, a distância assinada à superfície mais próxima, positiva fora do obstáculo e negativa dentro dele. São os campos de distância assinada, que produzem um campo contínuo em vez de blocos discretos e permitem ao planeamento saber, em qualquer ponto do espaço, a que distância fica o obstáculo mais próximo, garantindo assim uma folga mínima. Existem duas variantes com papéis distintos, a truncada (*Truncated Signed Distance Field* (TSDF)), mais

adequada à reconstrução da geometria, e a euclidiana (*Euclidean Signed Distance Field* (ESDF)), mais adequada ao planeamento, e é essa ponte entre as duas que sistemas como o Voxblox resolvem na prática [45].



(a) Grelha de elevação 2.5D (b) Representação volumétrica por octree

Figura 2.10: Duas formas de organizar espacialmente o ambiente. (a) adaptada de Fankhauser *et al.* [43]. (b) figura retirada de Hornung *et al.* [44].

A escolha entre 2.5D e volumétrico é, no fundo, um compromisso entre memória e computação de um lado e expressividade do outro. Uma grelha 2.5D custa pouco e chega para a maioria do terreno percorrido por um robô terrestre. Uma representação volumétrica custa mais mas é a única forma de capturar corretamente estruturas com saliências ou sobreposições, como acontece frequentemente sob um dossel florestal. Uma terceira via, intermédia entre estas duas, é representar apenas objetos discretos identificados no ambiente, como troncos individuais, por uma lista de marcos vetoriais em vez de os densificar numa grelha ou num volume, reservando a representação densa só para o terreno contínuo.

2.6 Planeamento de Trajetória e Decisão de Locomoção

Um robô autónomo que se move num ambiente que não conhece por completo tem de tomar decisões a três escalas diferentes. Primeiro precisa de decidir, de forma geral, que caminho seguir desde a posição atual até ao objetivo. Depois precisa de ajustar esse caminho a cada instante, porque os sensores vão revelando obstáculos que o plano inicial não previa. E, no caso de um robô capaz de se mover de mais do que uma forma, ainda precisa de decidir quando faz sentido mudar de modo de locomoção. As três subsecções seguintes seguem esta divisão. Convém desde já separar todas elas do problema de converter um caminho já aceite em comandos fiéis para os atuadores, que é um problema de controlo de seguimento e é tratado na Secção 2.7.

2.6.1 Planeamento global em ambientes parcialmente conhecidos

O planeamento global procura o caminho de menor custo entre a posição do robô e o objetivo, sobre uma representação do ambiente que varia de família para família, seja uma grelha de células ligadas entre si, seja uma amostragem direta de pontos no espaço contínuo. Essa representação, construída a partir dos sensores como descrito na Secção 2.5, é o mapa sobre o qual os algoritmos desta subsecção procuram o caminho, e é precisamente a existência ou não desse mapa que separa os casos mais simples do caso florestal.

O que distingue o caso florestal é que não existe mapa prévio. O robô só conhece o ambiente à medida que os sensores o revelam, pelo que a cada poucos metros surge informação nova que invalida parte do plano. A pergunta deixa por isso de ser qual o algoritmo que encontra o melhor caminho num mapa conhecido, e passa a ser qual deles sobrevive a replanear vezes sem conta sem que o custo se torne proibitivo. É este critério que elimina a maior parte das famílias disponíveis.

A procura clássica em grafo, de que o algoritmo de Dijkstra [46] e o A* [47] são os representantes canónicos, assume que o grafo não muda depois de a procura começar, e por isso qualquer célula que passe a ocupada obriga a recomeçar tudo do zero. A procura em grelha cinematicamente consciente, como o Hybrid-A* [48], incorpora na própria procura as curvas que o veículo consegue realmente fazer, o que resolve bem a manobra apertada mas continua a pressupor um mapa conhecido, e em floresta o que falta não é capacidade de manobra. Os métodos por amostragem, que constroem o caminho amostrando pontos no espaço contínuo em vez de discretizarem o ambiente numa grelha, como o *Rapidly-exploring Random Tree* (RRT) [49], o *Probabilistic Roadmap Method* (PRM) [50] e as suas variantes com garantias de otimalidade assintótica [51], são fortes em espaços de muitas dimensões mas não têm forma nativa de reaproveitar a árvore já construída quando o mapa muda, pelo que voltar a amostrar acaba por custar tanto como recomeçar.

Resta a procura incremental, que é a única família desenhada precisamente para este problema. Em vez de deitar fora o trabalho da procura anterior sempre que uma parte do mapa muda, mantém a maior parte da solução e corrige apenas a zona afetada. O Lifelong Planning A* [52] estabelece o mecanismo. Cada célula guarda dois valores de custo que coincidem quando a solução está correta, e quando uma célula passa de livre a ocupada esses valores deixam de coincidir apenas na vizinhança da mudança, o que permite usar uma fila de prioridade para recalcular só as células afetadas.

O D* Lite [53] constrói-se sobre esse mecanismo e acrescenta-lhe o que falta a um robô que se desloca. Enquanto o Lifelong Planning A* mede os custos a partir da posição inicial, o D* Lite mede-os a partir do objetivo, procurando de trás para a frente. A diferença parece pequena mas é decisiva, porque custos medidos a partir do objetivo não dependem de onde o robô está, e continuam por isso válidos à medida que ele avança pelo caminho. A Figura 2.11 compara os dois. Repare-se que, perante o mesmo obstáculo, as células que perdem validade ficam de lados opostos, além do obstáculo num caso e aquém dele no outro, precisamente porque a âncora do cálculo é diferente. Note-se ainda que as células recalculadas não são só as do caminho, são todas aquelas cujo custo até à âncora passava pelo corredor agora bloqueado, enquanto as linhas que já

contornavam o obstáculo mantêm os valores que tinham.

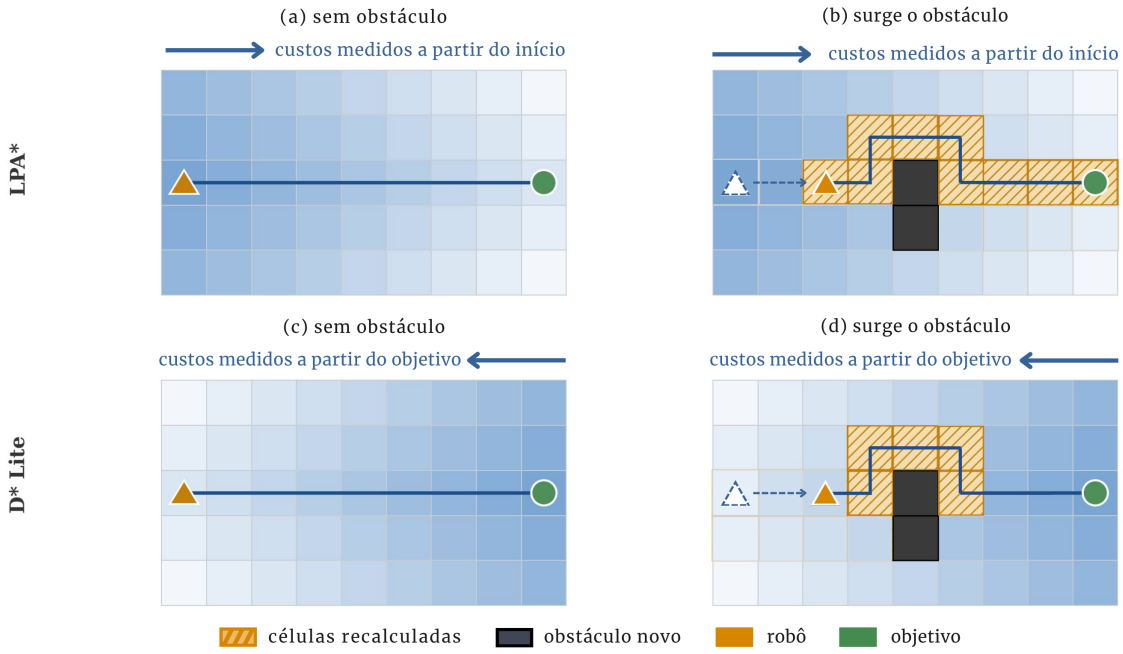


Figura 2.11: Procura incremental perante o mesmo obstáculo. Em cima o Lifelong Planning A*, medido a partir do início. Em baixo o D* Lite, medido a partir do objetivo. Ilustração própria dos conceitos descritos em Koenig et al. [52] e Koenig e Likhachev [53].

2.6.2 Planeamento local e evitamento

Junto do robô existem dois problemas distintos que a literatura nem sempre separa com clareza. O primeiro é o seguimento, que consiste em converter o caminho recebido numa sequência de comandos executados com fidelidade, e é um problema de controlo tratado na Secção 2.7. O segundo é o evitamento, que consiste em alterar a trajetória em resposta a obstáculos que o plano global desconhecia, e é o objeto desta subsecção. O critério que os separa é se o método consulta ou não o mapa de obstáculos no momento em que decide o comando. Um controlador de seguimento executa o caminho que lhe é entregue, seja ele qual for, ao passo que um planeador local pode recusar esse caminho quando ele conduz a uma colisão.

As famílias de planeamento local dividem-se conforme decidam apenas a partir do instante presente ou simulem o futuro antes de agir, e é essa a divisão seguida nesta subsecção.

As famílias reativas decidem a partir do instante presente, sem simular o futuro. O *Dynamic Window Approach* (DWA) [54] procura no espaço das velocidades lineares e angulares que o robô consegue mesmo atingir no próximo instante, dadas as suas acelerações máximas, e começa por descartar todas as que não lhe permitiriam travar antes de embater. De entre as que sobram, escolhe depois a que melhor concilia aproximar-se do objetivo, manter distância aos obstáculos e andar depressa, e é essa ponderação, ao favorecer as velocidades angulares que curvam para longe do obstáculo, que lhe permite contorná-lo e não apenas parar diante dele. Os *Artificial Potential*

Fields (APF) [55], ou campos potenciais, seguem uma lógica diferente e tratam o problema como um campo de forças, somando uma atração ao objetivo com uma repulsão de cada obstáculo. Ambas são muito baratas de calcular, mas também míopes, e os campos potenciais podem ainda ficar presos em mínimos locais, situações em que as forças se anulam antes de o robô chegar ao destino.

As famílias preditivas simulam o futuro antes de agir, e é daí que vem tanto a sua capacidade como o seu custo. O *Model Predictive Path Integral* (MPPI) [56] gera a cada ciclo um grande número de sequências de comandos ligeiramente diferentes, simula cada uma ao longo de um horizonte curto, avalia o seu custo e devolve uma média ponderada que dá muito mais peso às de menor custo. Como não precisa de calcular gradientes, lida bem com custos irregulares e não diferenciáveis, mas exige simular centenas ou milhares de trajetórias por ciclo, o que se torna pesado sem uma unidade de processamento gráfico dedicada.

O *Model Predictive Control* (MPC) clássico segue a mesma lógica preditiva mas resolve, em vez de amostrar, um problema de otimização exato a cada ciclo, prevendo a evolução do sistema a partir de um modelo linear da sua dinâmica e aplicando só o primeiro comando da solução antes de repetir o processo no ciclo seguinte [57]. Essa formulação linear e convexa resolve-se de forma rápida e barata através de programação quadrática, mas não serve bem ao caso florestal, porque o espaço livre à volta de um obstáculo de forma irregular não é convexo e a dinâmica de um veículo sobre terreno acidentado não se comporta bem como uma aproximação linear. O *Nonlinear Model Predictive Control* (NMPC) remove ambas as suposições ao resolver o problema sobre o modelo dinâmico real, não linear, do veículo, impondo restrições explícitas de velocidade, de aceleração e de distância mínima a cada obstáculo. É por via desta última que faz evitamento e não mero seguimento, e é essa formulação com restrições rígidas que garante, dentro do horizonte otimizado, que essas restrições são respeitadas, ao contrário dos métodos por amostragem, que só as respeitam de forma aproximada. Em troca, o algoritmo de otimização exige que o custo e as restrições sejam diferenciáveis, o que obriga a aproximar a forma irregular dos obstáculos por envolventes suaves, e o seu custo cresce rapidamente com o número de obstáculos, com a complexidade do modelo e com o horizonte escolhido. A Figura 2.12 mostra como o mesmo obstáculo entra na decisão em cada uma das quatro famílias.

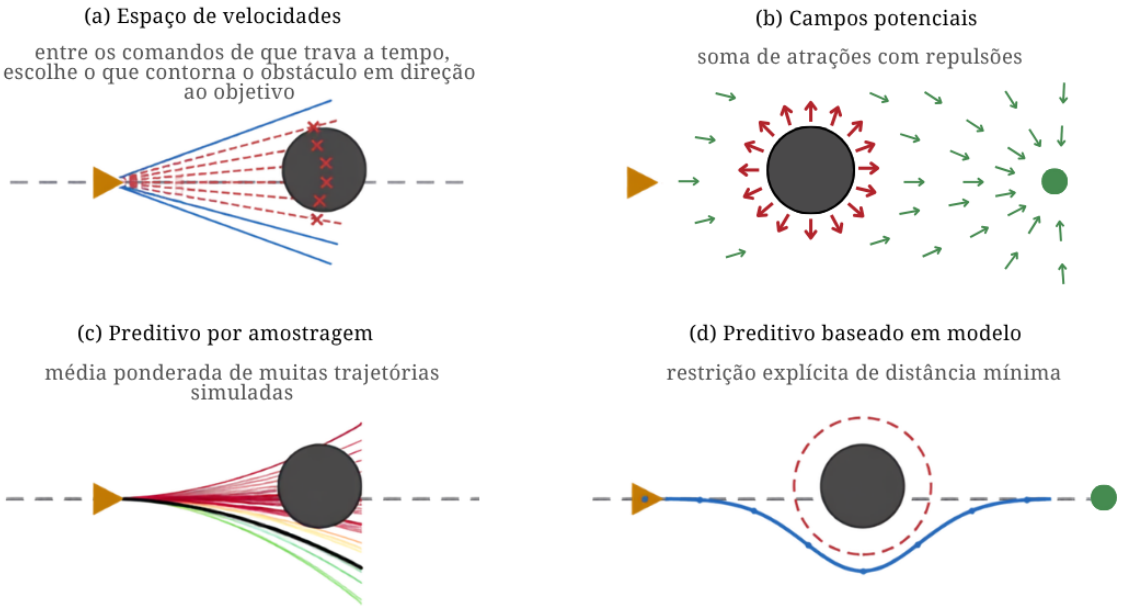


Figura 2.12: Como o mesmo obstáculo entra na decisão em cada família de planeamento local. Ilustração própria dos comportamentos descritos em Fox et al. [54], Khatib [55], Williams et al. [56] e Mayne [57].

2.6.3 Decisão híbrida

Esta subsecção trata de um problema que não se enquadra nas duas anteriores, porque não é encontrar um caminho nem reagir a obstáculos ao longo dele, mas decidir qual a forma de locomoção a usar em cada troço. Surge apenas em veículos capazes de se deslocar de mais do que uma forma, por exemplo em modo terrestre e em modo aéreo, e consiste em decidir quando e onde é vantajoso mudar de modo. Essa decisão implica custos próprios, como o tempo e a energia despendidos na transição. A literatura propõe três formas distintas de a resolver, descritas de seguida por ordem crescente de sofisticação.

- **Otimização do ponto de comutação** [6] formula o problema como uma procura em grafo sobre um espaço combinado, em que cada nó representa não só uma posição mas também o modo de locomoção associado a essa posição. O algoritmo alterna entre uma procura bidimensional, mais leve, enquanto o veículo se desloca em modo terrestre, e uma procura tridimensional, mais pesada, restrita aos troços em modo aéreo, reduzindo assim o custo computacional global. O resultado não é apenas um caminho, é também a determinação de em que pontos convém descolar e em que pontos convém aterrar.

- **Planeamento hierárquico com controlo unificado** [5] gera uma única trajetória que já integra troços terrestres e troços aéreos. O planeador de nível mais alto privilegia sistematicamente o modo terrestre por este consumir menos energia, e só atribui um troço ao modo aéreo quando a rota terrestre não é segura ou não é exequível. Uma camada de controlo

unificada, de nível mais baixo, ajusta depois dinamicamente o consumo de energia durante os troços percorridos em solo.

- **Modelo dinâmico unificado** [7] elimina a própria separação entre os dois modos. Em vez de tratar o troço terrestre e o troço aéreo como problemas distintos que depois se costuram, os autores derivam um único modelo dinâmico que descreve o veículo nas duas formas de locomoção, e mostram que esse modelo goza de planitude diferencial, uma propriedade que permite gerar trajetórias dinamicamente viáveis com pouco cálculo. Sobre esse modelo, um otimizador produz a trajetória e os autores recorrem ao NMPC, aqui no papel de controlador de seguimento e não no de evitamento visto na Secção 2.6.2, para a seguir e assegurar transições suaves entre modos. Como o modelo é o mesmo dos dois lados, a mudança de modo deixa de ser um evento a gerir e passa a ser mais um troço da mesma trajetória contínua.

2.7 Controlo e Seguimento de Trajetória

Depois de decidido o caminho, falta convertê-lo em comandos reais para os atuadores, respeitando a dinâmica do veículo e corrigindo os desvios que forem surgindo. É esse o papel do controlo, que se ocupa de como um comando de velocidade linear e angular se converte em movimento real, de forma estável e precisa, tanto em solo como no ar, sem qualquer noção de obstáculos. Como já ficou estabelecido na Secção 2.6.2, é essa ausência de noção de obstáculos que distingue um controlador de seguimento de um planeador local. O MPPI e o NMPC atravessam essa fronteira consoante a formulação escolhida, porque o mesmo formalismo pode incluir ou não um termo de obstáculos na sua função de custo, e por isso reaparecem nesta secção na versão sem esse termo. Já o *Pure Pursuit* nunca a atravessa, pois é um controlador de seguimento puro, e é como tal que é aqui apresentado.

2.7.1 Controlo de seguimento em solo

O problema desta subsecção é reduzir a zero o desvio entre o robô e um caminho de referência já dado como seguro, e existem três níveis de rigor para o atacar, por ordem crescente de exigência.

O *Pure Pursuit* escolhe um ponto no caminho de referência a uma certa distância à frente do robô, chamada distância de antecipação, e calcula a curvatura do arco de circunferência que liga a posição atual do robô a esse ponto [58]. Essa curvatura é convertida diretamente em velocidade angular de comando. O método é muito barato em termos de computação e não olha para a dinâmica do veículo, mas com uma distância de antecipação curta tende a oscilar, e com uma distância longa tende a cortar as curvas em vez de as seguir de perto. Este compromisso é o seu principal parâmetro de afinação, e várias extensões procuram resolvê-lo tornando a distância de antecipação variável com a velocidade e com a curvatura local do caminho [59].

Kanayama et al. definem o erro de seguimento no referencial do robô, como a diferença de posição lateral, longitudinal e de orientação face a um ponto de referência que avança ao longo do caminho a uma velocidade prevista [60]. A lei de controlo soma a essa velocidade uma correção

proporcional aos três erros, com três ganhos livres desde que positivos. Provam, com uma função de Lyapunov, que o erro converge para zero, e indicam como escolher os ganhos para uma resposta bem amortecida.

O terceiro nível é o NMPC de seguimento, uma forma mais restrita do formalismo já apresentado na Secção 2.6.2, que aqui não tem noção de obstáculos e se limita a minimizar o erro entre a trajetória prevista e a de referência, sujeito a restrições físicas como velocidade e aceleração máximas. A vantagem face às duas leis anteriores é que essas restrições nunca são violadas pela solução, ao contrário de uma lei simples, onde só a saturação do comando evita ultrapassar os limites. Ostafew et al. propõem o *Learning-Based Nonlinear Model Predictive Control* (LB-NMPC), que combina um modelo cinemático nominal com uma perturbação aprendida por processo Gaussiano a partir da experiência do robô. Numa versão robusta, as restrições tornam-se mais conservadoras enquanto essa estimativa não for fiável. Desta forma corrigem-se os erros de modelo causados pelo terreno irregular sem construir um modelo dinâmico mais complexo [61, 62]. A Figura 2.13 resume os três níveis lado a lado.

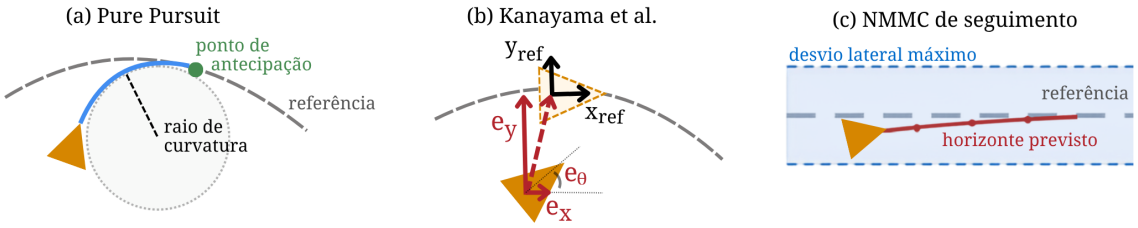


Figura 2.13: Os três níveis de rigor no controlo de seguimento em solo. Diagrama próprio, ilustrativo dos princípios descritos em Coulter [58], Kanayama et al. [60] e na formulação geral de NMPC de seguimento.

A diferença entre o *Pure Pursuit* e a lei de Kanayama está na garantia formal de convergência, que o primeiro não oferece porque nunca modela o erro que está a tentar anular. Em troca dessa garantia, a lei de Kanayama assume que o modelo cinemático do veículo é conhecido com precisão. Essa suposição é razoável para uma roda motriz por lado, que nunca escorrega porque nunca é obrigada a duas velocidades ao mesmo tempo, mas falha num veículo com vários pontos de contacto rígidos por lado, como um *skid-steer* ou uma lagarta, em que o *slip* cresce com a curvatura e com a irregularidade do terreno. Trivedi et al. propõem um modelo para este caso que aprende, por processo Gaussiano, a diferença entre o movimento previsto pelo modelo simples e o movimento real, dependente do tipo de terreno [63]. É a mesma ideia de correção aprendida já vista no LB-NMPC, aqui aplicada a uma lei mais simples e sem as garantias de restrição rígida que o formalismo de otimização oferece.

Um problema mais exigente surge num veículo cujas pernas servem duas funções distintas, ora como rodas em solo, ora como vetorização de empuxo no ar, e cujo controlador de voo tem por isso de decidir sobre ambas ao mesmo tempo. Salagame et al. desenvolvem um NMPC para o M4, um robô que transita entre solo e ar reaproveitando as pernas articuladas como vetorização de empuxo em voo [64]. Durante o voo, o mesmo problema de otimização decide a postura das

pernas e o comando dos rotores a cada instante, o que permite recuperar de falhas de atuador ao reconfigurar a postura em tempo real. Isto mostra que o NMPC já é tratado como um formalismo capaz de unificar, num único problema, decisões sobre atuadores que servem funções diferentes consoante o modo de locomoção.

2.7.2 Controlo aéreo

Um veículo aéreo é um problema de controlo mais exigente do que um terrestre porque não tem o solo a suportá-lo, é inerentemente instável em translação, e depende de controlo contínuo da orientação só para se manter no ar. Por isso organiza-se numa cascata de três malhas. A externa decide a posição desejada, a intermédia converte isso em orientação e empuxo total desejados, e a interna ajusta a velocidade de cada motor. É esta malha interna a mais rápida das três, a correr tipicamente perto de mil ciclos por segundo, uma frequência de atualização muito acima das dezenas típicas de um veículo terrestre.

A malha de orientação tem sido tradicionalmente descrita com três ângulos de Euler, um por eixo de rotação, o *roll*, o *pitch* e o *yaw*. Esta representação tem uma limitação conhecida, a singularidade ou bloqueio de cardan. Ela ocorre quando dois eixos ficam alinhados e dois ângulos passam a descrever o mesmo movimento, fazendo desaparecer da representação um grau de liberdade que continua a existir fisicamente. Isto acontece, por exemplo, quando o ângulo de *pitch* se aproxima de 90°, momento em que os eixos de *roll* e de *yaw* se alinham e uma rotação sobre um deixa de se distinguir de uma rotação sobre o outro [65]. A Figura 2.14 mostra quando é que isto acontece.

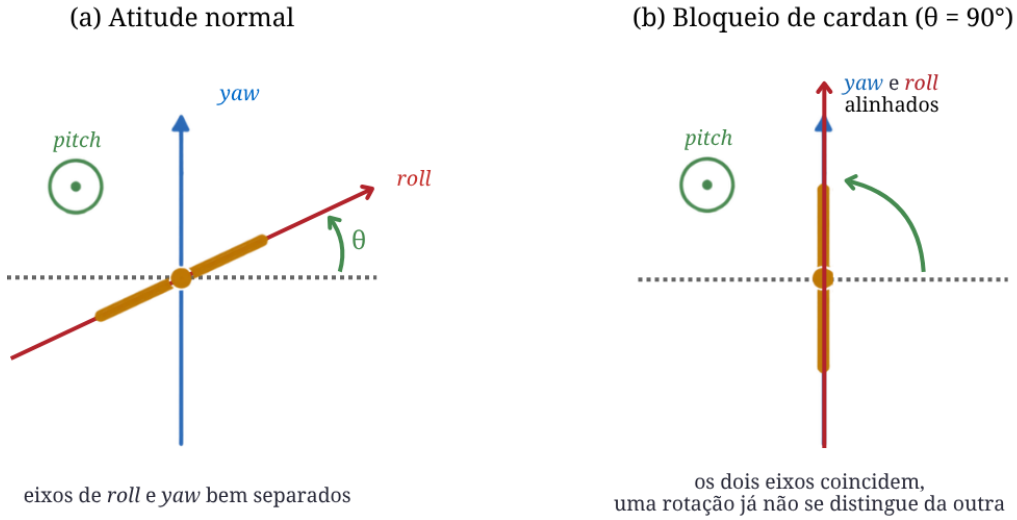


Figura 2.14: A singularidade da representação por ângulos de Euler (bloqueio de cardan). Diagrama próprio.

Para evitar esta singularidade, Lee, Leok e McClamroch descrevem a orientação diretamente como uma rotação em SO(3) e o movimento completo do veículo em SE(3), sem nunca a decompor em ângulos, e mostram que o erro converge para zero a partir de quase qualquer condição inicial,

não só perto do equilíbrio como acontece nos controladores lineares por ângulos de Euler [65]. Qualquer destes controladores depende de conhecer a orientação real do veículo, que não vem diretamente dos sensores, porque o giroscópio deriva ao longo do tempo e o acelerómetro é perturbado por qualquer aceleração que não seja a gravidade. É por isso necessário um filtro que combine as duas medidas e corrija essa deriva sem o custo de um filtro de Kalman completo, e em dados reais de voo um filtro complementar não linear mostrou-se mais próximo da referência do que uma alternativa por quaterniões ou um filtro de Kalman estendido [66, 67, 68]. Implementar de raiz toda esta cascata, da estimação de atitude à alocação de motores, é uma tarefa de engenharia considerável, e por isso a maioria dos projetos recorre antes a um autopiloto já desenvolvido e testado, dos quais o PX4 e o ArduPilot são os mais usados tanto em investigação como na indústria [69]. A Tabela 2.4 resume as suas características.

Tabela 2.4: Comparação entre o PX4 e o ArduPilot, os dois autopilotos *open source* mais usados, com base nos dados de Aliane [69].

Critério	**ArduPilot**	**PX4**
Ano / país de origem	2009, Estados Unidos	2011, Suíça
Veículos suportados	Multirrotor, asa fixa, terrestre, aquático	Multirrotor, asa fixa, VTOL, terrestre, aquático
RTOS	ChibiOS, NuttX	NuttX
Licença	GNU-GPL (redistribuição obrigatoriamente aberta)	BSD (redistribuição livre, mesmo fechada)
Protocolo de comunicação	MAVLink, CAN	MAVLink, UAVCAN, RTPS (para ROS 2)
Integração ROS	Forte	Forte
Suporte MATLAB/Simulink	Moderado	Forte
Simulação *Software In The Loop* (SITL) / *Hardware In The Loop* (HITL)	Forte	Forte
Computador companheiro	Forte	Forte

2.8 Simulação, Geração Procedimental e Distribuição Natural

2.8.1 Simulação em Robótica de Campo

Para um robô de campo, testar diretamente no ambiente real tem um custo elevado. Um teste falhado numa floresta pode significar uma peça partida, um sensor danificado ou várias horas perdidas em deslocação e preparação. A simulação surge como resposta a este custo, porque

permite repetir o mesmo cenário quantas vezes forem necessárias, controlar uma única variável de cada vez e criar situações raras à vontade, sem esperar que aconteçam naturalmente no terreno.

Existem hoje vários motores de simulação usados em robótica de campo, e cada um faz um compromisso diferente entre fidelidade física, integração com ROS e custo de utilização. O Gazebo é provavelmente o mais usado em investigação académica ligada a ROS, por ser gratuito, de código aberto e bem integrado com o ecossistema ROS 2. O Isaac Sim, da NVIDIA, oferece maior fidelidade de renderização e aceleração por GPU, à custa de um ecossistema mais fechado. Motores construídos sobre o Unity ou o Unreal Engine, como o AirSim, privilegiam sobretudo o realismo visual, o que os torna atrativos para gerar dados sintéticos de treino de visão por computador. O Webots é outra alternativa aberta, mais orientada à robótica em geral do que especificamente a ambientes exteriores.

Uma razão recorrente na literatura para recorrer a simuladores não é a comodidade, é a segurança e o custo. Wiberg *et al.* resumem isto de forma direta ao afirmarem que os simuladores são uma opção atrativa para treinar controladores porque aliviam o problema do tempo, da segurança e do custo [70]. Os mesmos autores sublinham que este problema se agrava em veículos pesados a operar em ambientes hostis, como na construção, na mineração ou na floresta, onde a recolha de dados reais é cara e demorada. O seu caso de estudo, uma máquina florestal real com um sistema de suspensão hidráulico complexo, treina o controlo inteiramente em simulação antes de o testar na máquina física.

O mesmo padrão aparece na robótica de pernas. O grupo de Marco Hutter, na ETH Zurique, treina o controlador de locomoção do robô quadrúpede ANYmal inteiramente em simulação e só depois o testa em ambiente real, incluindo trilhos de montanha, ribeiros, lama, neve e floresta húmida [71]. Uma versão posterior do mesmo controlador, que já combina perceção visual com o controlo de locomoção, identifica a vegetação como um dos tipos de obstáculo visualmente mais difíceis de interpretar corretamente [72]. Em nenhum dos dois casos o robô aprendeu a andar no terreno real, aprendeu tudo em simulação e só depois foi posto à prova fora dela.

Esta vantagem tem um preço. Nenhuma simulação reproduz exatamente o comportamento físico do mundo real, e a diferença entre os dois é conhecida como o fosso *sim-to-real*. Parâmetros como o atrito, a massa exata de cada componente ou o desgaste mecânico ao longo do tempo são difíceis de medir e de modelar com precisão, e um controlador que funciona bem em simulação pode falhar de forma imprevisível quando é transferido para o robô real.

2.8.2 Geração Procedimental de Terreno

Depois de decidido como simular e com que fidelidade, falta o próprio conteúdo do mundo, o terreno sobre o qual tudo o resto acontece. Gerar manualmente um terreno grande e variado é lento, por isso a solução comum é gerar o terreno por um processo procedimental, um algoritmo que produz uma superfície nova a cada execução, ou a partir de um valor de semente, em vez de um humano desenhar cada elevação à mão.

A técnica mais usada para isto tem origem numa função de ruído. Ken Perlin introduziu, no seu artigo de 1985, um gerador de ruído contínuo capaz de produzir texturas e superfícies com um

aspeto natural, ao contrário do ruído aleatório puro, que não tem qualquer coerência espacial e por isso parece estático de televisão em vez de uma textura real [73]. Perlin usou esta função para gerar nuvens, fogo, água e madeira, entre outros efeitos, mas a mesma ideia aplica-se diretamente a um terreno, uma superfície de altura em que cada ponto do plano recebe um valor suave e contínuo em vez de um valor desconexo do seu vizinho. Sobrepor várias camadas desta função de ruído, cada uma numa escala espacial diferente, é uma prática comum para dar a um terreno relevo em várias escalas ao mesmo tempo, colinas largas vindas de uma camada e aspereza fina vinda de outra. Esta mesma técnica continua a aparecer em trabalho recente de robótica de campo, Wiberg *et al.* usam ruído de Perlin para gerar o terreno onde treinam o controlo da sua máquina florestal, com pequenos elipsoides embebidos a representar pedregulhos [70].

O ruído coerente não esgota o campo da modelação de terreno. Uma revisão recente organiza os métodos existentes em três grandes famílias, nomeadamente os métodos procedimentais como o já descrito, os métodos baseados em simulação física de processos erosivos, e os métodos baseados em exemplos, que aprendem a gerar terreno novo a partir de dados de terreno real já existente [74].

Aplicada especificamente a ambientes florestais, a geração procedimental de terreno já tem pelo menos uma ferramenta de código aberto publicada. Newlands e Zauner apresentam um sistema que gera árvores individuais por L-systems especializados e depois distribui essas árvores pelo terreno através de um algoritmo de simulação de ecossistema, seguido de um *pipeline* de *rendering* diferido que produz cenas de floresta navegáveis e visualmente credíveis [75], como mostra a Figura 2.15. O código deste sistema está publicamente disponível, o que o torna um precedente direto e verificável de que gerar procedimentalmente um ambiente florestal completo, terreno e árvores incluídos, é uma abordagem já validada por outros grupos de investigação.



Figura 2.15: Cena de floresta gerada pela ferramenta de código aberto de Newlands & Zauner, com árvores individuais por L-systems distribuídas por simulação de ecossistema. Figura retirada do artigo original [75].

2.8.3 Distribuição Natural de Obstáculos em Floresta

Nem toda a floresta está distribuída da mesma forma no espaço, e a estatística espacial oferece vários modelos para descrever essa distribuição, uns mais realistas do que outros consoante o tipo de floresta em causa.

O modelo mais simples é o processo de Poisson homogéneo, em que cada árvore é colocada de forma independente e com a mesma probabilidade em qualquer ponto do espaço. Este modelo corresponde à ideia de aleatoriedade espacial completa, e serve sobretudo como modelo nulo de comparação, não como descrição realista de uma floresta. Ramón, Velázquez, Escudero e de la Cruz testaram este pressuposto contra dados reais de 229 espécies lenhosas na parcela de dinâmica florestal de Barro Colorado, no Panamá, e o resultado foi claro. Todas as 229 espécies mostraram um padrão agregado, e não aleatório, independentemente do seu mecanismo de dispersão de sementes [76]. A maioria delas, 63%, foi melhor descrita por um processo de Poisson por *clusters* inomogéneo, e não por qualquer variante de aleatoriedade uniforme. Este resultado sustenta a ideia de que agregação, e não uniformidade, é a norma esperada na distribuição espacial de árvores.

Se a agregação é a regra, o passo seguinte é escolher um modelo matemático capaz de a produzir. Ngo Bieng, Ginisty e Goreaud propõem, para povoamentos mistos de carvalho e pinheiro silvestre na floresta de Orleães, em França, um modelo combinado que junta um processo de Poisson, um processo de Neyman-Scott e um mecanismo de inibição de núcleo suave [77]. O processo de Neyman-Scott é a formalização matemática da agregação em torno de árvores-mãe, pontos-pai distribuídos aleatoriamente pelo espaço, cada um com um número aleatório de descendentes

espalhados à sua volta segundo uma distribuição gaussiana. O mecanismo de inibição resolve o problema oposto, o de duas árvores adultas não poderem fisicamente ocupar o mesmo lugar, ao reduzir a probabilidade de dois pontos aparecerem demasiado próximos um do outro em vez de a proibir de forma rígida. A combinação destes dois efeitos, agregação por um lado e inibição por proximidade por outro, é a família de modelos que a literatura de estatística espacial florestal considera hoje mais adequada para descrever povoamentos reais.

Além das árvores adultas, uma floresta natural tem também zonas de menor densidade, onde uma árvore antiga caiu ou morreu e abriu uma clareira. Estas clareiras não são zonas vazias por acaso, são zonas ecologicamente ativas. Zhu, Lu e Zhang reúnem numa meta-análise 527 observações de 42 estudos diferentes e concluem que a densidade de plantas lenhosas em regeneração é, em média, 359% maior dentro de clareiras do que no sub-bosque fechado à sua volta [78]. Uma clareira, portanto, não deve ser entendida como um vazio no modelo espacial, mas antes como uma zona onde a composição etária muda, com poucas árvores grandes mas potencialmente muita regeneração jovem.

Capítulo 3

Desenvolvimento Experimental

O sistema construído nesta dissertação divide-se em sete camadas, cada uma responsável por uma parte distinta do problema de navegação autónoma numa floresta. A Figura 3.1 mostra como essas camadas se encaixam e que dados cada uma entrega à seguinte, desde o ambiente simulado que gera os dados de sensor até à atuação final sobre as lagartas e as hélices, com as setas cheias a seguir esse percurso principal de dados e de volta ao ambiente. O Modo Híbrido não ocupa uma posição fixa neste fluxo porque é antes um sinal transversal, presente em quatro das outras camadas e responsável por avisar cada uma delas de que o robô está a conduzir ou a voar, o que a figura assinala com as setas tracejadas laranja.

A Tabela 3.1 resume a tecnologia central escolhida em cada camada. As secções seguintes deste capítulo percorrem cada uma delas por esta ordem e explicam a tecnologia escolhida, a justificação face às alternativas discutidas nos Fundamentos e o que foi tentado e descartado pelo caminho.

Tabela 3.1: Tecnologia central escolhida em cada camada do sistema.

Camada	**Tecnologia central**
Modo Híbrido Terrestre-Aéreo	Máquina de estados de transição terra-ar e autopiloto dedicado (ArduPilot) para o voo
Perceção	*Pipeline* geométrico sobre LiDAR 3D (segmentação CSF e classificação por PCA) e deteção por câmara (YOLOv8n), combinados por fusão assimétrica
Localização e Mapeamento Simultâneos	Filtro de Kalman estendido em SE(3) e Tree-SLAM por *landmarks* de tronco, com grafo de fatores otimizado por GTSAM e iSAM2
Representação do Terreno	Grelha de elevação 2.5D com mapa de custo contínuo, mantida em blocos persistentes ancorados às poses do SLAM
Planeamento e Decisão	D* Lite como planeador global incremental, com decisão híbrida por comparação direta do tempo estimado por terra e por ar
Controlo	Regulated Pure Pursuit para o seguimento em solo, autopiloto dedicado para o controlo de voo
Simulação e Geração Procedimental	Gazebo com mundos gerados proceduralmente pelo ForestGen

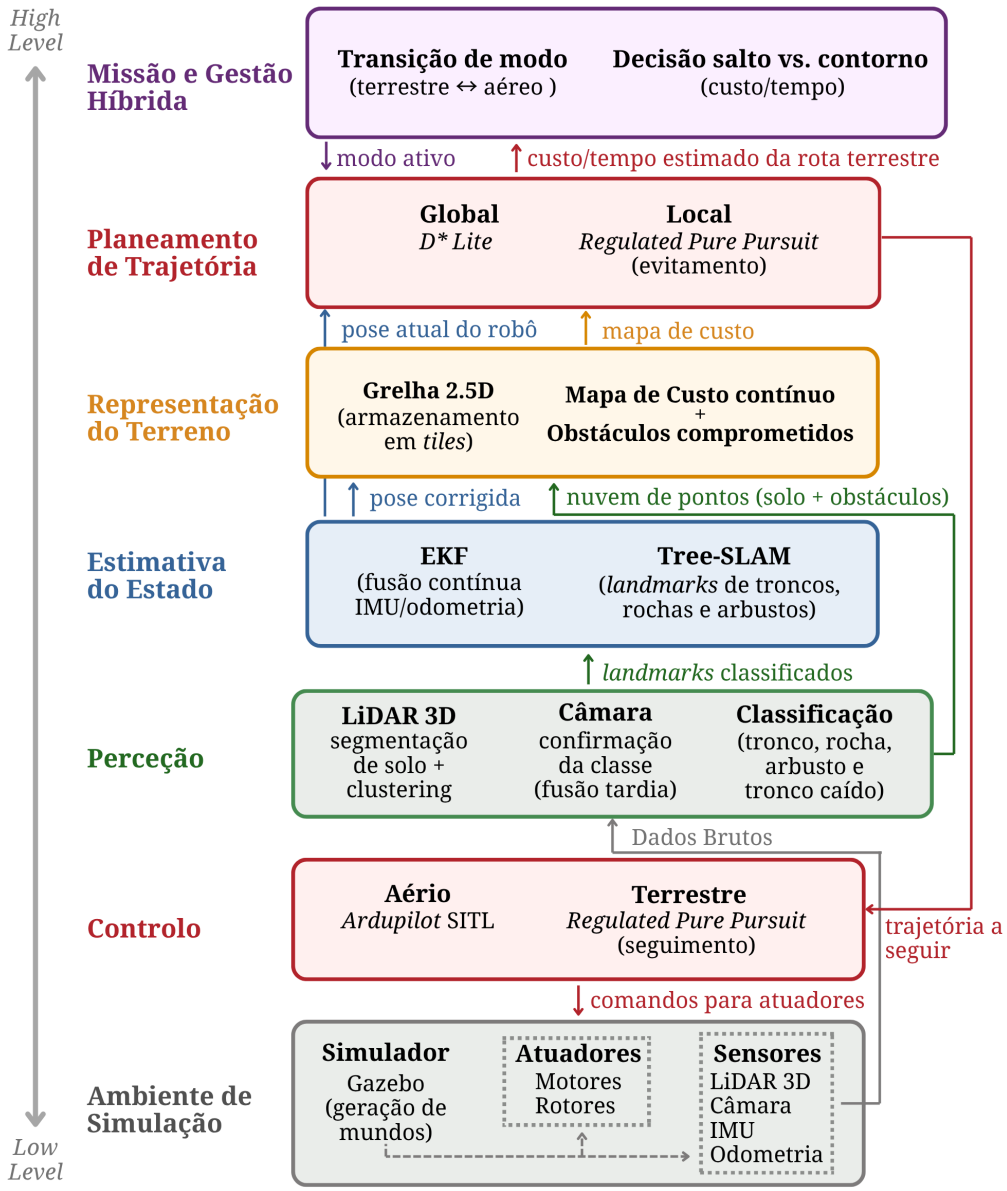


Figura 3.1: Arquitetura do sistema e fluxo de dados entre as sete camadas.

3.1 Idealização do Robô Híbrido

Esta secção descreve o robô tal como foi idealizado para este trabalho, um conceito de plataforma e não um modelo mecânico desenhado ao pormenor nem construído. A ideia de partida foi um único veículo que se desloca em terra sempre que pode e reserva o voo para os obstáculos que não consegue transpor pelo solo, à semelhança das plataformas híbridas discutidas nos Fundamentos.

A primeira intenção foi reaproveitar os motores da propulsão aérea também para a locomoção terrestre, numa lógica de propulsão partilhada que dispensaria atuadores dedicados ao solo. Essa via acabou por ser posta de lado depois de se perceber que atravessar terreno florestal sobre lagartas exige um binário que uma propulsão pensada para o voo dificilmente forneceria de forma adequada. Optou-se então por uma propulsão separada, com um motor dedicado a acionar cada lagarta no solo e outro a acionar cada rotor no voo. Ainda assim, em vez de acrescentar ao veículo estruturas de voo separadas e salientes, os rotores foram alojados dentro da própria estrutura das lagartas, o que mantém a plataforma compacta e integrada. A Figura 3.2 mostra o conceito resultante, em modo terrestre e em modo aéreo.

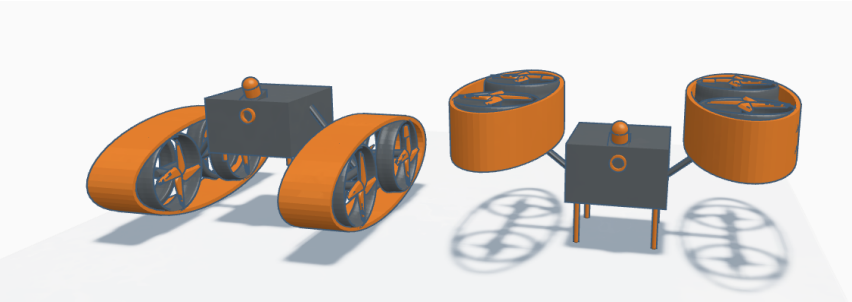


Figura 3.2: Robô híbrido idealizado, em modo terrestre (esquerda) e em modo aéreo (direita), com as lagartas rodadas e os pares de hélices coaxiais expostos.

O robô integra um LiDAR 3D como sensor geométrico principal. A alternativa mais barata seria um LiDAR 2D, e essa hipótese chegou a ser efetivamente tentada, mas revelou-se pouco adequada ao problema. Um LiDAR 2D mede um único plano de cada vez, portanto só produz estrutura tridimensional se for inclinado e for varrendo o terreno à medida que o robô avança, acumulando plano após plano ao longo do tempo. Essa reconstrução fica dependente da qualidade da odometria, uma vez que qualquer erro na pose do robô se traduz diretamente numa distorção do que é reconstruído, e fica também dependente do movimento, porque um robô parado ou a rodar sobre si mesmo deixa de acrescentar informação nova. Um LiDAR 3D resolve as duas coisas ao mesmo tempo, já que capta a estrutura vertical em torno do robô num único disparo, sem depender do movimento nem da odometria. O menor custo do sensor 2D não compensava a dificuldade acrescida que introduzia na perceção.

Independentemente de ser 3D, o LiDAR foi montado inclinado para a frente em vez de na horizontal. A Figura 3.3 ilustra o efeito dessa inclinação, aqui designada por α. Inclinar o sensor faz a sua parte da frente apontar para o solo à frente do robô e a sua parte de trás apontar para

cima, na direção das copas e do céu, pelo que o valor de α controla o compromisso entre estas duas direções.

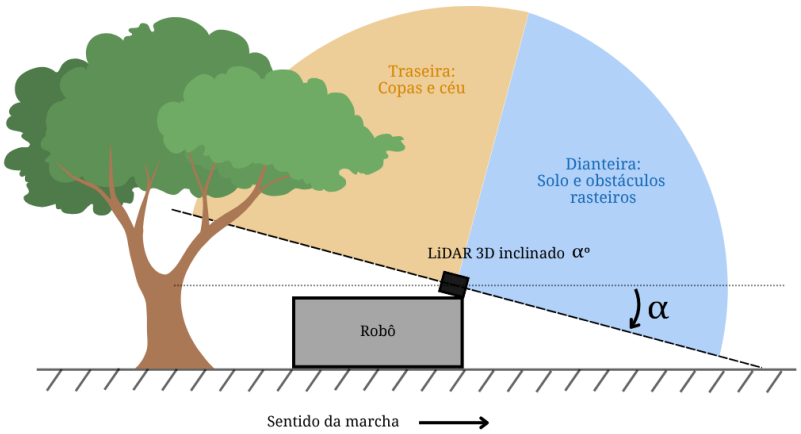


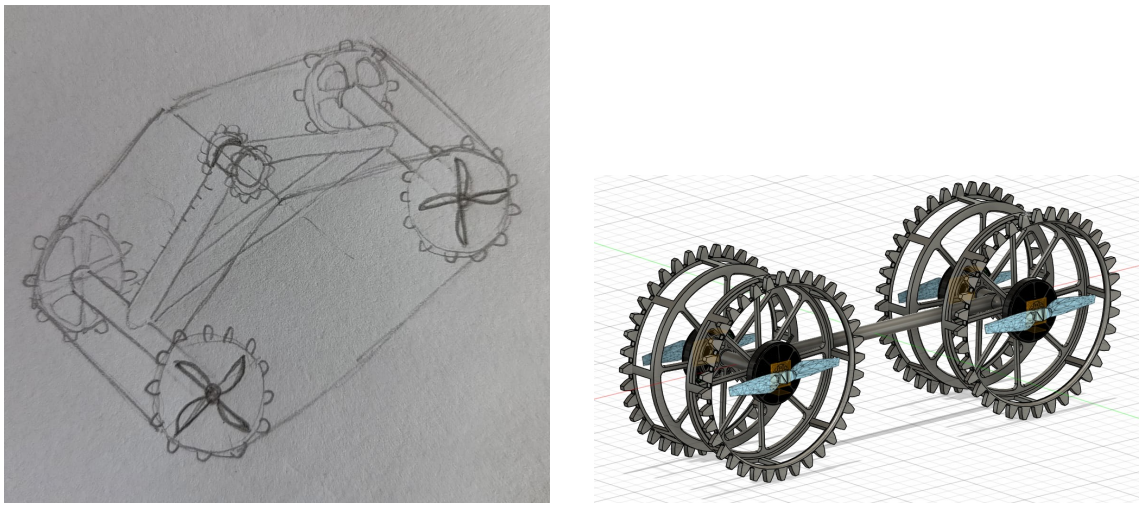
Figura 3.3: Efeito da inclinação α do LiDAR. A parte da frente do sensor lê o solo à frente do robô, enquanto a parte de trás aponta para as copas e o céu.

- Um α maior aproxima ainda mais do robô a leitura do solo à frente, mas faz a parte de trás do sensor apontar quase por inteiro para o céu e para as copas, desperdiçando esses feixes.
- Um α menor preserva mais informação atrás do robô, mas afasta para longe o ponto onde os feixes da frente encontram o solo, o que reduz a utilidade da leitura junto ao veículo.

Depois de experimentar vários ângulos, fixou-se α em 12 graus, um compromisso deliberado entre os dois extremos. Este valor não maximiza nenhum dos dois lados, mas mantém a leitura do solo suficientemente próxima do robô sem sacrificar por completo os dados úteis que a parte de trás do sensor ainda capta a curta distância.

Ao LiDAR juntou-se uma câmara, virada para a frente. Enquanto o LiDAR fornece a geometria, ou seja onde estão os pontos, a câmara traz a aparência que o LiDAR não capta e serve sobretudo para dar sentido a esses pontos, ajudando a distinguir o que é um tronco, uma rocha ou vegetação. Essa informação semântica é depois combinada com a geometria do LiDAR na fusão descrita na camada de Perceção.

Para a locomoção em solo optou-se por lagartas em vez de rodas, por duas razões. As lagartas oferecem melhor tração e estabilidade em terreno florestal irregular, coerente com as adversidades levantadas nos Fundamentos, e a sua geometria alongada dá o volume interno onde alojar os rotores e os restantes componentes da propulsão aérea sem estruturas salientes adicionais. O desenho das lagartas passou por uma fase intermédia de esboço à mão (Figura 3.4a) antes de chegar a uma modelação tridimensional (Figura 3.4b), como resume a Figura 3.4.



(a) (b)

Figura 3.4: Evolução do desenho da lagarta, com (a) o esboço à mão e (b) a modelação tridimensional da mesma.

Nas pontas de cada lagarta foram colocados pares de hélices coaxiais, em vez de uma só hélice por ponta. A intenção foi aumentar o empuxo máximo disponível em cada ponto de sustentação, sabendo à partida que o ganho ficaria abaixo do dobro de uma configuração de hélice única, porque duas hélices coaxiais interferem aerodinamicamente uma com a outra e perdem parte da eficiência.

Importa sublinhar que tudo isto corresponde ao conceito idealizado do robô e não a um modelo mecânico terminado. A modelação em CAD chegou a ser iniciada pelas lagartas, mas não foi concluída, porque se decidiu fazer uma pausa deliberada no desenho mecânico do veículo para concentrar o esforço na simulação, que é onde assenta a validação de todas as camadas seguintes desta dissertação.

3.1.1 Máquina de estados da transição de modo

A transição entre o modo terrestre e o modo aéreo foi implementada com uma *Finite State Machine* (FSM), que avança de um estado para o outro só depois de confirmar, através de sensores e não de temporizadores, que o passo anterior terminou de facto. A Figura 3.5 resume os estados e as transições descritos a seguir. A esta transição junta-se um elemento mecânico ainda não apresentado, as quatro pernas prismáticas do robô, cuja função é elevar o corpo para dar folga às lagartas rodarem sem as hélices tocarem no solo.

No caminho de terra para o ar, a máquina passa por `GROUND_DRIVE`, o modo normal de condução, e entra em `TRANSITION_LOCK` assim que recebe o pedido de transição, imobilizando o robô antes de qualquer alteração mecânica. Segue-se `LEGS_EXTENDING`, em que as quatro pernas descem até à posição alvo, e só depois `TRACKS_ROTATING`, em que as duas lagartas rodam sobre o seu próprio eixo longitudinal até ±90 graus, expondo os pares de hélices coaxiais. Quando as lagartas confirmam essa posição, a máquina entra em `AERIAL_READY` e entrega o controlo de voo a um autopiloto dedicado, cuja escolha e papel se detalham na camada de Controlo. Os dois estados

seguintes, `AERIAL_FLY` e `AERIAL_HOVER`, limitam-se a acompanhar a subida e a confirmar que o robô está efetivamente no ar, sem intervir no controlo de voo em si, que pertence inteiramente ao autopiloto.

O caminho de regresso ao solo reaproveita o estado `TRACKS_ROTATING` no sentido inverso, rodando as lagartas de volta a 0 graus, e segue depois para `LEGS_RETRACTING` até regressar a `GROUND_DRIVE`. Sempre que um dos passos mecânicos demora mais do que o tempo limite previsto, a máquina entra no estado `FAILED` em vez de continuar às cegas, e fica à espera de um novo pedido de transição para tentar outra vez.

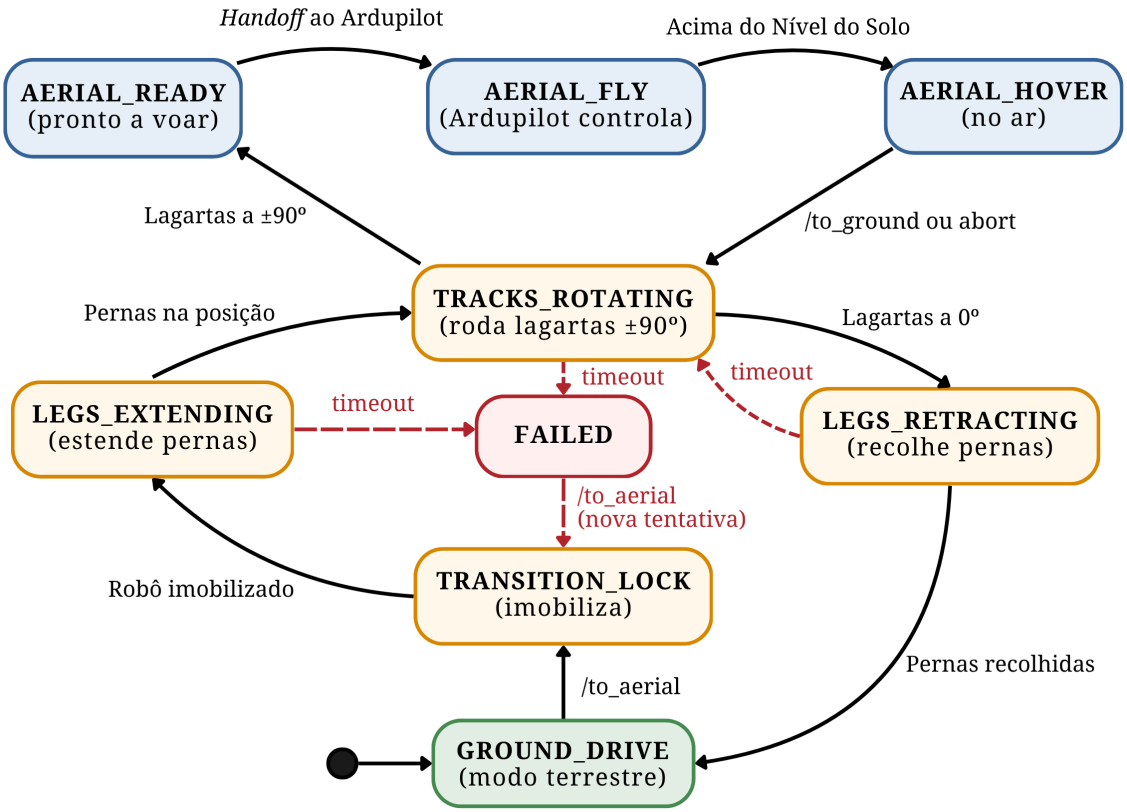


Figura 3.5: Diagrama de estados da máquina de transição de modo.

3.2 Perceção

A camada de perceção transforma os dados brutos dos sensores em objetos com uma classe e uma posição, que as camadas seguintes usam para mapear o ambiente e navegar. Esta dissertação resolve esse problema com dois sensores complementares, um LiDAR 3D que fornece a geometria da cena e uma câmara que lhe acrescenta a aparência, cujas modalidades foram comparadas nos Fundamentos (Secção 2.3.1). As três subsecções seguintes seguem essa divisão, primeiro o *pipeline* geométrico sobre o LiDAR, depois o detetor por imagem, e por fim a forma como as duas fontes se combinam por fusão assimétrica.

3.2.1 LiDAR

O *pipeline* de perceção geométrica implementado no nó `lidar3d_node` processa cada mensagem de nuvem de pontos de forma independente, sem manter estado entre frames, e resume-se nos seis passos da Figura 3.6.

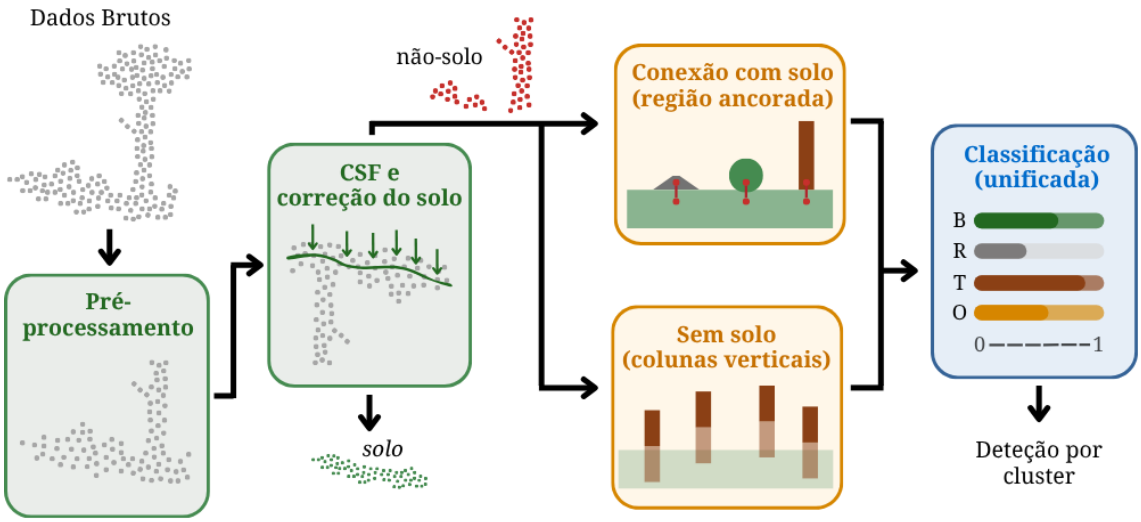


Figura 3.6: *Pipeline* de perceção geométrica implementado sobre o LiDAR, do frame bruto à deteção.

- **Dados brutos**: a nuvem de pontos 3D tal como chega do sensor, na frequência nativa do LiDAR.
- **Pré-processamento**: recorte por alcance e por altura, para remover pontos fora do intervalo útil, seguido de downsample por voxel grid, para reduzir a densidade sem perder a forma geral da cena e assim aliviar o custo dos passos seguintes.
- **CSF + correção do solo**: segmentação solo/não-solo pelo CSF, escolhido por lidar bem com o terreno irregular de uma floresta, onde um simples ajuste de plano falharia. O CSF tende a absorver como solo alguns objetos baixos e achatados que na realidade não o são, sobretudo rochas rasteiras e troncos caídos rentes ao chão. Um corretor devolve esses objetos ao não-solo, estimando a altura real do terreno em pequenas zonas do mapa e recuperando qualquer ponto de solo que fique acima dessa altura por mais de uns centímetros. O solo sai aqui, só o não-solo continua.
- **Bifurcação**: os pontos não-solo seguem um de dois caminhos, consoante o LiDAR tenha ou não visto solo diretamente por baixo deles. O primeiro caminho aplica-se quando há solo observado por baixo. Esse solo dá a cada objeto uma base conhecida, e um crescimento de região parte dessa base para reunir todos os pontos do mesmo objeto, prendendo-o ao ponto exato onde toca o chão. Esta ancoragem ao solo é o que dá ao objeto uma posição e uma altura estáveis, sem ela a base do cluster iria variar de frame para frame consoante os pontos

que o LiDAR captasse nesse instante. O segundo caminho aplica-se quando não há solo por baixo, o que acontece sobretudo atrás e ao lado do robô, onde os feixes traseiros do LiDAR inclinado sobem diretamente para as copas sem nunca varrer o chão. Sem essa base, deixa de existir uma raiz a partir da qual separar objetos vizinhos, e por isso este segundo caminho segue uma lógica diferente, filtra primeiro apenas os pontos que formam uma coluna vertical, própria de um tronco, e só depois agrupa esses pontos em clusters. É este segundo caminho que evita fundir num só bloco copas que se tocam, e que garante que os troncos vistos de lado ou de trás não ficam de fora, algo essencial porque o SLAM se orienta por troncos e ficaria sem marcos suficientes para trabalhar caso esses troncos se perdessem.

- **Classificação**: os clusters dos dois caminhos juntam-se numa só lista e passam pelo mesmo classificador, que atribui uma pontuação por classe, tronco, rocha, arbusto, tronco caído ou obstáculo, a partir da geometria de cada cluster. A única diferença entre os dois caminhos fica na incerteza atribuída na emissão, maior para os clusters vindos do caminho sem solo por baixo, por serem geometricamente menos completos.

- **Saída**: publica-se uma deteção por cluster estrutural, com posição, dimensões e classe proposta, mais nuvens de depuração usadas para diagnóstico do *pipeline*. Estas deteções ainda não são *landmarks*, são o que o Tree-SLAM consome para os criar e manter.

Para afinar os muitos parâmetros deste *pipeline*, limiares do CSF, tolerâncias de *clustering*, critérios de classificação, sem ter de reiniciar a simulação a cada mudança, foi construída uma ferramenta de afinação em tempo real (Figura 3.7), uma interface web local que expõe cada parâmetro como um controlo deslizante, agrupado por secção do *pipeline*, e aplica cada mudança ao nó em execução através de um cliente de parâmetros ROS 2. O efeito de cada mudança é visível de imediato na nuvem classificada, como mostra a Figura 3.8.

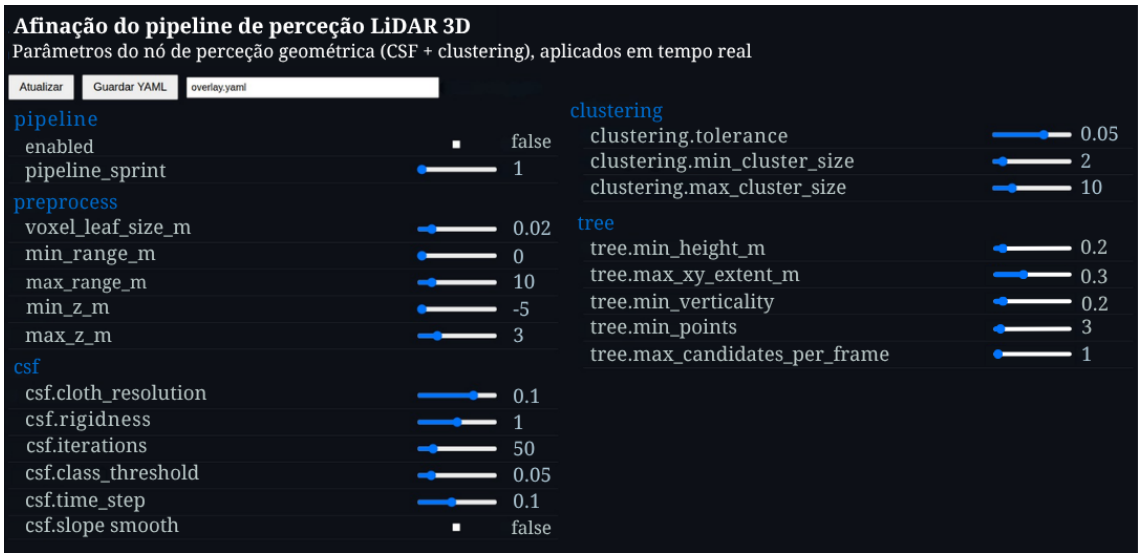
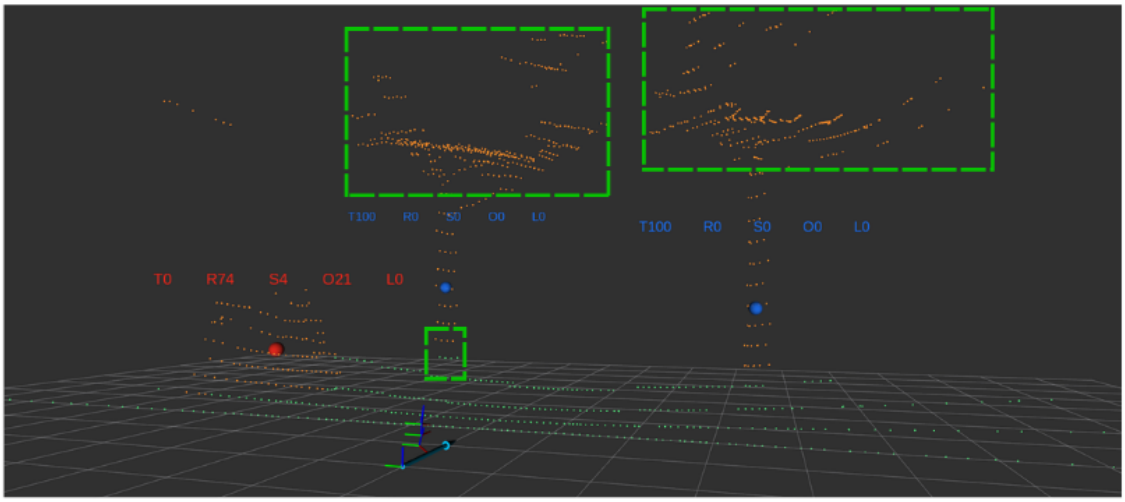


Figura 3.7: Interface web de afinação em tempo real, com cada parâmetro do *pipeline* exposto como um controlo deslizante, agrupado por secção.

(a) antes de ajustar valores pela interface



(b) depois de ajustar valores pela interface

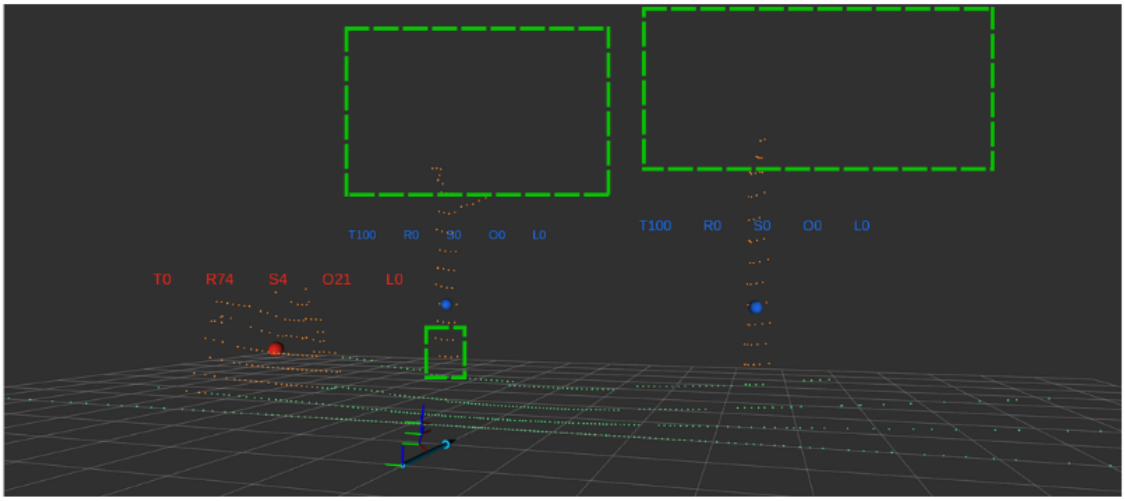


Figura 3.8: Efeito imediato, na nuvem classificada em simulação, de mexer num controlo deslizante da interface da Figura 3.7, antes e depois da alteração.

A distinção entre classes não depende de nenhum modelo aprendido. Assenta num pequeno conjunto de propriedades geométricas do cluster, a maior parte delas calculada por *Principal Component Analysis* (PCA) sobre os seus pontos. São quatro.

A **verticalidade** mede o alinhamento do eixo principal do cluster com a direção vertical. É elevada num tronco em pé e próxima de zero num objeto deitado.

A **linearidade** mede o quanto a nuvem se concentra numa única direção dominante. É elevada em qualquer objeto alongado, seja um tronco em pé ou um tronco caído, e baixa num objeto compacto como uma rocha ou um arbusto.

A **rugosidade** vem do mesmo PCA que dá a verticalidade e a linearidade, mas olha para o que sobra depois de descontar as duas direções dominantes do cluster. É baixa quando quase todos os pontos se explicam por essas duas direções, como uma casca lisa e contínua, e é alta quando sobra

dispersão fora delas, sinal de que os pontos não formam uma superfície limpa mas antes ecos a profundidades diferentes, como o interior poroso de um arbusto.

A **altura do cluster** distingue um tronco caído, baixo e achatado como o seu próprio diâmetro, de uma rocha que, mesmo alongada, se mantém volumosa e alta.

A Tabela 3.2 resume a assinatura de cada classe nestas quatro propriedades.

Tabela 3.2: Assinatura geométrica de cada classe usada pelo classificador.

Classe	**Verticalidade**	**Linearidade**	**Rugosidade**	**Altura**
Tronco (em pé)	alta	alta	baixa	alta
Rocha	baixa a média	baixa	baixa	baixa a alta
Arbusto	baixa	baixa	alta	baixa a média
Tronco caído	baixa	alta	baixa	baixa

Aplicando estas quatro propriedades geométricas, a Figura 3.9 mostra os quatro objetos dispostos em arco à frente do robô numa cena de teste, à esquerda a simulação, à direita a nuvem classificada em RViz com a pontuação de cada classe por cluster. O tronco (T100), o tronco caído (L86) e o arbusto (S78) são classificados com confiança clara, a rocha vence com menor margem (R56 contra S25), refletindo a maior dificuldade em separar rocha de arbusto, discutida a seguir.

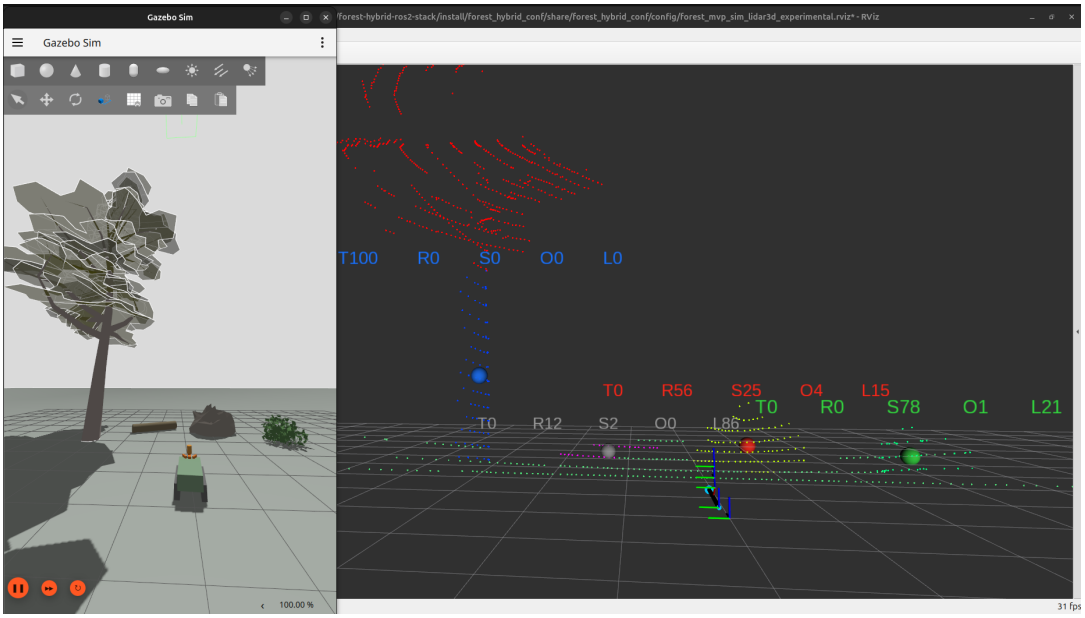


Figura 3.9: Os quatro objetos estruturais classificados numa mesma cena, tronco, rocha, arbusto e tronco caído. Vista da simulação (esquerda) e vista da nuvem classificada a partir do RViz (direita).

3.2.1.1 Dificuldades de classificação geométrica

Duas separações mostraram-se mais difíceis do que as restantes. Nenhuma delas exigiu introduzir uma técnica de classificação nova, mas as soluções não vieram do mesmo lado. Uma

resolveu-se corrigindo o próprio modelo de simulação que estava na origem da ambiguidade, a outra reorganizando a forma como as propriedades geométricas já existentes eram combinadas.

Rocha vs. arbusto. O primeiro modelo de arbusto usado nos mundos de simulação era um aglomerado de esferas lisas, geometricamente indistinguível de uma rocha, uma esfera lisa fechada não tem nenhuma propriedade que a separe de uma casca de rocha lisa e fechada. Várias tentativas de afinar só a fórmula de pontuação não resolveram o problema pela raiz, incluindo uma tentativa de medir a rugosidade a uma escala mais fina que acabou por ter o efeito inverso do pretendido. A correção definitiva não foi na fórmula mas no modelo, um arbusto novo com ramos e folhas separadas, com espaço vazio real entre elas, para que a porosidade medida pelo LiDAR reflita a porosidade real do objeto. Com este modelo, a mesma fórmula de pontuação original voltou a separar as duas classes de forma limpa. Numa rocha de superfície lisa e fechada, os feixes param na face frontal e devolvem uma casca contínua, com rugosidade baixa, enquanto num arbusto poroso os feixes atravessam os vãos entre as folhas e devolvem ecos dispersos a profundidades diferentes, com rugosidade alta. A Figura 3.10 ilustra este mecanismo físico que separa as duas classes.

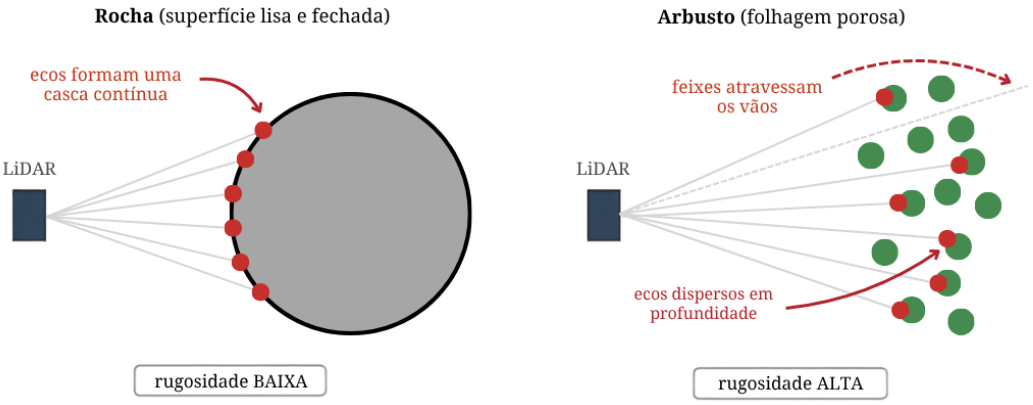


Figura 3.10: Mecanismo que separa rocha de arbusto no LiDAR.

Rocha vs. tronco caído. Ao introduzir o tronco caído como classe própria, a assinatura escolhida para ele apoiava-se sobretudo na linearidade, alta num tronco deitado e baixa numa rocha. Mas uma rocha vista parcialmente pelo LiDAR, com apenas uma fração da sua superfície devolvida em certos ângulos e distâncias, também pode medir linearidade alta, e como a pontuação da rocha era suprimida sempre que a linearidade subia, algumas dessas vistas passaram a ser classificadas como tronco caído. A medição direta das propriedades de várias rochas confirmou o padrão, a linearidade de uma rocha pode subir até valores próximos dos de um tronco caído nalgumas vistas, mas a sua altura mantém-se sempre próxima de 1 a 3 metros, muito acima da altura de um tronco deitado, que é aproximadamente o seu próprio diâmetro, poucas dezenas de centímetros. A correção passou a exigir a assinatura completa do tronco caído, linear e achatado, e só essa combinação suprime a pontuação da rocha, uma rocha que pareça alongada mas continue alta mantém-se classificada como rocha. A Figura 3.11 ilustra a ambiguidade e a forma de a resolver.

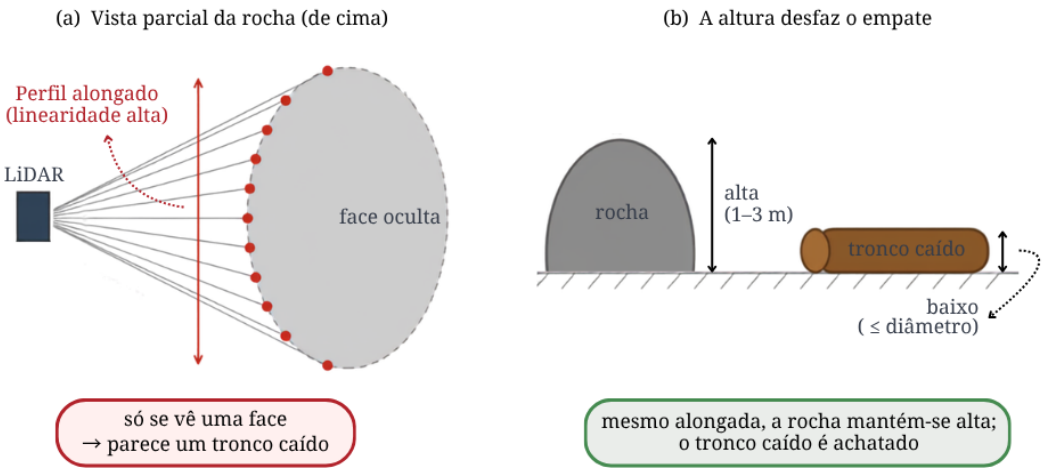


Figura 3.11: Ambiguidade entre rocha e tronco caído no LiDAR e a sua resolução pela altura.

Resta ainda um caso que nem a altura resolve. Uma rocha que seja ao mesmo tempo pequena e alongada é baixa e linear em simultâneo, ou seja reúne a assinatura completa de um tronco caído, e vai por isso muito provavelmente ser classificada como tal. Esta é uma limitação assumida e não um defeito a corrigir, porque geometricamente os dois objetos são de facto indistinguíveis, não há nenhuma propriedade de forma que os separe. É precisamente aqui que a forma sozinha esgota o que consegue fazer, e onde faz sentido trazer um segundo sensor capaz de ver aquilo a que o LiDAR não tem acesso, a aparência. Esse é o papel da câmara.

3.2.2 Câmara

Extrair essa aparência da imagem podia ter seguido uma via simples e sem aprendizagem, apoiada em regras sobre a cor, a textura e as proporções de cada objeto na imagem. Essa via foi posta de lado por assentar em regras manuais frágeis, que uma mudança de escala, de iluminação ou de fundo facilmente quebra. Optou-se então por treinar um classificador próprio o que trocou a dificuldade de inventar regras pela de obter dados anotados em quantidade.

Aqui surge o principal obstáculo. O detetor não vai operar sobre imagens genéricas de floresta, mas sobre o mundo simulado desta dissertação, povoado pelos objetos concretos que ele terá de reconhecer. Um classificador treinado com imagens de outra proveniência dificilmente identificaria esses objetos com fidelidade, porque a sua aparência não coincidiria com a das imagens de treino, e por isso fazia sentido treinar o detetor com dados vindos da própria simulação onde ele vai correr. A simulação torna isto possível a custo baixo, porque todas as poses e malhas dos objetos são conhecidas com exatidão, o que permite gerar caixas delimitadoras de referência sem qualquer anotação manual.

3.2.2.1 Rotulagem automática

O nó `gz_auto_labeler` liga-se ao Gazebo durante a simulação e transforma cada frame numa imagem já anotada, sem qualquer intervenção humana. Como o simulador conhece a pose e a

malha tridimensional de cada objeto do mundo, o nó projeta essa malha para o plano da imagem da câmara e toma o retângulo que a envolve como caixa delimitadora de referência. É esta a vantagem central de rotular dentro da simulação, a anotação deixa de ser um trabalho manual e passa a ser uma consequência exata do que o simulador já sabe. A Figura 3.12 resume o fluxo completo.

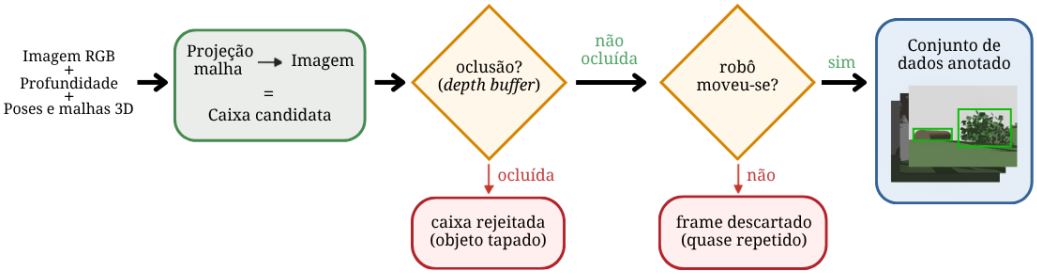
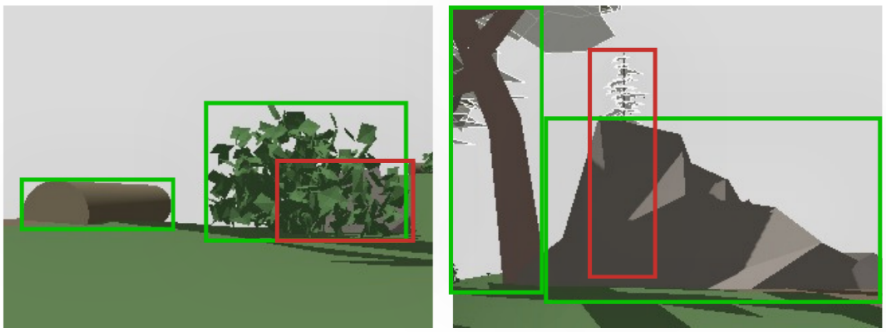


Figura 3.12: Fluxo do rotulador automático. Cada objeto conhecido do simulador é projetado para a imagem como caixa candidata, que passa por um teste de oclusão pelo buffer de profundidade e por um *gate* de movimento antes de entrar no conjunto de dados.

Duas fontes de erro impediam que estas caixas fossem fiáveis logo à partida, e cada uma foi tratada por um mecanismo próprio.

A primeira é a oclusão. Um objeto pode estar geometricamente dentro do campo de visão da câmara e, ainda assim, estar tapado por outro em primeiro plano, caso em que a caixa projetada não corresponde a nada de visível na imagem. Para distinguir os dois casos, o nó recorre ao buffer de profundidade da câmara, que dá a distância real da superfície visível em cada pixel. Quando a profundidade medida no centro da caixa coincide com a distância a que o objeto deveria estar, o objeto está à vista e a caixa é aceite. Quando a profundidade medida é bastante menor, há algo à frente e a caixa é rejeitada. A Figura 3.13 mostra dois casos em que dois objetos diferentes forem rejeitados.



(a) Rejeitada: Rocha ocluída por arbusto

(b) Rejeitada: Árvore ocluída por rocha

Figura 3.13: Teste de oclusão pelo buffer de profundidade. A verde estão as caixas aceites e a vermelho as rejeitadas.

A segunda é a redundância entre frames. Um robô parado produz uma sequência longa de imagens quase idênticas, que inflariam o conjunto de dados sem lhe acrescentar variedade e enviesariam o treino para essas poses repetidas. Para o evitar, o nó só guarda um frame quando o robô se deslocou o suficiente desde a última captura, o que mantém o conjunto variado sem esforço adicional. A Figura 3.14 mostra uma sequência assim capturada, já com as caixas de referência sobrepostas.

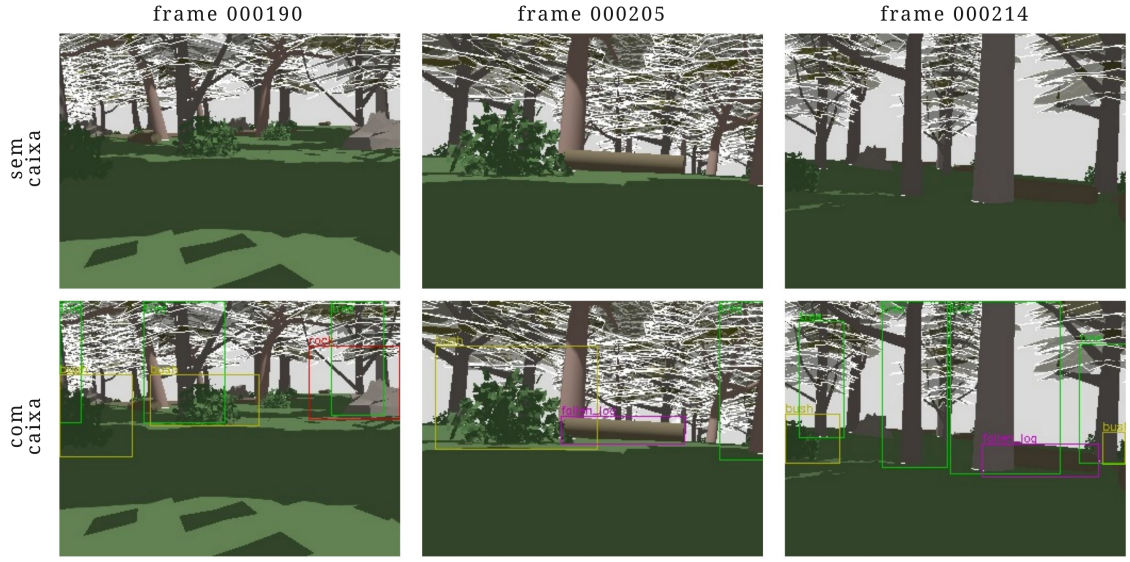


Figura 3.14: Sequência de frames capturados ao longo de um percurso, com as caixas de referência geradas automaticamente para cada objeto visível.

3.2.2.2 Dados, treino e saída do detetor

Foram gerados nove mundos de simulação para a captura, listados na Tabela 3.3 e ilustrados na Figura 3.15, seis mono-classe e três mistos. Cada mundo mono-classe contém só uma classe de objeto, o que garante dados abundantes e limpos dessa classe isolada, uma garantia que um mundo misto não dá às classes mais raras, como o arbusto ou o tronco caído. Apesar de existirem só quatro classes, há seis mundos mono-classe e não quatro porque a rocha, a classe que se revelou mais difícil para o detetor (Secção 3.2.2.3), recebeu três mundos só para ela em vez de um. Os três mundos mistos juntam as quatro classes num único mundo, cada um com um relevo de inclinação diferente, suave, moderado ou acentuado, para o detetor generalizar a vários declives e a situações reais de oclusão mútua entre objetos.

Tabela 3.3: Mundos de simulação usados na captura do conjunto de dados de câmara.

Mundo	**Composição**	**Relevo**	**Imagens**	**tronco**	**rocha**	**arbusto**	**t. caído**
trees_only	mono-classe	moderado	304	880	–	–	–
rocks_only	mono-classe	moderado	354	–	360	–	–
rocks_gentle	mono-classe	suave	551	–	799	–	–
rocks_steep	mono-classe	acentuado	554	–	641	–	–
bushes_only	mono-classe	moderado	404	–	–	1031	–
logs_only	mono-classe	moderado	456	–	–	–	822
mixed_gentle	misto	suave	356	537	161	282	126
mixed_moderate	misto	moderado	401	797	69	444	152
mixed_steep	misto	acentuado	356	372	124	227	99

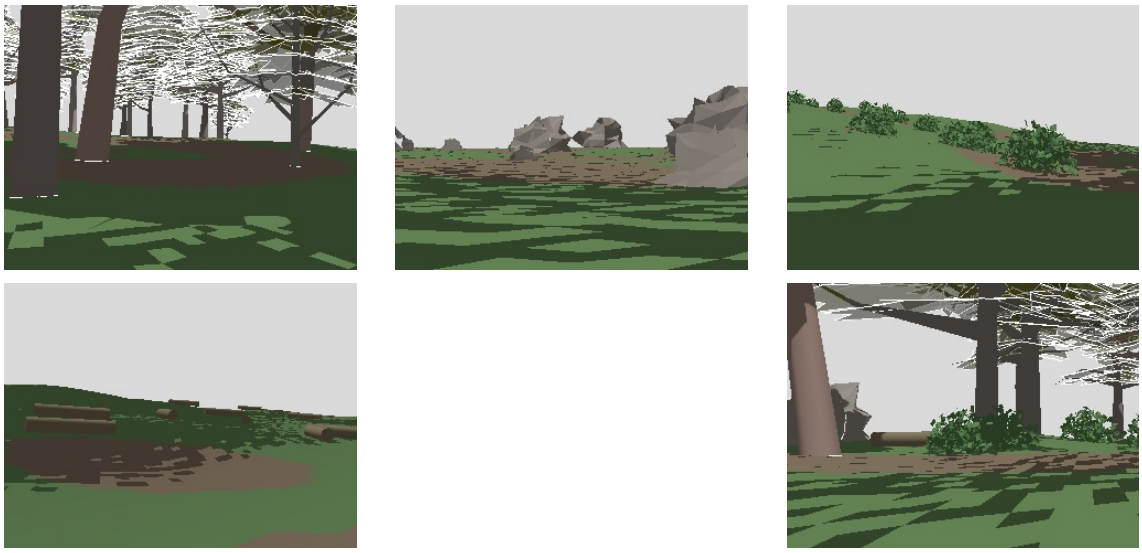


Figura 3.15: Vista da câmara em cada tipo de mundo de captura. Aparecem pela seguinte ordem: tronco, rocha, arbusto, tronco caído e mundo misto.

Depois de combinados e divididos em treino e validação, os seis mundos mono-classe mais os três mistos resultam num conjunto de 3736 imagens no total, com 2586 instâncias de tronco, 2154 de rocha, 1984 de arbusto e 1199 de tronco caído.

O detetor treinado é um YOLOv8n, a variante mais pequena da família, adequada a inferência embarcada. As imagens são redimensionadas a 320×320 e o treino corre durante 100 épocas com lote de 16, sobre o conjunto de dados descrito acima.

Treinado sobre este conjunto de dados, o detetor produz para cada imagem uma lista de caixas, cada uma com a sua classe e a confiança associada. A Figura 3.16 mostra exemplos de deteção sobre imagens de validação, com as quatro classes reconhecidas em cenas com relevo e oclusão mútua entre objetos. A avaliação quantitativa deste detetor, por classe, apresenta-se nos Resultados.

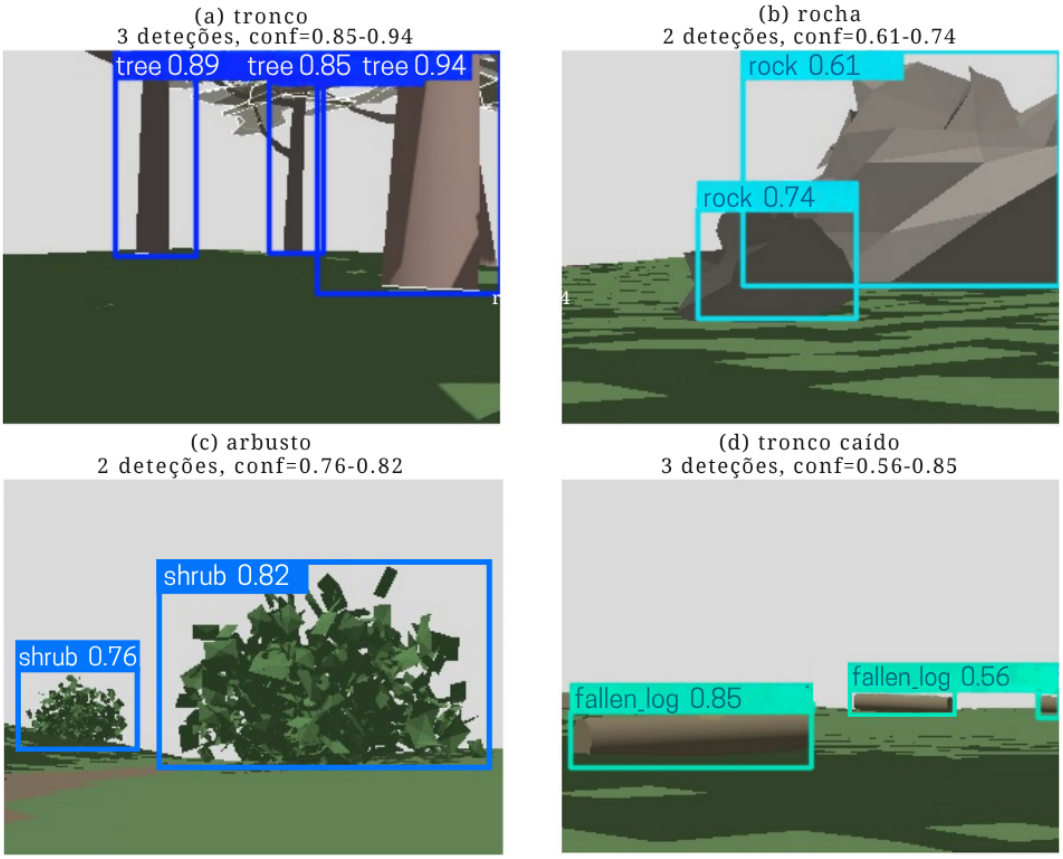


Figura 3.16: Exemplos de deteção do detetor treinado sobre imagens de validação, com a classe e a confiança de cada caixa.

3.2.2.3 Dificuldades do detetor por imagem

***Domain shift* do arbusto.** O modelo de arbusto usado nos primeiros mundos de captura era um aglomerado de esferas, geometricamente simples. Depois de o arbusto ter sido reconstruído com folhagem porosa para corrigir a confusão com rocha no LiDAR (Secção 3.2.1), o detetor treinado com as esferas deixou de reconhecer bem a nova aparência, um caso de mudança de domínio dentro do próprio conjunto de dados sintético. A correção não exigiu nada de novo, apenas recapturar os mundos com arbustos já com o modelo novo e retreinar.

Rocha, a classe mais difícil. A rocha foi de longe a classe mais problemática do detetor, por razões que se foram somando.

A primeira é a própria aparência. Uma rocha cinzenta recortada contra um horizonte também cinzento tem pouco que a distinga do fundo, ao contrário de um tronco ou de um arbusto.

A segunda vem da rotulagem automática. Para rochas pequenas e distantes, a caixa gerada pelo autolabeler ficava grande de mais em relação à rocha que devia conter, ao ponto de mostrar tudo menos a própria rocha, um alvo minúsculo perdido no meio de terreno e céu. Treinar com caixas assim não ensinava o detetor a reconhecer rochas, ensinava-o a reconhecer fundo. A correção foi passar a aceitar apenas as rochas de que era possível obter dados bons, ou seja as que apareciam

com tamanho suficiente na imagem para ocuparem a caixa de forma significativa, e descartar as demasiado pequenas e distantes.

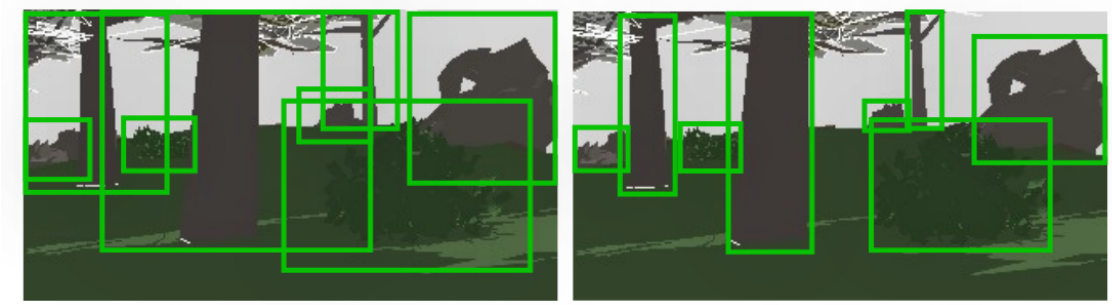
Mesmo assim, os resultados continuavam abaixo do aceitável. Ao contar as instâncias de cada classe, percebeu-se que a rocha estava muito menos representada do que as restantes, e foi essa a razão para gerar mais mundos dedicados só a rochas (Tabela 3.3), que reequilibraram o número de exemplos fiáveis sem inflar nenhuma outra classe. Os valores por classe do detetor final apresentam-se nos Resultados.

Aplicar este critério de tamanho às rochas já capturadas, sem ter de voltar a correr todos os mundos no simulador, exigiu uma ferramenta própria, descrita a seguir.

3.2.2.4 Rotulagem offline

Um critério de rotulagem raramente fica certo à primeira, e reabrir o simulador para recapturar um mundo inteiro a cada ajuste seria demorado. Foi para responder a isto, e entre outros para refiltrar as rochas descritas acima, que se construiu um rotulador offline, o `relabel_offline`. Este trabalha diretamente sobre os dados já gravados durante a captura, incluindo a imagem de profundidade de cada frame, e como aplica exatamente o mesmo teste de oclusão que o nó ao vivo aplicaria, a decisão que produz é indistinguível da que teria sido tomada em direto, agora sem simulador nenhum a correr.

O que isto permite é recalibrar as caixas de qualquer classe a partir do material já capturado, por exemplo apertar o critério de tamanho que decide a partir de que distância uma rocha deixa de ser fiável. A Figura 3.17 mostra o efeito de uma dessas recalibrações sobre o mesmo frame.



(a) Antes da Recalibração *offline* (b) Depois da Recalibração *offline*

Figura 3.17: Recalibração offline de um envelope de rotulagem sobre o mesmo frame, antes (esquerda) e depois (direita) de apertar o critério de tamanho, sem reabrir o simulador.

Importa sublinhar que todo este detetor foi treinado e validado dentro da simulação, e é aí que se prova nesta dissertação. Levá-lo a um robô real seria trabalho futuro, e passaria por recapturar e anotar imagens reais de floresta, ou por recorrer a conjuntos de dados próprios desse domínio, uma vez que a aparência das imagens sintéticas não coincide com a das reais. Como nesta arquitetura a câmara é apenas uma confirmação secundária do LiDAR, e não o classificador principal, esta dependência da simulação não compromete o funcionamento do sistema, apenas adia para trabalho futuro o ganho que a câmara traria num cenário real.

3.2.3 Fusão

Nenhum dos dois sensores, sozinho, responde à classificação dos quatro objetos com fiabilidade total em todas as situações. As duas secções anteriores mostram exatamente onde cada um falha, e é dessa complementaridade, não de uma exigência abstrata da literatura, que nasce a arquitetura de fusão desenhada nesta secção.

3.2.3.1 Escolha da fusão assimétrica

O LiDAR e a câmara não falham no mesmo eixo. Os erros residuais do classificador LiDAR (Secção 3.2.1) são ambiguidades geométricas na fronteira entre classes, uma rocha vista parcialmente pode parecer alongada como um tronco caído, uma rocha compacta pode competir com um arbusto poroso. São exatamente os casos em que cor e textura resolvem de forma trivial o que a forma sozinha não resolve, uma rocha é cinzenta e mineral, um arbusto é verde e foliar e um tronco é castanho e tem casca.

Já a câmara tem um problema diferente e mais difícil de contornar. O *domain gap* discutido nos Fundamentos é, em última análise, um problema de dados, resolvido nesta dissertação com um conjunto de imagens sintéticas gerado de propósito da forma descrita na Secção 3.2.2, e essa via funciona no sentido em que o detetor aprende de facto a reconhecer as quatro classes a partir de dados sintéticos, ainda que de forma desigual entre elas como mostram os Resultados. O problema real que impede a câmara de ser o método principal é outro, é um sensor ótico passivo, e por isso permanece intrinsecamente suscetível a falhar sempre que as condições ambientais não são favoráveis, contraluz, sombra profunda sob copa densa, pouca luz ao fim do dia, poeira ou névoa. Ao contrário do *domain gap*, este não é um problema que mais dados ou melhor treino resolvam, é uma limitação física do princípio de deteção da câmara. O LiDAR, um sensor ativo, é largamente indiferente a estas condições. É esta fragilidade estrutural, não a qualidade do detetor em si, que torna a câmara inadequada como método principal de classificação, e adequada como confirmação secundária, que só precisa de estar disponível quando as condições o permitem.

Esta separação de eixos é o argumento central para uma fusão assimétrica em vez de simétrica. Os dois sensores não fazem o mesmo trabalho com fiabilidades diferentes, fazem trabalhos diferentes, o LiDAR localiza no espaço e propõe uma primeira classe a partir da forma, a câmara confirma essa classe recorrendo a um eixo de informação, a aparência, que o LiDAR estruturalmente não tem acesso. Um segundo argumento reforça esta escolha, o de robustez, precisamente porque a fragilidade ambiental da câmara é inevitável, o sistema não pode depender dela para continuar a funcionar, se a câmara degradar, o LiDAR sozinho mantém uma classificação segura, ainda que mais conservadora, e a câmara limita-se a acelerar e a solidificar essa confiança quando está disponível, nunca é um ponto único de falha. Um terceiro argumento é arquitetural, a fusão assimétrica mantém os dois *pipelines* independentes e depuráveis, sem exigir calibração espácio-temporal apertada nem re-treino conjunto, o que se ajusta ao facto de os dois classificadores terem sido desenvolvidos e validados de forma totalmente independente ao longo desta dissertação.

3.2.3.2 Arquitetura de fusão desenhada

A arquitetura desenhada segue o princípio de que o LiDAR é a base da classificação e a câmara confirma e solidifica essa base, nunca a substitui nem a inverte sozinha num único frame. A Figura 3.18 resume o mecanismo. A fusão ocorre no tracker do SLAM, e não no nó de perceção por-frame. Cada deteção 3D é projetada para a imagem através da extrínseca estática entre o robô e a câmara, e associado à deteção 2D mais próxima do ponto projetado. O termo da câmara entra então na acumulação de confiança do *landmark*, em log-odds, com um teto suave que impede a câmara de dominar. Assim, o LiDAR sozinho continua a poder promover um *landmark*, apenas mais devagar, e a câmara acelera esse processo quando concorda. O *landmark* é, portanto, uma entidade do SLAM, e não uma saída da perceção.

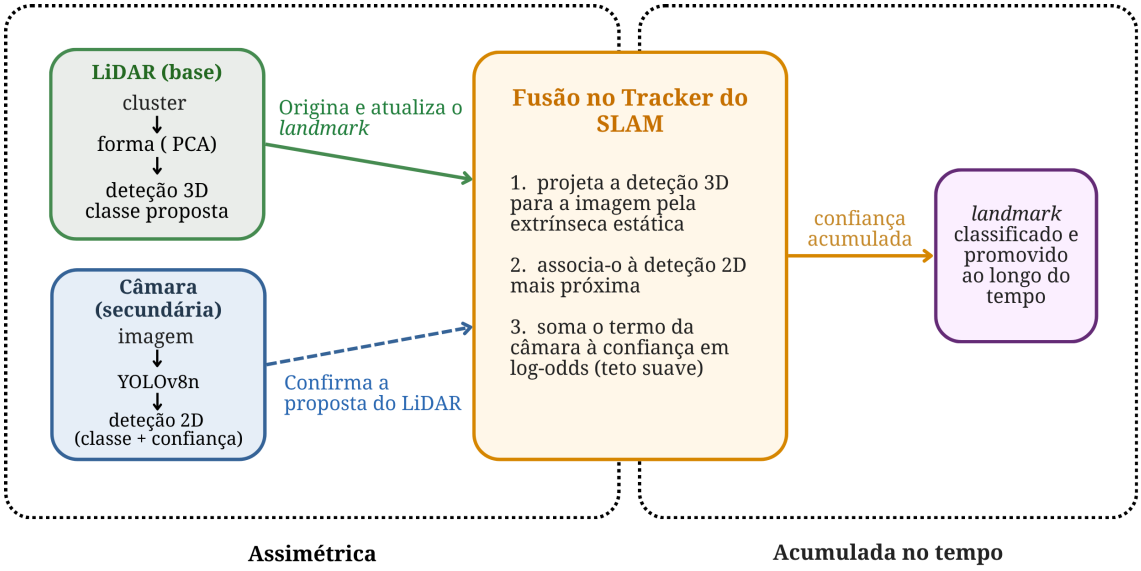


Figura 3.18: Arquitetura de fusão assimétrica desenhada.

Mapeada à taxonomia de fusão dos Fundamentos (Secção 2.3.4), esta é uma fusão assimétrica, os dois ramos não entram ao mesmo nível, é a deteção do LiDAR que faz nascer o *landmark* e que depois o atualiza a cada observação, enquanto a câmara apenas influencia a sua classe sem nunca gerar uma proposta independente por si só. A esta classificação junta-se uma característica própria desta arquitetura, e não da taxonomia em si, a confiança acumula-se no tempo porque cresce com observações repetidas dentro do tracker, e não num único instante.

3.3 Localização e Mapeamento Simultâneos

A localização deste robô assenta em duas camadas complementares. A primeira estima a pose de forma contínua a partir dos sensores próprios do robô e está sempre disponível, mesmo antes de o robô ver qualquer árvore. A segunda corrige a deriva dessa pose sempre que o robô reconhece os troncos e as rochas em torno de si. A camada contínua responde à pergunta de onde o robô está agora, a cada instante, e a camada de *landmarks* garante que essa resposta não se afasta da verdade

ao longo do tempo. A Figura 3.19 mostra como as duas se ligam, à entrada de *landmarks* vinda da Perceção e ao gestor que decide, em cada instante, qual delas tem autoridade sobre a pose global.

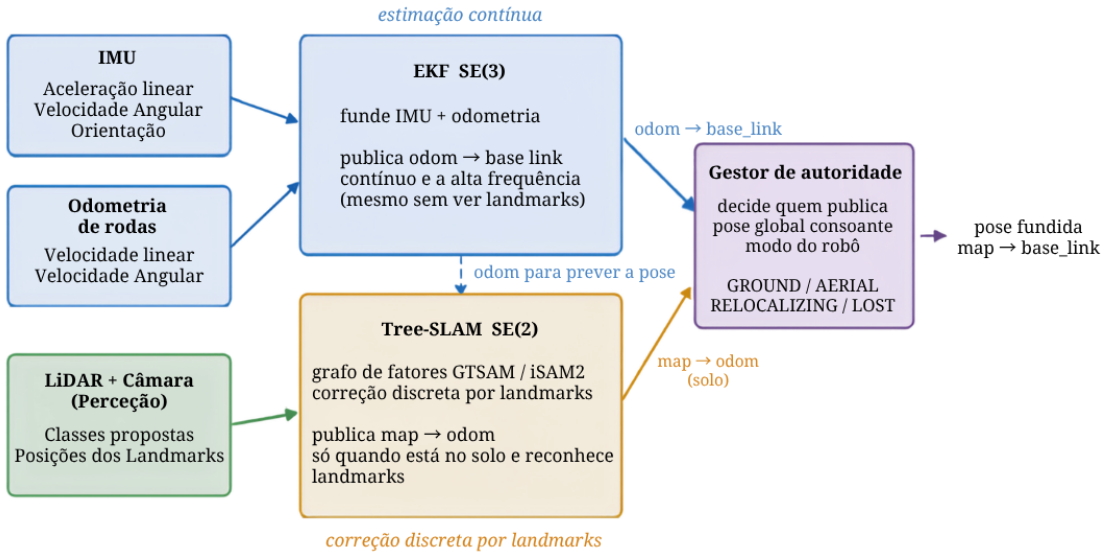


Figura 3.19: Arquitetura da localização, com o estimador contínuo e o Tree-SLAM.

3.3.1 Estimação de estado contínua

A base da pose é um filtro de Kalman estendido em SE(3), montado sobre a biblioteca `robot_localization`. Este filtro funde duas fontes. Uma é a odometria de rodas, que dá a velocidade linear e angular do robô. A outra é o IMU, que dá as três taxas do giroscópio e as acelerações lineares nos três eixos.

O filtro publica a transformação `odom`→`base_link` de forma contínua e a alta frequência. Essa pose não depende de o robô ter visto algum *landmark*, por isso está sempre disponível. É ela a fonte de pose usada por todas as camadas que precisam de uma estimativa a cada instante.

Fundir o roll e o pitch absolutos do IMU, e não apenas a sua taxa de variação, ajuda a impedir que a atitude do robô derive lentamente ao longo do tempo. Esse erro não fica só na orientação, e propaga-se também para as alturas que o robô calcula a partir da sua pose, incluindo a altura dos *landmarks* vistos pelo LiDAR, contaminando por isso também o mapa.

Para que as outras camadas não precisem de saber como esta pose é montada por dentro, um nó de contrato publica a pose final já composta. Essa pose resulta de encadear a correção por *landmarks* `map`→`odom` com a estimativa contínua `odom`→`base_link`. Assim a navegação e o mapa de custo recebem sempre uma única pose coerente, sem terem de saber qual das camadas a corrigiu por último.

3.3.2 Associação de dados

Sobre esta base contínua assenta o Tree-SLAM, o SLAM esparso por *landmarks* de troncos e rochas que constitui o contributo central desta camada. Por cada nova mensagem de *landmarks*

vinda da Perceção, o Tree-SLAM corre o ciclo da Figura 3.20, dividido em duas colunas. A coluna da esquerda corre em todos os ciclos, prevê a pose, associa as deteções, e atualiza e otimiza o grafo com a keyframe de odometria e as observações dos *landmarks* já confirmados. A coluna da direita gere o nascimento e a promoção dos *landmarks* ao longo do tempo. Note-se que a otimização do grafo, feita pela coluna da esquerda, acontece a cada ciclo, e não apenas quando um candidato é promovido. As três subsecções seguintes detalham os três blocos desse ciclo, a associação de cada deteção ao mapa já conhecido, a consolidação dos *landmarks* ao longo do tempo, e a otimização final no grafo de fatores.

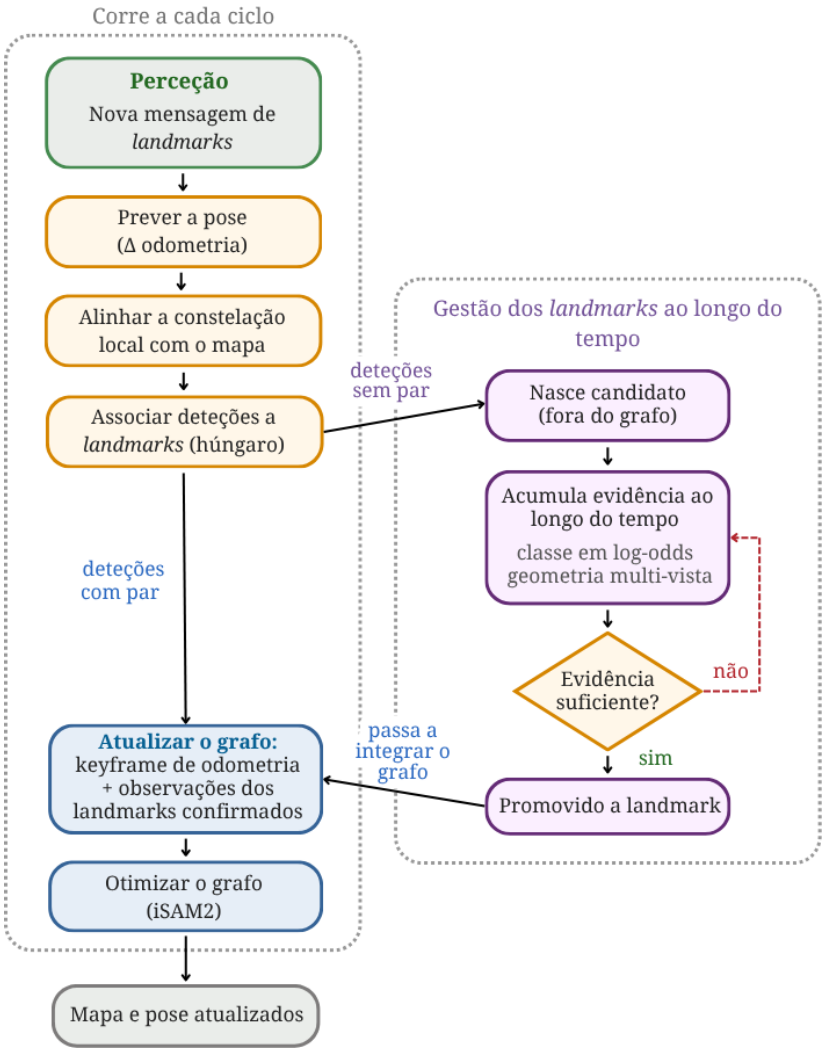


Figura 3.20: Ciclo que o Tree-SLAM corre por cada mensagem de *landmarks*.

O primeiro passo do ciclo é prever onde o robô está, a partir do deslocamento de odometria desde a última mensagem. O passo seguinte é associar cada deteção atual ao *landmark* do mapa que lhe corresponde, e é aqui que está a maior dificuldade da camada.

A dificuldade nasce de uma propriedade da própria floresta. Quanto mais repetitiva for a disposição das árvores vizinhas, mais difícil é distinguir uma configuração local de outra igual

noutro ponto do mapa. A primeira abordagem tentada foi um *gate* de Mahalanobis em coordenadas absolutas, que avalia cada par deteção-*landmark* isoladamente pela distância entre as suas posições no mapa. Essa abordagem revelou uma fragilidade estrutural sempre que existe deriva acumulada. Quando a deriva desloca o robô inteiro, todas as deteções se deslocam em bloco e todas caem fora do *gate* ao mesmo tempo, não apenas uma. O resultado é não associar nada exatamente no momento em que era mais preciso corrigir. A Figura 3.21 ilustra este contraste.

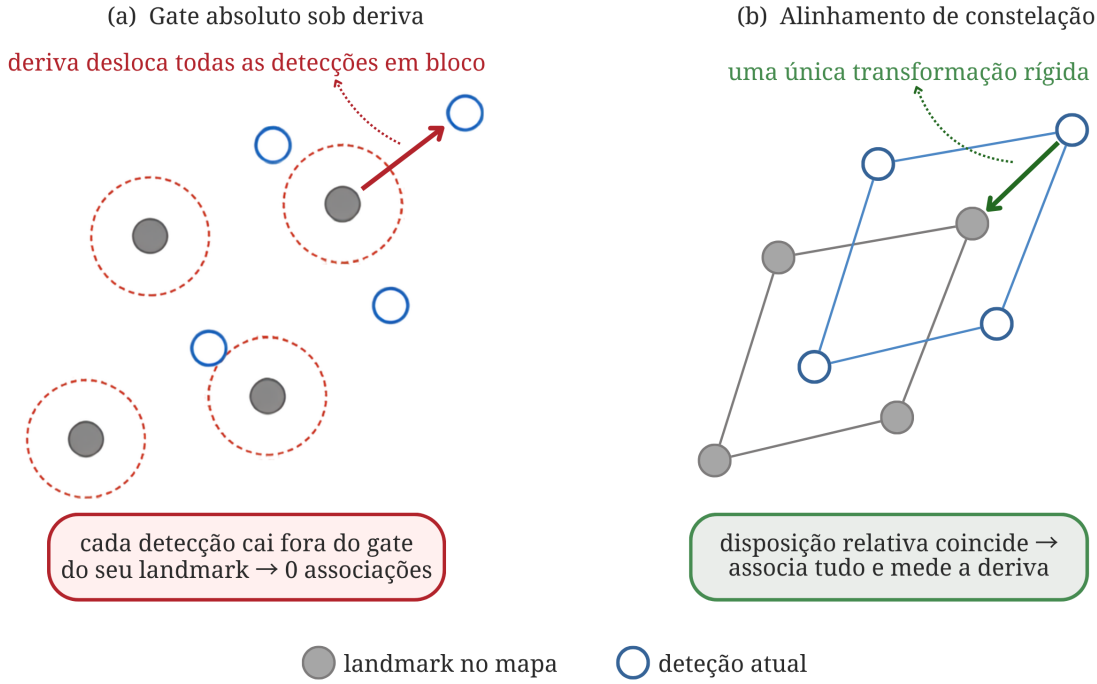


Figura 3.21: Porque a associação por constelação substitui o *gate* absoluto.

A solução foi deixar de comparar posições absolutas e passar a comparar a disposição relativa entre vizinhos. O alinhamento de constelação local toma as deteções atuais e os *landmarks* vizinhos e procura a transformação rígida que sobrepõe uma disposição na outra. Como compara a forma da constelação, e não a posição de cada ponto, resolve a associação e mede a deriva no mesmo passo. Depois deste alinhamento, constrói-se uma matriz de custos com a distância entre cada deteção e cada *landmark* já alinhados, e é sobre essa matriz que o algoritmo húngaro resolve a atribuição final de cada deteção ao seu *landmark*. Este mesmo motor de alinhamento é depois reutilizado pela relocalização, como se verá adiante.

3.3.3 Nascimento e consolidação dos *landmarks* ao longo do tempo

Uma deteção nova não entra logo no mapa. Se não tem par no mapa conhecido, nasce apenas como candidato, ainda fora do grafo. A partir daí, cada nova observação do mesmo objeto vai acumulando evidência, e só quando essa evidência é suficiente é que o candidato é promovido a *landmark* confirmado. Esta disciplina é propositadamente conservadora, para o mapa não encher de objetos que não existem. A Figura 3.22 acompanha um *landmark* ao longo deste processo.

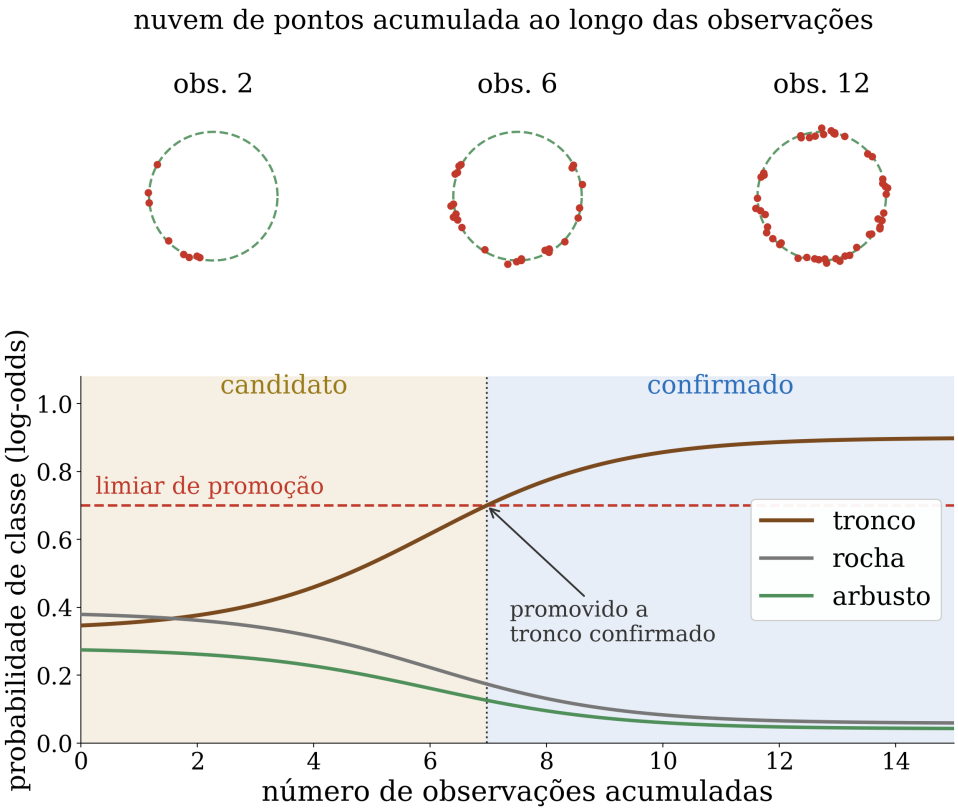


Figura 3.22: Vida de um *landmark* cuja classe verdadeira é tronco, tomado aqui como exemplo.

Duas evidências diferentes acumulam ao mesmo tempo. A primeira é a classe. Cada *landmark* trabalha com quatro classes possíveis, os troncos, as rochas, os arbustos e os troncos caídos. Os troncos são os *landmarks* principais, por serem os mais estáveis e mais fiáveis de medir. As rochas têm o mesmo estatuto dos troncos, por serem objetos igualmente rígidos e com contorno bem definido. Os arbustos e os troncos caídos também são considerados, mas com mais incerteza associada, por serem objetos irregulares ou alongados cuja posição não corresponde a uma medição tão exata. A cada observação, a Perceção propõe uma classe, e o *landmark* soma essa evidência em log-odds, uma forma de acumular probabilidades que dá peso à consistência ao longo do tempo em vez de reagir a uma única leitura. As quatro hipóteses de classe são concorrentes para o mesmo objeto e a sua probabilidade soma sempre um, por isso a evidência a favor de uma classe é sempre evidência contra as outras três. Um *landmark* nasce assim sem classe fixada. Só é promovido quando a classe vencedora reúne uma probabilidade acima de um limiar e com margem clara sobre a segunda mais provável, e nunca a partir de um único ponto de vista. Isto evita fixar no mapa um rótulo precipitado a partir de uma vista isolada e ambígua.

A segunda evidência é a geometria, e em particular o diâmetro do tronco, que é mal condicionado a partir de um só frame. O LiDAR nunca vê a circunferência completa de um tronco de uma única pose. Vê apenas um arco parcial. Ajustar um círculo a um arco parcial é sensível ao ruído, e pequenas variações na nuvem de pontos produzem diâmetros muito diferentes entre frames consecutivos do mesmo tronco. A solução não foi melhorar o ajuste por frame, mas deixar

de depender de um único frame. O acumulador multi-vista mantém a nuvem de pontos de cada *landmark* ao longo de todas as observações, com deduplicação por voxel, e só ajusta o círculo de Taubin, que dá em simultâneo o centro e o diâmetro do tronco, quando a cobertura angular já é suficiente. A Figura 3.23 mostra esta ideia. A posição resultante desse ajuste passa a servir de *prior* forte para o *landmark* quando a cobertura está garantida.

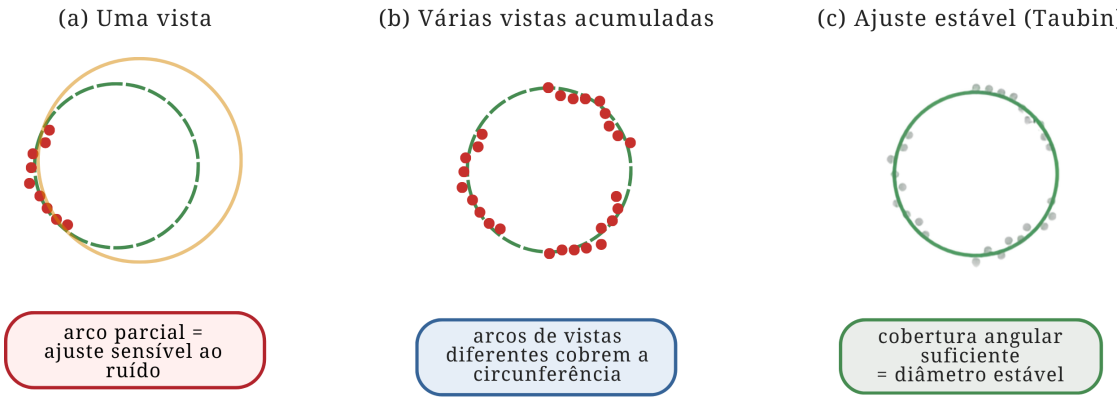


Figura 3.23: Estimação do diâmetro por acumulação multi-vista. (a) Um só frame vê apenas um arco parcial do tronco, e o ajuste do círculo a esse arco é sensível ao ruído. (b) Poses sucessivas trazem arcos diferentes que, acumulados, cobrem uma fração cada vez maior da circunferência. (c) Quando a cobertura angular acumulada é suficiente, o ajuste do círculo de Taubin estabiliza e o diâmetro deixa de variar entre frames.

Um *landmark* confirmado que deixa de ser observado num dado ciclo não é removido do mapa. Fica apenas marcado como dormente, pronto a ser reassociado quando o robô voltar a passar por ele. Assim o mapa preserva o que já aprendeu mesmo quando o robô se afasta e volta.

3.3.4 O grafo de fatores e a otimização

Os *landmarks* confirmados, junto com as poses do robô ao longo do tempo, formam o grafo que é finalmente otimizado. Este grafo é o mapa, e é construído com a biblioteca GTSAM e otimizado de forma incremental com o iSAM2, a combinação de *backend* já justificada nos Fundamentos (Secção 2.4.3) face às bibliotecas de otimização de uso geral como o g2o e o Ceres Solver, que recalculam a solução do grafo inteiro a cada nova otimização. É exatamente essa recomputação total que o iSAM2 evita, ao manter uma árvore de Bayes que isola apenas a parte do grafo afetada por cada novo fator, o que permite atualizar a solução a cada novo ciclo sem a recalcular de raiz. As variáveis do grafo são de dois tipos, as poses do robô guardadas como *keyframes* e as posições dos *landmarks* confirmados. A Figura 3.24 mostra a sua estrutura.

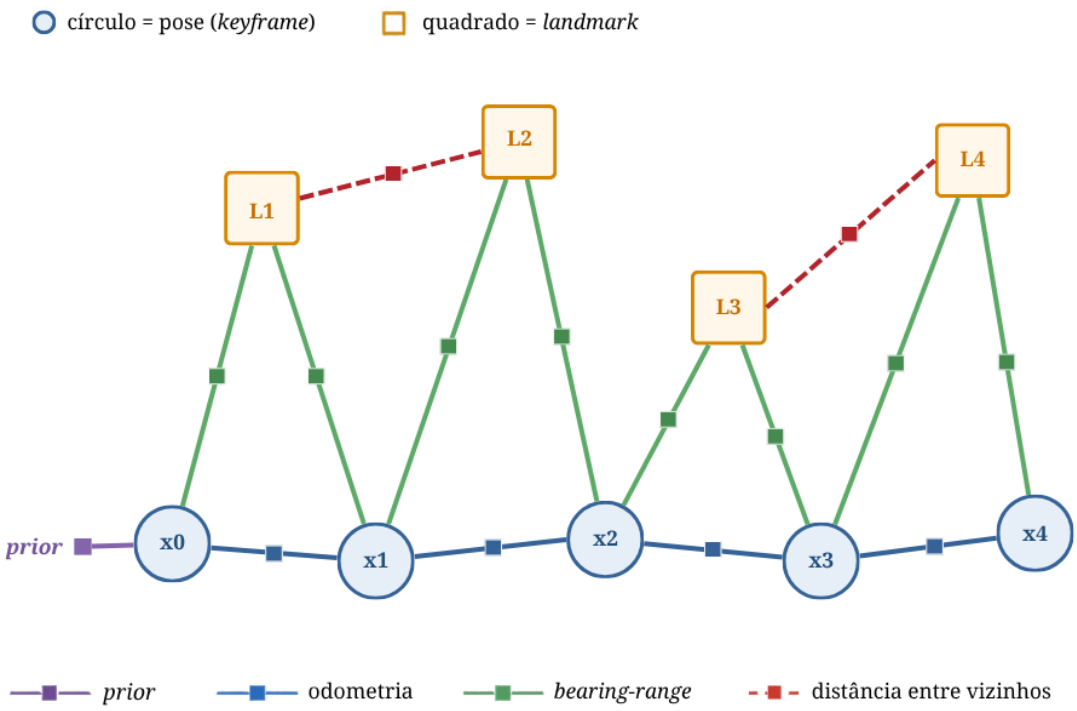


Figura 3.24: Grafo de fatores do Tree-SLAM, com as poses do robô e os *landmarks* como variáveis.

As variáveis estão ligadas por quatro tipos de fator, cada um a impor uma restrição diferente.

- **Fator *prior***: ancora a primeira pose na origem, para dar uma referência a todo o mapa.
- **Fatores de odometria**: ligam *keyframes* consecutivas, exprimindo o movimento medido entre elas.
- **Fatores de *bearing-range***: ligam cada *keyframe* aos *landmarks* nela observados, exprimindo o ângulo e a distância a que cada *landmark* foi visto.
- **Fatores de distância**: ligam *landmarks* vizinhos associados em conjunto pelo alinhamento de constelação, fixando a forma relativa da constelação e reforçando a associação de dados.

Todos os fatores de observação usam um núcleo robusto de Huber, para que uma associação errada isolada não domine a otimização sem ter de ser eliminada por completo.

Este grafo é resolvido em SE(2), no plano do solo, e não em SE(3) como o estimador contínuo. A escolha é deliberada. No solo, que é onde o Tree-SLAM tem autoridade, o que interessa corrigir é a posição do robô no plano e a orientação em torno da vertical, porque a altura e a inclinação instantâneas já vêm bem estimadas do IMU. Resolver o grafo em duas dimensões torna a otimização mais leve e mais estável. A correção que dele resulta é depois aplicada de forma a preservar o roll, o pitch e a altura da camada SE(3), sem os achatar.

3.3.5 Relocalização e autoridade de modo

A relocalização entra em ação sempre que o robô deixa de conseguir seguir de forma contínua a sua referência ao mapa. O caso mais claro é o regresso de um salto aéreo, em que o robô perde por completo a referência ao mapa e tem de se reconhecer nele de novo. O mesmo problema surge, de forma mais ligeira, se durante a condução no solo o robô passar tempo suficiente sem reconhecer nada à sua volta. A relocalização segue o desenho de duas fases já introduzido nos Fundamentos. Uma fase grosseira compara um histograma da distribuição de distâncias e diâmetros entre *landmarks* vizinhos, para reduzir depressa os candidatos possíveis. Uma fase fina verifica a geometria exata de triângulos formados por trios de *landmarks* próximos, com rejeição de correspondências por RANSAC. Este mesmo mecanismo é o que serve de motor ao alinhamento de constelação da associação. A diferença entre os dois usos está apenas na escala da busca, uma relocalização completa depois de um salto contra um alinhamento contínuo a cada ciclo.

A decisão de qual camada tem autoridade sobre a transformação `map`→`odom` em cada instante é gerida por uma máquina de estados com quatro modos, resumidos na Tabela 3.4. Importa notar que só esta transformação é disputada. A transformação `odom`→`base_link` é sempre publicada pelo estimador contínuo, aconteça o que acontecer, e a arbitragem de autoridade decide apenas quem corrige a deriva por cima dela. No solo essa correção pertence ao Tree-SLAM. No ar, essa autoridade vem diretamente da pose fundida do piloto automático. Nas transições em torno de um salto, o modo `RELOCALIZING` suspende a atualização da transformação até a relocalização terminar com sucesso, para nunca publicar uma pose de solo obtida a partir de uma associação ainda não verificada.

O modo `LOST` corresponde à situação em que o robô conduziu tempo suficiente sem reconhecer os *landmarks* à sua volta e perdeu a referência ao mapa. Nesse modo a transformação `map`→`odom` fica congelada no último valor válido, mas o robô não pára. Continua a andar guiado pela pose contínua do estimador, que nunca deixa de estar disponível, e volta ao modo `GROUND` assim que torna a reconhecer os *landmarks* à sua volta.

Tabela 3.4: Modos da máquina de estados e autoridade sobre a transformação `map`→`odom` em cada um.

Modo	**Situação**	**Autoridade sobre `map`→`odom`**
`GROUND`	Robô no solo, mapa referenciado	Tree-SLAM
`AERIAL`	Robô em voo	Pose fundida do piloto automático
`RELOCALIZING`	Transição em torno de um salto	Congelada até a relocalização terminar
`LOST`	Muitos ciclos sem reconhecer o mapa	Congelada, robô segue na pose contínua

3.4 Representação do Ambiente e Traversabilidade

Depois de o robô saber onde está e o que tem à sua volta, falta responder à pergunta que decide o movimento, que é se pode passar por um determinado sítio e a que custo. Esta camada existe

para responder a isso. Ela recebe o terreno e os objetos que as camadas anteriores identificaram, e produz um mapa onde cada célula tem um valor de custo, que o planeamento usa para escolher por onde ir.

Os fundamentos deixaram três perguntas em aberto sobre como representar o ambiente, nomeadamente o que se guarda em cada célula, para quem esse valor vale, e como a informação se organiza no espaço. As subsecções seguintes respondem a essas três perguntas para este sistema em concreto.

3.4.1 Arquitetura da camada

A camada está implementada no pacote `forest_traversability_mapping` e assenta na biblioteca `grid_map`, que mantém uma grelha 2.5D com várias camadas de dados sobrepostas na mesma geometria. A Figura 3.25 mostra o fluxo de dados desde as camadas anteriores até ao planeamento.

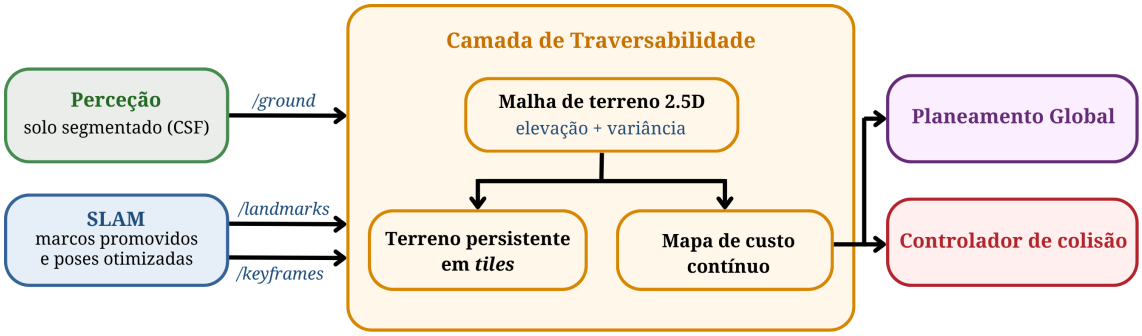


Figura 3.25: Fluxo de dados da camada de traversabilidade. As entradas são saídas de outras camadas e não sensores em bruto. O produto final é um mapa de custo contínuo que alimenta o planeamento e o controlo.

Vale a pena reparar que nenhuma das entradas é um sensor em bruto. O terreno chega já segmentado pela perceção, através da nuvem de pontos que o CSF classificou como solo, e os objetos chegam já promovidos pelo SLAM, na forma de marcos com identidade estável e posição otimizada. Isto evita repetir aqui o trabalho de segmentação que a perceção já fez, mas tem uma consequência que importa assumir, porque a qualidade deste mapa fica limitada pela qualidade das camadas anteriores. Se a perceção classificar mal a copa de uma árvore como solo, a malha de elevação ganha um relevo que não existe, e o custo de declive dispara num sítio onde o chão é plano.

Do outro lado, o consumidor do mapa de custo é o Nav2, a pilha de navegação onde vivem o planeador e o controlador, descrita em detalhe na Secção 3.5. Para esta camada o Nav2 é apenas o destino do produto final.

Das duas famílias de traversabilidade distinguidas nos Fundamentos, esta camada assenta inteiramente na traversabilidade aparente, aquela que se estima a partir da geometria observada do terreno e dos objetos. O custo de cada célula sai do declive, do degrau e dos obstáculos que a perceção e o SLAM já mediram, e não de qualquer registo de como o robô se comportou de facto

ao passar por ali. A traversabilidade relativa, que aprenderia com a experiência propriocetiva do próprio robô, exigiria fechar um ciclo de aprendizagem que este sistema não tem, e por isso ficou de fora. A geometria observada é a informação que já está disponível e é fiável, e é sobre ela que todo o modelo de custo é construído.

A grelha mantém sete camadas de dados. A camada `elevation` guarda a altura do solo fundida ao longo do tempo, e a `variance` guarda a incerteza dessa estimativa. A `elevation_smooth` é a elevação depois de filtrada. As três camadas `cost_terrain`, `cost_obstacle` e `cost_uncertainty` guardam cada uma das contribuições para o custo, e a camada `cost` guarda o valor final que sai para o planeamento.

3.4.2 A malha de terreno 2.5D

A primeira decisão foi representar o terreno como uma grelha 2.5D, com uma única altura por posição, em vez de um mapa volumétrico denso. A razão mais forte não é o custo de memória, embora ele exista, mas sim o facto de o volume ser redundante neste problema.

Um robô terrestre conduz sobre uma superfície e não dentro de um volume. O que ele precisa de saber é a altura do chão em cada ponto, o que é por definição uma altura por posição. O caso onde a representação 2.5D falha é a estrutura suspensa, tipicamente a copa de uma árvore por cima de espaço livre. Este robô também voa, mas o voo fica reservado a casos muito específicos, contornar obstáculos que são intransponíveis pelo solo, e não a uma navegação livre em três dimensões por entre o arvoredo, pelo que mapear a copa em volume para esse fim não se justifica. Mais importante ainda, a copa não é uma estrutura suficientemente estável para valer a pena guardá-la, e o desvio a um ramo ou a uma copa baixa sobre o caminho é tratado de forma reativa, diretamente a partir da nuvem de pontos não-solo entregue ao controlador de seguimento (Secção 3.5), sem que essa geometria precise de ser guardada nem revisitada mais tarde. Guardar a copa numa grelha volumétrica densa seria gastar memória a representar informação que já é tratada de outra forma, mais rápida e mais barata.

Além disso, os objetos que têm mesmo volume e que interessam à navegação já estão representados, e de forma melhor. Um tronco é descrito pelo SLAM como um marco vetorial, com posição e diâmetro, o que captura a sua extensão tridimensional com apenas quatro números, x, y, altura e o diâmetro. Uma grelha volumétrica densa gastaria milhares de voxels para descrever o mesmo cilindro, com menos precisão. O volume não é apenas caro, é redundante, porque já existe uma representação mais compacta e mais exata daquilo que precisa de volume. O modo de voo não altera esta conclusão, uma vez que o salto aéreo é uma travessia curta por cima de um obstáculo e não navegação livre em três dimensões por entre o arvoredo.

A malha é construída fundindo, célula a célula, os pontos de solo que a perceção entrega. Cada célula mantém uma estimativa de altura e a respetiva variância, atualizadas por um filtro de Kalman unidimensional. A variância atribuída a cada nova medição cresce com o quadrado da distância ao robô, porque o erro de altura de um ponto de LiDAR aumenta com o alcance, e por isso um ponto medido longe deve pesar menos do que um ponto medido perto. Só se fundem pontos até uma distância limite, e uma medição que se afaste mais do que três desvios-padrão da estimativa atual é

rejeitada, o que impede que um único eco espúrio contamine a célula. Este limiar não serve para cortar a variação natural do terreno, mas sim para travar ecos grosseiros que caem muito para lá dos três desvios-padrão, como um ponto de copa classificado por engano como solo. Um limiar mais apertado começaria a rejeitar degraus e desníveis verdadeiros, que produzem inovações grandes mas legítimas, e que são precisamente o que esta camada precisa de preservar. O *gate* é ainda normalizado pela variância da própria fusão, que cresce com a distância, e por isso a folga admitida adapta-se sozinha ao facto de uma medição feita ao longe ser menos fiável.

Sobre a elevação fundida aplica-se ainda um filtro de mediana numa vizinhança de três por três células. A mediana foi escolhida em vez da média porque remove picos isolados sem esbater os degraus reais do terreno, que são precisamente a informação que interessa preservar para detetar um buraco ou degrau. A Figura 3.26 mostra a malha de terreno resultante, construída desta forma em simulação.

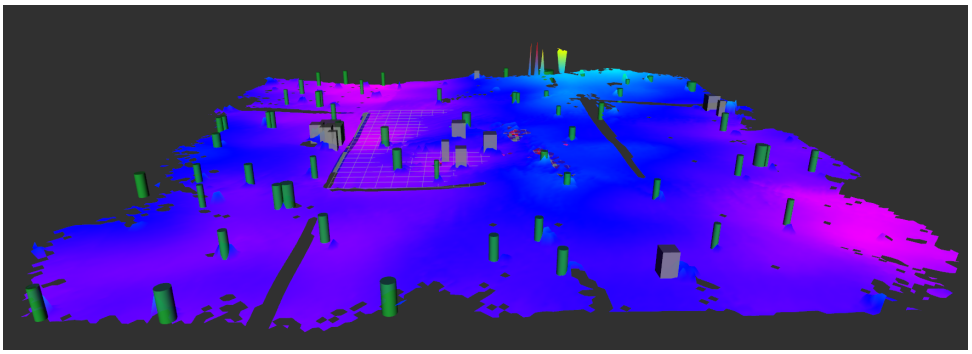


Figura 3.26: Malha de terreno 2.5D construída em simulação. A cor indica a altura estimada do solo em cada célula. Os marcadores sobrepostos assinalam *landmarks* já promovidos pelo SLAM, mostrados aqui apenas como referência, pois pertencem a uma camada distinta da mesma grelha e não a esta camada de elevação.

3.4.3 Terreno persistente em *tiles*

Uma grelha centrada no robô resolve o problema imediato, mas esquece tudo o que fica para trás. Se o robô voltar a um sítio por onde já passou, tem de reconstruir o terreno de raiz, como se nunca lá tivesse estado. Guardar simplesmente uma grelha cada vez maior também não serve, porque a memória cresceria sem limite e, pior ainda, o terreno guardado ficaria preso às poses antigas do robô, que o SLAM ainda vai corrigir.

A solução implementada divide o mundo numa grelha fixa de blocos quadrados, os *tiles*, com coordenadas absolutas que não dependem de onde o robô está. Cada bloco guarda a sua própria elevação e variância, e nunca é esquecido. A janela que sai para o planeamento é reconstruída a partir dos blocos vizinhos a cada ciclo, e por isso é apenas uma vista local sobre um terreno que continua guardado por inteiro.

O ponto delicado é o que acontece quando o SLAM fecha um laço e corrige a trajetória. Todo o terreno que foi escrito com as poses antigas passa a estar no sítio errado. Reintegrar as medições seria caro e, na prática, impossível, porque as nuvens de pontos originais já não existem. A solução

foi ancorar cada bloco a uma keyframe do grafo de poses, guardando qual era a pose dessa keyframe no momento em que o bloco nasceu, como mostra o painel (a) da Figura 3.27.

Quando a otimização move essa keyframe para a sua posição corrigida, mede-se quanto ela se deslocou e rodou, e aplica-se exatamente esse mesmo deslocamento e rotação ao bloco inteiro quando ele é lido, o efeito ilustrado no painel (b) da mesma figura. O bloco vai assim atrás da sua keyframe para o sítio certo, sem que o terreno guardado seja reescrito e sem que uma única medição tenha de ser reprocessada. A aproximação é válida porque dentro de um bloco o percurso do robô é curto e o desvio acumulado é pequeno, e por isso corrigir o bloco inteiro de forma rígida é suficiente.

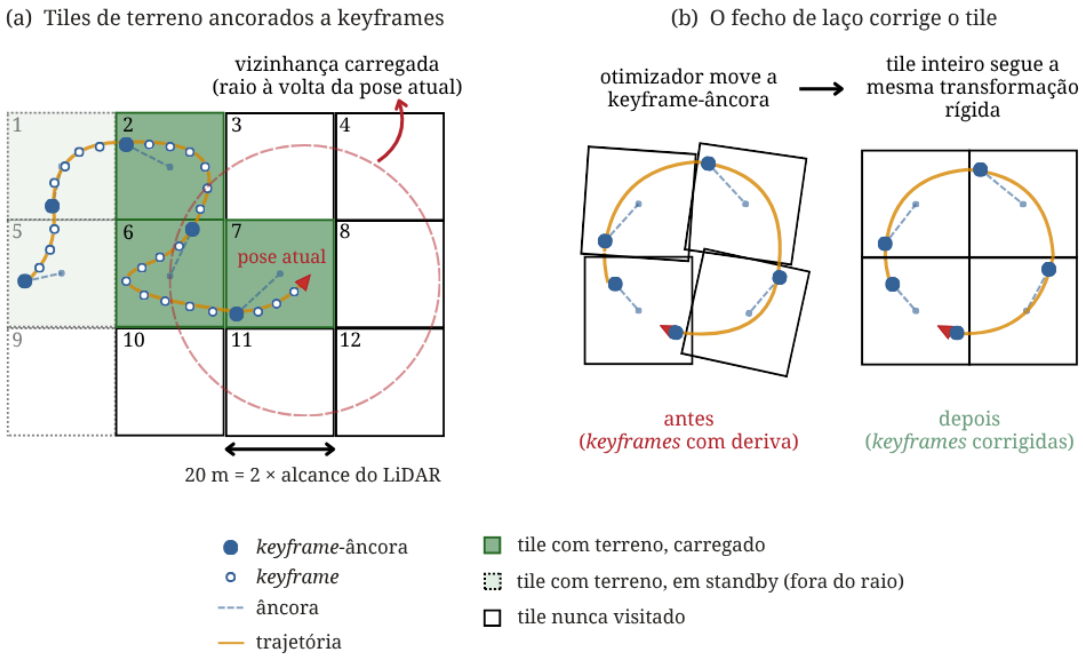


Figura 3.27: Terreno persistente em *tiles*. (a) A grelha fixa de blocos em coordenadas do mundo, com blocos ativos e inativos, cada um a guardar a sua elevação e variância e ancorado à keyframe ativa no momento em que nasceu. (b) O efeito de um fecho de laço, em que o bloco acompanha a correção da sua keyframe pela mesma transformação rígida, sem reprocessar nenhuma medição.

A mesma aritmética de divisão em blocos é partilhada com o Tree-SLAM, e não reimplementada aqui, o que garante que o terreno e os marcos caem sempre na mesma partição do mundo, como mostra a Figura 3.28 em simulação.

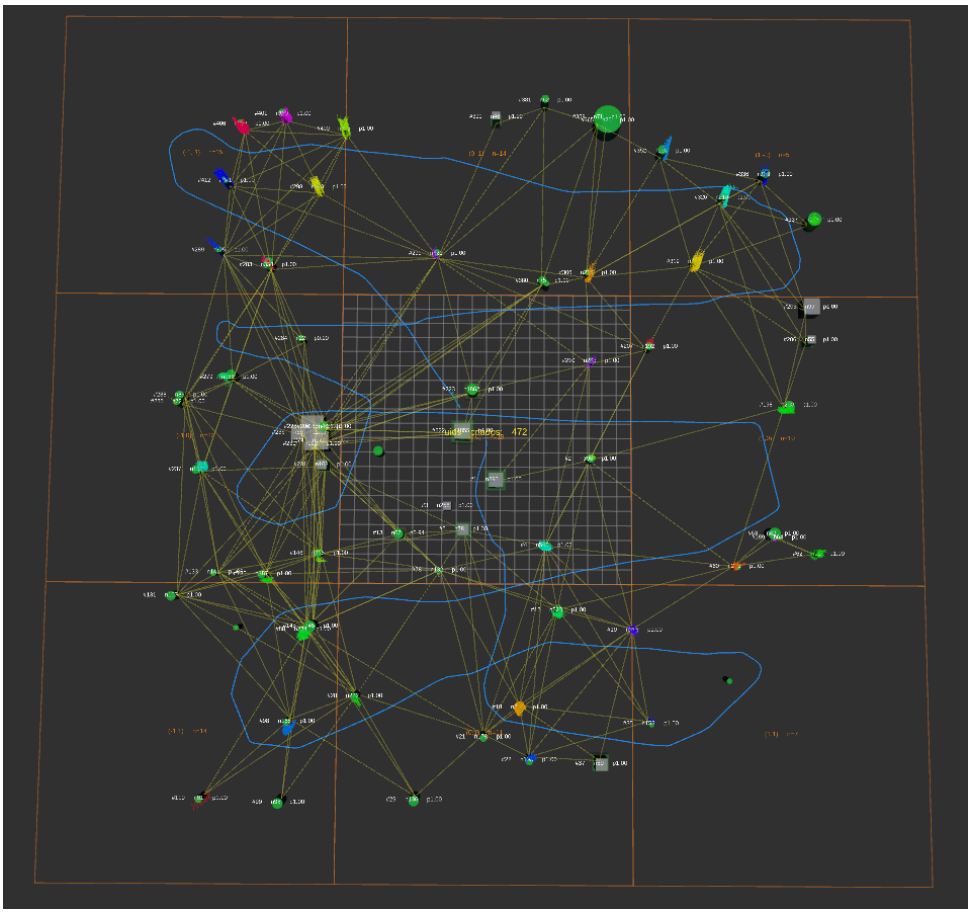


Figura 3.28: Grelha de *tiles* em simulação, com os *landmarks* promovidos pelo SLAM e as ligações de vizinhança entre eles sobrepostas à grelha de blocos, para mostrar como o terreno e os marcos partilham a mesma partição do mundo.

Desta arquitetura sai uma regra de desenho que vale a pena tornar explícita, que é guardar a evidência e recalcular o que dela deriva. A elevação e a variância são evidência, porque resultam de medições que não voltam a existir, e por isso são guardadas. Já o custo é uma quantidade derivada, uma função do terreno e dos objetos, e ambos já estão guardados. Guardar também o custo seria guardar uma cópia que envelhece mal, porque bastaria o fecho de um laço mover um tronco para o custo guardado continuar a marcar obstáculo no sítio antigo. Recalcular o custo em cada ciclo é barato e mantém-no sempre coerente com a última correção do SLAM.

3.4.4 O modelo de custo

O custo de cada célula combina três contribuições, que respondem a perguntas diferentes sobre o mesmo pedaço de terreno.

A primeira é a forma do terreno. Uma célula é tanto mais cara quanto mais inclinada estiver, e o custo de declive cresce de forma linear com a inclinação até saturar no declive máximo que o robô consegue subir. Junta-se-lhe o custo de degrau, que é binário, porque um salto de altura entre

células vizinhas acima de um certo limiar é simplesmente intransponível, e não existe meio-termo útil entre subir um degrau de dez centímetros e um de meio metro.

A segunda contribuição são os objetos, e é aqui que a classificação feita pela perceção ganha utilidade. Um tronco, uma rocha e um tronco caído são sólidos, e por isso recebem custo letal. Já a vegetação mole é penetrável, e recebe um custo alto mas abaixo do letal, o que permite ao planeamento atravessá-la quando a alternativa for um desvio muito longo. Tratar relva alta como se fosse um tronco de carvalho seria deitar fora informação que as camadas anteriores já produziram, e obrigaria o robô a contornar obstáculos que na verdade pode simplesmente empurrar.

A terceira contribuição é a confiança na própria estimativa. Uma célula observada uma única vez, e de longe, é mais arriscada do que uma célula vista dezenas de vezes de perto, mesmo que as duas tenham exatamente o mesmo declive. A variância acumulada na fusão da elevação já mede isso, e por isso é convertida numa margem de risco que se soma ao custo. Assim o robô prefere, entre dois caminhos igualmente planos, aquele que conhece melhor.

O custo final de cada célula é dado pela Equação 3.1, onde c_t é o custo de terreno, c_o o custo de objeto, σ o desvio-padrão da elevação estimada nessa célula, $\sigma_{\max}$ o desvio-padrão que satura a margem de risco e w o peso dessa margem.

$$c \;=\; \min\left(1,\; \underbrace{\left[1-(1-c_t)(1-c_o)\right]}_{\text{perigos combinados}} + w\cdot \underbrace{\min\left(1,\; \frac{\sigma}{\sigma_{\max}}\right)}_{\text{margem de risco}}\right) \tag{3.1}$$

O terreno e os objetos são tratados como causas independentes de risco, e por isso combinam-se da mesma forma que se combinam duas probabilidades de falha, ou seja, o robô só está a salvo se escapar às duas. Poderia parecer mais simples tomar apenas o maior dos dois custos, mas isso teria um efeito indesejado, porque o custo mais fraco seria completamente engolido pelo mais forte. Uma célula na zona de inflação à volta de uma árvore, onde o custo de objeto já decaiu do valor letal mas ainda é apreciável, ficaria com o mesmo custo quer o chão aí fosse plano quer fosse uma encosta acentuada, e a inclinação deixaria de contar precisamente na vizinhança do obstáculo, que é por onde o robô tem de passar. Com a combinação escolhida isso não acontece, e uma encosta junto a uma árvore custa mais do que chão plano junto à mesma árvore. A Figura 3.29 mostra este efeito com duas células à mesma distância de uma árvore.

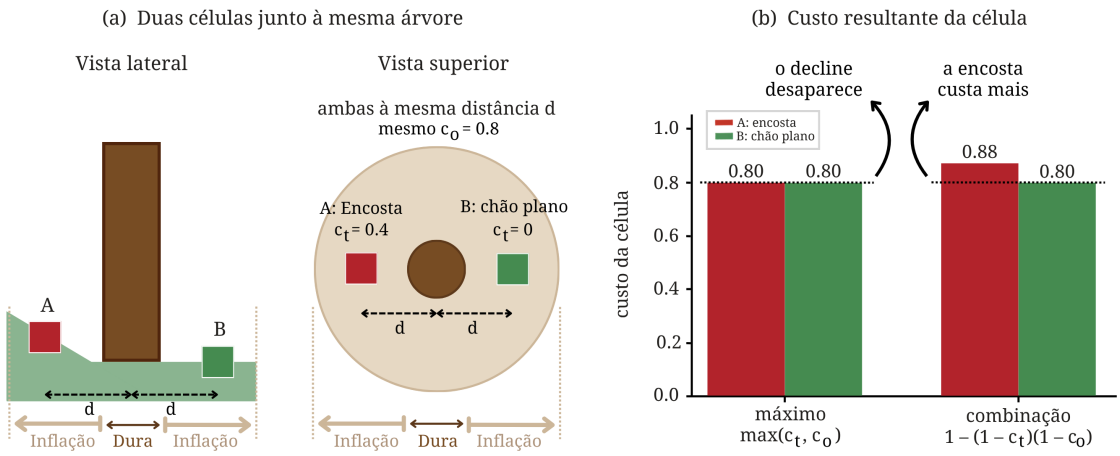


Figura 3.29: Porque a combinação substitui o máximo. Duas células estão à mesma distância de uma árvore, logo com o mesmo custo de objeto, mas uma está em chão plano e a outra numa encosta (a). Com o máximo, ambas ficam com o mesmo custo e o declive desaparece na vizinhança do obstáculo, enquanto a combinação $1-(1-c_t)(1-c_o)$ faz a encosta custar mais (b).

Esta combinação tem ainda duas propriedades que importam. Um obstáculo letal continua letal, porque se um dos custos for o máximo o resultado é sempre o máximo, e por isso um tronco permanece intransponível seja qual for o declive à sua volta. Ao mesmo tempo, dois riscos moderados não somam automaticamente para intransponível, uma vez que uma encosta a meio da escala com vegetação mole por cima resulta num custo alto mas ainda transponível, que é o comportamento pretendido.

Importa por isso deixar claro onde é que o efeito da Figura 3.29 vale, e onde não vale. Ele aplica-se na zona de inflação de qualquer obstáculo, e à volta de um obstáculo penetrável a qualquer distância, incluindo encostado a ele, porque nestes casos c_o nunca chega a 1.0 e por isso nunca engole por completo o termo do terreno. Já dentro do raio duro de um obstáculo sólido, tronco ou rocha, c_o satura exatamente em 1.0, e a primeira propriedade descrita acima, absorvente no letal, aplica-se por igual às duas fórmulas: uma célula em chão plano encostada ao tronco e outra numa encosta igualmente encostada ao mesmo tronco ficam ambas com custo 1.0, sem exceção, e a informação de declive desaparece de qualquer forma, como demonstrado nos resultados.

A margem de risco é somada em vez de combinada como perigo, porque a incerteza não é um perigo em si mas sim a desconfiança na estimativa dos perigos, e portanto agrava um custo que já existe em vez de acrescentar um perigo novo.

Todas as contribuições vivem no intervalo entre zero e um, e o custo final é limitado a esse mesmo intervalo. A Figura 3.30 mostra as curvas que resultam deste modelo.

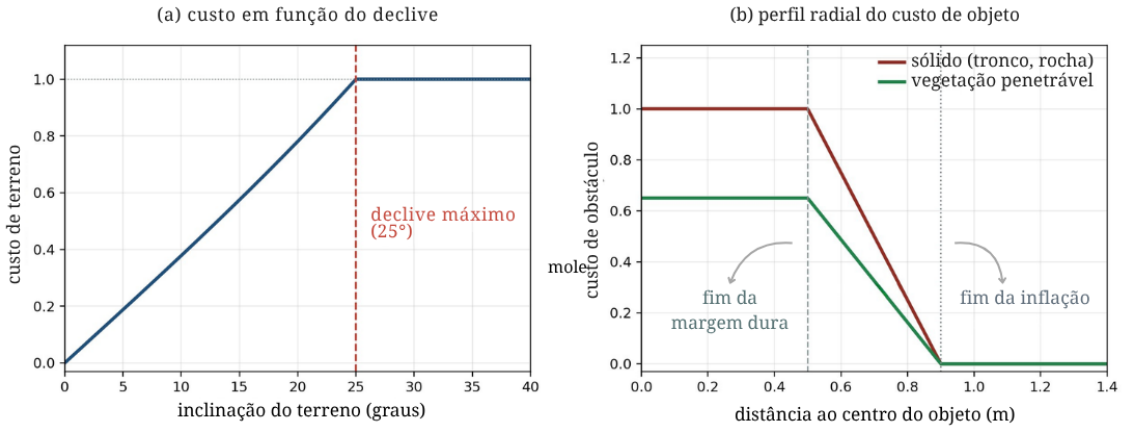


Figura 3.30: Funções de custo implementadas. À esquerda, o custo de terreno em função da inclinação, que satura no declive máximo admissível. À direita, o perfil radial do custo de um objeto, com o sólido a saturar no valor letal e a vegetação mole a saturar abaixo dele, ambos a decair na zona de inflação.

Falta ainda o caso das células que nunca foram observadas. Elas não são publicadas como livres, mas sim como desconhecidas, porque terreno por ver não é terreno seguro, e cabe ao planeamento decidir se quer arriscar entrar nele. A Figura 3.31 mostra o mapa de custo final resultante deste modelo em simulação.

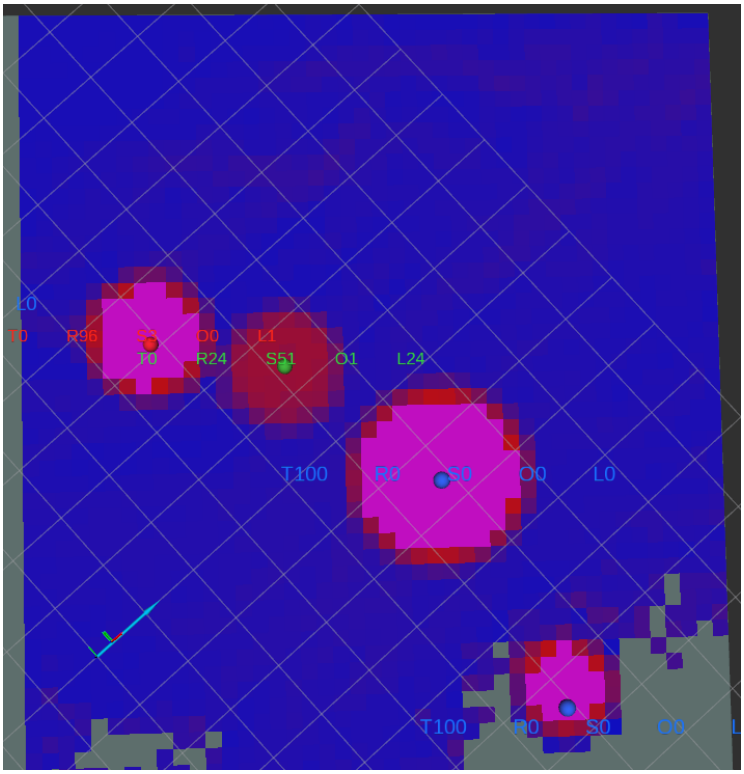


Figura 3.31: Mapa de custo final construído em simulação, combinando o custo de terreno, o custo de objetos e a margem de risco.

3.4.5 Integração com o planeamento

Um mapa de custo contínuo só tem valor se o planeamento o conseguir ler. O *costmap* do Nav2 é composto por camadas que se sobrepõem, e a forma mais comum de lhe entregar informação é através de marcações binárias, em que uma célula está ocupada ou está livre. Entregar o mapa desta camada por essa via destruiria exatamente aquilo que ela produz, porque um arbusto com custo intermédio chegaria ao planeador como uma parede, e uma encosta com 20º de inclinação chegaria como chão perfeitamente livre.

Por isso o mapa de custo é entregue ao Nav2 através de uma camada de *costmap* escrita para o efeito, que traduz o valor contínuo desta camada para a escala de custo do Nav2 preservando os valores intermédios. O planeamento passa assim a poder preferir um caminho menos arriscado, e não apenas um caminho livre, que era tudo o que conseguia distinguir antes. A Figura 3.32 mostra o que se perde por cada via.

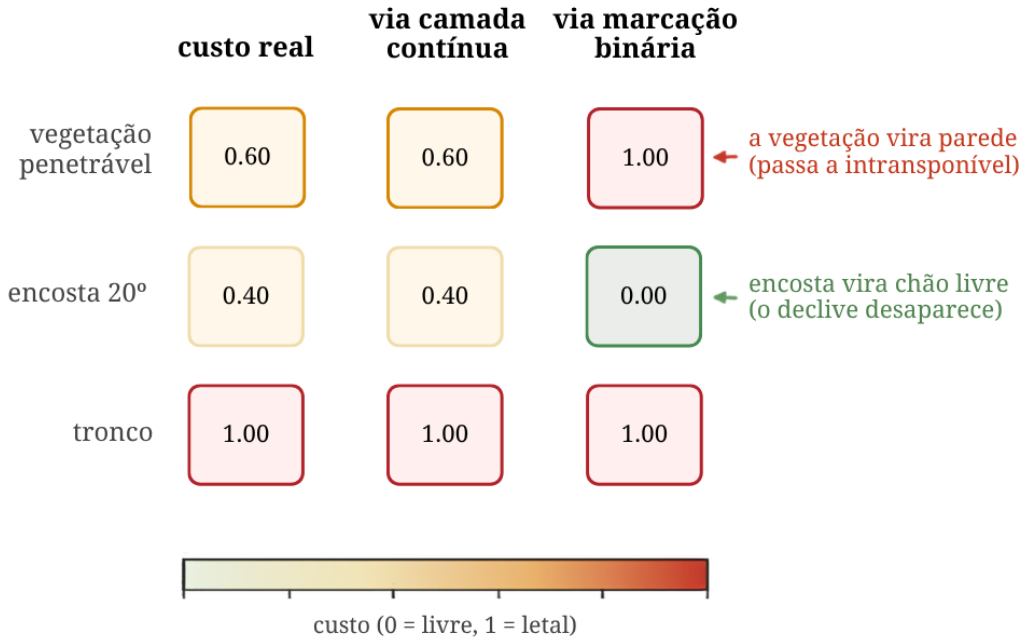


Figura 3.32: O que cada via de entrega ao Nav2 preserva, a camada de custo contínua ou a marcação binária.

Todo o custo produzido por esta camada, tanto o do terreno como o dos objetos, viaja por esta única via e no mesmo intervalo contínuo entre zero e um. Manter uma via separada para os objetos sólidos, entregues como marcações binárias, seria descrever a mesma realidade de duas maneiras diferentes no mesmo sítio, com o risco de a marcação binária se sobrepor ao custo graduado e o anular. A camada de custo combina-se com as restantes camadas do *costmap* pelo máximo, uma escolha diferente da combinação interna entre terreno e objetos descrita atrás e que não a contradiz, aqui o objetivo não é medir dois riscos em simultâneo mas sim garantir que a traversabilidade só pode agravar o custo de uma célula e nunca apagar um obstáculo que outra fonte tenha marcado.

Fica ainda uma ressalva sobre a vegetação penetrável. O caso que justifica um custo intermédio, alto mas abaixo do letal, é a vegetação verdadeiramente atravessável, como relva alta, que o robô pode simplesmente empurrar para passar. A simulação usada neste trabalho não chegou a incluir esse tipo de *asset*, e por isso é o arbusto que desempenha aqui esse papel, apenas para demonstrar como o mecanismo se comportaria perante vegetação penetrável. Um arbusto real não é tão atravessável como relva alta, e num sistema final não deveria receber um custo tão baixo. Modelar corretamente estas classes de vegetação penetrável, com *assets* próprios em simulação, fica como trabalho futuro.

3.5 Planeamento e Decisão

Esta camada decide por onde o robô vai e em que modo de locomoção. A sua raiz é o gestor de missão, que compara o tempo estimado de alcançar o objetivo por terra com o tempo estimado de o alcançar por ar, e a partir dessa comparação encaminha a execução por um de dois ramos. A Figura 3.33 mostra essa estrutura. O ramo terrestre passa pela árvore de comportamento e pelo planeador global D* Lite, entrega o caminho ao controlador e devolve em paralelo ao gestor o tempo estimado por terra e a prova de bloqueio, quando esta existe. O ramo aéreo não tem planeador nenhum, porque a máquina de estados do salto entrega o objetivo diretamente ao autopiloto. Os dois ramos atravessam depois a fronteira da camada e terminam já dentro do controlo, que é onde termina a responsabilidade desta camada.

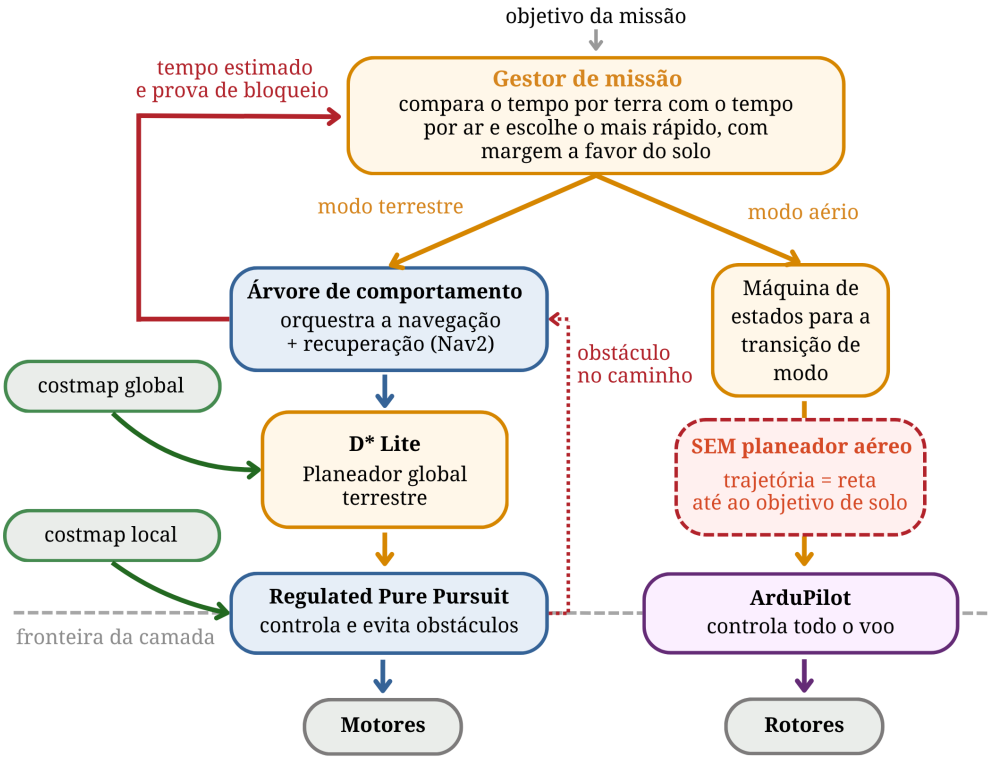


Figura 3.33: Arquitetura de planeamento e decisão, entre o gestor de missão e os dois ramos de execução.

A assimetria entre os dois ramos é deliberada e fica explicada ao longo desta secção. O ramo terrestre tem planeamento global, replaneamento incremental e evitamento local, enquanto o ramo aéreo tem apenas execução, porque a trajetória de voo é uma reta até ao mesmo objetivo de solo.

3.5.1 Planeamento global em ambientes parcialmente conhecidos

O planeamento global foi construído sobre o Nav2, integrado como um *plugin* de planeador do tipo `GridBased`. O grafo de procura sobre o qual este *plugin* raciocina é a grelha de custo produzida pela camada de traversabilidade (Secção 2.5), e não um mapa conhecido antecipadamente.

O algoritmo escolhido para esse *plugin* foi o D* Lite, da família de procura incremental já justificada nos Fundamentos face ao A* e ao RRT*. Dentro dessa família, a alternativa mais próxima discutida ali é o Lifelong Planning A*, do qual o D* Lite deriva, e é sobre essa escolha que vale a pena deter-se aqui. A diferença entre os dois está em relação a que ponto os custos são medidos, ao início do caminho ou ao objetivo. O Lifelong Planning A* mede-os a partir da posição inicial, pelo que cada passo dado pelo robô invalida uma parte crescente do trabalho já feito e obriga a repeti-lo a cada avanço. O D* Lite mede-os a partir do objetivo, um ponto que não se desloca com o robô, pelo que só a deteção de obstáculos novos invalida trabalho já feito, nunca o simples avançar pelo caminho. Além disso, a grelha de custo já existente na camada de traversabilidade encaixa diretamente na representação que o D* Lite espera, sem exigir nenhuma conversão para um espaço de amostragem contínuo.

Uma dificuldade concreta surgiu na integração deste *plugin* com o *costmap* do Nav2. As células do *costmap* que ainda não foram observadas por nenhum sensor são marcadas com um valor especial de sem informação, distinto de livre ou ocupado. O núcleo do D* Lite, tal como implementado, tratava esse valor como sendo maior ou igual ao custo letal, e portanto como um obstáculo. Isto significava que um *costmap* ainda por preencher, típico do início de uma missão, ficava todo bloqueado e o planeamento falhava por completo mesmo em espaço aberto. A correção passou por converter explicitamente esse valor antes de o entregar ao núcleo do D* Lite, tratando-o como espaço livre sempre que o parâmetro correspondente assim o permitir, ou como letal no modo mais conservador. Esta distinção é importante para uma missão de exploração florestal, em que o robô começa sempre com uma parte substancial do mapa por conhecer. A Figura 3.34 contrasta os dois comportamentos.

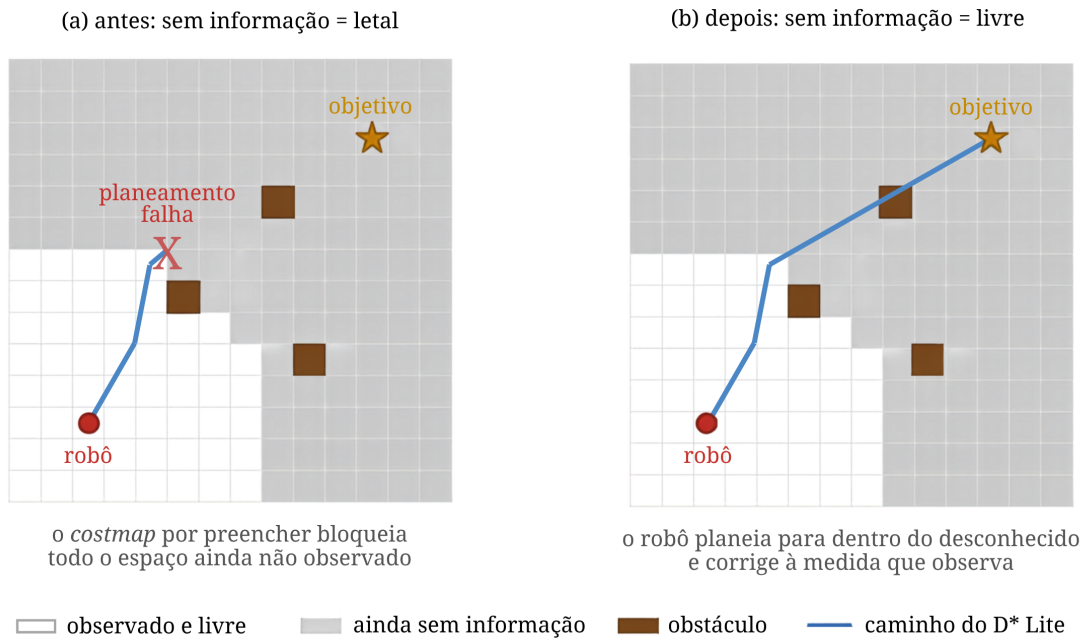


Figura 3.34: Efeito de como o planeador interpreta as células ainda não observadas. Tratadas como letais, todo o espaço por observar fica bloqueado e o planeamento falha logo no arranque da missão, mesmo em terreno aberto (a). Tratadas como livres, o robô planeia para dentro do desconhecido e corrige o caminho à medida que o vai observando (b).

3.5.2 Planeamento local e evitamento de obstáculos

O seguimento do caminho global e o evitamento de obstáculos a curto prazo foram resolvidos por um único componente, o controlador `RegulatedPurePursuitController` do Nav2, em vez de dois módulos independentes. Este controlador estende o *Pure Pursuit*, cuja geometria e base teórica são apresentadas na Secção 2.7.1, com duas capacidades adicionais. A primeira é a regulação da velocidade linear em função da curvatura do caminho, para que o robô abrande nas curvas apertadas e mantenha uma velocidade de cruzeiro maior nos troços retos. A segunda é uma verificação de colisão que projeta para a frente, ao longo de um arco de circunferência, o comando de velocidade prestes a ser enviado, e trava o robô caso essa projeção intersete o mapa de custo local antes de alcançar o ponto de antecipação [59].

Este mapa de custo local é alimentado pela nuvem de pontos não pertencentes ao solo, produzida pela perceção (Secção 2.3), e é atualizado a uma frequência mais alta do que o *costmap* global usado pelo D* Lite. A escolha de acoplar a verificação de colisão diretamente ao controlador de seguimento, em vez de implementar uma das famílias de planeamento reativo apresentadas nos Fundamentos, como o DWA ou os APF, deve-se à simplicidade desta solução face às necessidades reais do robô. Um veículo de lagartas a avançar a baixa velocidade numa floresta não precisa de recalcular constantemente uma trajetória alternativa de desvio, precisa sobretudo de deixar de avançar a tempo quando um obstáculo aparece perto, deixando o replaneamento global do D* Lite tratar de propor um caminho alternativo. Esta divisão de responsabilidades, em que o controlador local trava e o planeador global contorna, troca a capacidade de desviar suavemente de um obstáculo

pela simplicidade de um único componente bem testado, e mostrou-se suficiente para o caso de uso considerado.

Um controlador alternativo, discutido nos Fundamentos, foi tentado antes de se chegar a esta solução e acabou por ser posto de lado. O MPPI, disponibilizado também como *plugin* do Nav2, foi a primeira escolha, mas a sua utilização resultou em comandos de velocidade quase nulos e persistentes, que impediam o robô de avançar de forma fiável. A causa aparente era o custo computacional do próprio algoritmo, que teria deixado de convergir dentro do período de controlo disponível. Uma investigação posterior revelou que a causa real não era o algoritmo em si, mas antes limitações da máquina de simulação usada, nomeadamente a ausência de aceleração gráfica dedicada e memória insuficiente, que atrasavam toda a simulação e não apenas o MPPI. Ainda assim, não se voltou a este controlador, porque o Regulated Pure Pursuit já tinha entretanto mostrado ser adequado ao robô de lagartas considerado, e o tempo disponível para o projeto não justificava reverter uma solução que já funcionava para reavaliar outra que tinha sido posta de lado por um motivo entretanto invalidado.

Quando o robô fica bloqueado apesar da verificação de colisão do controlador e do replaneamento do D* Lite, o Nav2 recorre a comportamentos de recuperação genéricos, como rodar sobre si próprio, recuar ou aguardar, geridos pela árvore de comportamento que orquestra toda a missão. Estes comportamentos não são específicos deste robô nem desta floresta, são os comportamentos de recuperação disponibilizados de origem pelo Nav2, e foram adotados sem alterações por serem suficientes para desencravar o robô nos cenários considerados.

3.5.3 Decisão híbrida

A decisão híbrida não segue nenhum dos três métodos discutidos nos Fundamentos, pois nenhum deles chegou a ser adotado, mas também não se reduz a uma simples regra de limiar. Em vez de uma procura em grafo sobre um espaço combinado de posição e modo, ou de um planeamento hierárquico com controlo de energia unificado, a decisão compara diretamente o tempo estimado de completar a missão por terra com o tempo estimado de a completar por ar, e escolhe o modo mais rápido, com uma margem de segurança a favor do solo.

O tempo por terra é o comprimento do caminho que o D* Lite já encontrou a dividir pela velocidade de solo do robô. Este valor passa a infinito quando há prova de que não existe caminho, o que acontece de duas formas, o Nav2 esgota as suas tentativas de recuperação, ou o planeador deixa de produzir planos novos durante um intervalo de tempo depois de já ter produzido algum antes.

O tempo por ar soma três parcelas, o custo fixo da transição mecânica de descolagem e aterragem, a distância direta entre a posição atual e o objetivo a dividir pela velocidade de cruzeiro do voo, e uma penalização de risco e energia. A decisão só passa para o modo aéreo quando este tempo por ar, multiplicado pela margem de segurança, fica abaixo do tempo por terra. A Figura 3.35 ilustra esta regra.

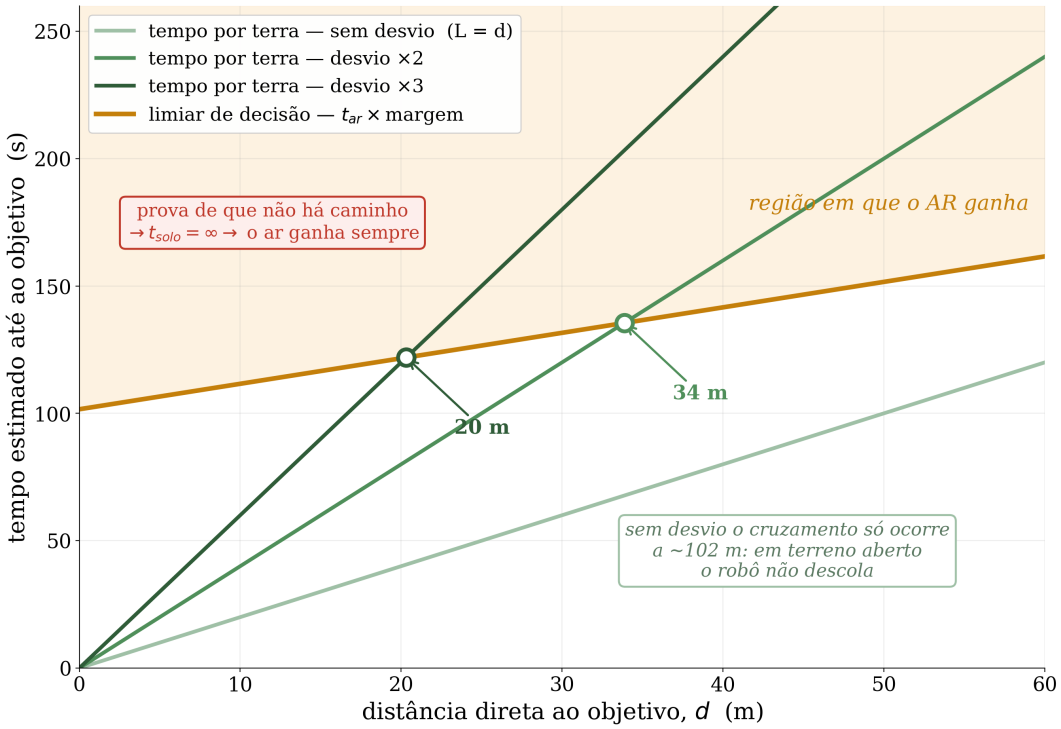


Figura 3.35: Regra de decisão entre os dois modos de locomoção.

Esta penalização de risco e energia substitui um termo que falta e que, num veículo híbrido real, seria o mais importante dos dois, o consumo de energia de cada modo. Voar consome muito mais energia do que deslocar-se por terra, mas sem um modelo de consumo medido não há forma de comparar diretamente as duas formas de locomoção nesse eixo. Por isso a penalização, medida em segundos, inclina deliberadamente a decisão a favor do solo enquanto esse modelo de energia não existir, e fica documentada aqui como o que é, um substituto declarado, em vez de ser apresentada como parte de um custo já completo.

A decisão inclui ainda três salvaguardas, necessárias para não decidir cedo demais nem em condições em que decidir seria arriscado.

- **Tentativa mínima por terra**: exige que o solo tenha tido uma oportunidade real de ser tentado antes de a ausência de caminho contar como prova de bloqueio. Sem esta salvaguarda, o silêncio inicial do *costmap*, ainda por preencher no arranque da missão, levaria o robô a descolar sem sequer tentar avançar por terra.

- **Limpeza pós-salto**: apaga toda a evidência acumulada assim que um salto termina. O salto desloca o robô para um local onde o *costmap* está de novo por preencher, e sem esta limpeza o Nav2 falhava de imediato nesse local novo, o que disparava um segundo salto em cadeia.

- ***Gate* de localização**: impede que o modo aéreo seja escolhido enquanto o SLAM não estiver a localizar-se com confiança, ou seja, sempre que se encontra em relocalização ou perdido, porque saltar nesse momento arriscaria aterrar sem uma pose fiável para retomar a condução.

Uma vez tomada a decisão, a execução do salto sai do âmbito desta camada e passa à máquina de estados de transição de modo já descrita na Secção 3.1.1. O que importa reter aqui é apenas o que essa execução pressupõe do lado do planeamento, ou seja, nada. Não existe planeador aéreo, a trajetória de voo é uma reta até ao mesmo ponto que era o objetivo em modo terrestre, e o robô aterra aí para retomar a condução.

Esta ausência foi deliberada e não uma limitação técnica intransponível. Os três métodos da literatura pressupõem uma integração de voo suficientemente madura para ser controlada com precisão a partir de um plano aéreo, o que inclui posicionamento fiável ao ar livre e uma ligação estável ao autopiloto por protocolo MAVLink. Construir e validar essa integração já consumiu uma parte substancial do tempo disponível para o projeto, pelo que se optou por deixar o voo em si simples e direto, e concentrar o esforço de desenho na comparação de custo que decide quando lhe recorrer. Planear a trajetória aérea, e com isso permitir que o ponto de aterragem seja escolhido em vez de imposto, fica identificado como o passo natural seguinte desta camada.

3.6 Controlo

O seguimento de trajetória e o evitamento de obstáculos em solo estão inteiramente a cargo do *Regulated Pure Pursuit* do Nav2, já descrito na camada de Planeamento (Secção 3.5), que acopla num único componente a geometria de seguimento, apresentada nos Fundamentos (Secção 2.7.1), e a verificação de colisão contra o mapa de custo local. Não existe nesta dissertação nenhuma malha de controlo terrestre de baixo nível separada deste componente, por isso esta secção não volta a tratar do lado terrestre. O que foi de facto decidido e construído ao nível do controlo, para além do que já foi descrito em Planeamento (Secção 3.5), está do lado do voo. Resume-se a uma única escolha, usar um autopiloto dedicado em vez de um controlador de atitude próprio.

3.6.1 Controlo de voo por autopiloto dedicado

O nó que executa o salto aéreo, `hybrid_hop_executor_node`, não contém nenhum controlador de atitude ou de posição. A sua função limita-se a enviar comandos de alto nível ao autopiloto de voo através do MAVLink, o protocolo de mensagens usado para comunicar com autopilotos deste tipo, nomeadamente armar, descolar até à altitude de cruzeiro, avançar até ao ponto-alvo, e aterrar. Quem decide a velocidade de cada rotor a cada instante, e quem mantém o robô estável no ar, é inteiramente o autopiloto, não código desta dissertação. A Figura 3.36 mostra esta fronteira de delegação.

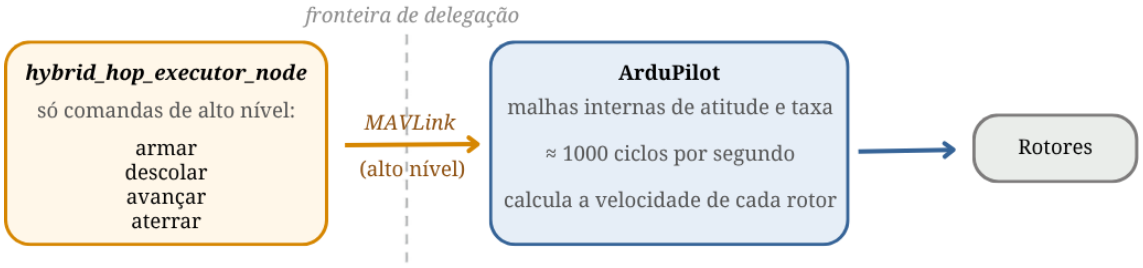


Figura 3.36: Fronteira de delegação do controlo de voo. O nó desta dissertação envia apenas comandos de alto nível por MAVLink, enquanto o ArduPilot corre as malhas internas de atitude e de taxa angular a cerca de mil ciclos por segundo e calcula a velocidade de cada rotor.

Esta divisão de responsabilidades não foi a primeira tentativa. Antes de se chegar ao autopiloto dedicado, foram experimentados dois caminhos de controlo de voo próprio, e ambos foram postos de lado. O primeiro usou um controlador geométrico próprio, uma implementação do controlador de Lee et al. [65] já descrito nos Fundamentos (Secção 2.7.2), mas essa via foi abandonada porque lê o estado do veículo diretamente da física do simulador, um recurso que não existe num robô real. O segundo foi uma malha de atitude própria em ROS 2/Python, a correr a cerca de 50 Hz, que serviu como degrau pedagógico em simulação mas não se mostrou suficientemente rápida para as malhas internas de um voo real, que precisam de correr perto de mil ciclos por segundo. Só depois destas duas tentativas descartadas se decidiu delegar o controlo de voo a um autopiloto já maduro.

Entre os dois autopilotos comparados nos Fundamentos, a escolha recaiu sobre o ArduPilot, mas não por um teste comparativo direto contra o PX4. O plano inicial era experimentar os dois, mas o ArduPilot mostrou-se estável e fiável logo nas primeiras validações em SITL. Além disso, o levantamento de Aliane [69], já discutido nos Fundamentos, mostra que os dois autopilotos são equivalentes em quase todos os critérios relevantes a este projeto, diferindo apenas na licença e no suporte a MATLAB/Simulink, nenhum dos quais pesava na decisão. Não se viu por isso valor em repetir a validação com o PX4 só para confirmar um resultado que a literatura já indicava como pouco provável de ser diferente.

3.7 Simulação, Geração Procedimental e Distribuição Natural

3.7.1 Escolha do Gazebo

Antes de fixar o Gazebo como motor de simulação, foi feita uma primeira tentativa com o Isaac Sim, da NVIDIA. Na altura em que essa avaliação foi feita, o Isaac Sim tinha muito menos material de apoio disponível online, menos tutoriais, menos exemplos de projetos parecidos e menos respostas a problemas comuns em fóruns e documentação. O Gazebo, pelo contrário, já tinha uma comunidade grande e madura à volta do ROS 2, com muito mais exemplos práticos de robôs de campo e de sensores semelhantes aos usados neste projeto. Esta diferença de apoio disponível, mais do que qualquer limitação técnica de um motor face ao outro, foi o que decidiu a

escolha a favor do Gazebo (a lista mais ampla de motores de simulação e o que cada um oferece está descrita em 2.8.1).

3.7.2 O CLI `forest`

Operar a simulação diretamente por comandos individuais do ROS 2 e do Gazebo rapidamente se tornou impraticável. Cada sessão de teste envolve arrancar o Gazebo com o mundo certo, lançar os nós de ponte entre o Gazebo e o ROS 2, esperar que todos os nós estejam prontos, abrir o RViz com a configuração certa, e no fim garantir que tudo é fechado e limpo antes da sessão seguinte, sem processos zombie a ocupar portas ou memória. Repetir esta sequência manualmente, várias vezes por dia, é lento e é fácil de errar um passo. O CLI `forest` nasceu para resolver exatamente este problema, embrulhando toda esta sequência num pequeno conjunto de comandos.

O CLI organiza-se à volta de perfis, ficheiros YAML que descrevem uma configuração completa de teste (por exemplo, simulação com LiDAR 3D e o Nav2 ligado, ou simulação com o robô híbrido a testar a transição de modo). Arrancar um perfil é um único comando:

```
forest up sim-nav2 --world forest_gentle_trees_rocks
```

Este comando arranca o Gazebo com o mundo indicado, liga as pontes de sensores e de pose, arranca o Nav2 e abre o RViz, tudo na ordem certa e à espera que cada componente esteja pronto antes de avançar para o seguinte. Para além de arrancar uma sessão, o CLI cobre todo o ciclo de vida de um teste, desde a preparação até à limpeza final, resumido na Tabela 3.5.

Tabela 3.5: Comandos principais do CLI `forest` e o que cada um faz.

Comando	**Função**
`forest up <perfil>`	Arranca um perfil completo (Gazebo, pontes, Nav2, RViz), em primeiro plano ou em segundo plano
`forest status`	Mostra se há uma sessão ativa e qual
`forest down`	Termina a sessão e verifica que nada ficou pendurado
`forest cleanup`	Limpeza forçada de processos presos, sem depender de uma sessão registada
`forest world list`	Lista os mundos disponíveis no `ForestGen` atual
`forest capture <mundo>`	Gera imagens de um mundo e fecha a simulação de seguida, sem intervenção manual
`forest profile list`	Lista os perfis YAML disponíveis e o que cada um ativa
`forest test <workflow>`	Corre um cenário de teste automatizado (por exemplo patrulha, ir-a-um-ponto, *benchmark*)
`forest diag <nome>`	Corre um diagnóstico dedicado (IMU, *Transform* (frames de coordenadas) (TF), LiDAR, navegação)
`forest panel` / `teleop`	Abre o painel de missão ou o painel de teleoperação sobre uma sessão já ativa
`forest logs [camada]`	Segue os registos da sessão em tempo real, por camada ou todos juntos

Esta tabela cobre só os comandos de topo, cada um dos quais aceita ainda opções próprias (por exemplo, escolher entre LiDAR 2D e 3D, ou entre modo com e sem interface gráfica). O ponto central não é o número de comandos, mas sim o facto de nenhum passo de arranque, verificação ou limpeza precisar de ser feito à mão, o que elimina a fonte mais comum de erro humano ao operar a simulação, como esquecer de fechar um processo ou de o fechar pela ordem errada.

3.7.3 Evolução da geração procedimental de mundos

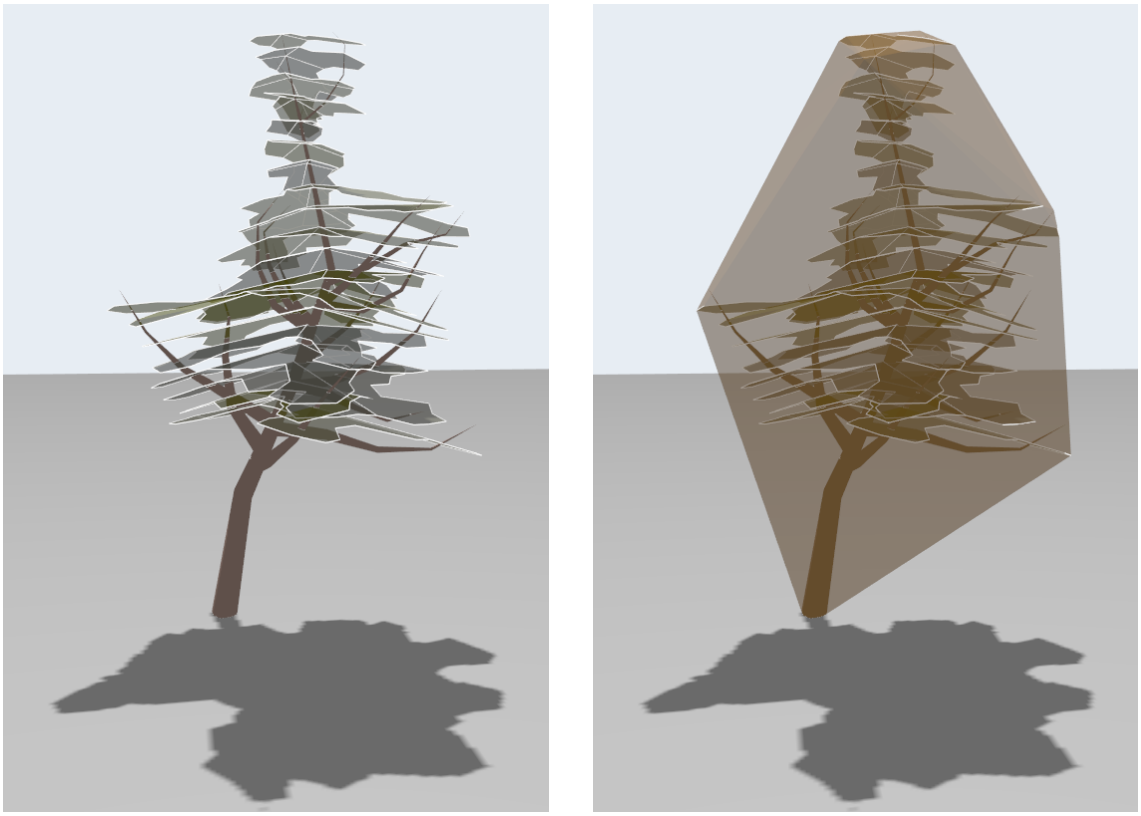
A geração procedimental de mundos florestais não nasceu completa. Foi construída em fases sucessivas, cada uma a acrescentar uma camada de realismo sobre a anterior.

3.7.3.1 Árvores

O ponto de partida foram as próprias árvores. Em vez de desenhar seis espécies de árvore do zero, usaram-se as seis árvores (`Tree1` a `Tree6`) do repositório `forest_gen` da ETH Zurique, da Autonomous Systems Lab, originalmente publicado para avaliação de replaneamento de trajetória de veículos aéreos [79], sem alteração ao aspeto visual das malhas originais.

A única adaptação feita a estas árvores não é visual, mas sim na malha de colisão. Cada árvore original chega com uma malha de colisão tão detalhada como a malha visual, o que torna a física da simulação muito mais pesada sem trazer benefício nenhum, já que o robô não precisa de colidir

com o desenho exato de cada ramo. Por isso, a malha de colisão de cada espécie foi substituída por um invólucro convexo simplificado, que mantém o mesmo volume aproximado da árvore mas reduz drasticamente o número de faces a testar em cada passo de física. A Figura 3.37 compara uma destas árvores tal como chega do repositório original com o invólucro convexo simplificado que substitui a sua malha de colisão.



(a) Malha visual original (b) Invólucro convexo de colisão

Figura 3.37: `Tree1`, repositório `forest_gen` da ETH Zurique [79]: (a) a malha visual original, usada tal e qual no ForestGen; (b) o invólucro convexo simplificado sobreposto à árvore, que substitui a malha de colisão original.

3.7.3.2 Terreno

A geração do próprio terreno, independente das árvores, veio a seguir, a cargo do `solo_gen.py`. A técnica de base é a mesma referida em 2.8.2, o ruído de Perlin, mas aqui usada em três camadas sobrepostas em vez de uma só, cada uma com a sua própria escala espacial e o seu próprio peso na soma final. Uma camada macro, de escala espacial larga e peso dominante, define o relevo geral do terreno, colinas e depressões amplas. Uma camada meso, de escala intermédia e peso menor, sobrepõe-lhe ondulações mais locais. Uma camada micro, de escala fina e peso pequeno, acrescenta por fim uma rugosidade de poucos centímetros que não se nota à distância mas evita que o solo pareça artificialmente liso de perto. A Figura 3.38 mostra o efeito de acrescentar cada camada isoladamente sobre as anteriores.

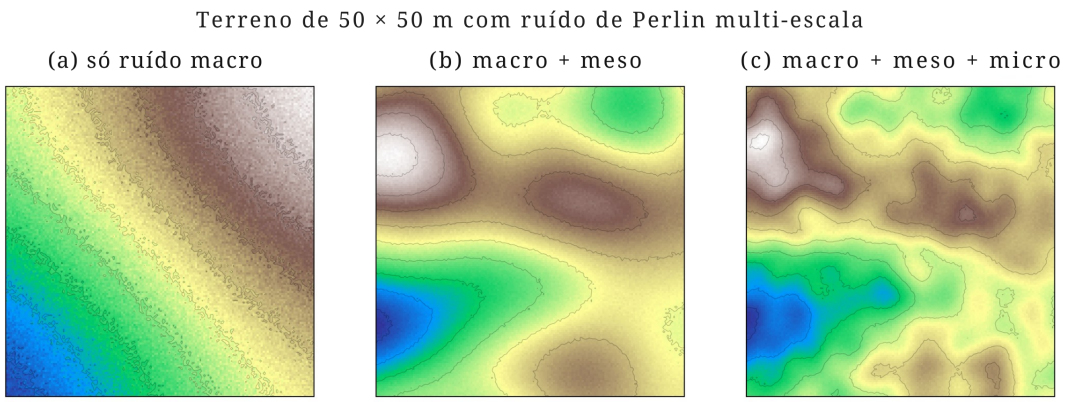
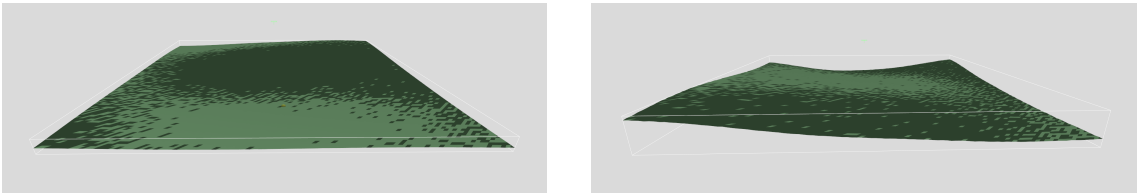


Figura 3.38: Progressão do terreno gerado por `solo_gen.py`: só ruído macro, macro com meso, e macro com meso e micro, para ilustrar o efeito de cada camada isoladamente.

Variando apenas a amplitude aplicada a estas camadas obtêm-se terrenos com relevos bastante diferentes a partir da mesma técnica, como mostra a Figura 3.39.



(a) Relevo suave (b) Relevo moderado

Figura 3.39: Dois terrenos gerados pelo `solo_gen.py` a partir da mesma configuração de camadas, só com a amplitude do relevo alterada, para mostrar a variação que a mesma técnica consegue produzir.

3.7.3.3 Rochas

As rochas (`Rock1` a `Rock3`) e o respetivo gerador (`rock_gen.py`) foram acrescentados pouco depois, ainda nesta fase inicial. Cada rocha parte de uma esfera de baixa resolução cuja superfície é deformada radialmente por ruído gaussiano, o que lhe dá um aspeto irregular sem exigir uma malha detalhada. O mesmo gerador produz três variantes de tamanho a partir do mesmo processo, pequena, média e grande, só mudando a escala de partida da esfera.

Só depois de terem estas três peças, terreno, árvores e rochas, separadas e a funcionar cada uma por si, é que foram fundidas num único gerador de mundo coerente, o `world_builder.py`, que já não gera apenas um mundo estático mas uma matriz inteira de mundos combinando diferentes relevos (plano, suave, moderado) com diferentes conjuntos de obstáculos.

3.7.3.4 Melhorias finais

Passado bastante tempo desde essa primeira versão, decidiu-se investir de novo na geração de mundos para os tornar mais realistas. Foram acrescentados troncos caídos e arbustos como novas

classes de obstáculo baixo, o solo passou a distinguir duas texturas diferentes (relva e terra solta, com atrito diferente) em vez de uma só, e a distribuição espacial das árvores passou a seguir um modelo de agregação com distância mínima em vez de colocação simples (esta parte está descrita em 3.7.4). A Figura 3.40 mostra esta distinção de texturas sobre o terreno gerado.

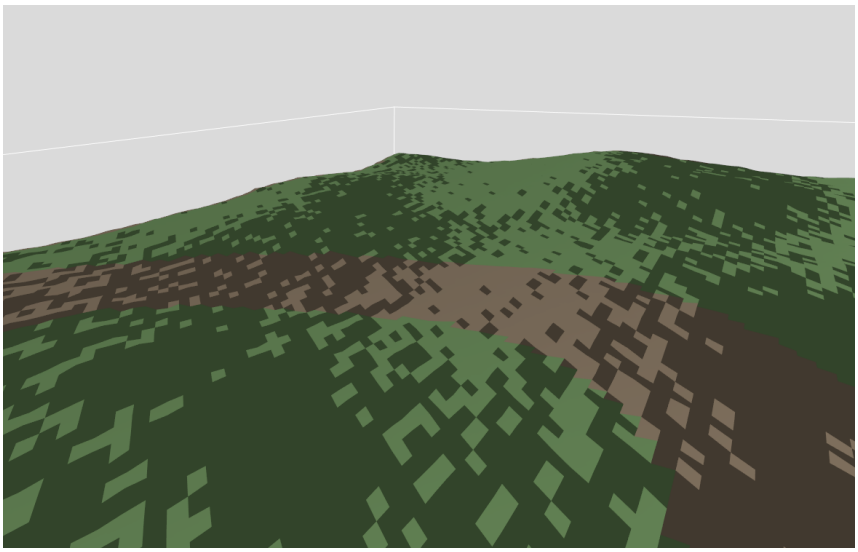
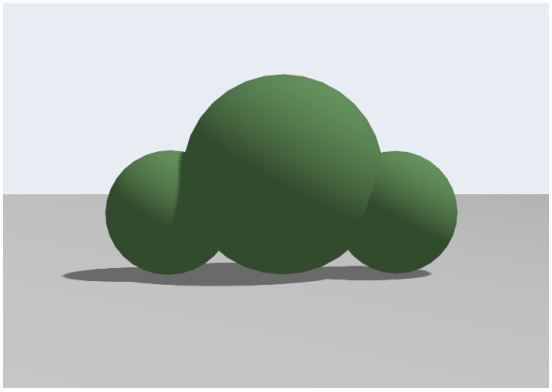
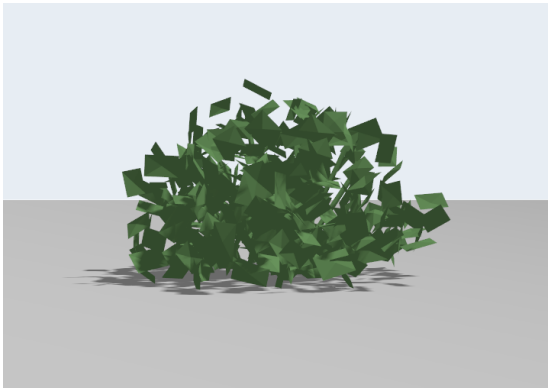


Figura 3.40: Distinção de texturas do solo em simulação: verde para relva, castanho para terra solta, cada uma com um coeficiente de atrito próprio.

Os arbustos, em particular, já existiam antes desta fase como simples aglomerados de esferas, e foram nesta revisão substituídos por uma malha visual mais parecida com um arbusto real, como mostra a comparação da Figura 3.41.



(a) Versão anterior, aglomerado de esferas



(b) Versão atual, malha `bush.stl`

Figura 3.41: Comparação do modelo de arbusto antes e depois do refinamento visual, na mesma posição do mundo, mantendo a mesma forma de colisão em ambas as versões.

A Figura 3.42 reúne o catálogo completo dos modelos base usados na geração de mundos, já com estas classes mais recentes incluídas.

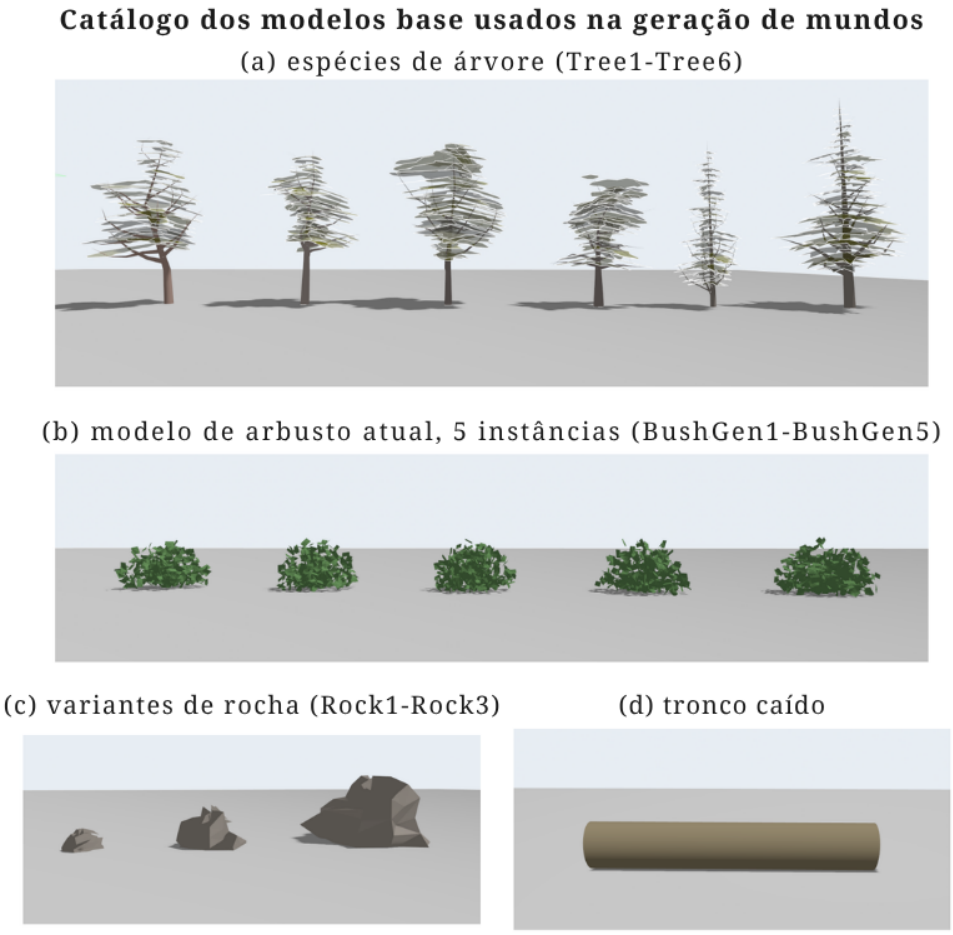


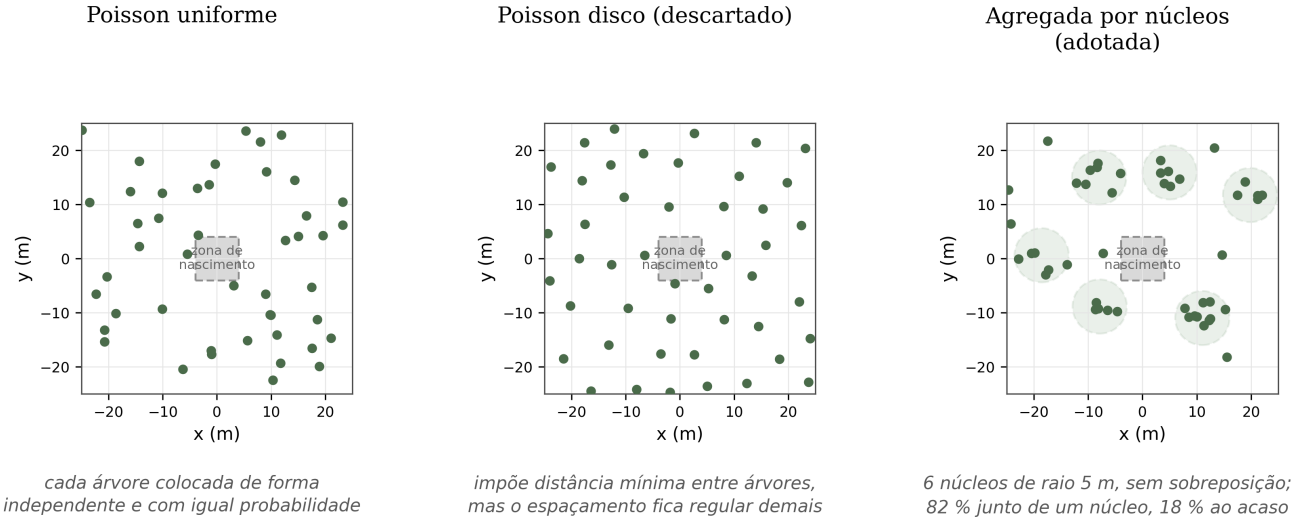
Figura 3.42: Catálogo dos modelos base usados na geração de mundos, cada categoria isolada sobre fundo neutro, sem sobreposição de copas ou de folhagem: (a) as seis espécies de árvore, (b) as cinco instâncias do modelo de arbusto atual, (c) as três variantes de rocha, (d) o tronco caído.

3.7.4 Distribuição natural dos obstáculos

A literatura reunida em 2.8.3 descreve a distribuição espacial de árvores em floresta real como um processo de agregação com inibição por proximidade, e não como uma distribuição uniforme. Esta camada tentou recriar exatamente esse comportamento, em vez de espalhar árvores de forma puramente aleatória pelo mundo.

A colocação segue duas fases. Primeiro, um conjunto de árvores-mãe (pontos-pai) é gerado, distribuído aleatoriamente pelo espaço, cada uma com um número aleatório de árvores descendentes dispersas à sua volta segundo uma distribuição gaussiana, no espírito do processo de Neyman-Scott descrito por Ngo Bieng, Ginisty e Goreaud [77]. Depois, qualquer par de árvores (ou de árvore e rocha, entre classes diferentes) que fique mais próximo do que a distância mínima definida é rejeitado e reposicionado, para que nenhum obstáculo apareça fisicamente dentro de outro. Este segundo passo é uma inibição rígida por distância mínima, mais simples do que o mecanismo de núcleo suave e probabilístico que Ngo Bieng, Ginisty e Goreaud descrevem no seu modelo,

mas resolve o mesmo problema de sobreposição física com muito menos custo computacional. Um primeiro modelo de distribuição, baseado só em amostragem de Poisson disco (Bridson) com distância mínima e sem qualquer agregação, foi testado e descartado por produzir um espaçamento demasiado regular entre árvores, visualmente artificial e sem o efeito de aglomerado que a literatura de Ramón, Velázquez, Escudero e de la Cruz mostra ser a norma em floresta real [76]. A Figura 3.43 compara os três modelos considerados sobre a mesma densidade total de pontos.



(a) Poisson uniforme (b) Poisson disco (descartado) (c) Cluster + distância mínima (final)

Figura 3.43: Comparação de três modelos de distribuição espacial com a mesma densidade total de pontos: Poisson uniforme, Poisson disco isolado (testado e descartado por regularidade excessiva) e o modelo final de agregação com distância mínima.

Além da colocação das árvores em si, o mundo inclui também clareiras, zonas circulares onde a densidade de árvores adultas é reduzida a zero, no espírito do papel ecológico das clareiras descrito por Zhu, Lu e Zhang [78]. Estas clareiras não servem apenas para aproximar o mundo gerado de uma floresta real. A escassez de *landmarks* no seu interior cria também um cenário mais exigente para o SLAM, com muito menos referências à sua volta do que no meio denso da floresta, e o espaço aberto que oferecem é um local mais seguro para a descolagem e aterragem do salto híbrido do que o meio de um povoamento fechado. Nenhuma destas duas utilizações é hoje explorada de forma deliberada pelo código, resultam apenas da geometria das clareiras, mas isso não as torna menos reais. A Figura 3.44 mostra a vista aérea de um mundo gerado com as clareiras assinaladas.

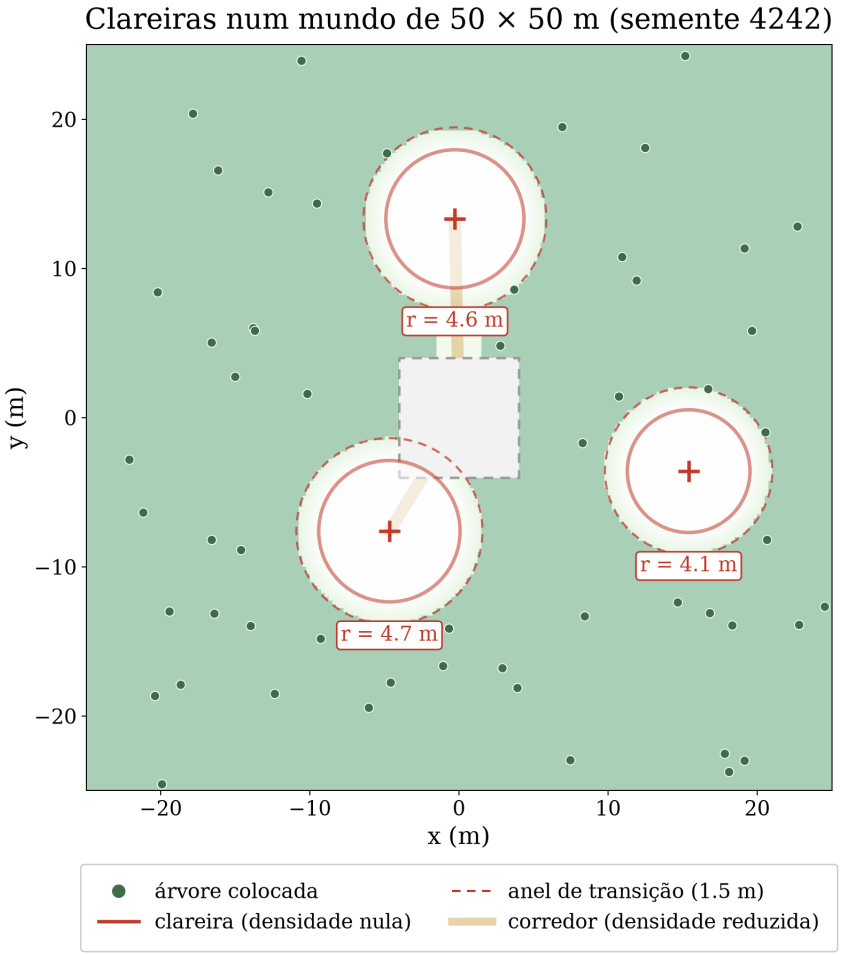


Figura 3.44: Vista aérea de um mundo gerado com as clareiras assinaladas (centro e raio de cada uma sobrepostos à imagem), mostrando as zonas de densidade de árvores reduzida a zero.

Capítulo 4

Resultados Experimentais

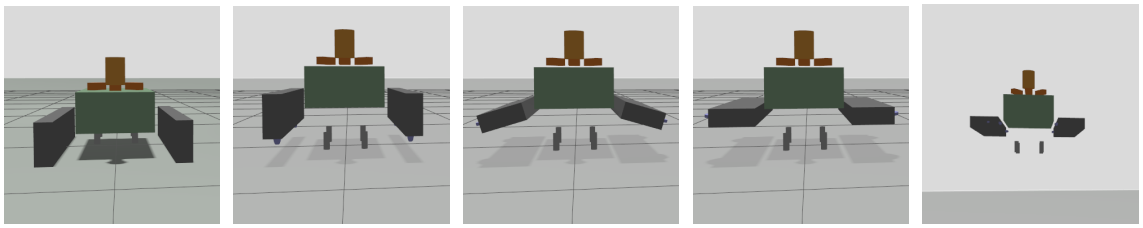
Os testes deste capítulo seguem, secção a secção, a mesma ordem de camadas usada no capítulo de Desenvolvimento Experimental, da robótica híbrida terrestre-aérea até ao controlo. Cada uma dessas secções isola a sua camada da arquitetura e testa-a com o menor número possível de outras camadas a interferir, de modo a que um resultado bom ou mau possa ser atribuído com confiança à camada em análise e não a um efeito indireto de outra parte do sistema. Sempre que isolar uma camada por completo não é possível, porque depende fisicamente de outra para produzir qualquer comportamento observável, essa dependência é assumida e explicada no momento, em vez de se fingir um isolamento que não existe. Depois dessas camadas isoladas, a última secção acrescenta um teste diferente por natureza, já não de uma camada isolada mas do robô completo, com a perceção, o SLAM, o planeamento, a decisão híbrida e o controlo todos ligados ao mesmo tempo numa única missão, a correr em simulação por ser essa a única forma disponível de testar o sistema de ponta a ponta nesta fase do projeto. Cada secção termina assim com uma análise crítica dos seus próprios resultados, e o capítulo fecha com uma síntese que cruza essas conclusões parciais com o resultado desse teste de stack completo.

4.1 Robótica Híbrida Terrestre-Aérea

Esta secção isola a camada mais mecânica de todo o sistema, a transição física entre o modo terrestre e o modo aéreo, e testa-a à parte de tudo o resto, através de dois testes. O primeiro é qualitativo, uma corrida única em simulação capturada em imagens, e confirma visualmente que a máquina de estados passa por todos os modos esperados sem ficar presa nem saltar etapas. Já o segundo é quantitativo, e consiste na repetição do ciclo completo terra-ar-ar-terra trinta vezes seguidas num mundo sem obstáculos, para medir a fiabilidade e a duração de cada estado da máquina isoladamente do resto da navegação. Não se testa aqui, portanto, se o salto é útil para chegar ao objetivo, nem se a decisão de saltar em vez de contornar é acertada perante um obstáculo real, já que essas perguntas dependem da perceção, do planeamento e da decisão híbrida a funcionar em conjunto durante uma missão real, e só podem por isso ser respondidas de forma válida no teste de missão completa em pilha integrada apresentado na Secção 4.7.

4.1.1 Transição observada em simulação

A Figura 4.1 mostra o robô em `GROUND_DRIVE`, com as pernas recolhidas e as lagartas encostadas ao solo, seguido do início da extensão das pernas em `LEGS_EXTENDING`, da rotação das lagartas em `TRACKS_ROTATING`, do robô já em `AERIAL_READY` com as lagartas nos ±90 graus, e por fim do robô em voo.



(a) (b) (c) (d) (e)

Figura 4.1: Transição de modo observada em simulação, do modo terrestre ao voo. (a) `GROUND_DRIVE`. (b) `LEGS_EXTENDING`. (c) `TRACKS_ROTATING`. (d) `AERIAL_READY`. (e) `AERIAL_FLY` / `AERIAL_HOVER`.

A transição completa foi validada com sucesso em simulação, com o robô a atingir um voo pairado estável depois da rotação das lagartas. A aterragem e o regresso ao modo terrestre, pelo caminho inverso da máquina de estados, também foram validados com sucesso.

4.1.2 Fiabilidade da transição

Para além desta validação pontual, o ciclo completo terra-ar-ar-terra foi repetido trinta vezes seguidas, isolando exclusivamente o desempenho da máquina de estados e do autopiloto. As trinta repetições terminaram todas com sucesso e nenhuma entrou no estado `FAILED`. A duração média do ciclo completo foi de 33,4 segundos, com um desvio-padrão de 3,3 segundos, entre um mínimo de 29 e um máximo de 42 segundos, como resume a Tabela 4.1.

Tabela 4.1: Fiabilidade da máquina de estados ao longo de trinta ciclos terra-ar-ar-terra.

Métrica	Valor
Ciclos completos	30/30 (100,0%)
Duração média do ciclo (s)	33,4
Desvio-padrão (s)	3,3
Mín. / Máx. (s)	29 / 42

A Tabela 4.2 e a Figura 4.2 detalham quanto tempo a máquina passa em cada um dos seus estados ao longo dessas trinta repetições. Os dois estados que dependem do autopiloto, `AERIAL_HOVER` e `AERIAL_FLY`, dominam a duração de cada ciclo, com médias de 16,3 e 8,4 segundos respetivamente. Já os estados puramente mecânicos, como a extensão das pernas ou a rotação das lagartas, terminam em pouco mais de um segundo cada, e fazem-no de forma consistente, com

desvios-padrão sempre abaixo de 0,1 segundos nesses casos. Isto mostra que a parte mecânica da transição em si é rápida e repetível, e que a maior parte do tempo de cada ciclo é gasta a voar e a confirmar que o robô está de facto no ar, e não na transformação mecânica. O estado `TRACKS_ROTATING` surge com o dobro das amostras dos restantes porque ocorre duas vezes em cada ciclo, uma vez na saída para o ar e outra vez no regresso ao solo.

Tabela 4.2: Tempo de permanência em cada estado da máquina, ao longo de trinta ciclos.

Estado	n	média (s)	σ (s)	máx. (s)
AERIAL_HOVER	30	16,34	0,17	16,73
AERIAL_FLY	30	8,36	1,80	13,86
TRACKS_ROTATING	60	1,36	0,06	1,52
GROUND_DRIVE	30	0,88	0,64	2,51
TRANSITION_LOCK	30	0,78	0,14	0,96
LEGS_RETRACTING	30	0,61	0,09	0,92
LEGS_EXTENDING	30	0,60	0,03	0,61
AERIAL_READY	30	0,10	0,00	0,10

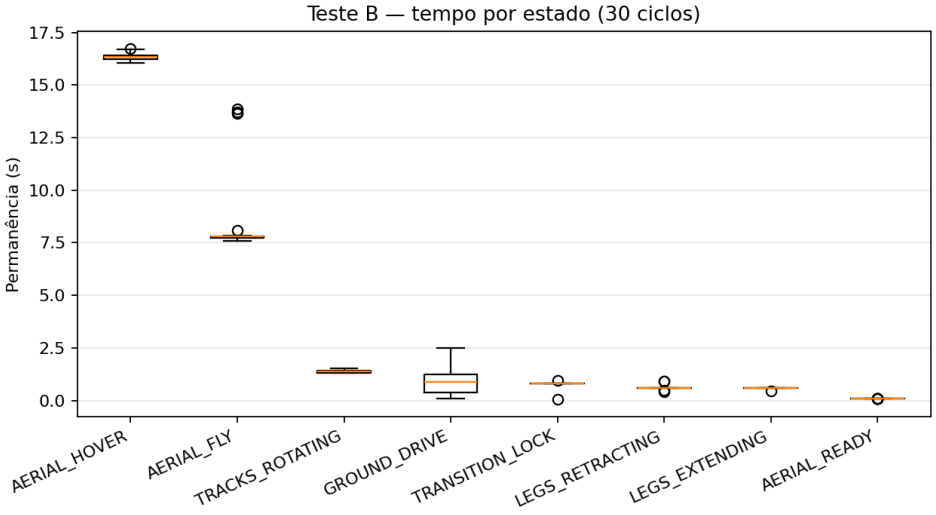


Figura 4.2: Distribuição do tempo de permanência em cada estado da máquina, ao longo de trinta ciclos terra-ar-ar-terra.

4.1.3 Análise crítica

O resultado de 30/30 é sólido dentro do que foi de facto testado, pois a máquina de estados nunca falhou uma transição nem ficou presa num estado, mas o cenário é o mais favorável possível, um mundo vazio, sem obstáculos, sem vento e sempre com a mesma distância e altitude de salto, o que isola bem o comportamento da própria máquina de estados e do autopiloto, que é o objetivo desta secção. Por isso é também de esperar que os resultados nestas condições, e ainda por cima

em simulação, dessem valores tão perfeitos, e é possível que em ambiente real, com todas as fontes de incerteza e de perturbação que a simulação não reproduz, esse valor fosse diferente.

4.2 Perceção

Esta secção não se limita a reportar um número final de acerto por sensor. Os resultados aqui apresentados nasceram de um processo de diagnóstico sucessivo, em que cada teste existe porque o teste anterior levantou uma pergunta concreta que só um novo ensaio, desenhado de propósito, conseguia responder. Primeiro mediu-se o classificador em bancos de ensaio offline isolados, cada um sobre um único *cluster* ou imagem já recortada, sem simulador nem ROS 2 a correr, o que isola por completo a perceção do resto do sistema mas paga esse isolamento com um cenário artificialmente favorável, de nuvens de pontos limpas e já segmentadas. Esse resultado offline definiu um teto de referência, e a pergunta natural a seguir foi saber quanto desse teto sobrevive com o robô a conduzir de facto por um mundo completo. Testou-se por isso o mesmo classificador ao vivo, nos três mundos de simulação, mantendo ainda assim a perceção isolada da localização, porque essa condução usa diretamente a pose *ground-truth* do simulador em vez do EKF ou do SLAM, de forma a que nenhum erro de localização seja contabilizado como erro de perceção. A diferença entre estes dois resultados, o teto offline e o valor ao vivo, é que levantou as perguntas mais específicas respondidas pelos ensaios seguintes, cada uma delas justificada no momento em que surge.

4.2.1 LiDAR

A qualidade da separação entre pontos de tronco e pontos de copa ou ramos dentro do mesmo *cluster* foi também validada, num banco de ensaio offline, sem simulador nem ROS 2 a correr, que faz *raycasting* sobre a mesh visual real das 6 árvores usadas nos mundos de simulação e testa a seleção geométrica de pontos do fuste em 96 cenas, combinando as 6 árvores com 4 distâncias (3, 4, 6 e 8m) e 4 orientações do sensor em torno de cada uma (0°, 90°, 180° e 270°). A Figura 4.3 mostra uma vista representativa por árvore, com os pontos de tronco, copa/ramos e solo a cores diferentes, e a azul-escuro os pontos que a seleção geométrica escolheu como fuste.

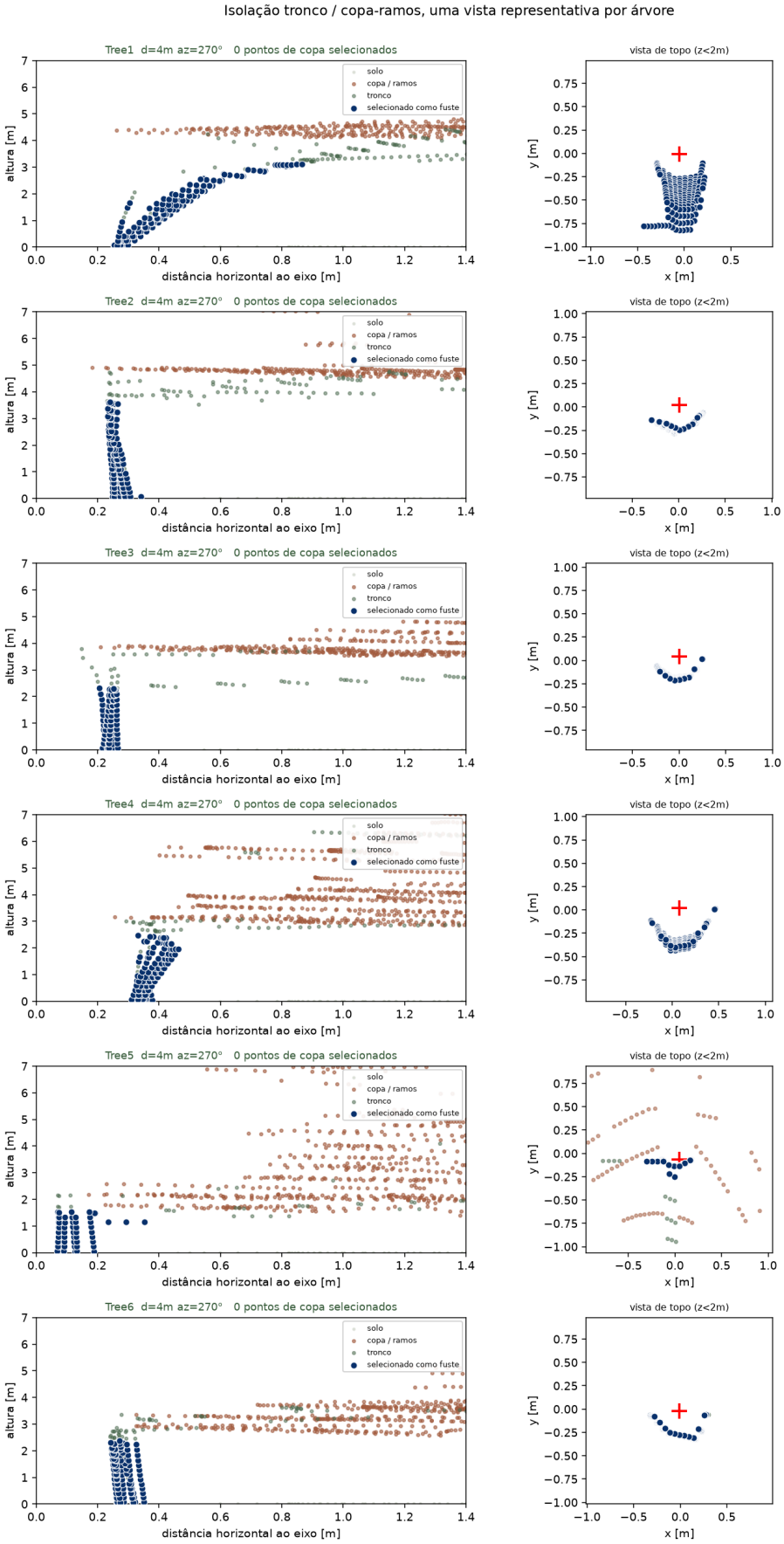


Figura 4.3: Isolação de pontos de tronco (fuste) em relação a copa e ramos, uma vista representativa por árvore.

Nas 96 cenas testadas, a seleção de fuste nunca incluiu um único ponto de copa ou ramo, tirando um ramo que não foi possível evitar na Tree5. A Tabela 4.3 mostra quantos pontos ficaram disponíveis para cada árvore a cada distância, em média sobre as 4 orientações, e esse número desce com a distância porque o LiDAR devolve sempre menos pontos quanto mais longe está o objeto. A Tree5 destaca-se por ter sempre menos pontos disponíveis do que as restantes, não por ser a árvore mais pequena do conjunto, mas porque a sua copa começa muito mais abaixo do que a das outras cinco, o que encurta bastante a zona de tronco limpo antes de a copa começar a interferir. É uma limitação da própria árvore, não do método de seleção, que mesmo assim continua a excluir toda a contaminação.

Tabela 4.3: Pontos médios selecionados como fuste por árvore e distância (sem pontos de copa/ramos, exceto um ramo residual na Tree5).

	3m	**4m**	**6m**	**8m**
Tree1	238	131	55	25
Tree2	262	139	58	29
Tree3	182	94	46	19
Tree4	237	99	47	22
Tree5	55	32	14	8
Tree6	248	104	48	18

4.2.1.1 Resultados por classe

A Figura 4.4 mostra a matriz de confusão do classificador contra o *ground truth*, medida no mesmo banco de ensaio offline com *raycasting* sobre as malhas reais, às distâncias de 3, 4, 6 e 8m e às quatro orientações do sensor. A matriz cobre apenas a árvore e a rocha, tanto na classe real como na classe prevista. O arbusto e o tronco caído ficam de fora dos dois lados porque, quando este ensaio foi feito, nenhum dos dois tinha ainda sido modelado nem existia lógica dedicada no classificador para os distinguir.

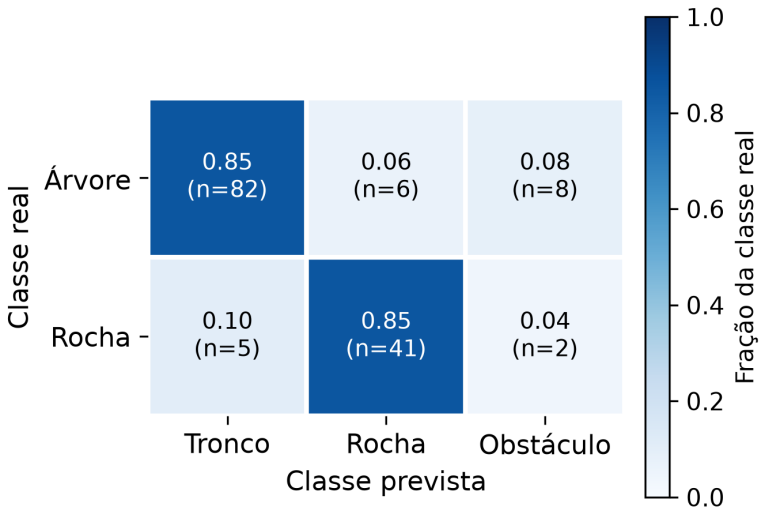


Figura 4.4: Matriz de confusão do classificador LiDAR, árvore e rocha (fração da classe real em cada classe prevista).

A árvore é reconhecida como tronco em 85% das cenas (82 em 96), e a rocha é reconhecida como rocha na mesma proporção (41 em 48). A árvore falha sobretudo para obstáculo (8%), sem evidência de copa acima do *cluster*, por exemplo a longa distância, onde a copa real já fica fora do alcance do sensor, e o classificador prefere não afirmar tronco com confiança em vez de arriscar uma classe errada. A rocha falha sobretudo para tronco (10%, 5 casos), porque algumas vistas de rocha, a certos ângulos, devolvem um perfil mais cilíndrico do que o habitual, apesar de a rocha ser bastante mais grossa do que um tronco real, e a análise por fatias interpreta esse perfil como núcleo de fuste.

4.2.2 Câmara

O detetor treinado descrito na Secção 3.2.2 foi avaliado no conjunto de validação separado dos mundos de captura, imagens que o modelo nunca viu durante o treino.

4.2.2.1 Resultados de deteção

A Tabela 4.4 mostra a precisão, a exaustividade (*recall*) e a média de precisão (mAP) por classe na validação. A Figura 4.5 mostra a matriz de confusão correspondente, que inclui uma linha e coluna extra de fundo (*background*), deteções que não correspondem a nenhum objeto real e objetos reais que o detetor não encontrou.

Tabela 4.4: Resultados de deteção por classe (conjunto de validação).

Classe	**Precisão**	**Recall**	**mAP@50**	**mAP@50-95**
Tronco	0.81	0.80	0.86	0.53
Rocha	0.71	0.60	0.67	0.33
Arbusto	0.70	0.74	0.74	0.48
Tronco caído	0.83	0.82	0.88	0.60
Global	0.76	0.74	0.79	0.48

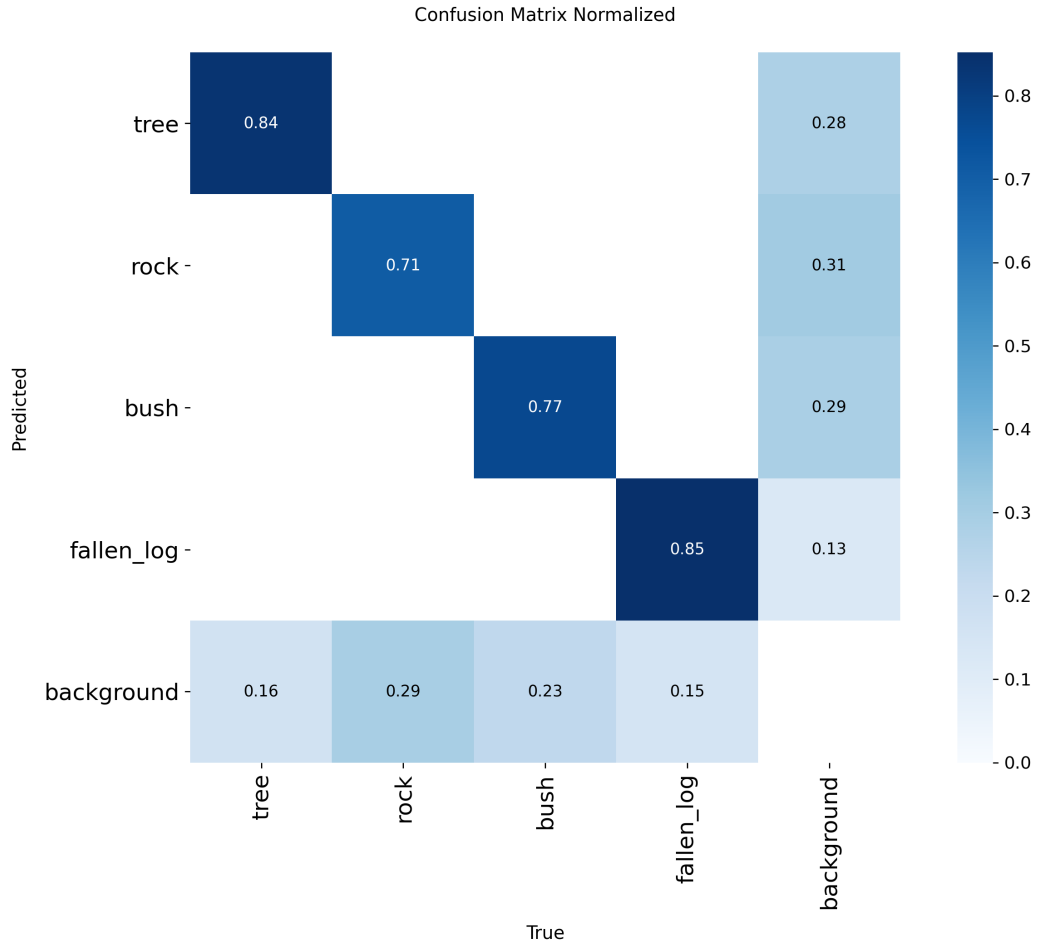


Figura 4.5: Matriz de confusão normalizada do detetor de câmara, incluindo fundo.

Não há confusão relevante entre pares de classes de objeto, toda a massa fora da diagonal cai na linha ou coluna de fundo, o detetor não confunde uma rocha com um arbusto nem um tronco caído com um tronco, os seus erros são de deteção (objeto perdido ou falso positivo), não de classificação entre categorias.

4.2.3 Deteção do LiDAR ao vivo, nos três mundos

Todos os resultados desta subsecção, até à comparação final com a câmara, dizem respeito só ao classificador LiDAR a funcionar ao vivo, com o robô em movimento.

4.2.3.1 Metodologia

O robô percorreu três mundos, `moderate`, `gentle` e `steep`, ao longo de uma rota de cobertura sem colisões, planeada com folga contínua à superfície de cada objeto e não por ocupação de célula, já que uma grelha discreta corta cantos em diagonal e permitiria rotas que na prática embatem. A Tabela 4.5 resume as três corridas.

Tabela 4.5: Execução das três corridas de avaliação ao vivo, robô conduzido pela pose *ground-truth*.

Mundo	Waypoints	Percurso	Presos	Frames LiDAR	Frames câmara
moderate	648/649	296 m	0	3434	13 722
gentle	649/650	296 m	0	3403	27 108
steep	644/645	292 m	0	3382	40 448

Este ensaio separa-se de um simples registo de deteções em dois pontos. Primeiro, deteção e classificação são medidas em separado. A deteção só pergunta se existe um *landmark* no sítio certo, a classificação pergunta além disso se a classe está certa, e uma árvore vista mas rotulada como rocha conta como erro de classificação, não como deteção falhada. Segundo, o denominador de cada métrica só conta objetos que o sensor podia fisicamente medir naquele instante, excluindo os que estavam fora de alcance, abaixo do cone de varrimento ou ocluídos, para que o classificador não seja penalizado por algo que nunca esteve ao seu alcance.

Esta segunda escolha revelou uma falha própria de metodologia, corrigida antes dos números finais aqui apresentados. Ter feixes do LiDAR a atravessar um objeto não basta para o classificar, o pipeline de perceção ainda tem de conseguir agrupar esses pontos num *cluster*, e cada um dos dois caminhos de deteção descritos nos Desenvolvimentos tem a sua própria condição mínima para o fazer. Sem essa condição satisfeita o objeto é estruturalmente impossível de detetar, e contá-lo no denominador mede o impossível em vez de medir o classificador. A primeira versão deste ensaio não fazia esta distinção, e a fração de observações afetadas era considerável, entre 30,8% e 32,5% no caso da árvore, entre 13,1% e 22,2% na rocha, e à volta de 6,5% a 6,9% no arbusto. O efeito recai quase todo sobre a árvore porque é a única classe com um caminho de deteção pela retaguarda do robô, o que a torna também a única a acumular muitas observações mesmo no limiar entre ser tocada pelos feixes e ser de facto agrupável.

4.2.3.2 Deteção e classificação por distância

Separar deteção de classificação levanta de imediato a pergunta seguinte, saber se as duas se comportam da mesma forma à medida que o objeto se afasta do robô, ou se decaem por razões

diferentes. A Tabela 4.6 mostra a deteção e a classificação por classe e por distância à superfície do objeto, e a Figura 4.6 traça a mesma informação em curva contínua.

Tabela 4.6: Deteção/classificação por distância à superfície, corrida `moderate`.

Distância	Árvore	Rocha	Arbusto	Tronco caído
0–2 m	0.74 / 0.60	0.78 / 0.21	0.64 / 0.44	0.43 / 0.09
2–4 m	0.81 / 0.73	0.85 / 0.25	0.78 / 0.42	0.55 / 0.26
4–6 m	0.85 / 0.72	0.65 / 0.21	0.62 / 0.28	0.23 / 0.01
6–8 m	0.86 / 0.77	0.38 / 0.13	0.23 / 0.01	0.24 / 0.00
8–10 m	0.83 / 0.67	0.15 / 0.04	0.09 / 0.00	0.20 / 0.00

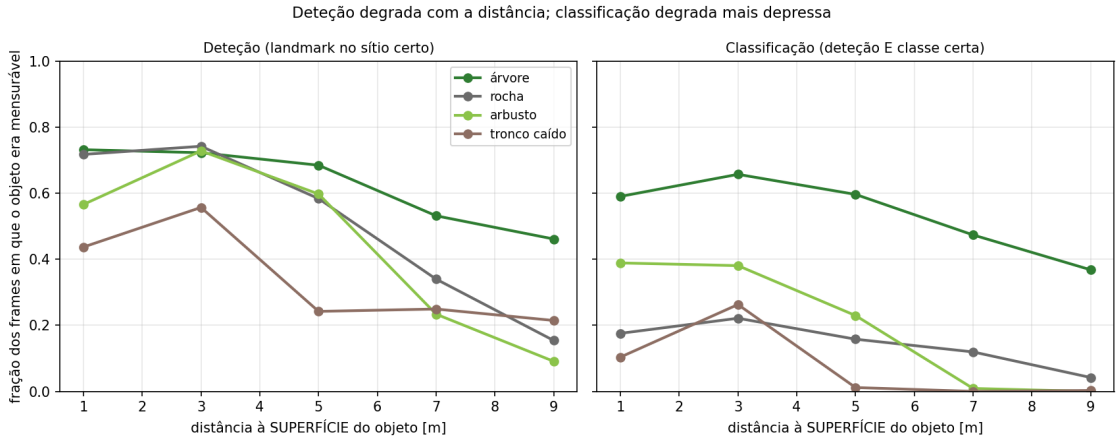


Figura 4.6: Deteção e classificação por distância à superfície, por classe.

A deteção é sempre bastante melhor do que a classificação nas três classes que não são árvore, a rocha por exemplo é detetada em cerca de cinco de cada seis *frames* entre os 2 e os 4 metros, mas só é classificada como rocha em um de cada quatro. Já a árvore deixa de decair com a distância assim que o denominador é corrigido, mantendo-se entre 0,81 e 0,86 de deteção dos 2 aos 10 metros, em vez da queda de 0,68 para 0,47 que aparecia antes da correção. Essa queda antiga era um artefacto de medição, porque a fração de observações fora do envelope de deteção cresce precisamente com a distância, o que inchava o denominador nos escalões mais longínquos. O que continua a decair de facto é a rocha e o arbusto, e para essas duas classes a causa é física, e não uma falha de medição, como confirma o argumento da densidade de pontos a seguir.

4.2.3.3 O argumento do multi-vista

Ficou por explicar por que é que a rocha e o arbusto continuam a decair com a distância mesmo depois de corrigido o denominador, enquanto a árvore não. A hipótese mais óbvia é a densidade de pontos, e é essa hipótese que este ensaio testa diretamente. A Tabela 4.7 relaciona a classificação com o número de pontos previsto pela geometria do sensor para cada observação, uma variável

que depende só do LiDAR e do tamanho do objeto, e nunca de qualquer decisão do *clustering* ou do classificador, o que evita qualquer circularidade na conclusão. A Figura 4.7 mostra a mesma relação em curva.

Tabela 4.7: Classificação em função do número de pontos previsto pela geometria do sensor. O tronco caído não tem valores nos três últimos escalões porque a sua geometria, um tubo fino e baixo deitado no chão, raramente devolve tantos pontos mesmo a curta distância, e por isso não há observações reais nesses escalões.

Pontos previstos	**Árvore**	**Rocha**	**Arbusto**	**Tronco caído**
0–25	0.02	0.08	0.00	0.02
25–50	0.32	0.10	0.07	0.24
50–75	0.63	0.14	0.38	0.30
75–100	0.70	0.17	0.33	–
100–125	0.64	0.21	0.55	–
125–150	0.60	0.34	0.47	–
200+	0.58	0.29	0.57	–

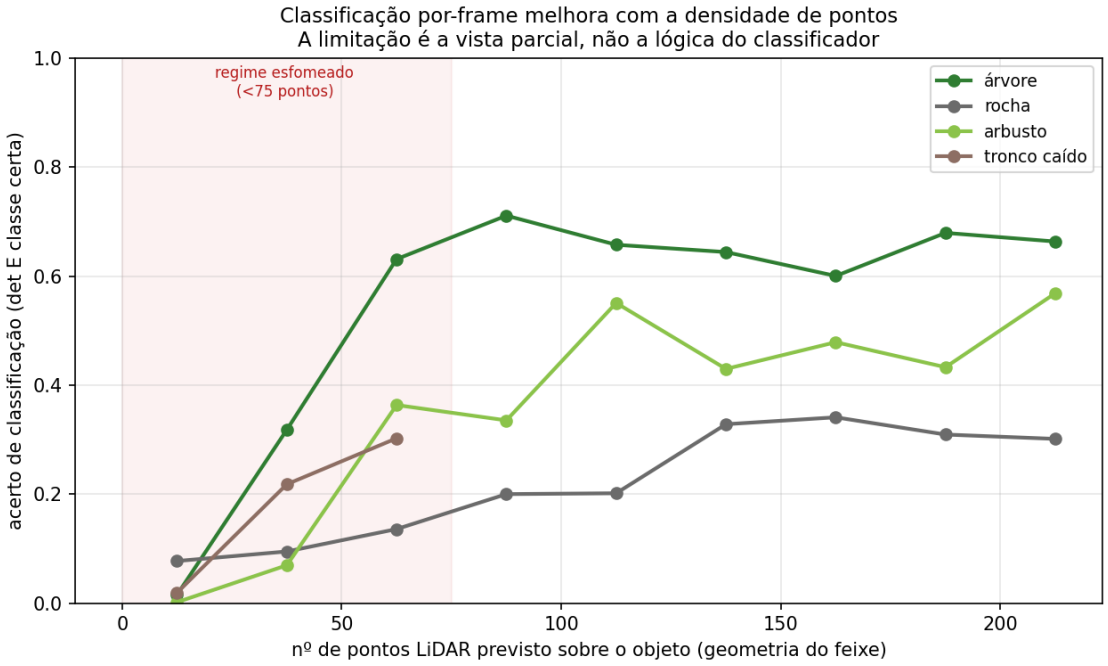


Figura 4.7: Acerto de classificação em função da densidade de pontos prevista pela geometria, argumento central para a acumulação multi-vista no SLAM.

A classificação melhora de forma acentuada com a densidade de pontos, e abaixo de cerca de 75 pontos previstos está claramente esfomeada, com a árvore a subir de 0,02 para 0,63 e o arbusto de 0,00 para 0,38. Como grande parte das observações reais cai precisamente nesse regime pobre, é ele que arrasta os números agregados para baixo.

Comparando com a classificação offline sobre *clusters* limpos e completos, a densidade de pontos explica uma parte grande da diferença entre os dois regimes, mas não toda. Mesmo com mais de 200 pontos previstos a rocha estabiliza perto de 0,30 ao vivo, longe do 0,81 medido offline, porque uma vista única, por mais densa que seja, continua a mostrar só uma face do objeto, e são as características de forma dessa face, e não do objeto inteiro, que dominam a decisão do classificador.

Este resultado aponta em duas direções distintas, que convém não misturar. Para a rocha, o arbusto e o tronco caído, a classificação está limitada pela entrada, pela vista parcial e esparsa que cada observação oferece, o que justifica de forma medida, e não apenas argumentada, a acumulação de evidência multi-vista já feita na camada de SLAM. Já a árvore não mostra essa folga entre o valor ao vivo e o offline, o que aponta antes para uma limitação do próprio classificador e não da entrada, e foi essa segunda pista que motivou uma investigação mais funda.

Essa investigação revelou um problema de agregação escondido na matriz de confusão offline da árvore. Os seis modelos de árvore usados no gerador de mundos comportam-se de forma marcadamente bimodal, e nenhum agregado único os descreve bem, como resume a Tabela 4.8 e ilustra a Figura 4.8.

Tabela 4.8: Classificação offline da árvore, discriminada por modelo (12 amostras cada).

Modelo	**Verticalidade**	**Linearidade**	**N.º de pontos**	**Acerto**
Tree1	0.983	0.976	119	1.00
Tree2	1.000	0.971	130	0.92
Tree3	0.964	0.893	117	0.83
Tree4	0.456	0.587	396	0.08
Tree5	0.374	0.325	622	0.17
Tree6	0.402	0.510	463	0.33
Agregado	–	–	–	0.56

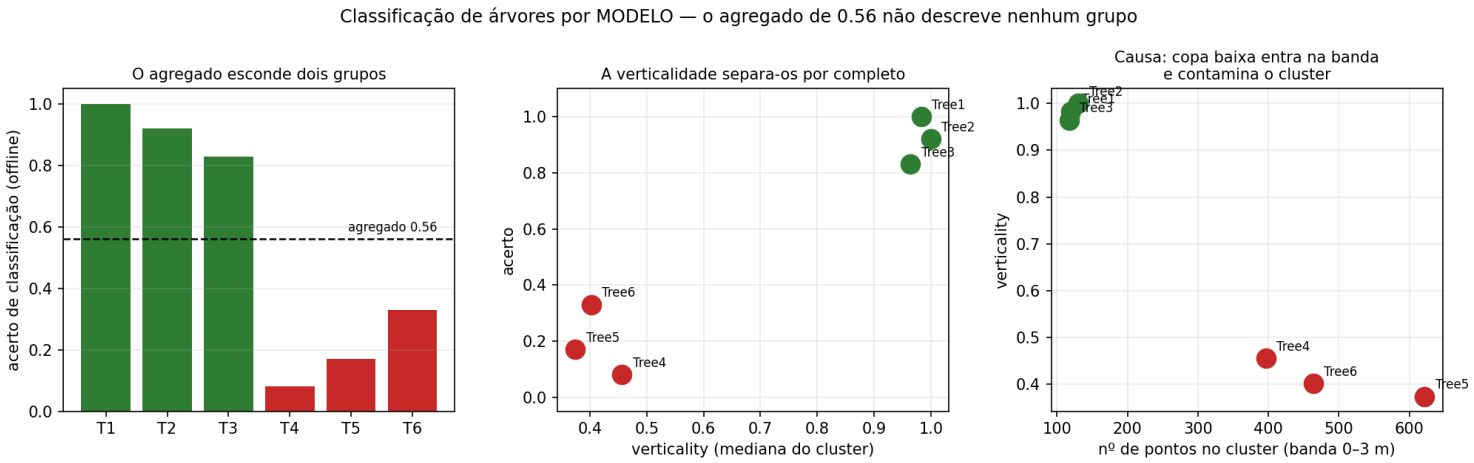


Figura 4.8: Classificação da árvore por modelo, evidenciando a bimodalidade escondida pelo agregado.

Todas as falhas caem em obstáculo, nunca em rocha, arbusto ou tronco caído, o que mostra que não se trata de confusão entre classes mas de rejeição do próprio *cluster* como estrutura de fuste. A causa está nos modelos que falham terem entre três e cinco vezes mais pontos no *cluster*, porque a sua ramagem desce para dentro da janela de crescimento de 3 metros usada para o isolar, o que faz a análise de componentes principais deixar de ler um fuste e passar a ler fuste mais folhagem. O núcleo de fuste continua lá dentro e continua a ser detetado, mas a análise de componentes principais é calculada sobre o *cluster* inteiro, e é a folhagem que domina os valores próprios resultantes.

Esta bimodalidade aponta uma causa concreta e corrigível, e não apenas uma limitação a assumir, o que justificou testar a correção e medir o seu efeito. Passou a calcular-se a verticalidade e a linearidade só sobre o núcleo de fuste, quando este existe e satisfaz duas condições que impedem uma rocha de forjar um núcleo semelhante, e não sobre o *cluster* completo. Repetindo o mesmo conjunto de nuvens offline com esta alteração ligada e desligada, a árvore sobe de 0,56 para 0,92 sem qualquer regressão nas restantes classes, como resume a Tabela 4.9.

Tabela 4.9: Efeito da análise de componentes principais restrita ao núcleo de fuste, mesmas nuvens offline, antes/depois.

Classe	**Antes**	**Depois**	**n**
Árvore	0.56	0.92	72
Rocha	1.00	1.00	36
Arbusto	0.33	0.33	36
Tronco caído	0.00	0.00	12

O tronco caído mantém 0,00 antes e depois desta correção, mas por uma razão distinta da que explica o efeito marginal ao vivo na árvore, discutido a seguir. Um tronco caído é um objeto pequeno e rente ao chão, com poucos pontos à partida, e o CSF usado na segmentação de solo (Secção 3.2.1) tende a absorver parte desses pontos como solo mesmo depois de aplicado o corretor que tenta devolvê-los ao não-solo. Sobra por isso uma nuvem residual escassa de mais para qualquer classificação funcionar, com ou sem a análise de componentes principais restrita ao núcleo de fuste, ao contrário do tronco vertical de uma árvore, onde há pontos suficientes para essa correção ter de facto algo sobre que atuar.

Ao vivo, contudo, o efeito desta mesma correção é marginal, a árvore sobe apenas de 0,50 para 0,52 na corrida `moderate`, bem longe do salto observado offline. A assinatura da correção existe, a fração de árvores classificadas como obstáculo cai 27%, mas o agregado quase não se move, porque a condição que permite calcular a análise de componentes principais sobre o núcleo exige um núcleo de fuste com pelo menos um metro de altura, e ao vivo a maioria das observações está no regime esfomeado de menos de 75 pontos onde esse núcleo simplesmente não chega a formar-se. A correção está certa e não regride nada, mas não resolve ao vivo o que os números offline prometiam, e essa distância entre o ganho offline e o ganho ao vivo é em si um resultado a reter.

4.2.3.4 Complementaridade entre LiDAR e câmara

A acumulação multi-vista no SLAM responde à limitação da rocha, do arbusto e do tronco caído ao longo do tempo, mas não muda o facto de, num único *frame*, o LiDAR continuar fraco nestas classes precisamente onde a nuvem de pontos é mais pobre. Isso levanta a pergunta de saber se a câmara falha nas mesmas condições ou se falha noutras, o que a tornaria um complemento útil em vez de uma simples repetição do mesmo problema. A Tabela 4.10 compara a classificação por classe entre os dois sensores, medidos na mesma passagem pelo mundo `moderate`, e a Figura 4.9 mostra a mesma comparação em função da distância.

Tabela 4.10: Classificação por classe, LiDAR contra câmara, mesma passagem.

Classe	LiDAR	Câmara
Árvore	0.71	0.82
Rocha	0.18	0.50
Arbusto	0.20	0.88
Tronco caído	0.06	0.87

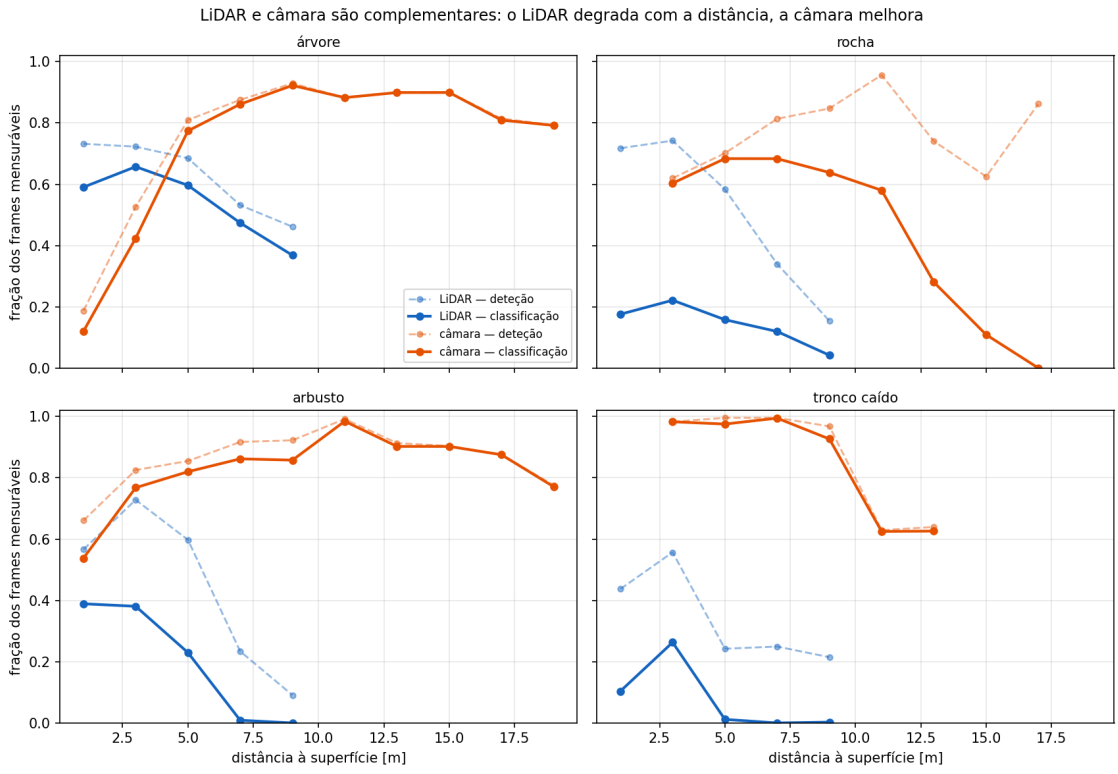


Figura 4.9: Complementaridade entre LiDAR e câmara, classificação em função da distância.

O padrão por distância é oposto entre os dois sensores, e é esse o resultado que de facto interessa, mais do que qualquer valor isolado. O LiDAR degrada com a distância nas classes baixas, porque fica sem pontos suficientes, enquanto a árvore já não degrada assim que o denominador está

corrigido. A câmara, pelo contrário, melhora com a distância, subindo de 0,19 na árvore entre 0 e 2 metros para 0,93 entre 8 e 10 metros, porque de perto o objeto transborda do enquadramento da imagem. As duas curvas cruzam-se por volta dos 4 a 5 metros, com o LiDAR a mandar abaixo dessa distância e a câmara acima dela, o que é uma justificação quantitativa medida para a fusão entre os dois sensores, e não apenas o argumento genérico de que uma câmara ajuda. A rocha continua a ser a classe mais fraca em ambos os sensores, a câmara mantém boa deteção mesmo longe mas a sua classificação da rocha colapsa para perto de zero a partir dos 16 metros, o sensor vê o objeto mas deixa de saber o que é.

Repetindo esta comparação nos três mundos, a árvore no LiDAR mostra-se notavelmente estável, entre 0,68 e 0,71 nos três terrenos, e as classes baixas até melhoram ligeiramente no terreno mais acidentado, provavelmente porque a inclinação expõe mais superfície ao LiDAR montado com uma inclinação fixa. A exceção que se destaca é o tronco caído na câmara, que cai de 0,99 no terreno suave para 0,53 no acidentado, a única degradação grande de todo este conjunto de resultados e um candidato natural a uma investigação futura.

Esta complementaridade motivou a implementação da fusão da câmara diretamente no classificador do *tracker* do SLAM, em que a câmara não substitui o LiDAR mas confirma ou reforça a sua classe quando a deteção geométrica está pouco confiante. Um primeiro teste ao vivo desta fusão, com o robô a conduzir o mesmo percurso duas vezes com a fusão ligada e duas vezes desligada, mediu o número de *landmarks* comprometidos no mapa final, a sua exatidão de classe e o número de fantasmas por corrida. Os quatro valores obtidos, com a fusão ligada e desligada, sobrepõem-se dentro da variação normal entre corridas e não mostram diferença mensurável. Com apenas duas corridas por condição esta amostra é claramente pequena demais para sustentar qualquer conclusão sobre o efeito real da fusão, mas fica registada como um primeiro indício, a confirmar ou a refutar com mais corridas no futuro, de que o benefício teórico sugerido pelo cruzamento das duas curvas ainda não se traduziu num ganho medido de qualidade do mapa.

4.2.4 Análise crítica

O quadro que este conjunto de ensaios desenha é consistente com o que já tinha aparecido no banco de ensaio offline, mas muda de forma nítida onde está o problema. A deteção geométrica funciona bem em quase todas as condições, o *clustering* encontra os objetos, o que falha é o classificador não saber com confiança de que classe se trata quando só tem uma vista parcial e pobre em pontos. Isto justifica, com números e não só com intuição, a decisão arquitetural de acumular evidência de várias vistas ao longo do tempo no SLAM em vez de exigir uma classificação perfeita a cada *frame*, e justifica também a fusão com a câmara nas distâncias onde o LiDAR fica esfomeado.

Há, no entanto, limitações que convém não esconder. A correção do denominador por envelope de deteção assenta na mesma geometria do sensor que a deriva, e não foi validada de forma independente, pelo que, se essa premissa for otimista, os valores de deteção aqui apresentados são um limite superior e não uma medida isenta de qualquer viés. A comparação entre a versão anterior e a corrigida do denominador também não é emparelhada, são duas passagens distintas pelo mesmo mundo e não o mesmo conjunto de *frames* repontuado, o que introduz algum ruído

entre corridas, ainda que a subida de dezanove pontos percentuais na árvore esteja bem acima desse ruído. A amostra offline é pequena, apenas 115 *clusters* em condições favoráveis, e deve por isso ser lida como um limite superior do que o classificador consegue e não como o seu desempenho típico. A própria classe árvore não é homogénea, como mostra a diferença entre os modelos de fuste limpo e os de copa baixa, e qualquer métrica agregada por classe corre o risco de esconder esta divisão em vez de a revelar. Por fim, os três mundos testados variam apenas o declive do terreno, e não a densidade de vegetação, que é um eixo de dificuldade diferente e que continua por medir. A própria fusão da câmara no SLAM, motivada por esta secção, ilustra bem os limites de tirar conclusões de amostras pequenas, com apenas duas corridas por condição não há como distinguir um efeito real de ruído entre corridas. Optou-se por não repetir este ensaio com mais corridas, precisamente porque o resultado nulo já apareceu desde a primeira comparação, e o tempo disponível foi investido noutras partes do sistema com sinal mais claro de melhoria em vez de insistir num ensaio que já não prometia diferença. Fica também por questionar a própria arquitetura assimétrica escolhida para esta fusão, em que a câmara só confirma ou limita a classe já proposta pelo LiDAR e nunca a substitui. Essa escolha pode estar a desperdiçar parte da vantagem que a Tabela 4.10 e a Figura 4.9 mostram a câmara ter em quase todas as classes e distâncias, e um acoplamento mais simétrico entre os dois sensores é um candidato mais promissor a explorar do que simplesmente repetir o mesmo ensaio com mais corridas.

4.3 Localização e Mapeamento Simultâneos

Os resultados desta secção isolam exclusivamente a camada de localização e mapeamento, estimador de estado SE(3) mais Tree-SLAM, sem qualquer contributo da navegação reativa ou dos mapas de custo, avaliados em separado nos seus próprios capítulos. Para isso o robô é conduzido ao longo de uma rota fixa a partir da sua pose de referência do simulador, e não pela pilha de navegação, de modo a que o erro medido reflita apenas a qualidade da localização em si.

Todas as medições reportadas usam um modelo de sensores deliberadamente não idealizado. A odometria é integrada a partir da velocidade *medida* das lagartas e não da velocidade comandada, de modo a que o escorregamento se manifeste como erro em vez de ser invisível por construção, e o giroscópio inclui um viés aleatório da ordem dos 20°/h, valor típico de uma unidade inercial MEMS de custo moderado. Esta distinção é material, porque com a velocidade comandada em vez da medida o estimador contínuo não acumula praticamente deriva nenhuma e a comparação com o Tree-SLAM perde significado.

Cada configuração é repetida em várias corridas independentes e os valores são apresentados como média mais ou menos desvio padrão entre corridas, uma vez que a variabilidade entre repetições da mesma configuração é da mesma ordem de grandeza dos efeitos em estudo e uma corrida isolada não permitiria distinguir um efeito real de ruído.

4.3.1 Erro de trajetória, Tree-SLAM contra estimador contínuo isolado

O ensaio principal percorre doze voltas consecutivas sobre a mesma rota fechada, repetido em cinco corridas independentes. O objetivo é distinguir dois regimes de erro qualitativamente diferentes, um erro *limitado*, que estabiliza num patamar e aí se mantém por muitas voltas que o robô dê, e um erro *não limitado*, que cresce indefinidamente com a distância percorrida. A Tabela 4.11 resume o resultado.

Tabela 4.11: Erro de posição ao longo de doze voltas, Tree-SLAM contra estimador contínuo isolado.

Métrica	**Tree-SLAM**	**Estimador contínuo**	**Razão**
Erro médio sobre as 12 voltas [m]	0.223 ± 0.037	0.843 ± 0.308	$3.8\times$
Erro médio na volta 1 [m]	0.294 ± 0.082	0.100 ± 0.015	$0.3\times$
Erro médio na volta 12 [m]	0.200 ± 0.023	1.613 ± 0.578	$8.1\times$
Coeficiente de variação [%]	16.5	36.5	
Corridas divergentes	0 / 5	0 / 5	

Na primeira volta o estimador contínuo isolado é cerca de três vezes mais exato do que o Tree-SLAM, com 0.100 m contra 0.294 m, um resultado que não é anómalo mas antes esperado, porque o mapa ainda está a ser construído e as poucas *landmarks* confirmadas ainda não constituem uma referência melhor do que a integração direta da odometria pouco depois do arranque. A partir da terceira volta a ordem inverte-se e não volta a mudar. O erro do Tree-SLAM estabiliza perto dos 0.20 m e mantém-se nesse patamar até ao fim, ao passo que o erro do estimador contínuo cresce de forma praticamente monótona em todas as cinco corridas, chegando a 1.613 m na décima segunda volta. A Figura 4.10 mostra esta separação, com a banda semitransparente a representar o desvio padrão entre corridas.

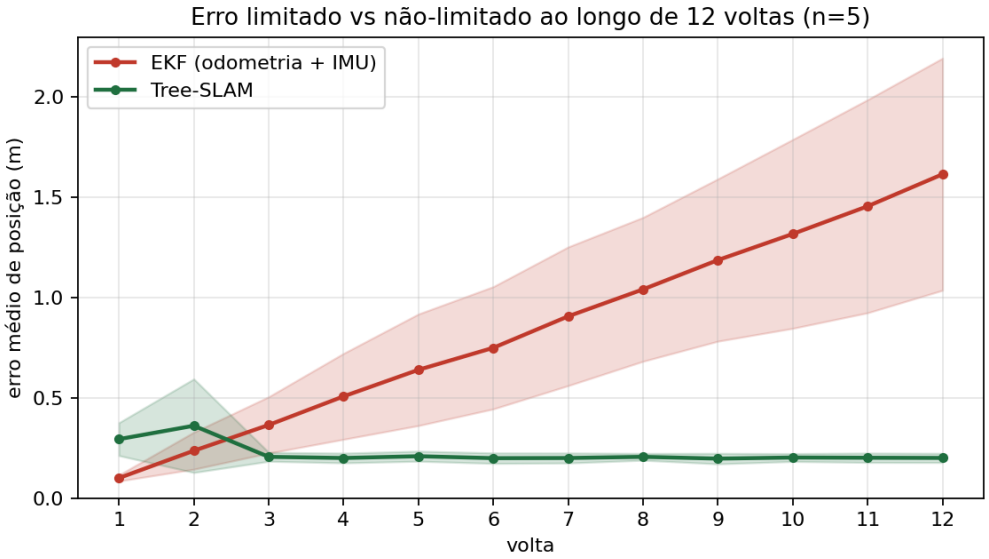


Figura 4.10: Erro médio de posição por volta ao longo de doze voltas, para o Tree-SLAM e para o estimador contínuo isolado. Média de cinco corridas independentes, com a banda a indicar o desvio padrão entre corridas.

A razão de 3.8× entre os dois erros médios não deve ser lida como uma constante do sistema, porque depende inteiramente da duração do ensaio. Se o mesmo ensaio fosse interrompido ao fim de três voltas a razão seria próxima da unidade, e se fosse prolongado para vinte e quatro voltas seria substancialmente maior, uma vez que o numerador continua a crescer e o denominador não. É esta assimetria, e não o valor concreto da razão, que constitui o resultado.

O coeficiente de variação entre corridas é também informativo. As cinco corridas do Tree-SLAM têm uma dispersão de 16.5% em torno da média e nenhuma delas diverge, enquanto em versões anteriores do sistema era comum uma em cada cinco corridas afastar-se para lá do metro de erro e nunca recuperar. Esta consistência resulta das correções descritas nas subsecções seguintes, em particular da parametrização do raio do alinhamento local.

O mesmo padrão qualitativo é visível de forma ainda mais direta numa corrida isolada, mais longa do que as doze voltas do ensaio agregado, em que se sobrepõem as três trajetórias no plano em vez de apenas o erro por volta. A Figura 4.11 mostra, à esquerda, a trajetória verdadeira do simulador, a trajetória do estimador contínuo isolado e a trajetória do Tree-SLAM com origem comum, e à direita o erro de posição de cada um ao longo do tempo dessa mesma corrida. Trata-se de uma corrida única e ilustrativa, não da mesma amostra estatística da Tabela 4.11, pelo que os valores médios da legenda (1.11 m e 0.20 m) não devem ser comparados diretamente com os 0.843 m e 0.223 m da tabela, mas o comportamento é o mesmo, o estimador contínuo afasta-se progressivamente da referência a cada volta, ao passo que o Tree-SLAM mantém o erro contido, com picos periódicos sempre que a deriva acumulada entre correções cresce um pouco antes de o alinhamento a corrigir de novo.

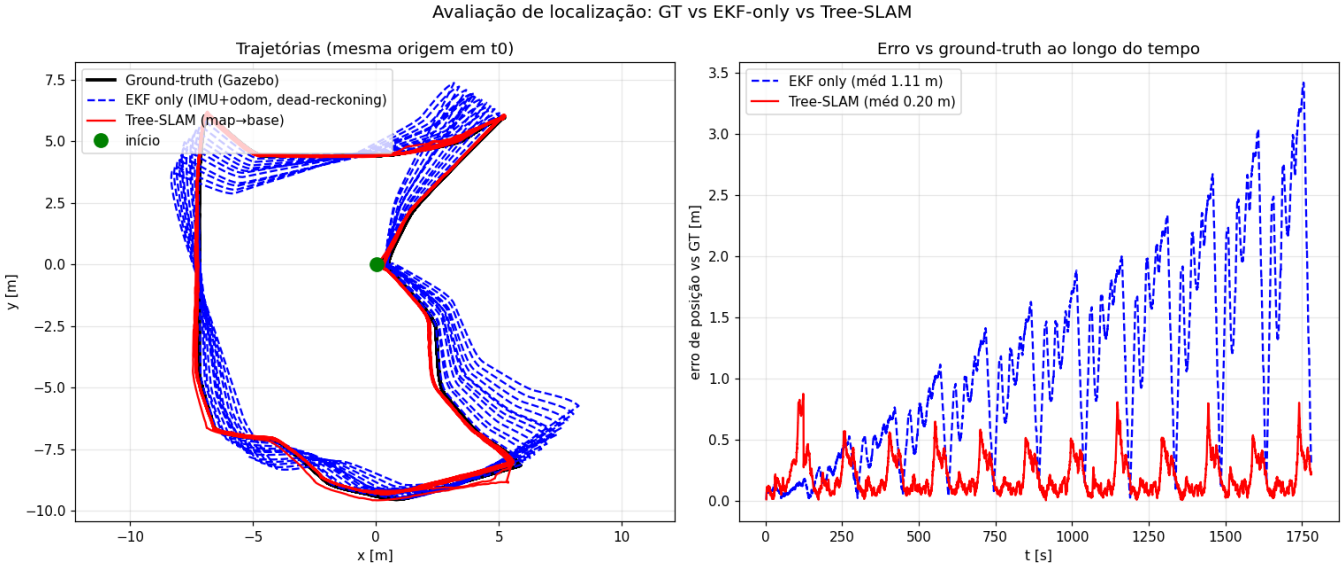


Figura 4.11: Corrida isolada e mais longa, sobreposição das trajetórias (esquerda) e erro de posição ao longo do tempo (direita), Tree-SLAM contra estimador contínuo isolado.

4.3.2 Fidelidade do mapa de *landmarks*

Além do erro de trajetória, avalia-se a qualidade do mapa construído, comparando as *landmarks* estimadas com as posições verdadeiras dos objetos do mundo simulado. Consideram-se apenas os objetos efetivamente visíveis a partir da rota, ou seja, os que passam a menos de 10 m do percurso, porque exigir a deteção de árvores que nunca entraram no alcance útil do sensor mediria a cobertura da rota e não a qualidade da perceção. A Tabela 4.12 resume as três métricas relevantes, todas medidas no mesmo ensaio de doze voltas.

Tabela 4.12: Fidelidade do mapa de *landmarks* ao fim de doze voltas, média de cinco corridas independentes.

Métrica	**Valor**
Erro quadrático médio de posição das *landmarks* emparelhadas [m]	0.359
Cobertura dos objetos visíveis a menos de 10 m da rota [%]	58.6
Landmarks sem correspondência no mundo real [%]	17.5

A Figura 4.12 mostra o mapa final de uma das corridas sobreposto às posições verdadeiras do mundo. A trajetória estimada acompanha de perto a trajetória de referência ao longo de todo o percurso, e as *landmarks* confirmadas caem sobre troncos reais na grande maioria dos casos.

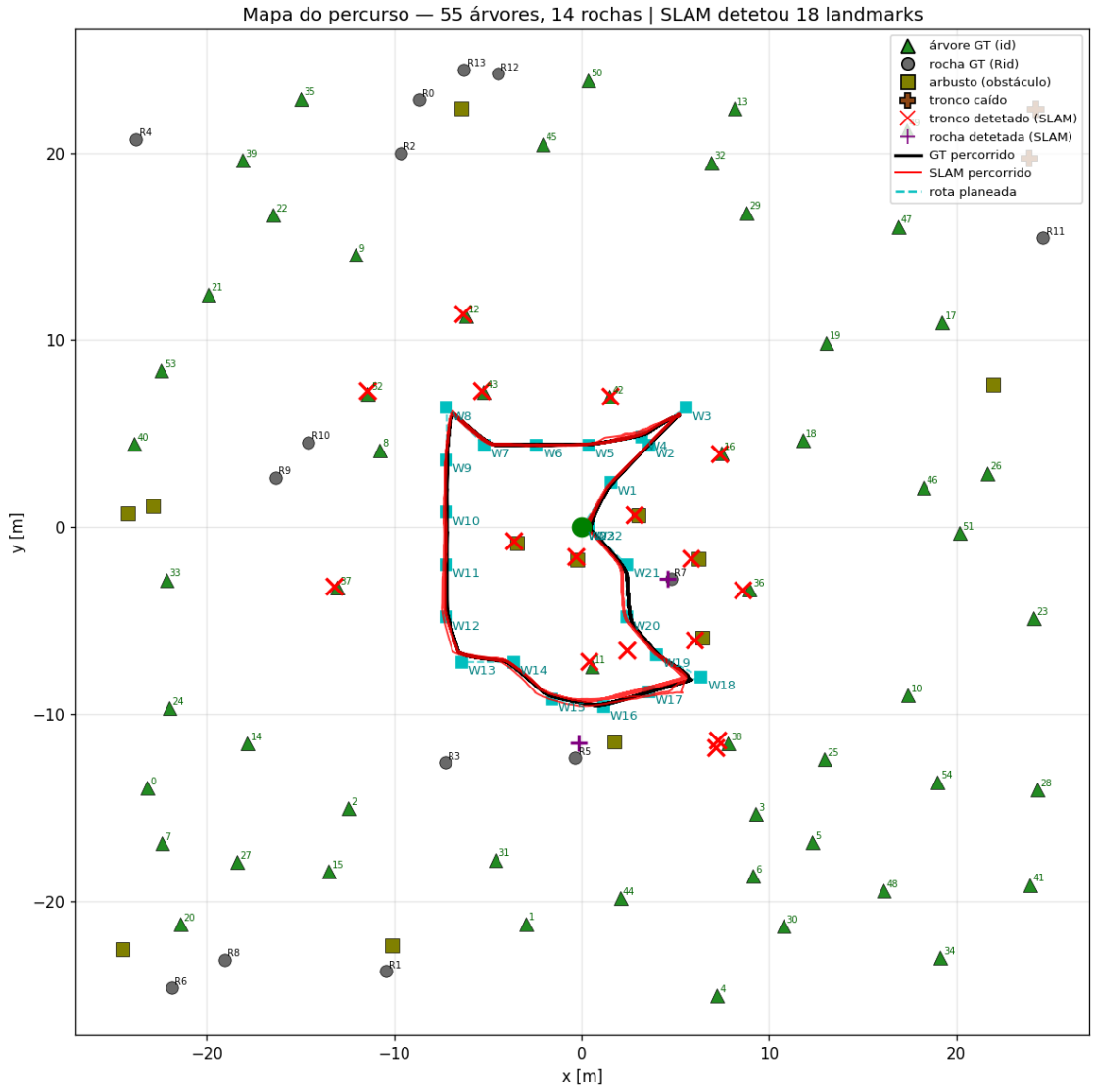


Figura 4.12: Mapa final de *landmarks* do Tree-SLAM sobreposto à referência do mundo simulado, com a trajetória estimada e a trajetória verdadeira.

Os dois valores menos favoráveis da tabela merecem leitura cuidada. A cobertura de 58.6% reflete sobretudo a geometria de observação, porque o sensor tem um campo de visão útil estreito na direção do movimento e vários troncos dentro do raio nominal de 10 m nunca chegam a ser observados de um ângulo que permita ajustar um cilindro com confiança suficiente para promover a deteção a *landmark*. Quanto aos 17.5% de *landmarks* sem correspondência, a inspeção do mapa mostra que uma parte apreciável destas não são invenções do estimador mas antes arbustos classificados como troncos, objetos que existem realmente no local indicado e cuja posição está correta, falhando apenas a classe atribuída.

4.3.3 Contributo do alinhamento local e do fecho de laço

Para isolar o contributo de cada uma das duas técnicas de correção do Tree-SLAM, o alinhamento de constelação local e o fecho de laço global, repetiu-se o mesmo ensaio nas quatro combinações possíveis, ligando e desligando cada uma delas de forma independente, com três corridas por combinação. A Tabela 4.13 reporta a mediana do erro em vez da média, porque duas das doze corridas divergiram e uma média seria dominada por esses valores extremos em vez de descrever o comportamento típico.

Tabela 4.13: Contributo do alinhamento local e do fecho de laço, quatro combinações com três corridas cada. Mediana do erro médio de posição sobre seis voltas.

Alinhamento local	**Fecho de laço**	**Erro mediano [m]**	**Divergências**
desligado	desligado	0.294	1 / 3
ligado	desligado	0.365	1 / 3
desligado	ligado	0.225	0 / 3
ligado	ligado	0.330	0 / 3

O resultado mais claro desta tabela diz respeito ao fecho de laço, que atua como rede de segurança. Nas seis corridas em que esteve ligado nenhuma divergiu, e nas seis em que esteve desligado duas divergiram, o que corresponde exatamente ao papel que dele se espera, ou seja, não melhorar o desempenho típico mas eliminar a cauda de corridas em que o sistema se perde e não recupera.

Já o alinhamento local produziu, nesta configuração, um resultado contrário ao esperado, degradando o erro mediano de 0.225 m para 0.330 m quando ligado sobre o fecho de laço. Este resultado motivou a investigação descrita na subsecção seguinte, uma vez que um componente cuja função é corrigir a pose não deveria piorá-la.

4.3.4 Parametrização do raio do alinhamento local

A degradação introduzida pelo alinhamento local não se devia ao método em si mas à sua parametrização. O raio dentro do qual se recolhem as *landmarks* candidatas a formar a constelação de referência estava fixado em 15 m, valor que excede a diagonal da própria rota de teste, que é de aproximadamente 21 m. Nestas condições o mapa dito local abrangia quase todo o mapa conhecido, com uma média de catorze candidatos, e a correspondência de três a quatro deteções contra catorze candidatos é ambígua por construção, produzindo associações erradas com frequência apreciável.

Para verificar esta hipótese variou-se o raio mantendo tudo o resto constante, com três corridas por valor. A Tabela 4.14 e a Figura 4.13 apresentam o resultado.

Tabela 4.14: Varrimento do raio do alinhamento local, três corridas por valor, seis voltas cada, com o fecho de laço sempre ligado.

Raio	**Erro mediano [m]**	**Erro médio [m]**	**Divergências**
desligado	0.225	0.221 ± 0.039	0 / 3
6 m	0.271	0.244 ± 0.053	0 / 3
8 m	0.281	0.912 ± 1.142	1 / 3
11 m	**0.221**	$\mathbf{0.225 \pm 0.008}$	0 / 3
15 m	0.330	0.369 ± 0.091	0 / 3

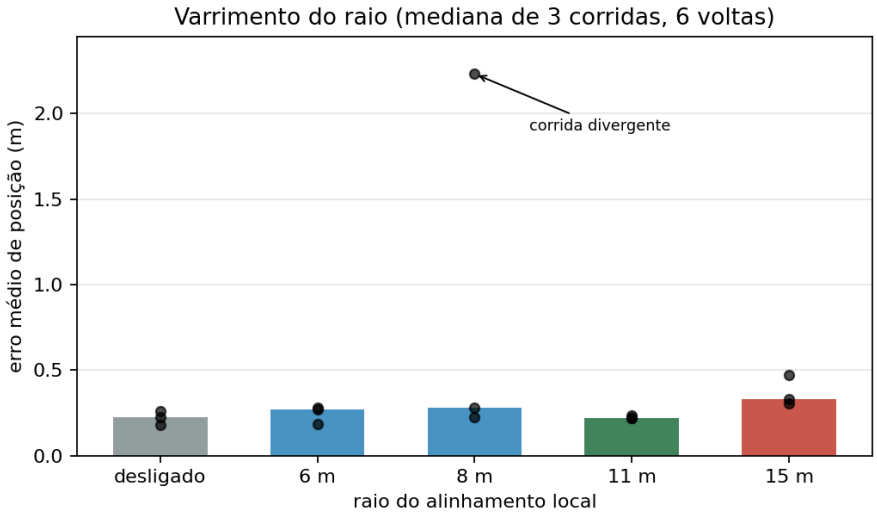


Figura 4.13: Erro mediano de posição em função do raio do alinhamento local. Os pontos escuros indicam as corridas individuais.

O raio de 11 m é o melhor valor testado, e o aspeto mais relevante do resultado não está na mediana mas na dispersão. Com 11 m as três corridas produziram 0.234, 0.221 e 0.221 m, um desvio padrão de apenas 0.008 m, ao passo que qualquer outro valor testado apresenta dispersão pelo menos cinco vezes maior. Um raio demasiado pequeno, de 6 m, deixa candidatos a menos para formar uma constelação distintiva e o componente deixa simplesmente de atuar, enquanto um raio demasiado grande, de 15 m, produz candidatos a mais e o componente passa a atuar mal. Vale a pena notar que mesmo o valor ótimo de 11 m já excede o alcance máximo de deteção da perceção, limitado a 10 m, pelo que o raio do alinhamento não está a alargar o que o sensor consegue ver, mas sim a decidir quantas *landmarks* já mapeadas entram na constelação de referência para associar o que o sensor vê agora. É esta a configuração usada em todos os resultados da Secção 4.3.1.

O raio de 8 m regista uma divergência em três corridas, o que também explica o erro médio muito mais alto e disperso desta linha (0.912 ± 1.142 m) face à sua mediana. Essa divergência não decorre do valor do raio em si, mas de a perceção não ter detetado *landmarks* suficientes na volta inicial dessa corrida em particular para o fecho de laço reconhecer o regresso a um local já mapeado. Sem esse reconhecimento, a correção que a Secção 4.3.3 mostra eliminar divergências

nunca teve oportunidade de atuar, e o erro cresceu sem controlo do mesmo modo que aconteceria com o fecho de laço desligado.

Convém registar que os resultados das Subsecções 4.3.3 e 4.3.5 foram obtidos com o raio de 15 m, anterior a esta descoberta, e constituem por isso um limite inferior conservador do desempenho do sistema na sua configuração final.

4.3.5 Generalização a outros cenários

Para avaliar se o desempenho depende do mundo específico em que o sistema foi desenvolvido, repetiu-se o ensaio em quatro mundos distintos, variando a densidade de vegetação e a irregularidade do terreno. Todos os mundos são percorridos pela mesma rota, deliberadamente construída para ser transponível em todos eles, de modo a que a comparação não seja contaminada por percursos de dificuldade diferente. A Tabela 4.15 apresenta os resultados.

Tabela 4.15: Desempenho em quatro mundos distintos, três corridas por mundo, seis voltas cada, com a mesma rota em todos.

Mundo	**Tree-SLAM [m]**	**Estimador contínuo [m]**	**Razão**
Referência	0.298 ± 0.071	0.366 ± 0.129	$1.2\times$
Vegetação densa	0.153 ± 0.008	0.513 ± 0.012	$3.4\times$
Terreno irregular denso	0.180 ± 0.023	0.750 ± 0.023	$4.2\times$
Terreno irregular	*rota não completada*		

A vantagem do Tree-SLAM sobre o estimador contínuo cresce com a dificuldade do cenário, de $1.2\times$ no mundo de referência para $4.2\times$ no mundo mais exigente. O mecanismo é direto, porque terreno mais irregular provoca mais escorregamento e portanto mais deriva na odometria, enquanto vegetação mais densa oferece mais *landmarks* por observação e portanto melhor correção. Os dois efeitos empurram a razão no mesmo sentido. É de notar que o Tree-SLAM obteve resultados absolutos *melhores* nos mundos mais densos do que no mundo de referência, o que é consistente com a densidade de *landmarks* ser o fator determinante do seu desempenho.

O quarto mundo, de terreno irregular sem adensamento de vegetação, não completou a rota em nenhuma das três tentativas, por existir no percurso um acidente de terreno intransponível para a plataforma. Não se trata de uma falha da localização, uma vez que o mundo homólogo com vegetação densa partilha exatamente os mesmos parâmetros de geração de terreno e completa as seis voltas sem dificuldade, mas antes de uma particularidade daquela realização concreta do terreno. Optou-se por reportar a ausência de dados em vez de substituir o mundo por outro mais favorável.

Duas ressalvas limitam o alcance desta comparação. Em primeiro lugar, os quatro mundos não isolam uma única variável, porque as malhas de terreno diferem entre si mesmo quando os parâmetros de geração coincidem, pelo que a tabela demonstra generalização a cenários variados mas não constitui um estudo fatorial limpo do efeito da densidade. Em segundo lugar, e como já referido, estes ensaios usaram o raio de alinhamento de 15 m, agora conhecido como sub-ótimo.

4.3.6 Análise crítica

Os resultados desta secção sustentam com solidez a conclusão central, o Tree-SLAM produz um erro limitado onde o estimador contínuo isolado produz um erro que cresce sem parar, e essa diferença mantém-se ou até se acentua em cenários mais difíceis do que o mundo de desenvolvimento. Ainda assim, vários aspetos do desenho experimental e do próprio sistema pedem uma leitura cautelosa.

Um primeiro ponto foi já assinalado ao longo do texto mas vale a pena reunir aqui, os resultados do contributo fatorial do alinhamento local e do fecho de laço (Secção 4.3.3) e da generalização a outros mundos (Secção 4.3.5) foram obtidos com o raio de alinhamento local de 15 m, mais tarde identificado como sub-ótimo. Isto não invalida essas conclusões, o fecho de laço continuaria a eliminar divergências e a vantagem continuaria a crescer com a dificuldade do mundo, mas os valores absolutos de erro reportados nessas duas secções são um limite inferior conservador do desempenho do sistema na sua configuração final de 11 m, e não devem ser lidos lado a lado com os da Secção 4.3.1 como se tivessem sido medidos na mesma configuração.

Um segundo ponto, não discutido até aqui, diz respeito à fusão de *landmarks* duplicados. Todos os resultados desta secção foram obtidos com esse mecanismo ativo por omissão, sem que nenhum dos ensaios o tenha desligado para medir o seu contributo isoladamente, ao contrário do que se fez para o alinhamento local e o fecho de laço. O limiar de distância que decide se duas *landmarks* são candidatas a fusão está fixado em 6 m no código, um valor que funciona bem para os duplicados típicos, tipicamente a menos de 1 m um do outro por pequenas variações de deteção, mas que é permissivo em excesso para decidir com confiança a 4 m ou mais, distância a que dois troncos genuinamente distintos numa floresta razoavelmente densa já podem estar. Não há neste momento evidência de que este limiar tenha produzido fusões erradas nos ensaios reportados, mas também não há um ensaio dedicado a excluir essa possibilidade, o que o deixa como um mecanismo ativo em todos os números desta secção e nunca isolado deles.

Um terceiro ponto é de desenho experimental antes de ser um resultado. Toda esta secção conduz o robô ao longo de uma rota fixa a partir da pose de referência do simulador, precisamente para isolar a localização do resto da pilha de navegação, mas isso significa que nenhum dos números aqui reportados diz respeito ao caso em que o próprio Tree-SLAM decide como o robô se move, seguindo um caminho escolhido pela navegação autónoma em vez desta rota fixa preparada de propósito para manter as *landmarks* bem distribuídas ao longo de todo o percurso. Só o ensaio de sistema completo, na secção final deste capítulo, expõe a localização a essas condições, e é lá que a validade destes resultados isolados deve ser finalmente confrontada com a realidade de uma missão completa.

Por fim, os 17.5% de *landmarks* sem correspondência e os 58.6% de cobertura da Secção 4.3.2 recordam que o mapa construído está limitado pela qualidade da perceção a montante, uma parte apreciável do primeiro valor vem de instabilidade na classificação de objetos reais do percurso e não de invenções do estimador, e a cobertura incompleta reflete sobretudo a geometria do sensor, não uma falha de associação. Nenhum destes dois pontos é por isso uma fragilidade do Tree-SLAM

em si, mas ambos lembram que a fidelidade do mapa é, no limite, um produto de duas camadas e não de uma só.

4.4 Representação do Ambiente e Traversabilidade

Os resultados desta secção dividem-se em dois grupos de natureza distinta. O primeiro valida o modelo de custo em si, isolado de qualquer mundo ou sensor, através de testes unitários e de uma comparação direta entre a versão inicial do modelo e a versão corrigida. O segundo aplica esse modelo já validado sobre um mapa de custo real, produzido ao vivo no simulador com dados de sensor, para avaliar a consistência da malha de terreno sob deriva e ruído e o efeito da escolha da fórmula de combinação de custo. Este segundo grupo correu apenas sobre o mundo florestal realista do banco de ensaio, que combina declive suave com troncos e rochas na mesma rota.

4.4.1 Validação do modelo de custo

O núcleo do modelo de custo, que combina o custo por classe semântica com o termo de incerteza e produz o valor final entre 0 e 1 usado pelo planeador, tem trinta testes unitários isolados, sem qualquer dependência do simulador ou do *costmap* do Nav2. Todos os trinta passam. Cobrem, entre outros casos, que um obstáculo letal se mantém letal mesmo com incerteza elevada, que um peso de incerteza a zero desliga por completo o termo de risco, e que o custo combinado nunca excede o limite de 1 independentemente da combinação de entradas.

Além dos testes unitários, existe uma comparação direta entre a versão inicial do modelo de custo e a versão corrigida, medida sobre o mesmo cenário. A Tabela 4.16 resume essa comparação.

Tabela 4.16: Modelo de custo, versão inicial contra versão corrigida, mesmo cenário.

Métrica	**Inicial**	**Corrigida**
Custo do arbusto ($n = 5$)	1,00 (letal)	0,65 (penetrável)
Custo do tronco ($n = 3$)	1,00 (letal)	1,00 (letal)
Células publicadas como desconhecidas [%]	0	40,8
Células publicadas como livres [%]	60,8	0
Células com margem de risco [%]	0	56,4

A diferença mais relevante desta tabela é a do arbusto, que na versão inicial era tratado como um obstáculo tão letal como um tronco, quando na realidade é uma vegetação que o robô pode atravessar com penalização mas sem colisão, e que a versão corrigida já reflete com um custo de 0,65 em vez de 1,00. A segunda diferença é a das células desconhecidas, que a versão inicial publicava sempre como livres por omissão, escondendo a incerteza real do mapa, e que a versão corrigida passa a marcar explicitamente como desconhecidas em vez de assumir que são seguras.

4.4.2 Consistência sob deriva, ruído e escolha de fórmula de custo

Os três testes seguintes já correm sobre um mapa de custo real, produzido ao vivo no simulador, e não sobre o modelo isolado da secção anterior. A condução é feita por controlo proporcional direto guiado pela pose de *ground-truth* do Gazebo, sem gestor de missão nem Nav2, precisamente para isolar a camada de traversabilidade do planeamento e da navegação. Há uma limitação metodológica comum aos três testes que convém explicar antes das tabelas. O robô não tem nenhum sensor que lhe diga a sua altitude absoluta, só sensores que medem o movimento a partir de onde arrancou, e por isso o referencial usado para a elevação, `map`, marca a cota zero no ponto de partida do robô e não na cota real do mundo. Todas as estimativas de elevação ficam assim deslocadas por um valor aproximadamente constante face ao *ground-truth*, medido em cerca de 1,2 m nesta rota. Esse desvio é calculado e subtraído antes de qualquer comparação, e por isso reporta-se sempre o *Root Mean Square Error* (RMSE) bruto e o RMSE alinhado, corrigido desse desvio, sendo este último a métrica relevante para discutir precisão.

4.4.2.1 Ancoragem de *tiles* a *keyframes* contra grelha ingénua

Este teste compara a estratégia de ancoragem da malha de terreno a *keyframes* do SLAM com uma grelha ingénua presa apenas à odometria, ao longo de uma volta completa da rota `realistic-loop` no mundo `forest_realistic_v2_trees_rocks`, com fecho de laço detetado em ambas as configurações. A Tabela 4.17 resume o RMSE de elevação alinhado antes e depois do fecho de laço.

Tabela 4.17: RMSE de elevação alinhado, ancoragem a *keyframes* contra grelha ingénua.

Configuração	**RMSE pré-fecho (m)**	**RMSE pós-fecho (m)**
Tiles ancorados a *keyframes*	0,112	0,108
Grelha ingénua (sem ancoragem)	0,133	0,129

A ancoragem a *keyframes* mantém um RMSE sistematicamente mais baixo que a grelha ingénua nas duas fases, embora a diferença seja modesta porque esta rota só percorre uma volta e o salto de *keyframe* no fecho de laço é pequeno, entre 0,15 e 0,17 m. Uma rota com mais voltas ou mais deriva acumulada tenderia a mostrar essa diferença de forma mais evidente. A Figura 4.14 mostra o mapa de erro de elevação resultante.

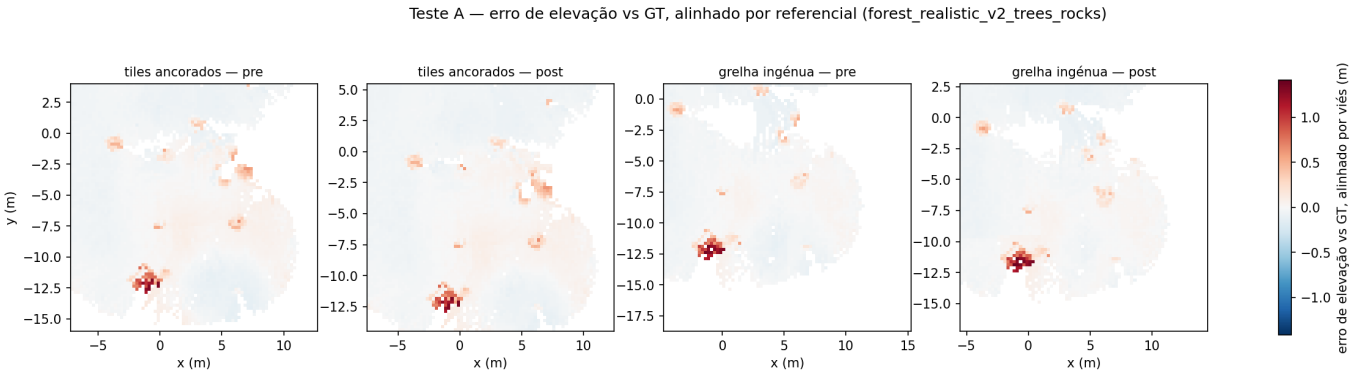


Figura 4.14: Mapa de erro de elevação alinhado, ancoragem a *keyframes* contra grelha ingénua, mundo `forest_realistic_v2_trees_rocks`.

4.4.2.2 Robustez a *outliers*, portão contra mediana

Este teste isola o efeito do *gate* de rejeição de *outliers* e do filtro de mediana na atualização de cada célula, sobre um plano inclinado sintético de 12° com *outliers* injetados artificialmente, sem depender de nenhum mundo do Gazebo. Correu-se em dois regimes de severidade, um regime severo com 15% das leituras contaminadas por um desvio de 1,5 m, e um regime moderado com 5% das leituras contaminadas por um desvio de 0,5 m. A Tabela 4.18 resume o RMSE face ao *ground-truth* nos dois regimes.

Tabela 4.18: RMSE de elevação por configuração de filtragem, regimes de *outliers* severo e moderado.

Configuração	**RMSE severo (m)**	**RMSE moderado (m)**
Sem portão, sem mediana	0,256	0,043
Só portão	0,650	0,107
Só mediana	0,136	0,022
Portão e mediana (predefinição)	0,326	0,051

O resultado mais relevante, e que se mantém em ambos os regimes, é que o portão de *outliers* sozinho piora sempre o RMSE face a não ter filtragem nenhuma. A causa foi identificada no código, o portão aceita sem verificação a primeira observação de cada célula, e se essa primeira leitura for já o *outlier*, o filtro fica ancorado a esse valor errado e passa a rejeitar como *outlier* as leituras corretas seguintes. A mediana isolada dá o melhor resultado nos dois regimes, severo (0,136 m) e moderado (0,022 m), e a combinação predefinida de portão e mediana fica sempre pior do que a mediana sozinha, embora continue bastante melhor do que o portão isolado nos dois casos. A configuração predefinida do sistema, portão e mediana em conjunto, não é portanto ótima, e usar apenas a mediana espacial, sem o portão de rejeição, é a alternativa a considerar em trabalho futuro.

4.4.2.3 Escolha da fórmula de combinação de custo

Este teste compara as duas fórmulas candidatas para combinar o custo de terreno com o custo de obstáculo semântico numa mesma célula. A primeira versão usada durante o desenvolvimento tomava o máximo dos dois termos, mas a análise da Secção 3.4.4 percebeu que essa escolha engole por completo o termo mais fraco sempre que o outro domina, apagando a informação do declive do terreno em toda a vizinhança de um obstáculo. Adotou-se por isso a fórmula independente $1-(1-a)(1-b)$ como predefinição do sistema, e este teste mede as duas situações previstas por essa análise sobre um cenário real, o recorte do mundo capturado no Teste A junto a uma árvore com 9,5° de declive de terreno a 2,3 m da rota, dentro do raio duro do tronco, onde as duas fórmulas deviam coincidir, e na zona de inflação à sua volta, onde deviam divergir.

Dentro do raio duro do obstáculo Dentro do raio duro do tronco, a faixa sólida junto ao seu centro onde o custo de objeto satura em 1,0, o custo de terreno varia célula a célula com o declive local, mas o custo resultante não. A Tabela 4.19 mostra três células reais desta captura, com custo de terreno baixo, médio e no máximo medido nesta captura, todas dentro do raio duro do mesmo tronco.

Tabela 4.19: Custo de terreno, custo de objeto e custo resultante, três células reais dentro do raio duro do tronco.

Custo terreno	**Custo objeto**	$1-(1-a)(1-b)$	$\max(a,b)$
0,000	1,000	1,000	1,000
0,103	1,000	1,000	1,000
0,200	1,000	1,000	1,000

Nas três células o custo de objeto está saturado no valor letal e o custo resultante é 1,0 nas duas fórmulas, apesar de o custo de terreno variar entre 0,000 e 0,200. Um obstáculo sólido absorve sempre o custo de terreno desta forma, nas duas fórmulas por igual, exatamente como esperado para um obstáculo letal e binário como um tronco. Um obstáculo penetrável como um arbusto teria um custo de objeto tipicamente perto de 0,6 em vez de 1,0 (Tabela 4.16), nunca satura o termo de perigo e por isso este efeito de absorção total não se aplicaria mesmo dentro do que seria o seu raio duro.

Na zona de inflação, fora do raio duro É fora do raio duro, na zona de inflação onde o custo de objeto já desceu do valor letal mas ainda é apreciável, que a Secção 3.4.4 prevê divergência entre as duas fórmulas, ilustrada ali de forma esquemática pela Figura 3.29. Esta captura tem 167 células nessa situação, com custo de objeto entre 0,05 e 0,95, e todas com o mesmo custo de terreno, 0,200, correspondente ao declive de 9,5° junto ao tronco. A Tabela 4.20 lista as três células com maior divergência absoluta entre as duas fórmulas.

Tabela 4.20: Custo de terreno, custo de objeto e custo resultante por fórmula, três células reais na zona de inflação com maior divergência entre fórmulas.

Custo terreno	**Custo objeto**	$1-(1-a)(1-b)$	$\max(a,b)$
0,200	0,212	0,370	0,212
0,200	0,238	0,390	0,238
0,200	0,185	0,348	0,200

Nas duas primeiras células o custo de objeto é ligeiramente maior do que o custo de terreno, e a fórmula do máximo devolve por isso o custo de objeto sozinho, descartando por completo a contribuição do terreno. Na terceira célula é o custo de terreno que é maior, e a fórmula do máximo devolve o custo de terreno sozinho, descartando por completo a contribuição do objeto. Em qualquer um dos três casos a fórmula do máximo apaga sempre um dos dois termos, ficando com o valor do outro sem alteração nenhuma, enquanto a fórmula independente combina sempre os dois e produz um valor mais alto do que qualquer um deles isoladamente, confirmando empiricamente o efeito de absorção previsto na Secção 3.4.4.

4.5 Planeamento e Decisão

Os resultados desta secção validam o núcleo do planeamento global e da decisão híbrida de forma isolada, ligando diretamente ao algoritmo D* Lite e a uma réplica exata da fórmula de custo do `hop_decision_node`, sem depender do Nav2, do simulador Gazebo ou de qualquer sensor real. Os mapas usados nos testes de planeamento global e de replaneamento são gerados sinteticamente por um gerador de obstáculos dedicado a este banco de ensaio, e não capturados de um mundo simulado do *ForestGen*, o gerador procedimental de florestas desenvolvido para esta dissertação e usado para construir os mundos do simulador ao longo de todo este capítulo, que também permite controlar a densidade de obstáculos. Optou-se por um gerador dedicado, mais simples, porque garante sempre a existência de um corredor livre entre início e objetivo por construção, mas isso significa que estes números isolam o algoritmo dos erros e do ruído reais do *costmap*, medidos na sua origem nas camadas de perceção (Secção 4.2) e de traversabilidade (Secção 4.4). O evitamento local e a decisão híbrida ao vivo, que dependem dessas camadas a montante e do resto da pilha de navegação, são reportados nas secções correspondentes e não são repetidos aqui.

4.5.1 Planeamento global em ambientes parcialmente conhecidos

Testes do núcleo do D* Lite O núcleo do D* Lite tem seis verificações unitárias isoladas, sem qualquer dependência do Nav2 ou do *costmap*. Todas passaram. A primeira confirma que uma grelha totalmente livre produz um caminho válido. A segunda confirma que o caminho extraído começa mesmo no início, termina no objetivo e tem vários pontos, o que evita que um caminho degenerado faça o controlador pensar que já chegou. A terceira e a quarta confirmam a correção

do valor especial de célula sem informação, discutida nos Desenvolvimentos (Secção 3.5), onde uma grelha por preencher tratada como livre encontra caminho e a mesma grelha tratada de forma conservadora não encontra, tal como esperado. A quinta e a sexta confirmam que uma parede com uma abertura é contornada pela abertura e que uma parede completa corretamente não tem caminho.

Planeamento em mapas com obstáculos ao nível do robô Para testar o planeamento numa escala de floresta, bem maior do que a grelha de doze por doze células dos testes unitários anteriores, gerou-se um banco de mapas de 50 metros por 20 metros, a uma resolução de 0,2 metros por célula, com obstáculos distribuídos aleatoriamente ao nível do robô, troncos finos com diâmetro entre 0,1 e 0,5 metros e rochas ou arbustos maiores com diâmetro entre 0,3 e 1,2 metros. O início e o objetivo estão sempre à mesma altura, a meio da largura do mapa, e o robô nunca se desvia mais do que alguns metros dessa linha central, pelo que o mapa é deliberadamente estreito, o suficiente para dar espaço a desvios em torno de obstáculos sem desperdiçar área nunca visitada. Cada mapa garante a existência de um corredor livre entre o início e o objetivo, com espaço suficiente para o robô passar entre os obstáculos. Testaram-se três densidades de obstáculos, com dez mapas gerados aleatoriamente por densidade. A Figura 4.15 mostra um exemplo de densidade média e o caminho encontrado pelo D* Lite. A Tabela 4.21 resume os trinta mapas.

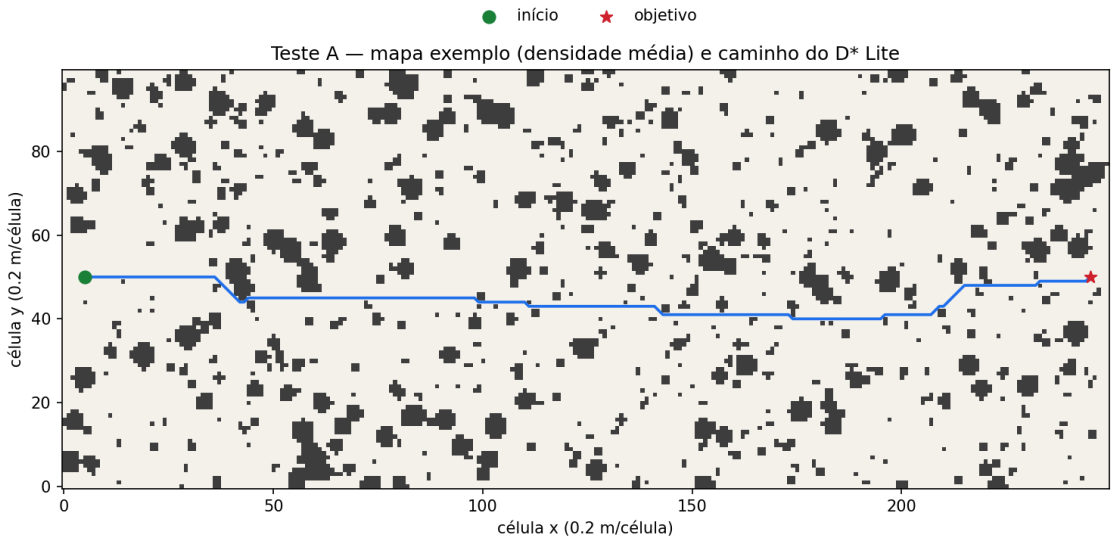


Figura 4.15: Exemplo de mapa gerado com densidade média de obstáculos ao nível do robô e o caminho encontrado pelo D* Lite entre o início e o objetivo.

Tabela 4.21: Planeamento global em 30 mapas (50 m x 20 m, obstáculos ao nível do robô/DBH), 10 sementes por densidade.

Densidade	Sucesso	Custo médio (células)	Desvio-padrão	Tempo médio (ms)
baixa	10/10	243,7	1,2	1,94
média	10/10	247,6	1,2	3,68
alta	10/10	251,8	1,3	5,47

O planeamento encontrou sempre um caminho válido nos trinta mapas. O custo do caminho ótimo sobe de forma consistente com a densidade de obstáculos, de cerca de 244 células equivalentes na densidade baixa para cerca de 252 na densidade alta, o que é o comportamento esperado de mais obstáculos a forçar desvios maiores. O tempo de cálculo também sobe com a densidade, mas mantém-se sempre abaixo dos 8 milissegundos mesmo no mapa mais denso, para uma grelha de 25 mil células.

Repetibilidade Repetiu-se o mesmo mapa, de densidade média, dez vezes seguidas, cada corrida com uma instância nova do planeador. As dez corridas produziram exatamente o mesmo caminho.

Replaneamento com revelação progressiva do mapa O robô nunca tem acesso ao mapa completo, só ao que o sensor já revelou até ao instante atual. Por isso a comparação que interessa não é entre o D* Lite incremental e resolver do zero o mapa final completo, essa alternativa não existe na realidade porque nenhum planeador tem alguma vez essa informação de uma só vez. A comparação justa é entre manter o D* Lite incremental ao longo da missão e resolver do zero, a cada instante, usando apenas a informação que já é conhecida nesse instante, a mesma para os dois lados.

Simulou-se por isso o robô a avançar ao longo de uma rota de cerca de 48 metros, no mesmo banco de mapas do teste anterior, com um sensor circular de alcance de oito metros a revelar progressivamente as células verdadeiras do mapa à medida que o robô se aproxima delas, tal como acontece com o *costmap* real alimentado pela perceção. A cada avanço de quatro metros, mediu-se o custo de continuar a atualizar a mesma instância do D* Lite face ao custo de resolver uma instância nova a partir do zero, com o mesmo conhecimento parcial do mapa nesse instante. A Figura 4.16 mostra a trajetória percorrida num destes ensaios. A Tabela 4.22 resume os resultados.

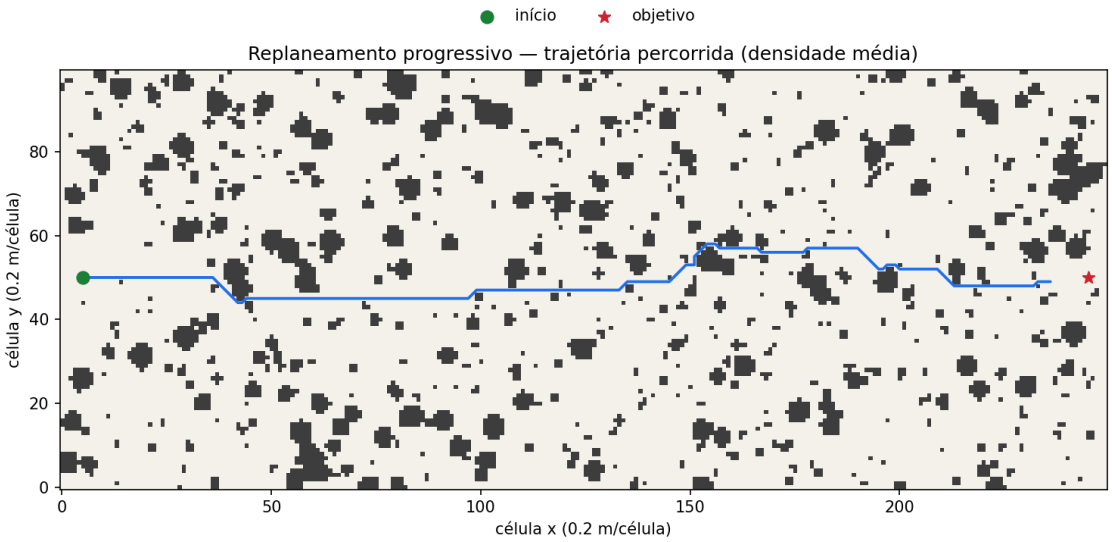


Figura 4.16: Trajetória percorrida num ensaio de replaneamento progressivo, densidade média.

Tabela 4.22: Replaneamento progressivo com revelação parcial do mapa, D* Lite incremental contra resolver do zero.

Densidade	Chegou ao objetivo	Avanços médios	Razão de células (do zero / incremental)
baixa	5/5	13,4	1,70x
média	5/5	13,2	1,65x
alta	5/5	13,8	1,53x

O robô chegou ao objetivo em todas as quinze combinações de densidade e semente testadas, entre 13 e 15 avanços. Em todas elas, manter a mesma instância do D* Lite a atualizar-se de forma incremental custou menos reavaliações de células do que resolver do zero a cada avanço, entre 1,33 e 2,28 vezes menos consoante a semente e a densidade. Este resultado só se confirmou depois de corrigir a comparação inicial, que testava um único corte no meio de um caminho já resolvido por inteiro contra resolver esse mesmo mapa final do zero. Essa primeira versão do teste dava uma vantagem artificial ao replaneamento do zero, porque lhe dava acesso a uma informação completa que nenhum planeador tem na realidade, e chegou a mostrar o incremental a custar mais do que o replaneamento do zero. A versão corrigida, com revelação progressiva e a mesma informação parcial para os dois lados, é a que corresponde à forma como o robô realmente opera numa floresta sem mapa prévio.

4.5.2 Planeamento local e evitamento de obstáculos

O evitamento de obstáculos a curto prazo está acoplado ao controlador de seguimento, tal como descrito nos Desenvolvimentos, e depende da nuvem de pontos da perceção e do *costmap* local

da traversabilidade para ter algum efeito. Por isso não é possível isolar esta parte da camada de planeamento sem arrastar consigo as outras camadas, e os seus resultados, erro de seguimento e contagem de travagens por colisão, são apresentados em conjunto com o resto da pilha de navegação na camada de Controlo.

4.5.3 Decisão híbrida

A decisão de mudar de modo de locomoção compara o tempo estimado de chegar ao objetivo por terra com o tempo estimado por ar, tal como descrito nos Desenvolvimentos. Para validar esta política isoladamente, sem depender da perceção, do SLAM nem de uma simulação de voo completa, replicou-se exatamente a fórmula usada no nó real, com os mesmos parâmetros por omissão, e construíram-se cinco cenários de propósito para a exercitar. A Tabela 4.23 resume os resultados.

Tabela 4.23: Política de custo do `hop_decision_node` (réplica exata da fórmula real), 5 cenários offline construídos de propósito.

Cenário	t_{solo} (s)	t_{ar} (s)	Decisão	Correto
sem caminho, distância curta	inf	58,3	salta	sim
caminho curto e direto	40,0	59,8	contorna	sim
contorno moderado	120,0	63,3	contorna	sim
contorno grande	300,0	60,8	salta	sim
sem caminho, distância grande	inf	200,8	salta	sim

A política decidiu corretamente nos cinco cenários. Quando não há caminho por terra, a decisão salta independentemente da distância direta até ao objetivo, porque um custo infinito por terra nunca é superado por um custo finito por ar, por muito longo que este seja. Quando há caminho, a decisão só passa a preferir o ar quando o desvio por terra é bastante maior do que a distância direta, porque o custo fixo da transição, da descolagem e da aterragem por si só já ultrapassa os trinta segundos, e a margem de segurança exige que o tempo por ar multiplicado por essa margem fique abaixo do tempo por terra antes de vencer. É por isso que mesmo um contorno já apreciável, mais do dobro da distância direta no cenário de contorno moderado, ainda fica do lado do solo, e só um desvio bastante maior, no cenário de contorno grande, é suficiente para inverter a decisão. Este espaçamento confirma que a margem exigida sobre o custo do ar está a funcionar como pretendido e que a decisão não muda de modo por diferenças pequenas de custo.

Este teste valida apenas o núcleo da comparação de custo. As três salvaguardas descritas nos Desenvolvimentos, tentativa mínima por terra antes de aceitar a ausência de caminho como prova de bloqueio, limpeza da evidência acumulada após um salto, e o *gate* que impede o modo aéreo enquanto o SLAM não estiver a localizar-se com confiança, não são exercitadas por este teste offline, que fixa diretamente as entradas da fórmula sem passar pela lógica de estado que as envolve no nó real.

4.5.4 Análise crítica

Os resultados desta secção confirmam a correção dos dois algoritmos centrais desta camada, o D* Lite e a política de decisão híbrida, mas fazem-no de forma isolada e é importante ter presentes os limites que essa escolha implica.

Os mapas usados no planeamento global e no replaneamento são gerados sinteticamente por um gerador aleatório de obstáculos, com corredor livre garantido por construção, e não capturados a partir de um mundo simulado do *ForestGen* nem alimentados pelo *costmap* real produzido pela perceção e pela traversabilidade. Isto isola bem o comportamento do algoritmo de procura em si, mas não diz nada sobre o que acontece quando o *costmap* de entrada tem os erros e o ruído reais dessas camadas a montante, nem sobre o comportamento do *plugin* do D* Lite já integrado no Nav2, só do seu núcleo isolado.

A validação da decisão híbrida tem a mesma natureza. O teste substitui diretamente os valores de tempo por terra e por ar na fórmula, o que confirma a lógica de comparação de custo em si, mas não exercita as três salvaguardas que o nó real acrescenta a essa fórmula, a tentativa mínima por terra, a limpeza pós-salto e o *gate* de localização do SLAM, nem o comportamento ao vivo do gestor de missão que invoca essa decisão.

Por fim, o evitamento local de obstáculos e o comportamento da decisão híbrida numa missão completa e ao vivo, com perceção e SLAM reais em vez de valores fixados de propósito, não são isolados nesta secção precisamente porque dependem de outras camadas, e são reportados na secção de Controlo e no teste de missão completa em pilha integrada (Secção 4.7).

4.6 Controlo

Esta secção mede o que já foi descrito no Desenvolvimento (Secção 3.6), o seguimento de trajetória em solo pelo *Regulated Pure Pursuit* (RPP) e o voo por autopiloto dedicado. Todos os ensaios correm isolados das restantes camadas, sem perceção, sem SLAM, sem mapeamento e sem planeamento, alimentados diretamente pela pose de *ground-truth* do simulador, para que o erro medido seja só do controlo e não se confunda com o de outra camada.

4.6.1 Precisão de seguimento em solo

O RPP seguiu quatro rotas offline num terreno plano sem obstáculos, uma reta pura, uma curva larga de 90 graus, uma reta seguida de uma viragem em L e um ziguezague de quatro troços, cada uma repetida três vezes. As 12 corridas terminaram com sucesso. A Figura 4.17 sobrepõe, para as quatro rotas, o caminho de referência à trajetória realmente percorrida numa das corridas.

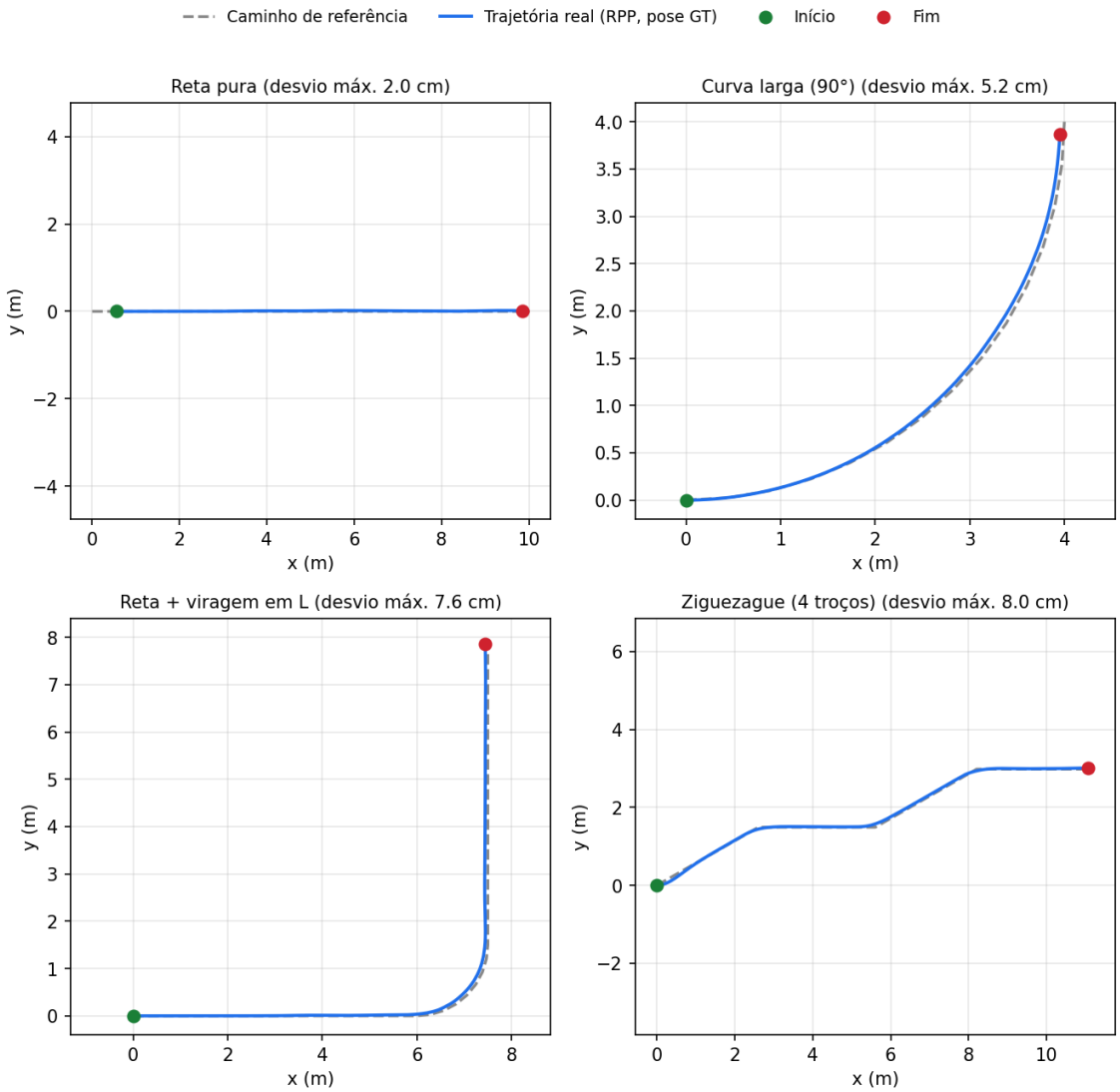


Figura 4.17: Caminho de referência contra a trajetória real do robô, medida pela pose de *ground-truth*, para as quatro rotas testadas. Desvio máximo indicado no título de cada painel.

A Tabela 4.24 resume o erro lateral e de orientação, separando amostras de troços retos e de troços em curva.

Tabela 4.24: Erro de seguimento do RPP, agregado pelas 12 corridas e separado por tipo de troço.

Troço	**Corridas**	**Erro lateral RMS**	**Erro de orientação RMS**
Reta	9	0.020 m	0.075 rad
Curva	9	0.038 m	0.046 rad

O erro lateral em curva é quase o dobro do erro em reta, o que confirma com dados reais o compromisso do *lookahead* já discutido nos Fundamentos (Secção 2.7.1), o RPP persegue um

ponto à frente sobre o arco em vez de cortar para dentro dele, o que basta para o manter dentro de poucos centímetros do caminho mesmo nas viragens mais apertadas do ziguezague.

4.6.2 Recuperação de perturbação

Com o robô a nascer com um desvio lateral de 0.5 m ou 1.0 m face a uma reta de referência, mediu-se o tempo e a distância percorrida até o erro lateral cair abaixo de 10 cm, repetindo para três valores fixos de *lookahead*, curto, o valor de produção e longo. As seis corridas terminaram com sucesso. A Figura 4.18 mostra a evolução do erro lateral ao longo do tempo para o desvio de 1.0 m.

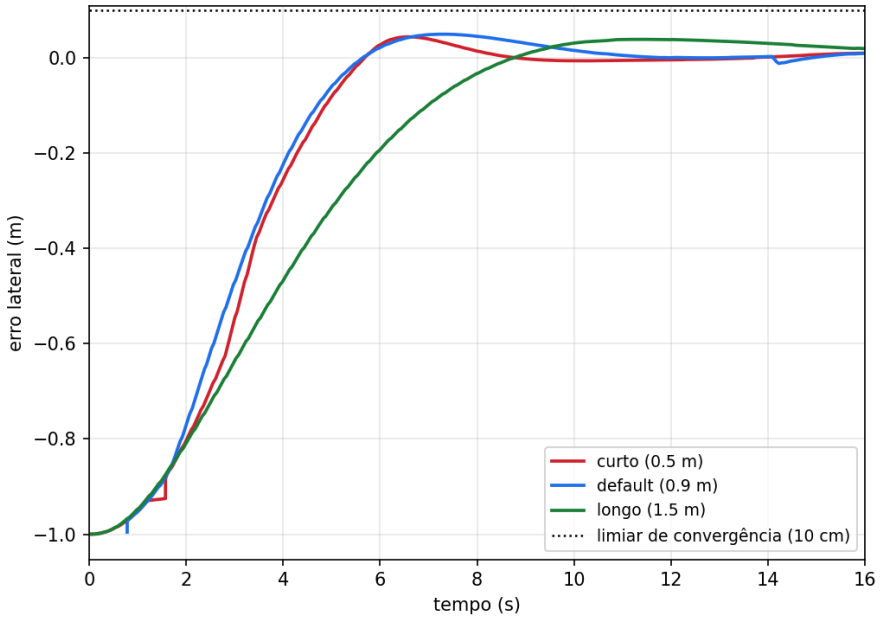


Figura 4.18: Erro lateral ao longo do tempo, a partir de um desvio inicial de 1.0 m, para três valores de *lookahead* do RPP.

Tabela 4.25: Tempo e distância até o erro lateral convergir para menos de 10 cm.

Lookahead	**Desvio inicial**	**Tempo até convergir**	**Distância percorrida**
Curto (0.5 m)	0.5 m	2.26 s	0.57 m
Curto (0.5 m)	1.0 m	4.91 s	0.78 m
Default (0.9 m)	0.5 m	3.57 s	1.32 m
Default (0.9 m)	1.0 m	4.73 s	1.26 m
Longo (1.5 m)	0.5 m	5.98 s	2.31 m
Longo (1.5 m)	1.0 m	6.98 s	2.57 m

A Tabela 4.25 resume o tempo e a distância até à convergência para os três valores de *lookahead* e os dois desvios iniciais testados. O *lookahead* curto reconverge em quase metade da distância do longo, 0.6 a 0.8 m contra 2.3 a 2.6 m, à custa de uma resposta mais nervosa que a literatura

associa a maior tendência para oscilar. Este é o mesmo compromisso já descrito nos Fundamentos (Secção 2.7.1), agora com números medidos em vez de só esperados.

4.6.3 Estabilidade em *hover* sob autopiloto dedicado

Este teste isola apenas os valores de controlo de trajetória aérea, altitude e atitude em *hover*, e não volta a validar a máquina de estados da transição em si, já validada em trinta ciclos terra-ar-ar-terra na Secção 4.1. Validou-se a estabilidade do *hover* sob ArduPilot em três cenários, cada um repetido duas vezes. O primeiro é o quadrotor de fábrica do Gazebo (*iris*), sem qualquer código desta dissertação, usado como referência do que o autopiloto consegue à partida. O segundo é o robô híbrido já montado em configuração de voo, pernas estendidas e lagartas fixas a $\pm 90°$ desde o início, o que isola o desempenho do controlo de voo sobre a plataforma real sem incluir a transição mecânica. O terceiro é o mesmo robô híbrido, mas a nascer em configuração de solo, lagartas a 0°, e a executar a transição mecânica completa, pernas a estender e lagartas a rodar $\pm 90°$, antes de o autopiloto armar e descolar, para verificar se os valores de *hover* se mantêm bons depois dessa transição. A Tabela 4.26 resume os valores medidos por MAVLink durante o hover.

Tabela 4.26: Estabilidade em *hover* sob ArduPilot, altitude alvo de 5 m.

Cenário	**Altitude média**	**Desvio-padrão**	**Roll / pitch RMS**
Quadrotor de fábrica	4.998 m	0.022 m	0.03° / 0.01°
Robô híbrido a voar	4.981 m	0.031 m	0.14–0.21° / 0.20–0.23°
Robô híbrido, transição completa	4.981 m	0.025 m	0.005–0.006° / 0.006°

Os três cenários mantêm a altitude dentro de poucos centímetros do alvo. O robô híbrido em configuração de voo apresenta o desvio-padrão de atitude mais alto dos três, coerente com ser uma plataforma mais pesada do que o quadrotor de fábrica. Já o cenário de transição completa reproduz de forma consistente entre as duas corridas uma atitude ainda mais estável do que a do próprio quadrotor de fábrica, roll e pitch RMS de cerca de 0,005–0,006°, contra 0,03°/0,01° do quadrotor.

Por ser um resultado inesperado, um robô híbrido mais complexo a sair mais estável do que a referência mais simples, este cenário foi corrido de novo, com o mesmo procedimento, para esta dissertação, e reproduziu o mesmo valor, roll e pitch RMS de 0,006°. Confirmou-se também que a IMU simulada dos dois robôs híbridos (o de voo nivelado e o de transição completa) usa exatamente a mesma pose de montagem, a mesma taxa de amostragem e o mesmo modelo de ruído nos ficheiros do simulador, pelo que a diferença não vem de uma vantagem artificial na instrumentação. O que de facto difere entre os dois modelos é a massa e a inércia do corpo, o corpo dedicado ao voo nivelado tem 5,84 kg e inércia de guinada de 0,30 kg m², enquanto o corpo híbrido usado na transição completa, mais leve por ser o mesmo que serve de base às lagartas, tem 3,5 kg e uma inércia consideravelmente menor. É por isso plausível que um corpo mais leve e com menor inércia responda de forma mais suave ao mesmo controlador de atitude do autopiloto, mas este

ensaio não isola a razão exata da diferença, só confirma que ela é real e reprodutível, e não um erro de medição ou uma configuração inconsistente entre os dois modelos.

4.6.4 Precisão de aterragem

Comandaram-se sete saltos aéreos completos a pontos de aterragem diferentes, três repetições da mesma distância na diagonal e quatro combinações distintas de distância, orientação e altitude de cruzeiro, medindo a distância entre o ponto de aterragem comandado e o ponto onde o robô realmente pousou. As sete corridas terminaram com sucesso. A Figura 4.19 contrasta os pontos de aterragem comandados com os pontos reais, e a Tabela 4.27 detalha o erro de aterragem obtido em cada um dos sete saltos.

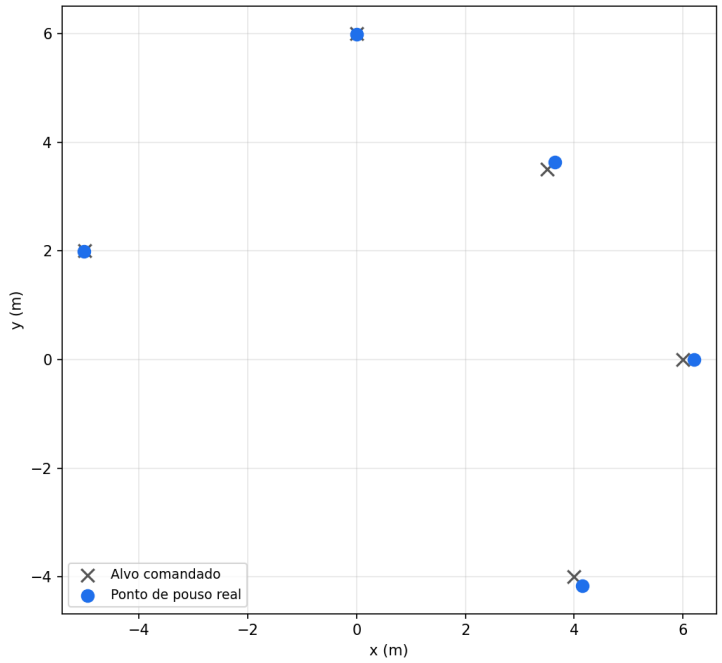


Figura 4.19: Ponto de aterragem comandado contra o ponto de aterragem real, para os sete saltos.

Tabela 4.27: Erro de aterragem por salto.

Salto	**Alvo (x, y)**	**Erro de aterragem**
Diagonal, repetição 1	(3.5, 3.5)	0.202 m
Diagonal, repetição 2	(3.5, 3.5)	0.207 m
Diagonal, repetição 3	(3.5, 3.5)	0.201 m
Leste	(6.0, 0.0)	0.209 m
Norte	(0.0, 6.0)	0.011 m
Sudeste, 4 m de altitude	(4.0, −4.0)	0.225 m
Oeste, baixa altitude	(−5.0, 2.0)	0.012 m

O erro médio de aterragem é de 0.153 m, com desvio-padrão de 0.089 m entre um mínimo de 1.1 cm e um máximo de 22.5 cm. As três repetições do mesmo salto diagonal são consistentes entre si, 0.201 a 0.207 m, o que mostra que a variação entre saltos diferentes reflete a distância e a orientação de cada um e não ruído do próprio ensaio.

4.6.5 Seguimento de trajetória em voo GUIDED

Para os mesmos sete saltos, mediu-se o desvio entre a trajetória realmente voada e a linha reta ideal entre o ponto de descolagem e o alvo de aterragem, durante a fase de cruzeiro. A Figura 4.20 sobrepõe as trajetórias voadas à linha reta ideal para cinco dos sete saltos, e a Tabela 4.28 resume o desvio médio e máximo obtido em cada salto.

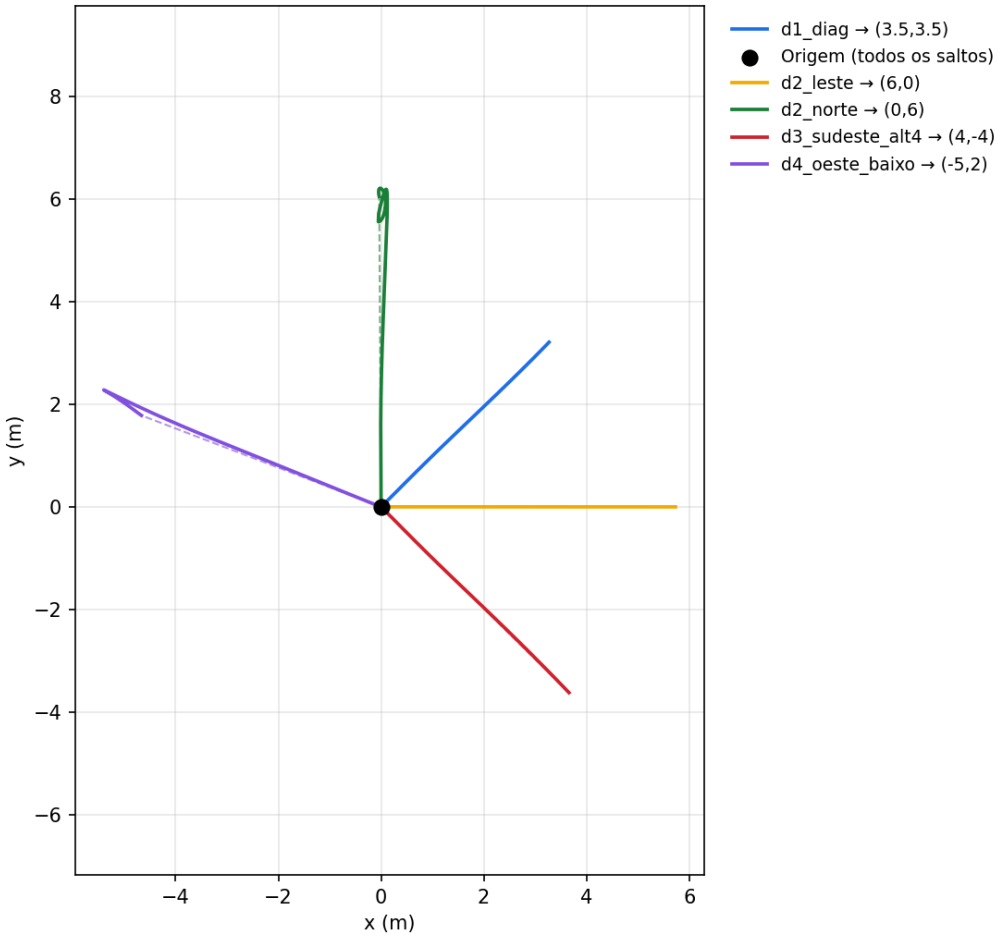


Figura 4.20: Trajetórias de voo reais (linha cheia) contra a linha reta ideal para cada alvo (tracejado), cinco dos sete saltos.

Tabela 4.28: Desvio da trajetória voada face à linha reta ideal, por salto.

Salto	**Desvio médio**	**Desvio máximo**
Diagonal, repetição 1	0.016 m	0.050 m
Diagonal, repetição 2	0.015 m	0.051 m
Diagonal, repetição 3	0.015 m	0.050 m
Leste	0.0002 m	0.0003 m
Norte	0.078 m	0.213 m
Sudeste, 4 m de altitude	0.013 m	0.040 m
Oeste, baixa altitude	0.091 m	0.488 m

O desvio varia muito conforme a direção do salto, de menos de um milímetro no salto para leste até quase meio metro no salto para oeste. Isto é esperado dado que o executor não corrige a trajetória em voo, só comanda o ponto-alvo em modo GUIDED, quem decide o caminho voado entre os dois pontos é inteiramente o autopiloto. O robô nasce sempre virado para leste, e o executor comanda ao autopiloto o rumo que aponta diretamente ao alvo. O salto para leste não exige por isso qualquer rotação antes de avançar, o que explica o desvio quase nulo. Os saltos diagonal e sudeste exigem uma rotação de 45°, o salto para norte uma rotação de 90°, e o salto para oeste uma rotação de cerca de 158°, a maior de todos os sete saltos. O desvio médio cresce exatamente por esta ordem, de 0,0002 m sem rotação a 0,091 m com quase meia-volta, o que é consistente com o autopiloto a desviar-se da linha reta ideal enquanto ainda está a rodar para o rumo certo, antes de convergir para a trajetória direta.

4.6.6 Análise crítica

Os resultados desta secção confirmam que os dois controladores em uso, o RPP em solo e o ArduPilot em voo, seguem o caminho ou o ponto-alvo comandado com precisão de poucos centímetros, mas fazem-no em condições isoladas que valem a pena ter presentes. É também de esperar que ambos tivessem um desempenho bom, o *plugin* do RPP é disponibilizado e mantido pelo próprio Nav2 e o ArduPilot é um autopiloto de voo maduro e amplamente usado fora desta dissertação, este projeto limita-se a configurar e a invocar cada um dos dois, não a implementar a lei de controlo por trás de nenhum deles.

O seguimento em solo e a recuperação de perturbação foram medidos com a pose de *ground-truth* do simulador, não com a pose estimada pelo EKF ou pelo Tree-SLAM reportada na Secção 4.3. O erro aqui reportado é por isso só o do próprio RPP a seguir um caminho já conhecido, e não inclui o ruído de localização que se soma quando o controlador corre a jusante de uma pose estimada em vez de uma pose perfeita. Os ensaios também correram em terreno plano e sem obstáculos, pelo que o evitamento local em resposta a um mapa de custo real, que depende da perceção e da traversabilidade, não foi exercitado aqui e é reportado nas secções correspondentes.

A validação do voo tem limites semelhantes. Só o ArduPilot foi testado, a comparação com o PX4 planeada no Desenvolvimento (Secção 3.6) nunca chegou a ser executada, pelo que estes

resultados nada dizem sobre como um autopiloto alternativo se comportaria nos mesmos cenários. Dos sete saltos, só o cenário diagonal foi repetido três vezes para confirmar a consistência entre corridas, 0.201 a 0.207 m de erro de aterragem e 0.015 a 0.016 m de desvio médio de trajetória. As outras quatro combinações de distância, orientação e altitude, incluindo a que mostrou o maior desvio de trajetória, a oeste, correu cada uma uma única vez, pelo que não se sabe se esses valores se repetiriam de forma igualmente consistente numa segunda tentativa.

4.7 Simulação de Missão Completa em Pilha Integrada

As secções anteriores deste capítulo validaram cada camada isolada das outras, com localização de *ground-truth*, mapas sintéticos ou entradas fixadas de propósito, precisamente para atribuir a causa de cada resultado a uma única camada. Esta secção fecha o capítulo e liga tudo ao mesmo tempo, perceção, SLAM, EKF, mapa de traversabilidade, planeamento, decisão híbrida e voo, numa missão real conduzida pelo gestor de missão e pelo Nav2, sem qualquer atalho de *ground-truth* a guiar o robô. Os mundos usados são gerados pelo *ForestGen*, o gerador procedimental de florestas descrito nos Desenvolvimentos, e cada corrida é orquestrada por um único perfil do `forest` CLI que sobe a pilha completa e arranca o gravador de métricas a 10Hz.

Testaram-se três cenários distintos, cada um a exercitar uma capacidade diferente da integração. O primeiro é uma missão em ciclo fechado de longa duração, para medir a deriva e o efeito do fecho de laço ao longo do tempo. O segundo é uma travessia direta de ponta a ponta num mundo grande gerado aleatoriamente, sem nenhuma rota pré-desenhada a orientar o Nav2. O terceiro é um salto aéreo desencadeado durante uma missão real por um obstáculo genuinamente intransponível, em vez de por uma decisão de custo isolada como na Secção 4.5.

4.7.1 Missão em ciclo fechado

O primeiro cenário usa um mundo com 55 árvores, 17 rochas, 12 arbustos e 2 troncos caídos espalhados por uma área de cerca de 50 por 50 metros. A missão é uma rota fechada pelos quatro quadrantes do mundo, com regresso à origem, repetida dez vezes seguidas, o que junta deriva acumulada ao longo de mais de meia hora de condução autónoma com dez oportunidades de fecho de laço a corrigi-la. A Tabela 4.29 resume o resultado.

Tabela 4.29: Missão em ciclo fechado, 10 voltas, mundo de referência (83 obstáculos).

Métrica	Valor
Resultado da missão	completa
Duração total	1683,0 s (∼28 min)
Distância de referência (rota fechada)	567,2 m
Distância percorrida (odometria fundida)	614,6 m
Excesso sobre a rota de referência	8,4%
Erro de localização, média	0,297 m
Erro de localização, mediana	0,270 m
Erro de localização, percentil 95	0,561 m
Erro de localização, máximo	0,729 m
Replaneamentos acumulados	1314

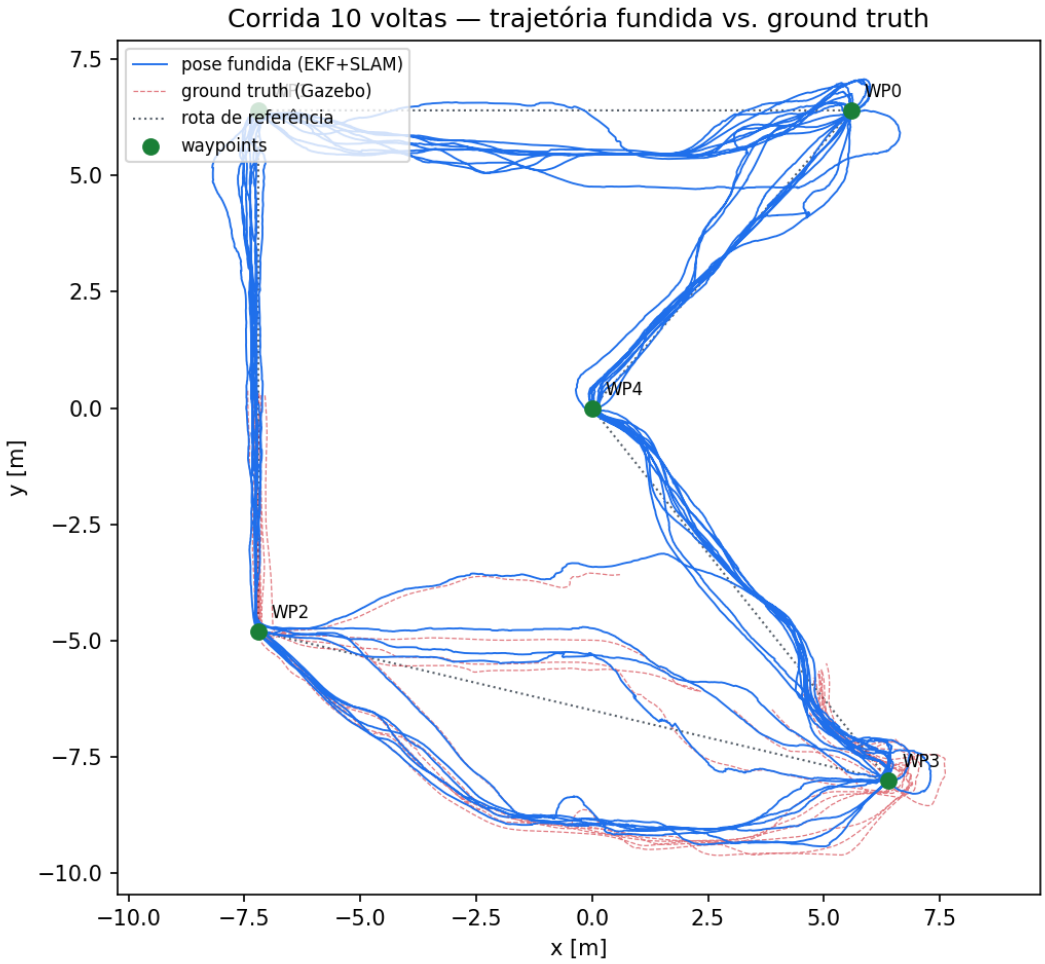


Figura 4.21: Trajetória fundida sobreposta ao *ground-truth* e à rota de referência, dez voltas ao mundo de 83 obstáculos.

A missão completou sem nenhum acidente de sistema, sem o `controller_server` nem o `planner_server` do Nav2 alguma vez colapsarem. O erro de localização mantém-se limitado ao longo de toda a corrida, com um padrão periódico em dente de serra visível na Figura 4.22, a subir dentro de cada volta e a descer sempre que o fecho de laço reconhece uma zona já mapeada, sem nenhuma tendência de crescimento continuado ao longo das dez repetições.

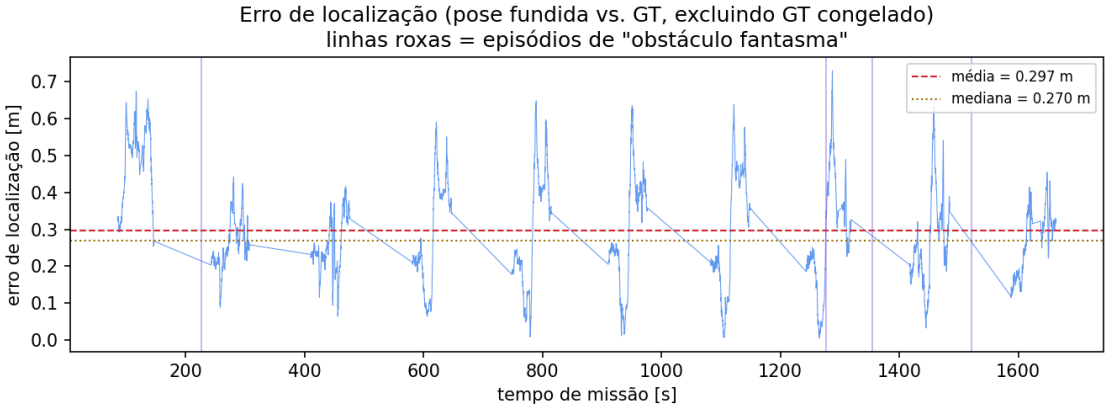


Figura 4.22: Erro de localização ao longo da missão, com os quatro episódios de obstáculo fantasma assinalados a roxo.

Nas voltas intermédias, o controlador do Nav2 assinalou por quatro vezes uma colisão iminente perto da copa de uma árvore, obrigando o D* Lite a recalcular um desvio local. As quatro vezes o sistema recuperou sozinho, sem intervenção do operador, e a missão nunca deixou de progredir. A origem mais provável, discutida na análise crítica desta secção, é arquitetural e não um erro de deteção isolado, a nuvem de pontos não classificados como solo que alimenta o *costmap* do Nav2 é uma via separada da classificação probabilística do Tree-SLAM, e por isso pontos de copa a baixa altura podem entrar no mapa de custo sem nunca passar pelo classificador semântico do SLAM.

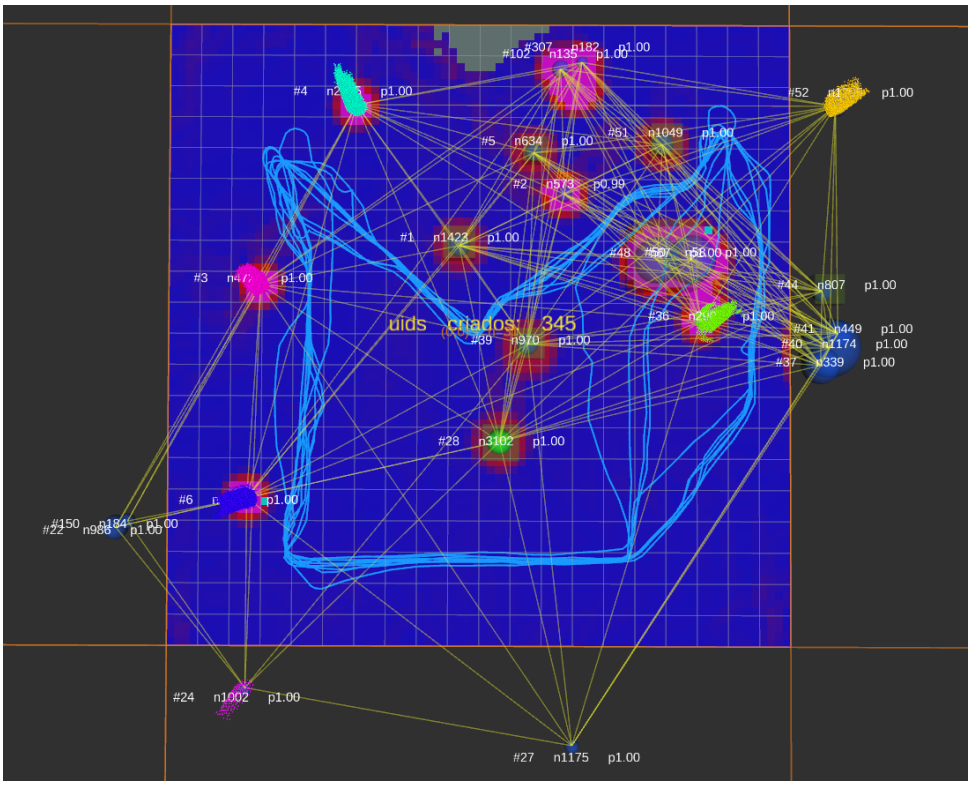
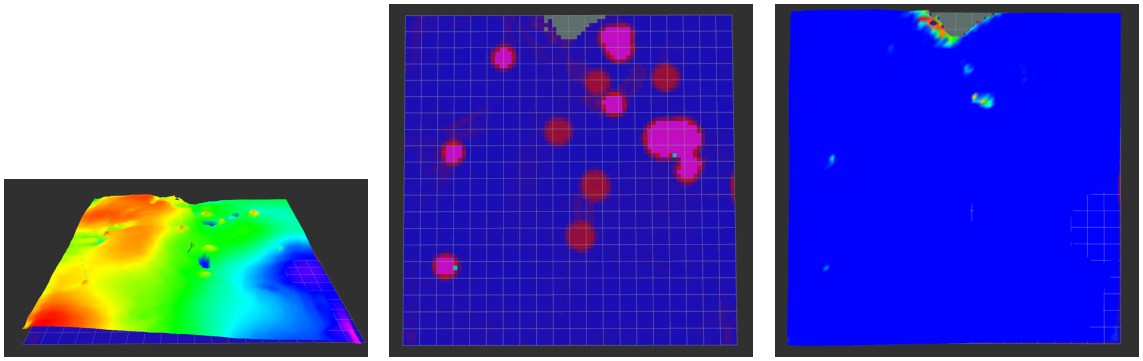


Figura 4.23: Vista do RViz no final da décima volta.



(a) Mapa de landmarks (b) *Costmap* (c) Incerteza

Figura 4.24: Mapa de landmarks, *costmap* de traversabilidade e mapa de incerteza no mesmo instante final da Figura 4.23.

4.7.2 Travessia direta ponta a ponta

O segundo cenário testa o sistema numa configuração deliberadamente diferente da anterior, um único objetivo distante em vez de uma sequência de pontos intermédios escolhidos de propósito, para que o Nav2 tenha de fazer sozinho todo o trabalho de planear o caminho ao longo de uma travessia longa. Gerou-se para isso um mundo novo, aleatório, de 100 por 100 metros, com o robô

a nascer numa borda e o objetivo na borda oposta, 90 metros em linha reta. A Tabela 4.30 resume o resultado desta segunda tentativa.

Tabela 4.30: Travessia direta, 90 m em linha reta, mundo novo gerado aleatoriamente (350 obstáculos).

Métrica	Valor
Resultado da missão	completa
Duração total	542,1 s (∼9,0 min)
Distância de referência (linha reta)	90,0 m
Distância percorrida (odometria fundida)	97,7 m
Excesso sobre a distância direta	8,6%
Erro de localização, média	1,466 m
Erro de localização, mediana	0,888 m
Erro de localização, percentil 95	2,837 m
Replaneamentos acumulados	63

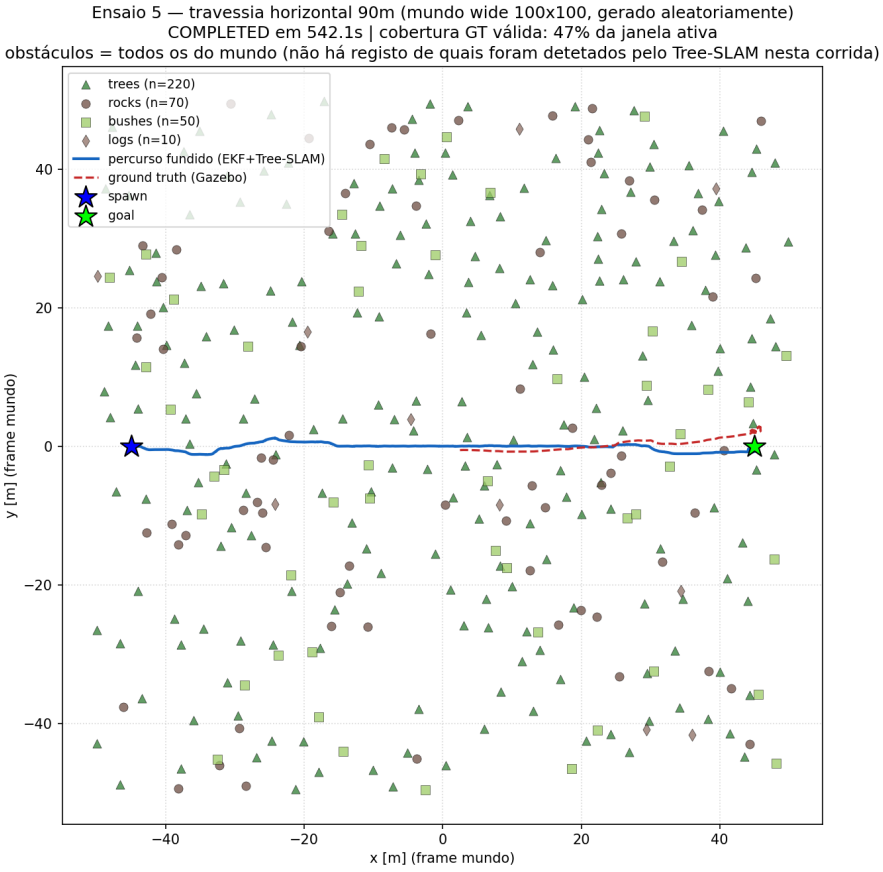


Figura 4.25: Percurso fundido e *ground-truth* sobre os 350 obstáculos do mundo gerado, spawn à esquerda e objetivo à direita.

A missão completou sem nenhum bloqueio real, com o percurso a manter-se quase reto ao longo de toda a travessia, visível na Figura 4.25. O erro de localização é bastante maior do que no cenário de ciclo fechado, com uma média de cerca de 1,5 metros contra os cerca de 0,3 metros da Secção 4.7.1, o que é o resultado esperado de uma travessia em linha reta sem nenhuma oportunidade de fecho de laço a corrigir a deriva acumulada ao longo do percurso, ao contrário da rota fechada que volta sempre a uma zona já conhecida.

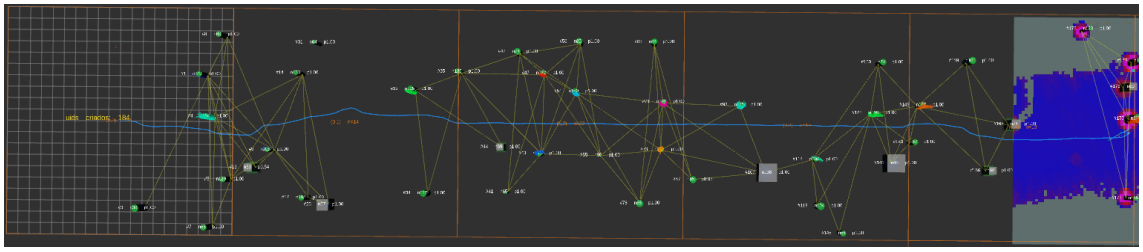


Figura 4.26: Mapa de landmarks no RViz no final da travessia.

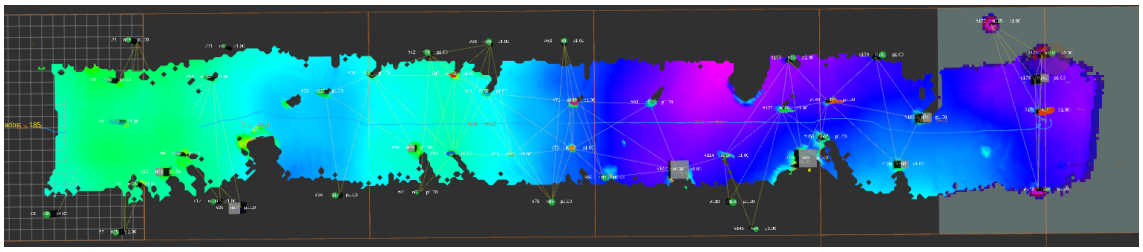


Figura 4.27: Grelha de elevação 2.5D no RViz no mesmo instante da Figura 4.26.

4.7.3 Salto sobre obstáculo real durante uma missão

O terceiro cenário testa a decisão híbrida e o salto aéreo dentro de uma missão real, com um obstáculo genuinamente intransponível por terra, em vez da réplica isolada da fórmula de custo usada na Secção 4.5. Construiu-se um mundo pequeno de propósito, 22 por 22 metros, com uma população normal de árvores, arbustos e um tronco caído, e uma parede de nove rochas de tamanho médio colocada a meio do percurso entre o ponto de partida e o objetivo. As rochas estão espaçadas 2,3 metros centro a centro, perto o suficiente para que o robô, com um raio de planeamento de 0,69 metros no Nav2, não caiba em nenhum intervalo, mas longe o suficiente entre as superfícies para não se tocarem, o que garantiria a colocação real de um obstáculo físico único a olhos do classificador de perceção em vez de uma parede sólida. O corredor de voo por cima da parede, e o espaço à volta do ponto de descolagem, ficam deliberadamente livres de árvores, para que a transição para o modo aéreo não seja impedida pela copa de uma árvore próxima.

A missão envia um único objetivo do outro lado da parede. O robô conduz até ficar bloqueado, o gestor de missão esgota duas tentativas de replaneamento, um valor propositadamente baixo porque este bloqueio é físico e permanente e não vale a pena esperar que se resolva sozinho como uma oscilação de *costmap*, e dispara o salto automaticamente. Repetiu-se a corrida 3 vezes, sem alterar mundo nem parâmetros entre repetições. A Tabela 4.31 resume essas três corridas.

Tabela 4.31: Salto sobre parede real de rochas durante uma missão, 3 repetições válidas, mesmo mundo e parâmetros.

Corrida	Duração	Fase aérea (descolagem→pouso)	Resultado
1	156,7 s	37,1 s	completa
2	142,2 s	31,5 s	completa
3	158,0 s	31,1 s	completa

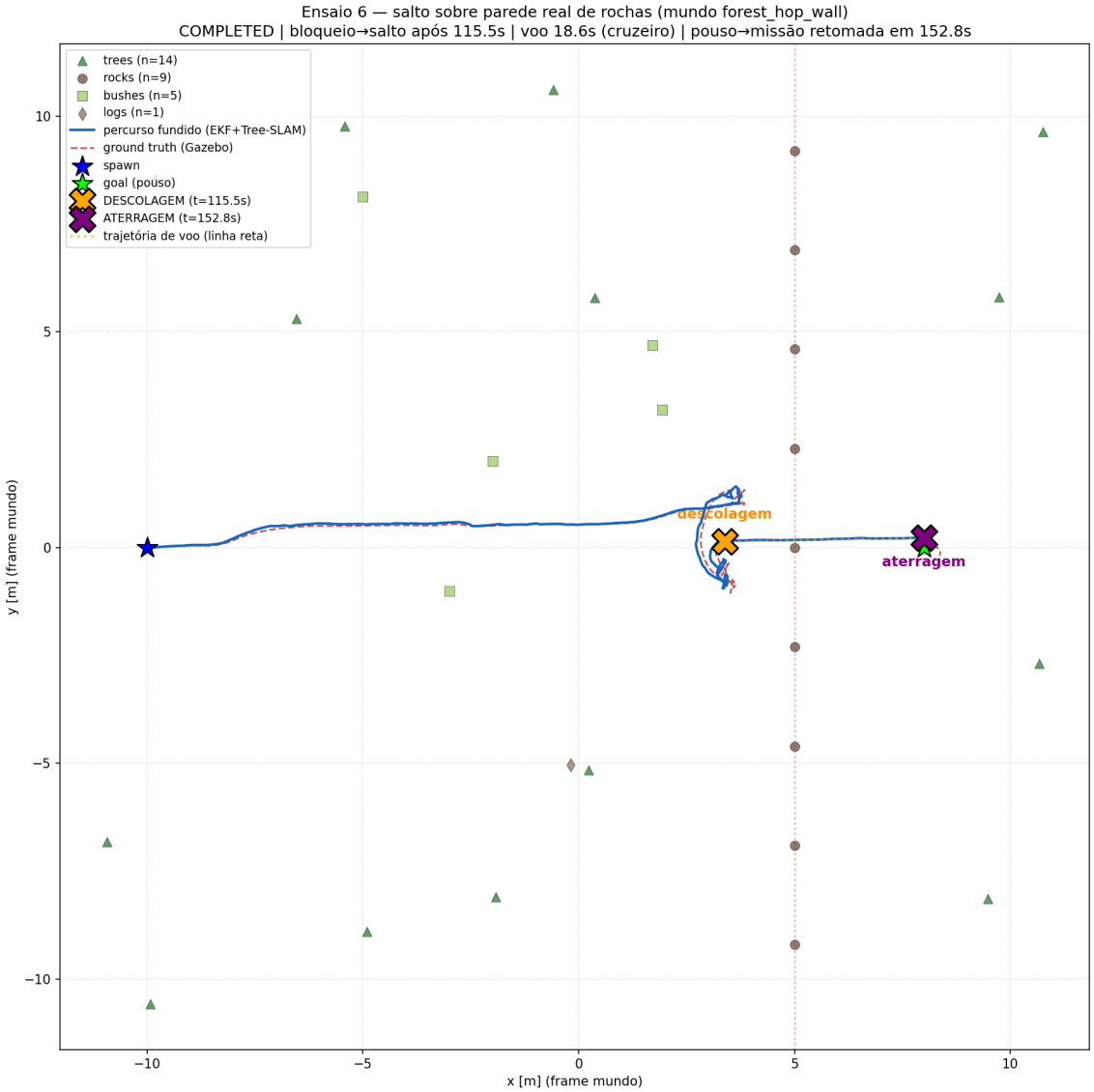


Figura 4.28: Corrida 1, percurso fundido e *ground-truth* com o ponto exato de descolagem e de aterragem assinalados, lidos diretamente da fase do salto na mesma amostra de telemetria.

Em todas as corridas o robô conduz até junto da parede, para, transforma-se para o modo aéreo, sobrevoa as rochas em linha reta e pousa a poucos metros do objetivo, retomando a missão de

imediato depois de aterrar, tal como ilustrado na Figura 4.28 para a primeira corrida. Apesar da taxa de sucesso medida ser 100% não se pode alargar sozinha a nenhuma conclusão porque a amostra é pequena.

Em todos os casos o Tree-SLAM relocalizou-se depois da aterragem com dez a onze landmarks reais no mapa, dentro da gama medida como suficiente para uma relocalização fiável nos ensaios isolados da camada de localização e mapeamento.

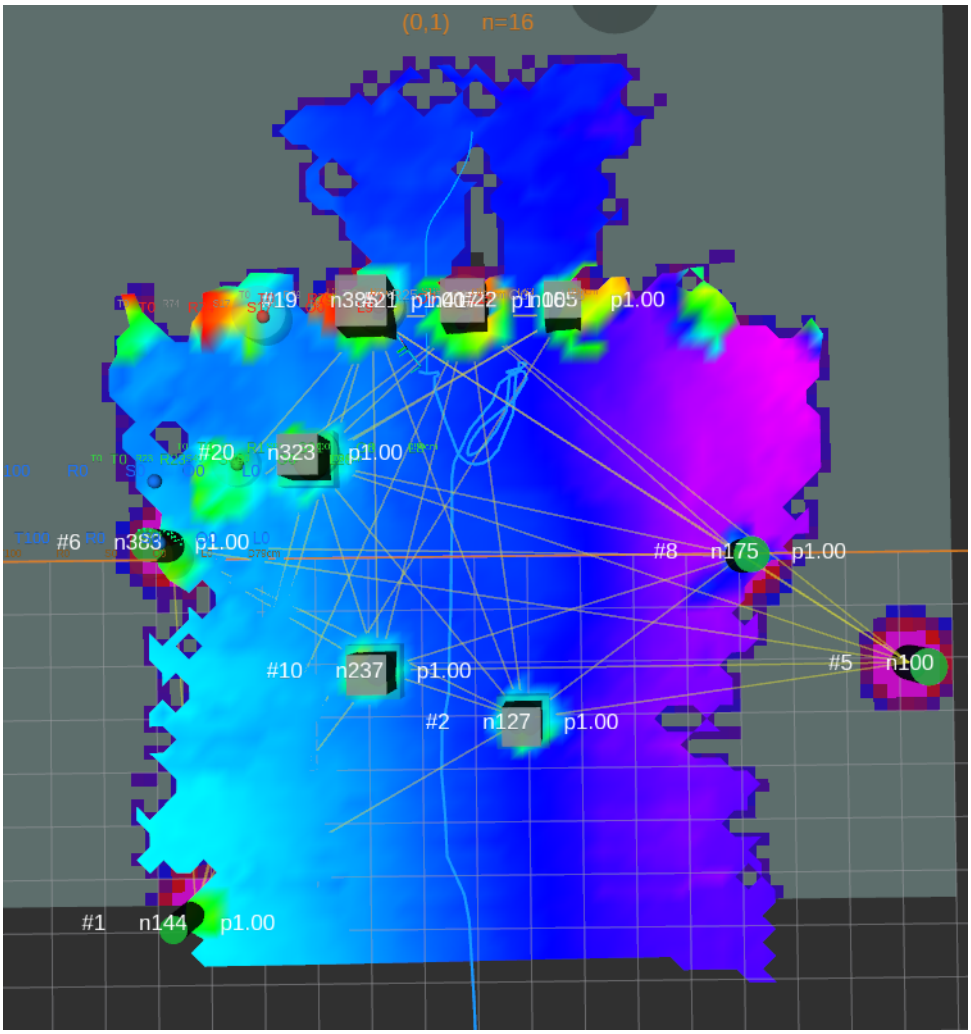


Figura 4.29: Vista do RViz depois da aterragem, com o robô já relocalizado do outro lado da parede de rochas.

4.7.4 Análise crítica

Os três cenários desta secção confirmam que o sistema completo, com todas as camadas ligadas e sem nenhum atalho de *ground-truth*, consegue conduzir missões reais de longa duração, atravessar um mundo grande de ponta a ponta e decidir e executar um salto aéreo a meio de uma missão. É precisamente a integração que a Introdução aponta como pouco explorada na literatura, e é a

peça que faltava para ligar os resultados isolados de cada camada anterior a um comportamento de sistema.

Alguns limites ficam por resolver. O ciclo fechado e a travessia correram cada um uma única vez, e o salto três vezes, amostras pequenas de mais para sustentar qualquer conclusão estatística sobre o sistema completo. Isto foi uma escolha deliberada e não um descuido, o tempo disponível para o projeto não permitia multiplicar por cinco ou dez o número de corridas de uma missão que já demora sozinha entre nove e trinta minutos a completar, e o objetivo destes três cenários é sobretudo qualitativo, mostrar que o sistema integrado consegue de facto conduzir uma missão real do início ao fim sem nenhum atalho de *ground-truth*, e não quantificar com confiança estatística a frequência com que isso acontece. O gravador de métricas usado para medir o erro de localização tem também um problema de instrumentação já identificado, o *ground-truth* do Gazebo congela por vezes durante dezenas de segundos seguidos sem sinalizar o erro, o que obrigou a filtrar essas amostras congeladas antes de calcular qualquer estatística de erro. É um problema do gravador e não do robô, a pose fundida usada para navegar nunca deixou de atualizar durante esses episódios, mas reduz a fração da missão coberta por medidas de erro fiáveis. Tentou-se também validar a generalização a mundos mais irregulares ou mais densos, gerados pelo mesmo *ForestGen* com parâmetros de terreno diferentes, mas encontrou-se nesses mundos um bloqueio de *costmap* reprodutível sem obstáculo físico correspondente, provavelmente ligado à mesma via arquitetural do obstáculo fantasma discutida na Secção 4.7.1, agravada pela maior variação real de solo desses terrenos. Não se investigou a causa raiz a tempo, e essa validação de generalização fica como trabalho pendente. Por fim, e como em todo este capítulo, os três cenários correram inteiramente em simulação, sem nenhuma validação em *hardware* real, pelo que a integração aqui demonstrada não garante por si só que o mesmo comportamento se reproduza sem alteração num robô físico.

Capítulo 5

Conclusão e Trabalhos Futuros

5.1 Síntese do Trabalho Realizado

Esta dissertação propôs-se conceber, implementar e avaliar uma arquitetura modular de navegação e decisão autónoma para um robô híbrido terrestre-aéreo aplicado a cenários florestais, dividida em seis objetivos específicos apresentados no Capítulo 1. Fechado o capítulo de Resultados, cabe agora confrontar cada um desses objetivos com o que foi de facto medido, célula a célula, sem arredondar as partes que ficaram incompletas.

A **conceção e modelação do robô híbrido** (Secção 3.1) idealizou uma plataforma de propulsão separada, com os rotores alojados dentro das próprias lagartas, e implementou a máquina de estados que rege a transição mecânica entre o modo de solo e o modo de voo. Essa máquina foi validada em trinta ciclos terra-ar-ar-terra consecutivos, todos concluídos com sucesso e nenhum preso num estado, com uma duração média de 33,4 segundos por ciclo (Secção 4.1). Este objetivo não incluiu, no entanto, a definição de um modelo cinemático explícito para além do princípio geral de um veículo *skid-steer*, nem a integração de sensores reais, e a modelação mecânica em CAD ficou deliberadamente por terminar para concentrar o esforço na simulação.

O **pipeline de perceção multimodal** (Secção 4.2) combina LiDAR 3D e câmara RGB para detetar e classificar quatro classes estruturais, tronco, rocha, arbusto e tronco caído. O LiDAR isola o fuste de um tronco sem contaminação de copa em 96 de 96 cenas testadas, salvo um ramo residual numa única árvore mais desfavorável do conjunto, e classifica corretamente a árvore e a rocha em cerca de 85% dos casos num banco de ensaio offline favorável, mas essa classificação degrada-se de forma acentuada ao vivo sempre que a nuvem de pontos de uma única vista fica abaixo de cerca de 75 pontos, o que motivou e justificou, com números e não apenas com intuição, a acumulação de evidência multi-vista implementada no Tree-SLAM. A câmara, por seu lado, atinge um *mAP@50* global de 0,79 em validação e mostra um padrão de erro complementar ao do LiDAR, degradando-se em condições diferentes das que degradam o sensor geométrico. A fusão entre os dois sensores no *tracker* do Tree-SLAM foi implementada e testada, mas o primeiro ensaio, com apenas duas corridas por condição, não mostrou nenhuma diferença mensurável entre a ter

ligada e desligada, pelo que o benefício teórico da complementaridade entre sensores ainda não está confirmado na prática.

A **implementação do SLAM por *landmarks*** (Secção 4.3) é o resultado mais sólido desta dissertação. Ao longo de doze voltas consecutivas sobre a mesma rota, já com o raio do alinhamento local corrigido para o seu valor ótimo de 11 m, o Tree-SLAM mantém um erro de posição limitado, estável perto dos 0,20 m, enquanto o estimador contínuo isolado cresce de forma praticamente monótona até 1,613 m na última volta, uma vantagem de 3,8 vezes (Secção 4.3.1). Essa mesma parametrização reduziu a dispersão entre corridas em mais de cinco vezes face ao valor inicial de 15 m. Um ensaio de generalização a quatro mundos distintos, ainda com o raio no valor sub-ótimo de 15 m e por isso um limite inferior conservador do desempenho final do sistema, mostra a vantagem a crescer com a dificuldade do cenário até 4,2 vezes no mundo de terreno mais irregular (Secção 4.3.5), mas os dois números não devem ser lidos como uma progressão contínua de uma só medição, porque vêm de configurações e amostras diferentes. O fecho de laço elimina por completo as divergências observadas quando desligado.

A **representação do ambiente e o custo de traversabilidade** (Secção 4.4) tem o seu núcleo validado por trinta testes unitários. Uma comparação direta entre a versão inicial do modelo e a versão corrigida confirma a correção mais importante feita ao longo do trabalho, a versão inicial tratava vegetação penetrável como obstáculo letal e a versão corrigida já distingue as duas coisas. Testou-se depois a consistência da malha de terreno já produzida ao vivo, num mapa de custo real sob deriva e ruído, e a escolha da fórmula de combinação de custo (Secção 4.4.2). Esse teste corrigiu uma suposição anterior. O portão de rejeição de *outliers* e o filtro de mediana combinados, a configuração predefinida do sistema, só são a melhor escolha no regime moderado de contaminação. No regime severo são piores do que não filtrar nada. O que continua por fazer nesta camada é alargar essa validação a um conjunto de mundos escolhidos de propósito para isolar cada tipo de terreno, em vez do único mundo florestal usado até agora.

O **planeamento de trajetória e a decisão de modo** (Secção 4.5) validam o núcleo do D* Lite em trinta mapas sintéticos, sempre com sucesso, com o replaneamento incremental a custar sempre menos recálculo do que resolver o mapa do zero a cada avanço, e a política de decisão híbrida acerta os cinco cenários offline construídos de propósito para a exercitar. O **controlo** que executa estas decisões ao vivo (Secção 4.6) segue as rotas de referência com desvio lateral máximo na ordem dos centímetros e mantém a altitude do voo pairado dentro de poucos centímetros do alvo nos três cenários testados (Secção 4.6.3).

Por fim, a **simulação e a geração procedimental de mundos** produziram, através do *ForestGen*, mundos florestais com distribuição espacial de árvores e rochas por agregação e inibição por proximidade, em vez de colocação uniforme, usados para gerar os mundos de todos os ensaios deste capítulo. Sobre eles correu-se, já no final do trabalho, o teste de missão completa em pilha integrada (Secção 4.7), com a perceção, o SLAM, o mapa de traversabilidade, o planeamento, a decisão híbrida e o controlo todos ligados ao mesmo tempo e sem qualquer atalho de *ground-truth* a guiar o robô. Os três cenários testados, uma missão em ciclo fechado de dez voltas, uma travessia direta de 90 m de ponta a ponta e um salto aéreo desencadeado por um obstáculo real durante

uma missão, completaram todos com sucesso. É precisamente esta integração conjunta, apontada na Introdução (Secção 1.1) como pouco explorada na literatura, que fica assim demonstrada a funcionar de ponta a ponta, embora com amostras ainda pequenas, uma corrida no primeiro cenário, uma no segundo e três no terceiro, que a Secção 5.2 discute adiante.

5.2 Limitações

As limitações identificadas ao longo do capítulo de Resultados agrupam-se em três tipos, as que resultam de o teste ter sido feito de forma isolada e só parcialmente confrontado com a pilha completa, as que resultam de uma amostra pequena de mais para sustentar uma conclusão forte, e as que são limitações genuínas e ainda não resolvidas do próprio sistema.

Do primeiro tipo fazem parte quase todos os ensaios deste capítulo. O Tree-SLAM foi medido a conduzir uma rota fixa a partir da pose de referência do simulador, e não pela navegação autónoma real, exceto no cenário de ciclo fechado do teste de missão completa (Secção 4.3.6). O D* Lite e a decisão híbrida foram validados sobre mapas sintéticos e uma réplica exata da fórmula de custo, sem o *costmap* nem as salvaguardas do nó real (Secção 4.5.4). A fiabilidade da máquina de estados híbrida foi medida num mundo vazio, sem obstáculos nem vento (Secção 4.1). Em todos estes casos o isolamento foi uma escolha deliberada para atribuir a causa do resultado a uma única camada, e o teste de missão completa em pilha integrada (Secção 4.7) já respondeu, de forma positiva, à pergunta de saber se essas camadas isoladas se comportam bem quando ligadas ao mesmo tempo, embora a amostra pequena de cada um dos seus três cenários, discutida a seguir, ainda não permita generalizar essa resposta com confiança total.

Do segundo tipo, a amostra offline de classificação por LiDAR tem apenas 115 *clusters* (Secção 4.2), o primeiro teste da fusão câmara-LiDAR no *tracker* do SLAM tem apenas duas corridas por condição e não mostra diferença mensurável entre ligada e desligada, os quatro mundos usados para testar a generalização do SLAM não isolam uma única variável entre si (Secção 4.3.5), e os três cenários do teste de missão completa correram cada um uma única vez, com exceção do salto sobre obstáculo real, repetido três vezes (Secção 4.7).

Do terceiro tipo, e sem tentar escondê-las, ficam por resolver: a cobertura de apenas 58,6% do mapa de *landmarks* e os 17,5% de *landmarks* sem correspondência real, dos quais uma parte é classificação errada de arbustos como troncos (Secção 4.3.2); o limiar de 6 m usado na fusão de *landmarks* duplicados, nunca isolado num ensaio dedicado (Secção 4.3.6); a rocha, que continua a ser a classe mais fraca tanto no LiDAR como na câmara; a estabilidade de atitude inesperadamente elevada do robô a fazer a transição mecânica completa antes de armar, só parcialmente explicada pela diferença de massa e inércia face ao robô já em modo de voo (Secção 4.6.3); a assimetria observada no erro de aterragem entre direções de aproximação, que não segue o mesmo padrão do erro de seguimento em voo e permanece sem explicação satisfatória (Secções 4.6.4 e 4.6.5); e um bloqueio de *costmap* reprodutível, sem obstáculo físico correspondente, encontrado ao tentar validar o teste de missão completa em mundos mais densos ou mais irregulares, provavelmente

ligado à mesma via arquitetural do obstáculo fantasma já identificado no cenário de ciclo fechado, mas cuja causa raiz não foi investigada a tempo (Secção 4.7.4).

5.3 Trabalhos Futuros

A prioridade imediata é melhorar o pipeline de perceção, porque é aí que se espera o maior ganho para o resto do sistema, com o SLAM, o mapa de traversabilidade e a decisão híbrida a herdarem diretamente qualquer melhoria na deteção e classificação. Do lado da fusão entre sensores, vale a pena experimentar um tipo de acoplamento diferente do assimétrico usado hoje, em que a câmara apenas confirma ou limita a classificação do LiDAR, testando formas mais simétricas de combinar os dois sensores. Do lado do LiDAR isolado, a assinatura geométrica usada para classificar cada *cluster* é escolhida manualmente e limitada a quatro variáveis. Vale a pena testar uma abordagem de aprendizagem automática, por exemplo uma regressão ou uma rede neuronal simples, para selecionar automaticamente as variáveis mais discriminativas para cada classe, em vez de as escolher à mão. Numa escala menor, falta também investigar a causa raiz do bloqueio de *costmap* sem obstáculo físico correspondente encontrado nos mundos mais densos ou mais irregulares, que atualmente impede alargar o teste de missão completa a esses mundos, e correr até ao fim a validação formal da geração procedimental de mundos e alargar a validação do modelo de custo de traversabilidade (Secção 4.4) a um conjunto de mundos que isolem cada tipo de terreno em separado.

Segue-se um conjunto de investigações mais específicas, motivadas diretamente pelas limitações do terceiro tipo listadas acima. Vale a pena repetir o ensaio de fusão câmara-LiDAR com mais corridas por condição, para confirmar ou refutar de forma conclusiva o benefício teórico que a complementaridade entre os dois sensores sugere. Vale também a pena isolar o efeito do limiar de distância da fusão de *landmarks* duplicados, testando-o ligado e desligado tal como já foi feito para o alinhamento local e o fecho de laço. A assinatura geométrica usada para distinguir rocha de tronco caído e de arbusto, que continua a ser a fronteira mais frágil do classificador, é diretamente o alvo da abordagem de aprendizagem automática proposta acima para o LiDAR.

Numa escala mais larga, o SLAM já foi reavaliado sob navegação autónoma real no cenário de ciclo fechado do teste de missão completa, mas apenas nessa configuração, e valeria a pena repetir essa reavaliação nos outros cenários e mundos, para confirmar que o seu desempenho isolado se mantém quando é o próprio sistema a escolher onde o robô passa em condições mais variadas. Os mundos usados para avaliar a perceção variam até agora só o declive do terreno, e ficaria por medir o efeito de variar também a densidade de vegetação, um eixo de dificuldade distinto e igualmente relevante em floresta real.

Por fim, ficam três direções de mais longo prazo. A primeira é retomar e terminar a modelação mecânica do robô, incluindo a definição formal de um modelo cinemático e a integração de sensores reais, hoje apenas idealizados conceptualmente. A segunda é uma validação, ainda que parcial, em hardware real, começando pelas partes mais isoláveis, como a máquina de estados da transição de modo ou o controlo de seguimento em solo. A terceira é considerar a disponibilização como código

aberto do *ForestGen* e do CLI *forest*, que hoje só existem no repositório privado deste projeto, mas que poderiam servir outros grupos de investigação em robótica florestal da mesma forma que o repositório da ETH Zurique serviu esta dissertação (Secção 3.7.3.1).

Referências

[1] J. L. Strunk, S. E. Reutebuch, R. J. McGaughey e H.-E. Andersen, «An Examination of GNSS Positioning under Dense Conifer Forest Canopy in the Pacific Northwest, USA,» *Remote Sensing Applications: Society and Environment*, vol. 37, p. 101 428, 2024. DOI: 10.1016/j.rsase.2024.101428.

[2] D. Baril et al., «Kilometer-Scale Autonomous Navigation in Subarctic Forests: Challenges and Lessons Learned,» *Field Robotics*, vol. 2, pp. 1628–1660, 2022. DOI: 10.55417/fr.2022050.

[3] P. Nazate-Burgos, M. Torres-Torriti, S. Aguilera-Marinovic, T. Arévalo, S. Huang e F. Auat Cheein, *Robust 2D lidar-based SLAM in arboreal environments without IMU/GNSS*, https://arxiv.org/abs/2505.10847, arXiv:2505.10847, 2025.

[4] D. D. Fan et al., «Autonomous Hybrid Ground/Aerial Mobility in Unknown Environments,» em *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, 2019. DOI: 10.1109/IROS40897.2019.8968276.

[5] R. Zhang, Y. Wu, L. Zhang, C. Xu e F. Gao, *Autonomous and Adaptive Navigation for Terrestrial-Aerial Bimodal Vehicles*, https://arxiv.org/abs/2109.04706, arXiv:2109.04706, accepted to IEEE Robotics and Automation Letters / ICRA 2022, 2021.

[6] X. Wang et al., «Path Planning for Air-Ground Robot Considering Modal Switching Point Optimization,» em *Proc. Int. Conf. Unmanned Aircraft Systems (ICUAS)*, 2023, pp. 87–94. DOI: 10.1109/ICUAS57906.2023.10156162.

[7] R. Zhang et al., *Model-Based Planning and Control for Terrestrial-Aerial Bimodal Vehicles with Passive Wheels*, https://arxiv.org/abs/2403.00322, arXiv:2403.00322, accepted to IROS 2023, 2024.

[8] F. Zellweger et al., «Estimating below-Canopy Light Regimes Using Airborne Laser Scanning: An Application to Plant Community Analysis,» *Ecology and Evolution*, vol. 9, n.º 16, 2019. DOI: 10.1002/ece3.5462.

[9] A. E. Carvalho, D. Portugal e P. Peixoto, «On Terrain Traversability Analysis in Unstructured Environments: Recent Advances in Forest Applications,» *Intelligent Service Robotics*, vol. 18, pp. 195–213, 2025. DOI: 10.1007/s11370-025-00591-4.

[10] D. Rapado-Rincon e G. Kootstra, *Tree-SLAM: semantic object SLAM for efficient mapping of individual trees in orchards*, https://arxiv.org/abs/2507.12093, arXiv:2507.12093, submetido a Smart Agricultural Technology, 2025.

[11] A. J. Sathyamoorthy et al., «VERN: Vegetation-Aware Robot Navigation in Dense Unstructured Outdoor Environments,» em *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, 2023. DOI: 10.1109/IROS55552.2023.10341395.

[12] CERVI Robotics. «HUUVER: Hybrid UAV-UGV for Efficient Relocation of Vessels,» acedido em 3 de jul. de 2026. URL: https://cordis.europa.eu/project/id/870236.

[13] A. Kalantari e M. Spenko, «Modeling and Performance Assessment of the HyTAQ, a Hybrid Terrestrial/Aerial Quadrotor,» *IEEE Transactions on Robotics*, vol. 30, n.º 5, pp. 1278–1285, 2014. DOI: 10.1109/TRO.2014.2337555.

[14] I. Mandralis, R. Nemovi, A. Ramezani, R. M. Murray e M. Gharib, «ATMO: An Aerially Transforming Morphobot for Dynamic Ground-Aerial Transition,» *Communications Engineering*, vol. 4, 2025. DOI: 10.1038/s44172-025-00413-6.

[15] J. F. Ferreira, D. Portugal, M. E. Andrada, P. Machado, R. P. Rocha e P. Peixoto, «Sensing and Artificial Perception for Robots in Precision Forestry: A Survey,» *Robotics*, vol. 12, n.º 5, p. 139, 2023. DOI: 10.3390/robotics12050139.

[16] S. Krisanski et al., «Forest Structural Complexity Tool—An Open Source, Fully-Automated Tool for Measuring Forest Point Clouds,» *Remote Sensing*, vol. 13, n.º 22, p. 4677, 2021. DOI: 10.3390/rs13224677.

[17] K. Zhang, S.-C. Chen, D. Whitman, M.-L. Shyu, J. Yan e C. Zhang, «A Progressive Morphological Filter for Removing Nonground Measurements from Airborne LIDAR Data,» *IEEE Transactions on Geoscience and Remote Sensing*, vol. 41, n.º 4, pp. 872–882, 2003. DOI: 10.1109/TGRS.2003.810682.

[18] W. Zhang et al., «An Easy-to-Use Airborne LiDAR Data Filtering Method Based on Cloth Simulation,» *Remote Sensing*, vol. 8, n.º 6, p. 501, 2016. DOI: 10.3390/rs8060501.

[19] J.-F. Lalonde, N. Vandapel, D. F. Huber e M. Hebert, «Natural Terrain Classification Using Three-Dimensional Ladar Data for Ground Robot Mobility,» *Journal of Field Robotics*, vol. 23, n.º 10, pp. 839–861, 2006. DOI: 10.1002/rob.20134.

[20] M. A. Fischler e R. C. Bolles, «Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography,» *Communications of the ACM*, vol. 24, n.º 6, pp. 381–395, 1981. DOI: 10.1145/358669.358692.

[21] P. Jiang, P. R. Osteen, M. Wigness e S. Saripalli, «RELLIS-3D Dataset: Data, Benchmarks and Analysis,» em *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2021, pp. 1110–1116. DOI: 10.1109/ICRA48506.2021.9561251.

[22] K. Huang, B. Shi, X. Li, X. Li, S. Huang e Y. Li, *Multi-Modal Sensor Fusion for Auto Driving Perception: A Survey*, https://arxiv.org/abs/2202.02703, arXiv:2202.02703, 2022.

[23] X. Yue, Y. Zhang e M. He, «LiDAR-based SLAM for robotic mapping: state of the art and new frontiers,» *Industrial Robot*, vol. 51, n.º 2, pp. 196–205, 2024. DOI: 10.1108/IR-09-2023-0225.

[24] W. Xu, Y. Cai, D. He, J. Lin e F. Zhang, «FAST-LIO2: Fast Direct LiDAR-Inertial Odometry,» *IEEE Transactions on Robotics*, vol. 38, n.º 4, pp. 2053–2073, 2022. DOI: 10.1109/TRO.2022.3141876.

[25] T. Shan, B. Englot, D. Meyers, W. Wang, C. Ratti e D. Rus, «LIO-SAM: Tightly-coupled Lidar Inertial Odometry via Smoothing and Mapping,» em *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, 2020, pp. 5135–5142. DOI: 10.1109/IROS45743.2020.9341176.

[26] C. Heupel, P. Wolf e K. Berns, «Tree-SLAM: Localization and Mapping in Dense Forest Environments for Autonomous Vehicles,» em *Commercial Vehicle Technology 2024*, 2024. DOI: 10.1007/978-3-658-45699-3_20.

[27] I. Vizzo, T. Guadagnino, B. Mersch, L. Wiesmann, J. Behley e C. Stachniss, *KISS-ICP: In Defense of Point-to-Point ICP – Simple, Accurate, and Robust Registration If Done the Right Way*, https://arxiv.org/abs/2209.15397, arXiv:2209.15397, publicado em IEEE Robotics and Automation Letters, 2023.

[28] T. Shan e B. Englot, «LeGO-LOAM: Lightweight and Ground-Optimized Lidar Odometry and Mapping on Variable Terrain,» em *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, 2018, pp. 4758–4765. DOI: 10.1109/IROS.2018.8594299.

[29] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. J. Leonard e F. Dellaert, «iSAM2: Incremental smoothing and mapping using the Bayes tree,» *The International Journal of Robotics Research*, vol. 31, n.º 2, pp. 216–235, 2012. DOI: 10.1177/0278364911430419.

[30] M. Jung, N. Chebrolu, L. C. de Lima, H. Oh, M. Fallon e A. Kim, *TreeLoc: 6-DoF LiDAR Global Localization in Forests via Inter-Tree Geometric Matching*, https://arxiv.org/abs/2602.01501, arXiv:2602.01501, aceite para ICRA 2026, 2026.

[31] M. Jung, D. Lee, N. Chebrolu, H. Oh, M. Fallon e A. Kim, *TreeLoc++: Robust 6-DoF LiDAR Localization in Forests with a Compact Digital Forest Inventory*, https://arxiv.org/abs/2603.03695, arXiv:2603.03695, 2026.

[32] L. C. de Lima et al., *Online 6DoF Global Localisation in Forests using Semantically-Guided Re-Localisation and Cross-View Factor-Graph Optimisation*, https://arxiv.org/abs/2409.16680, arXiv:2409.16680, submetido ao IROS 2025, 2024.

[33] H. P. Moravec e A. Elfes, «High Resolution Maps from Wide Angle Sonar,» em *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 1985, pp. 116–121. DOI: 10.1109/ROBOT.1985.1087316.

[34] A. Elfes, «Using Occupancy Grids for Mobile Robot Perception and Navigation,» *Computer*, vol. 22, n.º 6, pp. 46–57, 1989. DOI: 10.1109/2.30720.

[35] M. Benrabah, C. Orou Mousse, E. Randriamiarintsoa, R. Chapuis e R. Aufrère, «A Review on Traversability Risk Assessments for Autonomous Ground Vehicles: Methods and Metrics,» *Sensors*, vol. 24, n.º 6, p. 1909, 2024. DOI: 10.3390/s24061909.

[36] J. Laconte, A. Kasmi, F. Pomerleau, R. Chapuis e L. Malaterre, «A Novel Occupancy Mapping Framework for Risk-Aware Path Planning in Unstructured Environments,» *Sensors*, vol. 21, n.º 22, p. 7562, 2021. DOI: 10.3390/s21227562.

[37] D. D. Fan, S. Dey, A.-a. Agha-mohammadi e E. A. Theodorou, «Learning Risk-Aware Costmaps for Traversability in Challenging Environments,» *IEEE Robotics and Automation Letters*, vol. 7, n.º 1, pp. 279–286, 2022. DOI: 10.1109/LRA.2021.3125047.

[38] S.-W. Yoo, E. I. Son e S.-W. Seo, «Traversability-Aware Adaptive Optimization for Path Planning and Control in Mountainous Terrain,» *IEEE Robotics and Automation Letters*, vol. 9, n.º 6, pp. 5078–5085, 2024. DOI: 10.1109/LRA.2024.3387042.

[39] P. Papadakis, «Terrain Traversability Analysis Methods for Unmanned Ground Vehicles: A Survey,» *Engineering Applications of Artificial Intelligence*, vol. 26, n.º 4, pp. 1373–1385, 2013. DOI: 10.1016/j.engappai.2013.01.006.

[40] F. Ruetz, N. Lawrance, E. Hernández, P. Borges e T. Peynot, «ForestTrav: 3D LiDAR-only Forest Traversability Estimation for Autonomous Ground Vehicles,» *IEEE Access*, vol. 12, 2024. DOI: 10.1109/ACCESS.2024.3373004.

[41] Q. Zhu et al., *Learning-based Traversability Costmap for Autonomous Off-road Navigation*, https://arxiv.org/abs/2406.08187, arXiv:2406.08187, 2024.

[42] S. Triest, M. Sivaprakasam, S. Aich, D. Fan, W. Wang e S. Scherer, «Velociraptor: Leveraging Visual Foundation Models for Label-Free, Risk-Aware Off-Road Navigation,» em *Proc. 8th Conf. on Robot Learning (CoRL)*, 2024. acedido em 4 de jul. de 2026. URL: https://proceedings.mlr.press/v270/triest25a.html.

[43] P. Fankhauser, M. Bloesch e M. Hutter, «Probabilistic Terrain Mapping for Mobile Robots with Uncertain Localization,» *IEEE Robotics and Automation Letters*, vol. 3, n.º 4, pp. 3019–3026, 2018. DOI: 10.1109/LRA.2018.2849506.

[44] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss e W. Burgard, «OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees,» *Autonomous Robots*, vol. 34, n.º 3, pp. 189–206, 2013. DOI: 10.1007/s10514-012-9321-0.

[45] H. Oleynikova, Z. Taylor, M. Fehr, R. Siegwart e J. Nieto, «Voxblox: Incremental 3D Euclidean Signed Distance Fields for On-Board MAV Planning,» em *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, 2017. DOI: 10.1109/IROS.2017.8202315.

[46] E. W. Dijkstra, «A Note on Two Problems in Connexion with Graphs,» *Numerische Mathematik*, vol. 1, n.º 1, pp. 269–271, 1959. DOI: 10.1007/BF01386390.

[47] P. E. Hart, N. J. Nilsson e B. Raphael, «A Formal Basis for the Heuristic Determination of Minimum Cost Paths,» *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, n.º 2, pp. 100–107, 1968. DOI: 10.1109/TSSC.1968.300136.

[48] D. Dolgov, S. Thrun, M. Montemerlo e J. Diebel, «Practical Search Techniques in Path Planning for Autonomous Driving,» em *Proc. First Int. Symp. Search Techniques in Artificial Intelligence and Robotics (STAIR-08), AAAI Workshop*, 2008. acedido em 4 de jul. de 2026. URL: https://aaai.org/papers/ws08-10-006-practical-search-techniques-in-path-planning-for-autonomous-driving/.

[49] S. M. LaValle, «Rapidly-exploring Random Trees: A New Tool for Path Planning,» Computer Science Department, Iowa State University, rel. téc. TR 98-11, 1998. acedido em 6 de jul. de 2026. URL: https://lavalle.pl/rrtpubs.html.

[50] L. E. Kavraki, P. Svestka, J.-C. Latombe e M. H. Overmars, «Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces,» *IEEE Transactions on Robotics and Automation*, vol. 12, n.º 4, pp. 566–580, 1996. DOI: 10.1109/70.508439.

[51] S. Karaman e E. Frazzoli, «Sampling-based Algorithms for Optimal Motion Planning,» *The International Journal of Robotics Research*, vol. 30, n.º 7, pp. 846–894, 2011. DOI: 10.1177/0278364911406761.

[52] S. Koenig, M. Likhachev e D. Furcy, «Lifelong Planning A*,» *Artificial Intelligence*, vol. 155, n.º 1-2, pp. 93–146, 2004. DOI: 10.1016/j.artint.2003.12.001.

[53] S. Koenig e M. Likhachev, «D* Lite,» em *Proc. 18th National Conf. on Artificial Intelligence (AAAI)*, 2002, pp. 476–483. acedido em 4 de jul. de 2026. URL: https://cdn.aaai.org/AAAI/2002/AAAI02-072.pdf.

[54] D. Fox, W. Burgard e S. Thrun, «The Dynamic Window Approach to Collision Avoidance,» *IEEE Robotics & Automation Magazine*, vol. 4, n.º 1, pp. 23–33, 1997. DOI: 10.1109/100.580977.

[55] O. Khatib, «Real-Time Obstacle Avoidance for Manipulators and Mobile Robots,» *The International Journal of Robotics Research*, vol. 5, n.º 1, pp. 90–98, 1986. DOI: 10.1177/027836498600500106.

[56] G. Williams, A. Aldrich e E. A. Theodorou, *Model Predictive Path Integral Control using Covariance Variable Importance Sampling*, https://arxiv.org/abs/1509.01149, arXiv:1509.01149, 2015.

[57] D. Q. Mayne, «Model Predictive Control: Recent Developments and Future Promise,» *Automatica*, vol. 50, n.º 12, pp. 2967–2986, 2014. DOI: 10.1016/j.automatica.2014.10.128.

[58] R. C. Coulter, «Implementation of the Pure Pursuit Path Tracking Algorithm,» Robotics Institute, Carnegie Mellon University, rel. téc. CMU-RI-TR-92-01, 1992. acedido em 4 de jul. de 2026. URL: https://www.ri.cmu.edu/publications/implementation-of-the-pure-pursuit-path-tracking-algorithm/.

[59] S. Macenski, S. Singh, F. Martín e J. Ginés, «Regulated Pure Pursuit for Robot Path Tracking,» *Autonomous Robots*, vol. 47, n.º 6, pp. 685–694, 2023. DOI: 10.1007/s10514-023-10097-6.

[60] Y. Kanayama, Y. Kimura, F. Miyazaki e T. Noguchi, «A Stable Tracking Control Method for an Autonomous Mobile Robot,» em *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 1990, pp. 384–389. DOI: 10.1109/ROBOT.1990.126006.

[61] C. J. Ostafew, A. P. Schoellig, T. D. Barfoot e J. Collier, «Learning-based Nonlinear Model Predictive Control to Improve Vision-based Mobile Robot Path Tracking,» *Journal of Field Robotics*, vol. 33, n.º 1, pp. 133–152, 2016. DOI: 10.1002/rob.21587.

[62] C. J. Ostafew, A. P. Schoellig e T. D. Barfoot, «Robust Constrained Learning-based NMPC Enabling Reliable Mobile Robot Path Tracking,» *The International Journal of Robotics Research*, vol. 35, n.º 13, pp. 1547–1563, 2016. DOI: 10.1177/0278364916645661.

[63] A. Trivedi, M. Zolotas, A. Abbas, S. Prajapati, S. Bazzi e T. Padır, *A Probabilistic Motion Model for Skid-Steer Wheeled Mobile Robot Navigation on Off-Road Terrains*, https://arxiv.org/abs/2402.18065, arXiv:2402.18065, 2024.

[64] A. Salagame, S. Pandya, I. Mandralis, E. Sihite, A. Ramezani e M. Gharib, «NMPC-based Unified Posture Manipulation and Thrust Vectoring for Fault Recovery,» *IEEE Control Systems Letters*, vol. 9, pp. 2012–2017, 2025. DOI: 10.1109/LCSYS.2025.3589412.

[65] T. Lee, M. Leok e N. H. McClamroch, «Geometric Tracking Control of a Quadrotor UAV on SE(3),» em *Proc. 49th IEEE Conf. Decision and Control (CDC)*, 2010, pp. 5420–5425. DOI: 10.1109/CDC.2010.5717652.

[66] R. Mahony, T. Hamel e J.-M. Pflimlin, «Nonlinear Complementary Filters on the Special Orthogonal Group,» *IEEE Transactions on Automatic Control*, vol. 53, n.º 5, pp. 1203–1217, 2008. DOI: 10.1109/TAC.2008.923738.

[67] S. O. H. Madgwick, A. J. L. Harrison e R. Vaidyanathan, «Estimation of IMU and MARG Orientation Using a Gradient Descent Algorithm,» em *Proc. IEEE Int. Conf. Rehabilitation Robotics (ICORR)*, 2011, pp. 1–7. DOI: 10.1109/ICORR.2011.5975346.

[68] S. A. Ludwig e K. D. Burnham, «Comparison of Euler Estimate Using Extended Kalman Filter, Madgwick and Mahony on Quadcopter Flight Data,» em *Proc. Int. Conf. Unmanned Aircraft Systems (ICUAS)*, 2018, pp. 1236–1241. DOI: 10.1109/ICUAS.2018.8453465.

[69] N. Aliane, «A Survey of Open-Source UAV Autopilots,» *Electronics*, vol. 13, n.º 23, p. 4785, 2024. DOI: 10.3390/electronics13234785.

[70] V. Wiberg et al., *Sim-to-real Transfer of Active Suspension Control Using Deep Reinforcement Learning*, https://arxiv.org/abs/2306.11171, arXiv:2306.11171, 2023.

[71] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun e M. Hutter, «Learning Quadrupedal Locomotion over Challenging Terrain,» *Science Robotics*, vol. 5, n.º 47, eabc5986, 2020. DOI: 10.1126/scirobotics.abc5986.

[72] T. Miki, J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun e M. Hutter, «Learning Robust Perceptive Locomotion for Quadrupedal Robots in the Wild,» *Science Robotics*, vol. 7, n.º 62, eabk2822, 2022. DOI: 10.1126/scirobotics.abk2822.

[73] K. Perlin, «An Image Synthesizer,» *ACM SIGGRAPH Computer Graphics*, vol. 19, n.º 3, pp. 287–296, 1985. DOI: 10.1145/325165.325247.

[74] E. Galin et al., «A Review of Digital Terrain Modeling,» *Computer Graphics Forum*, vol. 38, n.º 2, pp. 553–577, 2019. DOI: 10.1111/cgf.13657.

[75] C. Newlands e K.-P. Zauner, *Procedural Generation and Rendering of Realistic, Navigable Forest Environments: An Open-Source Tool*, https://arxiv.org/abs/2208.01471, arXiv:2208.01471, 2022.

[76] P. Ramón, E. Velázquez, A. Escudero e M. de la Cruz, «Environmental Heterogeneity Blurs the Signature of Dispersal Syndromes on Spatial Patterns of Woody Species in a Moist Tropical Forest,» *PLOS ONE*, vol. 13, n.º 2, e0192341, 2018. DOI: 10.1371/journal.pone.0192341.

[77] M.-A. Ngo Bieng, C. Ginisty e F. Goreaud, «Point Process Models for Mixed Sessile Forest Stands,» *Annals of Forest Science*, vol. 68, n.º 2, pp. 267–274, 2011. DOI: 10.1007/s13595-011-0033-y.

[78] J. Zhu, D. Lu e W. Zhang, «Effects of Gaps on Regeneration of Woody Plants: A Meta-Analysis,» *Journal of Forestry Research*, vol. 25, pp. 501–510, 2014. DOI: 10.1007/s11676-014-0489-3.

[79] H. Oleynikova, M. Burri, Z. Taylor, J. Nieto, R. Siegwart e E. Galceran, «Continuous-Time Trajectory Optimization for Online UAV Replanning,» em *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, 2016. acedido em 15 de jul. de 2026. URL: http://helenol.github.io/publications/iros_2016_replanning.pdf.