Universidade do Minho
Escola de Engenharia

Daniel Fernando Costa Ferreira

Sistema de Visão Robótico Baseado no Sensor Kinect para Orientação no Espaço e Contorno de Obstáculos

outubro de 2013

Sistema de Visão Robótico Baseado no Sensor Kinect para Orientação no Espaço e Contorno de Obstáculos
Daniel Fernando Costa Ferreira
UMinho | 2013

Universidade do Minho
Escola de Engenharia

Daniel Fernando Costa Ferreira

Sistema de Visão Robótico Baseado no Sensor Kinect para Orientação no Espaço e Contorno de Obstáculos

Dissertação de Mestrado
Mestrado em Engenharia Eletrónica Industrial e Computadores
Área de Especiliazação em Robótica

Trabalho efetuado sob a orientação do
Professor Doutor Agostinho Gil Teixeira Lopes

outubro de 2013

DECLARAÇÃO

Nome: Daniel Fernando Costa Ferreira

Endereço eletrónico: a52683@alunos.uminho.pt

Telefone: 965275140

Número do Bilhete de Identidade: 13574990

Título dissertação: Sistema de Visão Robótico Baseado no Sensor Kinect para Orientação no Espaço e Contorno de Obstáculos

Ano de conclusão: 2013

Orientador: Professor Doutor Agostinho Gil Teixeira Lopes

Designação do Mestrado: Mestrado em Engenharia Eletrónica Industrial e Computadores

Ciclo de Estudos Integrados Conducentes ao Grau de Mestre em Engenharia

Área de Especialização: Robótica

Escola: Universidade do Minho – Azurém

Departamento: Departamento de Eletrónica Industrial

Universidade do Minho, ___/___/_____

Assinatura:_____

Agradecimentos

Ao meu orientador, Professor Doutor Gil Lopes bem como ao Professor Doutor Fernando Ribeiro, pelo apoio, acompanhamento, e pela confiança que sempre depositaram em mim, e todos os colegas de laboratório para que pudéssemos ter o melhor ambiente e rendimento. Agradeço-lhes a disponibilidade e sugestões valiosas que sempre me conduziram no melhor caminho.

Ao Laboratório de Automação e Robótica do Departamento de Eletrônica Industrial da Universidade do Minho, pelo apoio e recursos facultados.

Ao Aníbal, José e Ricardo, amigos e colegas de laboratório pelo ambiente e momentos de descontração que prestaram.

A todos os meus amigos e familiares, por se preocuparem comigo e pelas palavras encorajadoras, em especial aos meus amigos Vergílio Pereira e Luís Aguiar.

À minha namorada, Sara, pelo apoio e carinho em todos os momentos.

Por último, um obrigado muito especial aos meus pais e irmã, pelo amor, apoio, compreensão incondicionais e por me terem sempre proporcionado o melhor ao seu alcance.

Resumo

A robótica de serviços é uma área em forte crescimento, cada vez com mais aplicações, nomeadamente na substituição de algumas das tarefas domésticas mais repetitivas e, cada vez mais, com o envelhecimento da população, no apoio a pessoas de locomoção reduzida em inúmeras tarefas.

Um dos maiores problemas encontrados nesta área é a navegação do robô num ambiente não determinístico, onde mapas *a priori* desenvolvidos de um determinado espaço são dinamicamente alterados pelos utilizadores do mesmo, levando a que o robô seja forçado a desenvolver um novo mapa local enquanto, simultaneamente, reconhece a sua posição atual a partir do mapa construído, problema conhecido por SLAM (*Simultaneous Localization And Mapping*). Apesar de teórica e concetualmente este se considerar um problema resolvido, questões substanciais mantêm-se na realização prática de soluções mais genéricas [1]. Muitos dos algoritmos para mapeamento e localização simultânea existentes são baseados em sensores de *scanner* a *laser,* pela sua alta precisão, contudo são bastante dispendiosos. Com o Microsoft Kinect, sensor originalmente desenvolvido para uma consola de jogos, existe agora uma alternativa de baixo custo, permitindo a criação de mapas 3D, a cores do ambiente, e o seu uso para localização [2].

Posto isto, este projeto de dissertação teve como primeiro grande foco, o desenvolvimento de um sistema de visão por computador para um robô de serviços já existente (robô Mary), utilizando o sensor Microsoft Kinect, para construir o mapa do ambiente que o rodeia enquanto ao mesmo tempo reconhece a sua posição no mapa construído, resolvendo o problema SLAM.

A segunda contribuição deste projeto de dissertação consistiu na localização do robô a partir de um mapa previamente construído.

O último grande objetivo deste trabalho passou pela movimentação do robô para um determinado local, quando solicitado, criando rotas virtuais livres de obstáculos, tendo conhecimento do mapa do ambiente onde se encontra.

Palavras-chave: Kinect, SLAM, Planeamento de Trajetórias, Robô, Localização.

Abstract

The service robotics is an area in frank expansion, with more and more applications, including the replacement of some of the more repetitive tasks and, increasingly, with the aging population, supporting people of reduced mobility in numerous tasks.

One of the major problems encountered in this area is robot navigation in non-deterministic environment, which previously developed maps in a given space are dynamically changed by users, meaning that the robot is forced to incrementally build a consistent map of the environment while simultaneously determines its location within this map, problem known as SLAM (Simultaneous Localization And Mapping). At a theoretical and conceptual level, SLAM can now be considered a solved problem. However, substantial issues remain in practically realizing more general SLAM solutions [1]. Many of the existing algorithms for simultaneous localization and mapping are based on laser scanners as sensor because their distance measurements are very precise, but they are also quite expensive. With the Microsoft Kinect sensor, originally developed for a console game, there is now a low-cost alternative, allowing the creation of colored 3D maps of the environment, and its use for localization [2].

That said, the first major focus of this dissertation project, consisted in the development of a computer vision system for an existing service robot (Robot MARy), using the Microsoft Kinect sensor to incrementally build the map of the surroundings, while simultaneously recognizes his position, solving the SLAM problem.

The second contribution of this project consisted on the robot's localization from a previously constructed map, and also with the possibility to plan obstacle-free paths to a given location when prompted.

Keywords: Kinect, SLAM, Path Planning, Robot, Localization.

Conteúdo

Abreviaturas

RGB	Espaço de cores Red, Green e Blue
RGB-D	Espaço de cores Red, Green e Blue mais profundidade Depth
TOF	Time Of Flight
LRF	Laser Range Finder
NIR	Near Infrared
DoG	Difference of Gaussian
LoG	Laplacian of Gaussian
ICP	Iterative Closest Point
PCL	Point Cloud Library
RANSAC	RANdom Sample Consensus
SIFT	Scalar Invariant Feature Transform
SURF	Speeded Up Robust Feature
SLAM	Simultaneous Localization And Mapping
VSLAM	Visual Simultaneous Localization And Mapping
OMPL	Open Motion Planning Library
RRT	Rapidly Exploring Random Tree
CAD	Computer Aided Design
API	Application Programming Interface
IMU	Inertial Measurement Unit

Capítulo 1
Introdução

1.1 Motivação e Enquadramento

Durante os últimos 25 anos os robôs têm-se difundido em diferentes áreas e processos industriais, nomeadamente na medicina, em aplicações militares e segurança pública. Contudo só agora começam a ser introduzidos em ambiente doméstico como produtos de consumo, desde robôs aspiradores em milhões de casas até robôs rececionistas em gabinetes japoneses. Da mesma maneira que hoje encontramos computadores no dia-a-dia, as pessoas podem em breve ter pouca escolha na questão de interagir com robôs [3]. Existe um desejo de aperfeiçoar estes robôs dotando-os de novas funções e melhorando o seu funcionamento.

A equipa Minho@home, no âmbito da competição anual a nível mundial Robocup [4], que em 2008 criou a liga RoboCup@home para robôs que desenvolvem as mais diversas tarefas domésticas, desenvolveu o robô MARy (Minho Autonomous Robot) dotado de uma base omnidirecional [5] e um braço robótico [6] para participar no evento.

A necessidade do robô de se localizar e orientar e o facto de grande parte dos sistemas de localização existentes apenas funcionar em ambiente exterior, torna interessante a implementação de técnicas que o permitam fazer no interior.

Existe também uma motivação pessoal de aumentar as capacidades de um robô existente tornando-o mais parecido com um ser humano, dotando-o de um sistema de visão para localização e orientação no espaço, permitindo ainda que este se desvie dos obstáculos planeando previamente uma rota, utilizando o sensor Kinect.

1.2 Objetivos

Este projeto de dissertação tem como primeiro grande objetivo o desenvolvimento de um sistema de visão por computador para um robô de serviços já existente (robô Mary), utilizando o sensor Microsoft Kinect, que faça com que este construa um mapa enquanto, ao mesmo tempo, se localiza.

O segundo objetivo consiste na movimentação do robô para um determinado local, quando solicitado, sem colidir com os obstáculos.

Capítulo 2

Estado da Arte nos Robôs de Serviço

São apresentados de seguida alguns exemplos de robôs de Serviços existentes que utilizam, entre outras, técnicas aplicadas neste projeto de dissertação, dando ênfase às mesmas nas suas descrições. Os robôs apresentados obtiveram boas classificações na liga *Robocup@home* do evento *Robocup* [4], edições de 2011 e 2012. A competição deste evento consiste em testes com um procedimento predefinido, demonstrações abertas e um desafio técnico. Os testes regulares cobrem manipulação móvel e interação robô-humano. Nestes testes, os robôs devem comportar-se de uma maneira autónoma e finalizar as tarefas num determinado período de tempo. Na demonstração aberta, as equipas podem escolher as suas próprias tarefas de modo a demonstrar resultados da sua própria investigação. Finalmente, o desafio técnico é introduzido para testar o desempenho do robô num aspeto técnico específico. É permitido às equipas, durante os dois primeiros dias de competição, a aquisição do mapa da arena de competição que se assemelha a um apartamento, e o treino de reconhecimento de um conjunto de objetos, usados como objetos conhecidos com nomes ao longo dos testes de reconhecimento e manipulação. Durante a competição, a arena é submetida desde pequenas a grandes mudanças e contém também objetos desconhecidos.

2.1 Care-O-Bot 3

O robô *Care-O-Bot* foi desenvolvido pela equipa *b-it-bots* [7]. Incorpora um par de câmaras para sistema de visão estéreo, um sensor *Kinect*, um sensor do tipo *time-of-flight* (TOF), e ainda três sensores *Laser Range Finder* (LRF) para mapeamento.

Para se localizar baseia-se em dois princípios. Primeiro, a posição e orientação atual é estimada recorrendo à odometria. Contudo pequenos erros são inevitáveis e acumuláveis ao longo do tempo. Tendo em conta isto, informação de sensores LRF colocados à frente e atrás do robô são utilizados, de modo a detetar características significantes como paredes e postes. Estas características são comparadas com as suas posições de referência, que estão guardadas num mapa global do ambiente. Finalmente, a posição do robô é determinada em relação às mesmas.

Para planear a sua trajetória, é utilizado o mapa adquirido anteriormente como base. Este planeamento, tendo em consideração a geometria e cinemática do robô, otimiza o trajeto conformemente. A otimização do caminho, não só tem em consideração o mapa do ambiente em redor, mas também de dados sensoriais atuais, possibilitando a adaptação do trajeto a mudanças de parâmetros, ou seja, se um ponto no trajeto planeado é inacessível por causa de um obstáculo dinâmico, como uma pessoa ou peça de mobília, o trajeto é adaptado.

Figura 2.1: Care-O-Bot foi desenvolvido pela equipa b-it-bots[7].

2.2 KeJia

O robô *KeJia* desenvolvido pela equipa *Wright Eagle* [8], está equipado com um sensor *Mircosoft Kinect*, para perceção do ambiente em tempo real, uma câmara RGB de alta resolução e dois LRF. Para localização e navegação, é inicialmente criado um mapa 2D a partir dos dados recolhidos pelos sensores LRF através de uma viagem pelos quartos. É depois anotado manualmente no mesmo a aproximada localização e/ou área dos espaços, portas, mobília e outros objetos de interesse. Finalmente um mapa topológico é automaticamente gerado, sendo depois utilizado para um planeamento de trajetória global.

Com este mapa, são aplicadas técnicas probabilísticas bem como de correspondência para localização. Além disso é capaz de se desviar de obstáculos locais enquanto navega pelos quartos, e aplica estratégias de exploração para adquirir informação de ambientes desconhecidos.

Figura 2.2: Robô KeJia desenvolvido pela equipa Wright Eagle[8].

2.3 Dynamaid e Cosero

Os robôs *Dynamaid* e *Cosero,* desenvolvidos pela equipa *NimbRo@home* [9] , foram os vencedores do evento *Robocup@home*, edição de 2011 e 2012.

Estão equipados com múltiplos LRF 2D colocados no chão, por cima da base móvel, e no torso de modo a medir distâncias a objetos, pessoas ou obstáculos para propósitos de navegação. Os LRF colocados no torso podem rodar sobre os eixos longitudinal e transversal para evitar obstáculos no espaço 3D. Possui duas câmaras RGB, uma outra do tipo TOF e ainda um sensor *Microsoft Kinect* para reconhecer pessoas e objetos colocados em cima de uma mesa.

Durante os testes do evento, a arena de competição é assumida como estática. É adquirido um mapa 2D de grelhas de células de um ambiente desconhecido utilizando técnicas SLAM. São depois implementados métodos para localização e planeamento de trajetórias, assegurando uma condução livre de obstáculos ao longo do caminho planeado, ao incorporar os dados recebidos de todos os sensores de distância.

Figura 2.3: Robôs Cosero e Dynamaid da equipa NimbRo@home[10].

2.4 eR@sers

O sistema de visão utilizado pelos robôs da equipa eR@sers [11] é constituído por duas câmaras RGB, para capturar duas imagens em simultâneo, sendo depois utilizadas para processamento estéreo de modo a obter informação de profundidade. Tanto a informação de cor como a profundidade são utilizadas para reconhecimento de objetos e faces, bem como para aprendizagem e tarefas de reconhecimento.

Inclui ainda câmaras do tipo TOF *near-infrared* (NIR) para capturar informação 3D com precisão, e sensores de distância, LRF. Todas estas câmaras estão calibradas entre si de modo a que os pixéis correspondentes consigam ser facilmente encontrados.

Figura 2.4: Robô DiGORO da equipa eR@sers [11]

2.5 ToBi

O robô *ToBi* [12] possui um sensor de distância LRF, colocado a 30 centímetros do solo com um ângulo de 180 graus, que é capaz de medir objetos até 50 metros de profundidade e com uma precisão de 18 milímetros para distâncias inferiores a 18 metros. Para classificação de compartimentos, gestos e objetos 3D, *ToBi* está equipado com um sistema ótico para aquisição de imagens 3D em tempo real.

Utiliza a técnica SLAM, para criar um mapa caracterizado em termos de regiões ocupadas ou não ocupadas, formando uma grelha de células, conhecido por *Occupancy grid map*. Este, apenas contém obstáculos detetados pelo sensor LRF (colocado 30 centímetros acima do solo), como paredes e mobília. Adicionalmente sobre este mapa SLAM é introduzido um novo mapa de grelha de células que funde informação dada pelo detetor de cor, e um outro de superfícies horizontais (acima do nível do solo). Potencialmente, este mapa representa objetos, alvos de uma pesquisa, colocados numa mesa acima do chão.

Figura 2.5: ToBi com os seus componentes: câmara, sensores 3D, LRF, e ainda um braço robótico KATANA e microfone. [12]

Capítulo 3

Fundamentos Teóricos

Neste capítulo serão apresentadas em primeiro lugar técnicas e tecnologias que permitam a um robô localizar-se e orientar-se no espaço.

Do mesmo modo, serão apresentadas soluções existentes para um robô que necessite de traçar uma rota livre de obstáculos, conhecendo o mapa envolvente, desde a sua posição até ao destino, seguindo-se de uma apresentação do sensor Kinect bem como técnicas e métodos existentes usadas em Visão por Computador.

3.1 Localização e Orientação

As principais estratégias de localização de um robô podem ser definidas como: Localização Relativa, em que a posição e orientação são determinadas de forma incremental a partir de um ponto inicial utilizando sensores como acelerómetros, *encoders*, giroscópios, etc.; Localização Global, em que a posição do robô é determinada em relação a uma referência global, utilizando, por exemplo, balizas ou pontos de referência e Localização Probabilística, baseada na estimativa da localização do robô combinando os dados de medição e o conhecimento anterior do sistema. [13]

3.1.1 Odometria

Odometria é uma técnica de localização relativa que utiliza equações geométricas simples e dados de *encoders* que fornecem a velocidade angular das rodas de um robô para calcular a sua orientação e localização.

As grandes desvantagens desta técnica passam pelo facto das medidas sensoriais serem integradas causando um aumento do erro sem limites ao longo do tempo e distância e pelo facto da revolução das rodas não poder ser sempre traduzida em deslocamento linear relativamente ao chão, por exemplo quando este estiver escorregadio. [13]

Esta técnica não permite conhecer a posição de um robô em relação a um referencial global.

3.1.2 Técnicas de Localização Global

A técnica mais popular de localização global é o *GPS* (*Global Positioning System*), baseada em sinais por satélites para determinar a posição absoluta (longitude, latitude e altitude) de um objeto na Terra. No âmbito desta tese apresenta como grande desvantagem o facto do sinal GPS poder ser perdido em ambientes fechados. Apenas é útil no exterior.

Em relação às outras técnicas existentes de localização global, as desvantagens passam pela necessidade da implementação dispendiosa de pontos de referência no ambiente onde o robô vai navegar e do restringimento da navegação à área onde foram colocados. [13]

3.1.3 SLAM (Simultaneous Localization and Mapping)

A resolução ao problema SLAM (simultâneo mapeamento e localização), permite que um robô construa incrementalmente um mapa do ambiente que o rodeia enquanto simultaneamente determina a sua localização, a partir do conhecimento do mapa criado anteriormente [1]. Um dos inconvenientes de SLAM é o associado esforço computacional [13].

Considerando que num instante k,

- x_k é o estado do robô que descreve a sua posição e orientação;
- z_k é uma observação (por exemplo capturas de cor e profundidade obtidas a partir do sensor *Kinect*);
- u_k o controlo que representa mudanças de estado.

O problema SLAM, na sua forma probabilística, requer que a distribuição de probabilidade, condicionada por todas as observações passadas $z_{0:k}$ e todos os controlos passados $u_{0:k}$,

$$p(x_k|z_{0:k}, u_{0:k}) \tag{3.1}$$

seja calculada para todos os tempos k [1].

No caso do SLAM visual, VSLAM com o sensor *Kinect*, cada observação z_k é composta por uma captura RGB e profundidade

3.2 Planeamento de Trajetória

Planeamento consiste em encontrar uma sequência de ações que transformem um dado estado inicial num estado final desejado. No planeamento de trajetória, os estados são posições, e transições entre estados representam ações que o robô pode tomar, cada uma delas com um custo associado.

Um caminho é considerado ótimo se a soma do custo das suas transições é mínima ao longo de todos os caminhos possíveis desde a posição (estado) inicial à posição final. [14] Por exemplo, se existir um trajeto ótimo (o mais curto) desde uma dada posição inicial a uma dada posição final, e se existir uma posição intermédia no caminho, o segmento desde a posição inicial à intermédia é o caminho ótimo entre a posição inicial e intermédia. Do mesmo modo, o segmento desde a posição intermédia à final, é também o caminho ótimo entre estas duas posições (Figura 3.1).

Similarmente, um algoritmo de planeamento de trajetória é considerado completo se conseguir sempre encontrar um trajeto, se existir, em tempo finito. [14]

O planeamento de trajetória permite que um caminho seja previamente planeado ao invés do robô se movimentar em direção ao alvo sem planos, e esperar até ao último momento para perceber que existe um problema.

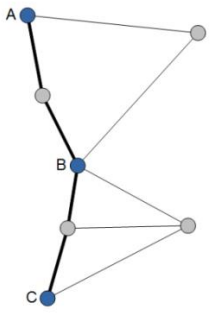

Figura 3.1: Exemplo de caminho ótimo. Se um caminho é ótimo entre A e C, é também ótimo entre A e B e B e C, por exemplo.

3.2.1 A*

A-star é um algoritmo de planeamento de trajetórias discreto e determinístico, que deve encontrar o melhor caminho entre dois pontos no plano/espaço envolvendo a exploração de grafos, representados por nós ou vértices, e arestas. É ótimo e completo, é também o mais comum e eficiente algoritmo para problemas que exijam um planeamento de percurso com o caminho mais curto. [15]

O algoritmo A* utiliza a mínima distância entre dois dados nós, obtida por meio de uma função heurística. Formalmente esta função devolve a medida garantidamente menor ou igual à menor distância entre duas posições dadas. Na maioria dos casos a distância euclidiana é usada pela simplicidade e facilidade de cálculo, mas a função heurística pode ser qualquer uma que satisfaça os critérios anteriores.

Utiliza duas estratégias principais para aumentar a sua eficiência. Primeiro, utiliza a distância do trajeto desde a fonte mais a mínima distância heurística até ao destino para determinar a ordem de pesquisa de potenciais nós, dada por,

$$f(n) = g(n) + h(n) \tag{3.2}$$

- $g(n)$: custo (distância) do trajeto desde a fonte ao nó atual
- $h(n)$: mínima distância heurística desde o nó atual até ao destino

Portanto, $f(n)$ vai ser a estimativa de custo da melhor solução que passa por *n*.

Esta estratégia funciona pois este valor representa a menor distância possível através de um dado nó, e geralmente serve também como uma boa estimativa da distancia total do percurso. Em segundo lugar, utiliza a heurística para podar a árvore de busca de galhos desnecessários, comparando a menor distância possível de um trajeto (novamente, a distância do trajeto desde a fonte mais a mínima distância heurística ate ao destino) através de um dado nó, com comprimentos de trajetos calculados anteriormente.

O algoritmo A* fica aquém do desejado quando o caminho por si só não é suficiente e são necessárias entradas de controlo que levem o robô de nó para nó, por outras palavras, apenas resolve parte do problema. Para navegar o trajeto encontrado, um segundo algoritmo é necessário para examinar o caminho, e derivar os comandos apropriados para fornecer aos

atuadores do robô, não havendo garantias de que um conjunto apropriado de entradas de controlo exista.

3.2.2 RRT

Árvore Aleatória de Exploração Rápida, ou *Rapidly Exploring Random Tree* (RRT) [16] é uma estrutura de dados probabilística e algoritmo desenvolvido para uma ampla gama de problemas de planeamento de trajeto. Enquanto partilham muitas das propriedades benéficas de técnicas de planeamento probabilísticas existentes, são especificamente desenvolvidas para lidar com restrições não-holonómicas. Na robótica um sistema é considerado não-holonómico se os graus de liberdade controláveis são menores que o total número de graus de liberdade. Um carro é um tipo de veículo não-holonómico, consegue alcançar uma qualquer posição e orientação no espaço 2D, por isso precisa de três graus de liberdade para descrever a sua pose, mas apenas tem dois graus de liberdade controláveis, movimento para a frente e o ângulo de rotação da direção.

Uma RRT é expandida iterativamente, aplicando entradas de controlo que conduzem o sistema suavemente até pontos escolhidos aleatoriamente.

Para algoritmos de planeamento estáticos, como o anterior descrito, *A-star*, um espaço de configuração que representa a orientação e posição de um robô é suficiente. No entanto, para facilitar a inclusão de restrições diferenciais no processo de planear um trajeto, como velocidade máxima ou raio mínimo de viragem, o espaço de configuração é estendido, ao incluir a derivada em função do tempo de cada uma das dimensões do espaço de configuração. O novo espaço é conhecido como espaço de estados, X, utilizado para indicar uma maior generalidade da que é usualmente considerada no planeamento de trajetórias. Por exemplo, para um carro, o espaço de configuração pode ser dado por (x, y, θ) enquanto que o seu espaço de estados é dado por, $X = (x, y, \theta, \dot{x}, \dot{y}, \dot{\theta})$, permitindo planear com restrições diferenciais.

Assumindo que uma região fixa de obstáculos, X_{obs} deve ser evitada, uma Árvore Aleatória de exploração Rápida (RRT), é construída de maneira a que todos os seus vértices são estados pertencentes a um espaço livre X_{free}, complemento de X_{obs}, e cada aresta, da RRT corresponderá a um trajeto que se encontra totalmente em X_{free}.

Para um dado estado inicial x_{init}, uma RRT, G, com K vértices, é construída como demonstram os algoritmos seguintes:

Algoritmo 1 RRT

Input: estado inicial, x_{init}, número máximo de vértices K, intervalo fixo de tempo T

```
1   G.Init(x_init)
2   for i = 1 to K do
3       x_rand ← RandomState();
4       Extend(G, x_rand);
5   end for
6   return G
```

Algoritmo 2 RRT_EXTEND (G, x)

```
1   x_nearest ← NearestNeighbor(G, x );
2   u ← SelectInput(x_nearest, x);
3   x_new ← NewState(x_nearest, u, T );
4   if ObstacleFree(x_nearest, x_new ) then
5       G.AddVertex(x_new );
6       G.AddEdge(x_nearest, x_new, u);
```

O primeiro algoritmo, Algoritmo 1 descreve o corpo da RRT. O primeiro vértice da árvore, G, é o estado inicial $x_{init} \in X_{free}$. Em cada iteração, *RandomState* seleciona um estado aleatoriamente, x_{rand} a partir do espaço de estados, X (Linha 3). A rotina *Extend*, tenta que a árvore se expanda em direção a x_{rand} (Linha 4).

A expanção da RRT está detalhada no Algoritmo 2. Primeiro, é calculado o vértice da árvore, $x_{nearest}$, mais próximo de x_{rand}, correndo todos os vértices da árvore (Linha 1). Em seguida são selecionadas as entradas de controlo, u, que conduzem $x_{nearest}$ mais próximo de x_{rand} (Linha 2), sendo depois integradas durante um intervalo de tempo T, alcançando-se uma nova posição, x_{new}, no espaço de estados. As linhas 2 e 3 devem ser ignoradas caso não sejam consideradas restrições diferenciais, e a nova posição x_{new} deve ser adicionada a uma determinada distância v, desde $x_{nearest}$ em direção a x_{rand}. Se o trajeto até este novo nó for livre de obstáculos (Linha 4), então o novo nó, x_{new}, é adicionado à árvore (Linha 5) e uma aresta desde $x_{nearest}$ até x_{new} é também adicionada a G, juntamente com as entradas de controlo associadas (Linha 6).

O algoritmo deve terminar, assim que a árvore estiver suficientemente perto do estado final desejado, x_{goal}. Para executar o trajeto planeado deve-se recuar ao longo da árvore para identificar as arestas que conduzem x_{init} a x_{goal}. Deve-se depois dirigir o robô até ao destino utilizando as entradas de controlo guardadas juntamente com as arestas identificadas.

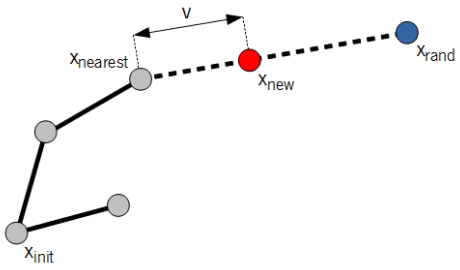

Figura 3.2: Extensão da RRT caso não sejam consideradas restrições diferenciais.

A Figura 3.2 exemplifica a extensão da RRT caso não sejam consideradas restrições diferenciais. Primeiro é selecionado o vértice, $x_{nearest}$, que está mais próximo de x_{rand}. O vértice x_{new}, deve ser adicionado a uma determinada distância v, desde $x_{nearest}$ em direção a x_{rand}. Se o trajeto até este novo nó for livre de obstáculos, x_{new}, é adicionado à árvore bem como a aresta entre $x_{nearest}$ e x_{new}.

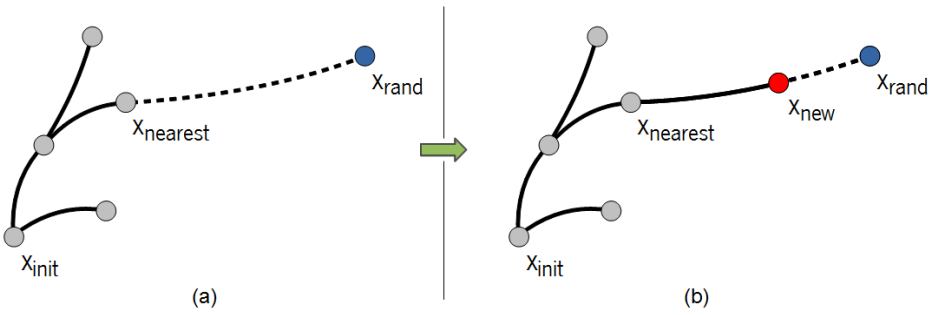

Figura 3.3: Extensão da RRT, caso sejam consideradas restrições diferenciais.

A Figura 3.3 representa a extensão da RRT (Algoritmo 2) considerando restrições diferenciais. Primeiro é selecionado o vértice, $x_{nearest}$, que está mais próximo de x_{rand}. Em seguida, são selecionadas as entradas de controlo, que conduzem $x_{nearest}$ até, ou mais próximo, de x_{rand} (Figura 3.3a). Por fim, as entradas de controlo são integradas durante um

intervalo de fixo de tempo *T*, e, se o trajeto até este novo nó for livre de obstáculos, x_{new}, é adicionado à árvore bem como a aresta que o conecta a $x_{nearest}$ (Figura 3.3b).

3.2.3 RRT*

Embora o algoritmo RRT seja conhecido por encontrar um trajeto rapidamente, não possui garantias na qualidade da sua solução. RRT* (*Optimal RRT)*, é um algoritmo de planeamento de trajetória introduzido por Karaman[17] para resolver esta questão.

Tira vantagem da rápida expansão RRT, introduzindo custo do trajeto e otimização. É probabilisticamente completo, isto é, a probabilidade que irá encontrar uma solução aproxima-se de '1' à medida que o tempo computacional tende para infinito. É assintoticamente ótimo, por outras palavras, o custo da solução retornada converge quase garantidamente para uma solução ótima e o seu tempo processamento é garantido ser um fator constante do tempo de processamento do algoritmo RRT.

A Figura 3.4 demonstra o algoritmo RRT* se não forem consideradas restrições dinâmicas. Primeiro é selecionado um ponto aleatoriamente, x_{rand}, e é criado um ponto x_{new}, que está a uma distância, *v*, desde o ponto mais próximo, $x_{nearest}$ até x_{rand} (Figura 3.2). É determinado o conjunto de vértices vizinhos, x_{near}, dentro de um determinado raio de x_{new} (Figura 3.4a). De todos os vértices vizinhos, é selecionado como pai de x_{new}, x_{min}, aquele que, conectado a x_{new}, produz o trajeto de menor custo (Figura 3.4b). Se o trajeto for livre de obstáculos, x_{new}, é inserido na árvore juntamente com a aresta que o conecta ao vértice pai (Figura 3.4c).

A operação de extensão tenta também conectar x_{new} a vértices já presentes na árvore. Isto é, para todo o conjunto de vértices vizinhos dentro de um determinado raio de $x_{new,}$ exceto x_{min} (Figura 3.4d), são religados aqueles que apresentarem um trajeto com menor custo e livre de obstáculos, se passarem por *xnew* (Figura 3.4e). Resultado final após uma iteração (Figura 3.4f).

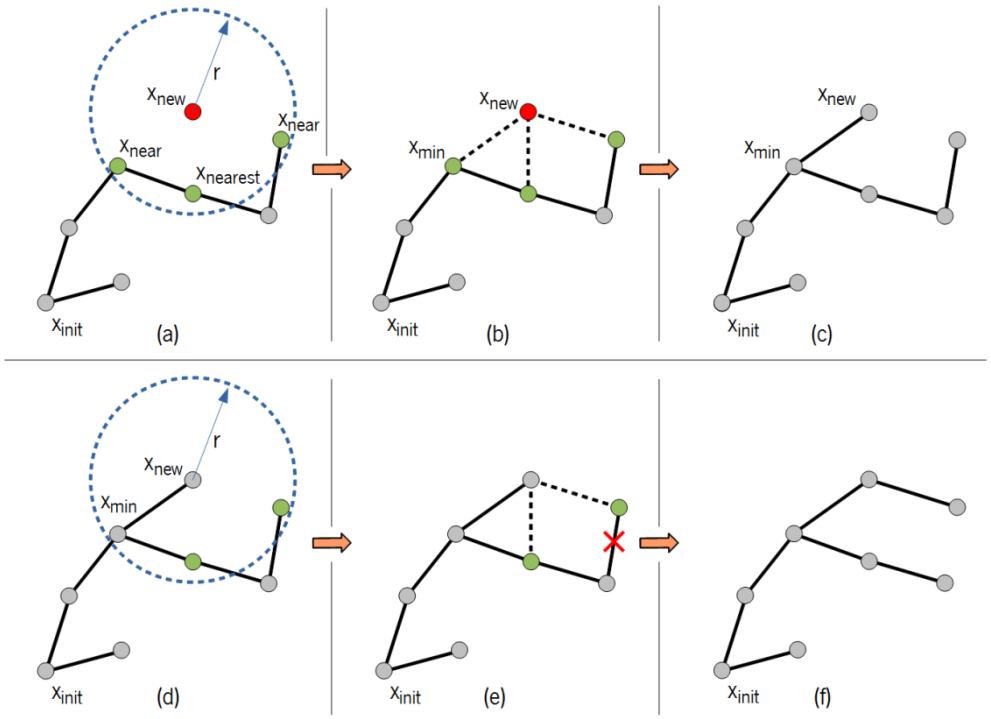

Figura 3.4: Representação da extensão do algoritmo RRT Ótimo, caso não sejam consideradas restrições dinâmicas.

O algoritmo RRT* com restrições diferenciais é dado no Algoritmo 1 e Algoritmo 3, bem como na Figura 3.5.

Antes de fornecer os detalhes deste algoritmo, é preciso delinear os procedimentos primitivos de que depende.

A função *Steer*, dados dois estados $x1$ e $x2$, retorna a trajetória ótima que começa em $x1$ e termina em $x2$ utilizando uma entrada de controlo u, durante um certo tempo, T.

Cost, calcula o custo acumulado desde o vértice inicial ao longo da árvore e *CostIm* calcula o custo da ida de um vértice para outro.

Algoritmo 3 RRTS_EXTEND (G, x)

```
1   x_nearest ← NearestNeighbor (G, x);
2   (u_new, T_new) ← Steer(x_nearest, x);
3   x_new ← NewState(x_nearest, u_new, T_new);
4   if ObstacleFree(x_nearest, x_new) then
5       G.AddVertex (x_new);
6       xmin ← xnearest;
7       T ← T_new;
8       u ← u_new;
9       X_near ← Near(G, x_new);
10      for all x_near ∈ X_near do
11          (u_near, T_near) ← Steer(x_near, x_new );
12          z_new ← NewState(x_near, u_near, T_near);
13          if ObstacleFree(x_near, x_new) and znew = xnew then
14              if Cost(x_near) + CostIm(x_near, x_new) < Cost(x_new) then
15                  x_min ← x_near;
16                  T ← T_near;
17                  u ← u_near;
18              end if
19          end if
20      end for
21      G.AddEdge(x_min, x_new, u, T);
22      for all x_near ∈ X_near\x_min do
23          (u_near, T_near) ← Steer(x_new, x_near);
24          z_new ← NewState(x_near, u_near, T_near);
25          T ← T_near;
26          u ← u_near
27          if z_new = x_new and ObstacleFree(x_new, x_near) and
            Cost(x_new) + CostIm(x_new, x_near) < Cost(x_near) then
28              x_parent ← Parent(x_near);
29              G.RemoveEdge(x_parent, x_near);
30              G.AddEdge(x_new, x_near, u, T);
31          end if
32      end for
33  end if
```

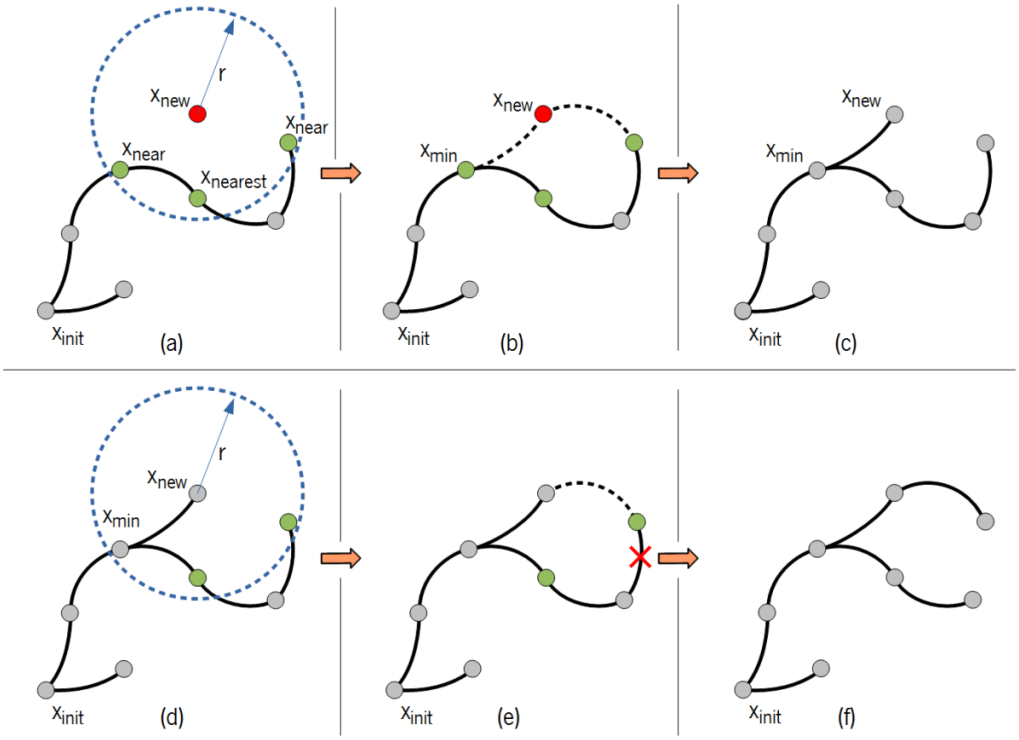

Figura 3.5: Representação da extensão do algoritmo RRT Ótimo, caso sejam consideradas restrições dinâmicas.

O Algoritmo 1 Inicializa-se com uma árvore que apenas contém o vértice, x_{init}, e sem arestas. Iterativamente constrói uma árvore com trajetórias livres de obstáculos, selecionando um estado aleatório, $x_{rand} \in X_{free}$ (Linha 3) expandindo-a depois em direção a este estado (Linha 4). A expansão da árvore está presente no Algoritmo 3. Primeiro, o algoritmo estende o vértice mais próximo, $x_{nearest}$ até ao estado aleatório, x_{rand} (Linhas 1-3) alcançando-se uma nova posição, x_{new}, no espaço de estados. Se o caminho entre $x_{nearest}$ e x_{new} for livre de obstáculos (Linha 4), x_{new} é adicionado à árvore (Linha 5) e o seu pai (vértice a que se vai ligar formando uma aresta) é decidido do seguinte modo. Primeiro, a função *Near* é invocada para determinar o conjunto de vértices vizinhos dentro de um determinado raio de x_{new} (Linha 9, Figura 3.5a). Depois, de entre todos os vértices vizinhos, é selecionado como pai, aquele que pode ser conduzido exatamente para x_{new} com o custo mínimo associado (Linhas 9-21, Figura 3.5b). Assim que o novo vértice, x_{new}, é inserido na árvore juntamente com a aresta que o conecta ao vértice pai (Figura 3.5c), a operação de extensão tenta também conectar x_{new} a vértices já presentes na árvore. Isto é, para todo o conjunto de vértices vizinhos dentro de um

determinado raio de x_{new} (Figura 3.5d), o algoritmo tenta conduzir x_{new} em direção a x_{near} (Linha 23, Figura 3.5e). Se conseguir conectar exatamente x_{new} e x_{near}, com uma trajetória livre de obstáculos com um custo menor ao custo atual de x_{near} (Linha 27), então x_{new} torna-se o novo pai de x_{near} (Linhas 28-30, Figura 3.5f), "religando-o".

3.3 Microsoft Kinect

Como mencionado na Introdução, o sensor utilizado foi o *Microsoft Kinect.*, um dispositivo lançado em Novembro de 2010. Composto por uma câmara RGB, sensores de profundidade, um conjunto de microfones e um motor para inclinação.

As principais características deste dispositivo são:

- Câmara RGB, com 8 *bits* por cada canal de cor, uma resolução de 640x480 *pixéis* com um *framerate* máximo de 30 Hz.
- Composto por um sensor infravermelho que consiste num projetor *laser* combinado com um sensor monocromático CMOS, que captura informação 3D. Possui um alcance entre os 0.8 e os 4 metros
- Possui um campo de visão de 57° na horizontal e 43° na vertical.

No âmbito deste trabalho foram utilizados os *drivers* desenvolvidos pela OpenNI[1], uma organização criada em Novembro de 2010. A *framework* multi-plataforma OpenNI é uma camada de abstração que providencia uma interface com os dispositivos físicos, neste caso o sensor *Kinect*. Possui ainda uma camada média, que proporciona componentes de *software* que analisam o áudio e a imagem e o compreendem. Por exemplo, *software* que recebe dados visuais, e retorna a localização da palma da mão, entre outros, como reconhecimento gestual e rastreamento do movimento do corpo.

[1] http://www.openni.org/

Figura 3.6: Microsoft Kinect

3.4 Deteção de Características (*Feature Dectection*)

Neste trabalho, para construir um mapa, e para que o robô se localizasse, foi necessário utilizar técnicas que permitissem detetar e acompanhar partes da imagem observada pela câmara.

Em visão por computador, mais especificamente no reconhecimento de objetos, muitas técnicas são baseadas na deteção de pontos de interesse em objetos ou superfícies. Isto é feito através da extração de características, que devem ser invariantes em relação à orientação, escala e localização, permitindo ao robô movimentar-se livremente sem restrições. Há dois aspetos a ter em conta no que diz respeito a uma característica: a deteção de um ponto-chave, que identifica uma área de interesse, e a sua descrição que caracteriza a sua região. Normalmente é identificada uma região que contém uma grande variação de intensidade, como um canto ou uma aresta, e o seu centro é designado ponto-chave. A sua descrição é geralmente determinada, medindo a orientação dos seus pontos mais próximos, dando origem a um vetor descritor que identifica um dado ponto-chave.

3.4.1 SIFT (Scalar Invariant Feature Transform)

O método SIFT de deteção de características, apresentado por David Lowe [18] é um algoritmo largamente utilizado em visão por computador para detetar e descrever características numa dada imagem. Não só é invariante em relação à escala como à rotação e iluminação. As características detetadas podem ser facilmente correspondidas entre imagens, para deteção e reconhecimento de objetos, bem como para determinar transformações geométricas entre as mesmas.

SIFT pode ser dividido nas etapas:

- **Construção de um espaço de escala –** A preparação inicial. São criadas representações da imagem original para assegurar invariância à escala.
- **Localização de pontos-chave** (*Keypoints*)
- **Atribuição de uma ou mais orientação aos pontos-chave –** São recolhidas as direções e magnitudes do gradiente à volta de cada ponto-chave. São depois selecionadas as orientações mais proeminentes que vão ser atribuídas aos pontos-chave. Isto assegura invariância em relação à orientação.
- **Descrição dos pontos-chave –** É criada mais uma representação que possibilita a identificação única de um dado ponto-chave.

3.4.1.1 Espaço de Escala

Objetos reais só fazem sentido a uma dada escala. É possível observar perfeitamente um cubo de açúcar numa mesa, mas ele deixa de existir se estivermos a observar a Via Láctea. O espaço de escala tenta replicar este conceito em imagens digitais.

Para criar um espaço de escala, é preciso pegar numa dada imagem, e progressivamente criar imagens desfocadas ou suavizadas, utilizando um desfoque de Gauss (*Gaussian Blur*). Este processo permite tirar detalhe de uma imagem, intencionalmente. Podemos querer observar uma árvore, e deixar de fora algum detalhe, como folhas ou galhos, por exemplo. Depois disto, é necessário redimensionar a imagem original para metade do tamanho e repetir o processo de desfocagem. As imagens do mesmo tamanho formam uma oitava, e cada oitava é formada por imagens com um progressivo aumento do nível de desfoque (escalas).

Matematicamente, o "desfoque" de uma imagem, é definido como a convolução de uma imagem I, com uma função gaussiana G, e uma variância σ,

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y) \tag{3.3}$$

em que,

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2} \tag{3.4}$$

3.4.1.2 Localização de Pontos-Chave

A operação *Laplacian of Gaussian* (LoG) calcula a segunda derivada de uma imagem (ou o seu Laplaciano), que foi primeiro desfocada utilizando um filtro de Gauss, de maneira a reduzir a sua sensibilidade ao ruído. Esta operação permite encontrar cantos e contornos de uma imagem, ótimos para encontrar pontos-chave na mesma.

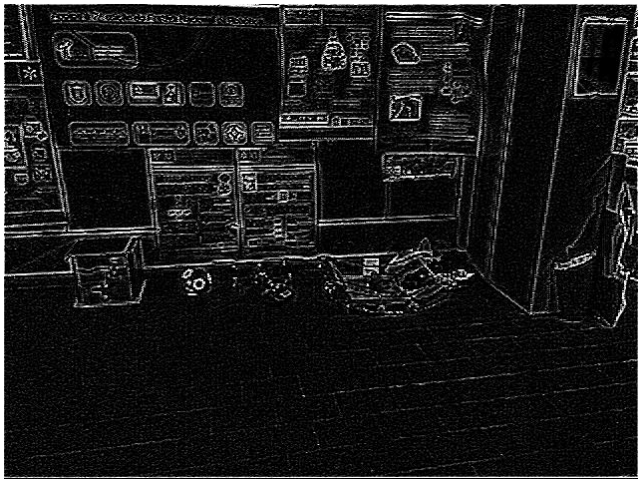

Figura 3.7: Resultado depois de aplicado o *Laplacian of Gaussian* numa dada imagem, com os seus contornos e cantos realçados.

Para criar imagens LoG de uma forma mais rápida, é utilizado o espaço de escala. É calculada a diferença Gaussiana, *Diffence of Gaussians* (DoG) entre duas escalas consecutivas, proporcionando uma aproximação à operação LoG descrita acima.

A diferença Gaussiana DoG é dada por,

$$\begin{aligned} D(x, y, \sigma) &= \big(G(x, y, k\sigma) - G(x, y, \sigma)\big) * I(x, y) \\ &= L(x, y, k\sigma) - L(x, y, \sigma) \end{aligned} \tag{3.5}$$

Que proporciona uma aproximação a LoG invariante à escala $\sigma^2\nabla^2 G$, como mostrado por Lindeberg [19],

$$\sigma\nabla^2 G = \frac{\partial G}{\partial \sigma} \approx \frac{G(x, y, k\sigma) - G(x, y, \sigma)}{k\sigma - \sigma} \tag{3.6}$$

Consequentemente:

$$G(x, y, k\sigma) - G(x, y, \sigma) \approx (k - 1)\sigma^2\nabla^2 G \tag{3.7}$$

Lindeberg [19] mostrou que o fator σ^2, permite uma verdadeira invariância em relação à escala, que já está presente nas imagens resultantes após a operação DoG. A localização dos pontos-chave é feita encontrando os máximos e mínimos destas imagens, como demonstra a Figura 3.9. Foi mostrado por Lowe [18], que o restante fator $(k - 1)$ não tem influência nessa localização.

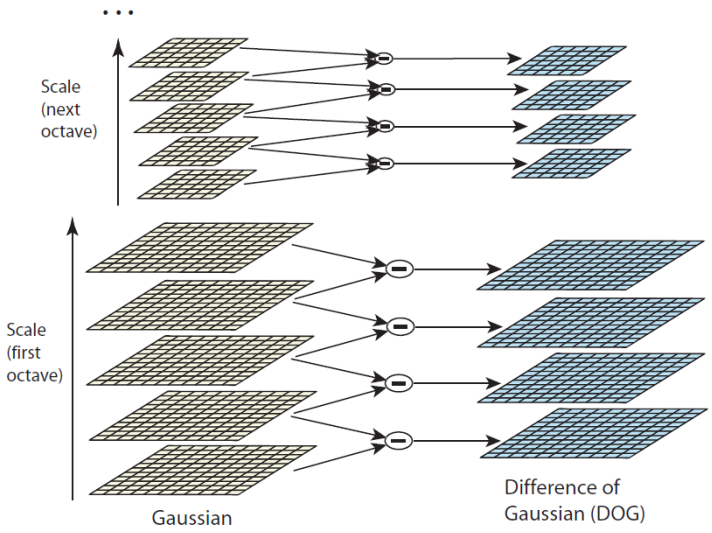

Figura 3.8: Processo de criação das imagens DoG [18].

A Figura 3.8 ilustra o processo de criação das imagens DoG. Para cada oitava do espaço de escala, é repetidamente feita a convolução da imagem inicial com funções Gaussianas, para produzir o conjunto de imagens que formam o espaço de escala, presentes na metade esquerda da figura. As imagens resultantes adjacentes são subtraídas, produzindo as imagens DoG da metade direita. Depois de cada oitava, a imagem é reduzida para metade, e o processo é repetido duplicando a variância σ. O valor inicial de σ pode ser modificado de acordo com o tipo de aplicação.[18]

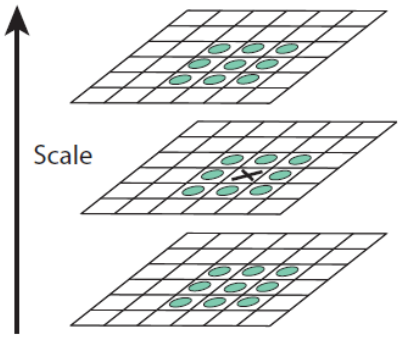

Figura 3.9: Máxima e mínima das imagens DoG são detetadas comparando um píxel (marcado com um X) com os seus 26 vizinhos (marcados como círculos) em regiões 3x3 na escala atual e nas duas adjacentes [18]

3.4.1.3 Atribuição de uma ou mais orientação aos pontos-chave

A escala do ponto-chave encontrado é utilizada para selecionar a imagem $L(x,y)$, de escala mais próxima, de modo a que todos os cálculos sejam feitos de uma maneira invariante à escala.

Para uma imagem suavizada por um filtro de Gauss, $L(x,y)$, a magnitude do gradiente, $m(x,y)$, e a sua orientação, $\theta(x,y)$ são calculadas da seguinte maneira,

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2} \quad (3.8)$$

$$\theta(x,y) = \tan^{-1}((L(x,y+1) - L(x,y-1))/(L(x+1,y) - L(x-1,y))) \quad (3.9)$$

É construído um histograma de orientação dos gradientes de 36 barras, cobrindo um alcance de 360 graus (a primeira barra corresponde a um alcance de 0 a 10 graus, a segunda, de 10 a 20 graus, e assim por diante), a partir da orientação e magnitude dos gradientes à volta do ponto-chave. As magnitudes dos gradientes adicionadas ao histograma são ponderadas por uma janela circular gaussiana com uma variância, σ, 1.5 vezes superior à da escala do ponto-chave.

Depois é identificada e atribuída ao ponto-chave a orientação mais proeminente, ou seja, o pico ou a maior barra do histograma construído. Qualquer outra orientação, acima de 80% deste limite é convertida num novo ponto-chave.

3.4.1.4 Descrição dos pontos-chave

É necessário criar uma única "impressão digital", o descritor, que identifique cada um dos pontos-chave.

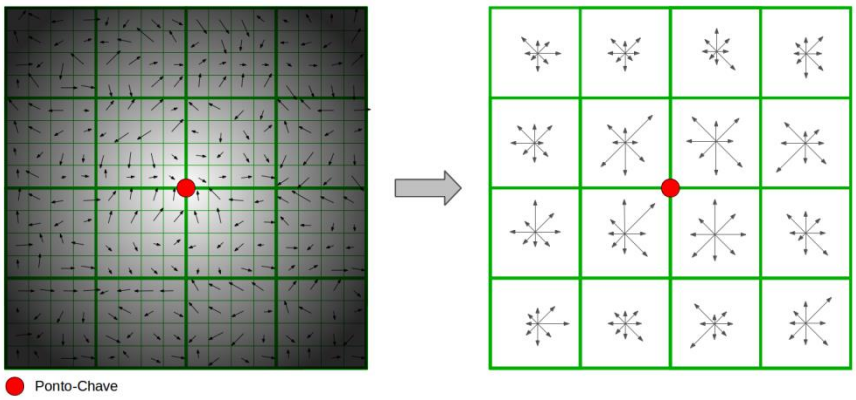

Figura 3.10: Processo de criação do vetor descritivo de um ponto-chave.

Na Figura 3.10, está exemplificado este processo. Primeiro, a escala do ponto-chave é utilizada para selecionar o nível de desfoque Gaussiano para a imagem em que se vão efetuar os cálculos, depois as coordenadas do descritor e as orientações do gradiente são rodadas relativamente à orientação do ponto-chave, de modo a obter invariância à orientação,

Em seguida, é criada uma janela 16x16 à volta de um dado ponto-chave (metade esquerda da Figura 3.10). Esta janela é subdividida em 16 janelas 4x4. Para cada uma das janelas 4x4, são calculadas as orientações e magnitudes dos gradientes. As quantidades adicionadas são ponderadas utilizando uma função gaussiana (a quantidade adicionada depende da distância ao ponto-chave), representado pelo gradiente radial da metade esquerda da Figura 3.10. As orientações e magnitudes do gradiente são acumuladas em histogramas de orientação de oito barras sumarizando o conteúdo das sub-regiões 4x4 (metade direita da Figura 3.10), com o comprimento de cada seta a corresponder à soma da magnitude dos gradientes nesse sentido, dentro da região.

Para cada ponto-chave obtém-se um vetor descritor de tamanho 128.

3.4.2 SURF (Speeded-Up Robust Features)

SURF [20], é um detetor e descritor robusto de características, baseado nos mesmos princípios e passos do anteriormente descrito, SIFT, superando-o em praticamente todos os aspetos, sendo diversas vezes mais rápido que este. Do mesmo modo do anterior, SURF divide o espaço de escala em oitavas e escalas. Uma oitava corresponde ao dobro da variância, σ, da oitava anterior, e cada oitava é dividida entre escalas uniformemente separadas.

De modo a tornar a descrição deste método claro por si, é necessário primeiro examinar os conceitos de **imagens integrais** e **máscaras**, no âmbito de processamento de imagem.

3.4.2.1 Máscaras

Máscaras, são filtros de convolução, representados por uma 'janela' ou matriz, usados para filtrar uma imagem. Alteram a intensidade de um pixel para refletir a intensidade dos pixéis vizinhos. É possível observar, na Figura 3.12, a resultante depois de aplicada uma máscara 5x5 com '1's em todas as suas células,

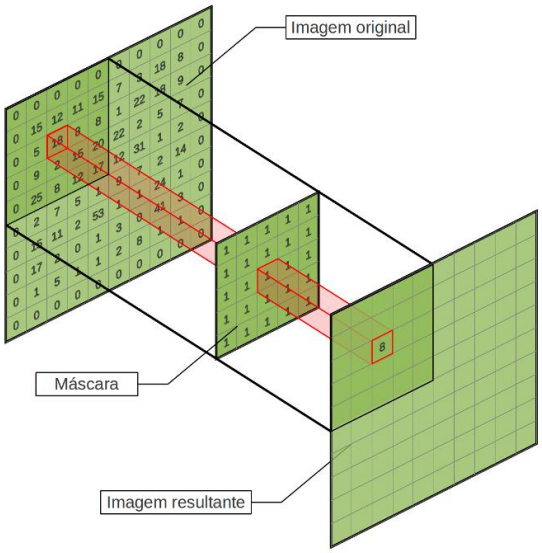

Figura 3.11: Máscara ou filtro de convolução.

A Figura 3.11 mostra a aplicação de uma máscara numa dada imagem. Durante a convolução, o centro da janela 5x5, passa por cada um dos pixéis. O processo multiplica cada

número da janela pela intensidade do pixel da imagem original imediatamente abaixo, resultando em tantos produtos quanto a quantidade de números existentes na máscara. No passo final, estes produtos são somados, e este valor torna-se a nova intensidade do píxel que estava diretamente por baixo do centro do filtro. No exemplo da Figura 3.11, o resultado foi dividido pela soma dos valores das células da máscara de modo a manter o brilho da imagem original, evitando a saturação dos pixeis.

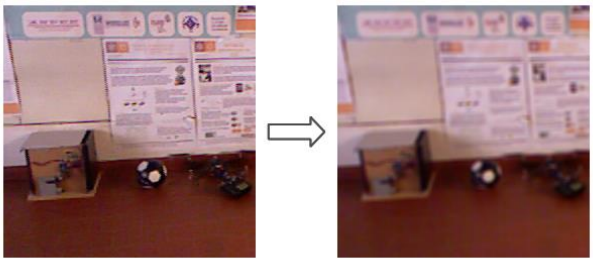

Figura 3.12: Imagem resultante, ligeiramente alisada ou desfocada, à direita, depois de aplicado um filtro de janela 5x5, na imagem original, à esquerda.

3.4.2.2 Imagens Integrais

As imagens integrais permitem a computação rápida dos filtros de caixa. Uma imagem integral $I_\Sigma(\mathrm{x})$ a uma localização $\mathrm{x} = (x, y)^T$, representa a soma de todos os pixels na imagem I, dentro de uma região retangular, formada pela origem e x.

$$I_\Sigma(x) = \sum_{i=0}^{i \le x} \sum_{j=0}^{j \le y} I(i,j) \tag{3.10}$$

Assim que a imagem integral é calculada, são apenas necessárias três somas para calcular a soma das intensidades de qualquer área retangular (Figura 3.13). Por isso, o tempo de cálculo é independente do tamanho. Isto é importante na abordagem do SURF, já que são usadas máscaras de grandes dimensões.

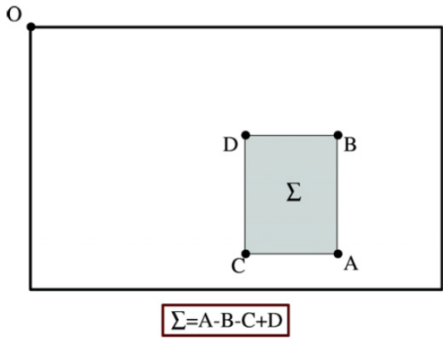

Figura 3.13: Com o uso de imagens integrais, são necessárias apenas três adições para calcular a soma das intensidades dentro de uma região retangular de qualquer tamanho. [20]

3.4.2.3 Matriz Hessiana

SURF baseia-se na matriz Hessiana para deteção de pontos-chave. O determinante de uma matriz hessiana expressa a mudança local em torno de uma área

Dado um ponto $\mathrm{x} = (x, y)$ numa imagem I, a matriz Hessiana $H(\mathrm{x}, \sigma)$ em x a uma escala σ é definida da seguinte maneira,

$$H(x, \sigma) = \begin{bmatrix} L_{xx}(x, \sigma) & L_{xy}(x, \sigma) \\ L_{xy}(x, \sigma) & L_{yy}(x, \sigma) \end{bmatrix} \tag{3.11}$$

Onde $L_{xx}(\mathrm{x}, \sigma)$ é a convolução da derivada gaussiana de segunda ordem $\frac{\partial^2}{\partial x^2} g(\sigma)$ com a imagem I num ponto x, e similarmente para $L_{xy}(\mathrm{x}, \sigma)$ e $L_{yy}(\mathrm{x}, \sigma)$. A base do método SURF é a deteção dos máximos e mínimos do determinante da matriz Hessiana. No entanto, a convolução é muito dispendiosa, e as derivadas gaussianas de segunda ordem, usadas para a matriz hessiana, na prática, têm de ser discretizadas e cortadas, antes de poderem ser aplicadas (Figura 3.14, metade esquerda). Sendo que o esforço computacional aumenta com o aumento do tamanho dos filtros. O algoritmo SURF aproxima estes filtros com caixas retangulares, chamados filtros de caixa. Estes filtros de caixa, são aproximações às derivadas gaussianas de segunda ordem, que podem ser calculadas com um custo computacional muito baixo utilizando imagens integrais. O tempo de cálculo é por isto, independente do tamanho do filtro.

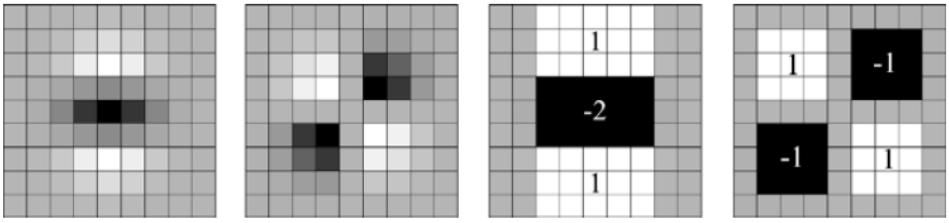

Figura 3.14: Ilustração das máscaras, discretizadas e cortadas das derivadas parciais de segunda ordem em contraste com as aproximações propostas por este método.[20]

A Figura 3.14 representa, da esquerda para a direita: A máscara, discretizada e cortada da derivada parcial de segunda ordem em $y, (L_{yy})$ e $xy, (L_{xy})$, respetivamente; a aproximação proposta por este método à derivada parcial gaussiana de segunda ordem em $y, (D_{yy})$ e na direção $xy, (D_{xy})$. As regiões a cinzento correspondem a zero.

O determinante aproximado da matriz Hessiana, utilizando os filtros de caixa (Figura 3.14, metade direita), numa localização $\mathrm{x} = (x, y, \sigma)$, pode ser dado por:

$$det(H_{aprox}) = D_{xx}D_{yy} - (0.9D_{xy})^2 \tag{3.12}$$

$$\sigma = (\, tamanhodofiltro/9) \times 1.2 \tag{3.13}$$

A partir daqui é possível calcular o determinante da matriz Hessiana para cada um dos píxeis, formando um mapa de resposta, *blob response map*, para diferentes escalas. Este mapa serve para a localização de pontos-chave, como explicado mais à frente.

3.4.2.4 Espaço de Escala

As máscaras 9x9 presentes na Figura 3.14 são aproximações de uma função gaussiana com $\sigma = 1.2$ e representam a escala mais baixa do espaço de escala.

No método SIFT, as imagens são iterativamente alisadas através de uma função gaussiana para depois se determinarem as imagens DoG, onde os contornos e outros pontos de interesse podem ser encontrados (Figura 3.8). Devido ao uso dos filtros de caixa e imagens

integrais, o método SURF, pode aplicar os filtros de caixa diretamente na imagem original. Por isso o espaço de escala é construído aumentando gradualmente o tamanho da máscara.

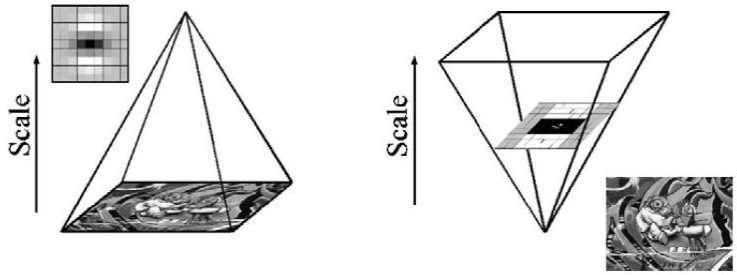

Figura 3.15: Em vez de iterativamente reduzir o tamanho da imagem (esquerda), o uso de imagens integrais permite a expansão do filtro a um custo constante (direita). [20]

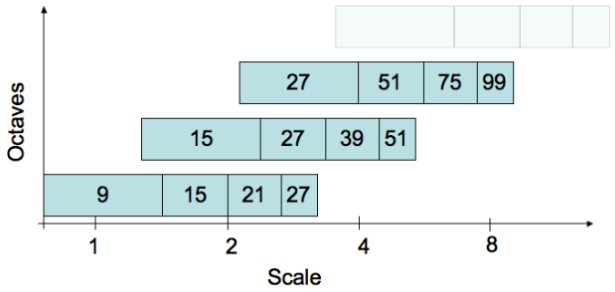

Figura 3.16: Representação gráfica dos tamanhos do lado do filtro para três diferentes oitavas numa escala logarítmica. As oitavas sobrepõem-se para garantir a cobertura completa de cada escala [20]

3.4.2.5 Localização dos pontos-chave

A localização dos pontos-chave é feita encontrando os máximos e mínimos no mapa de resposta, *blob response map*, comparando cada pixel com os seus 26 vizinhos em regiões 3x3 (Figura 3.9).

3.4.2.6 Atribuição de uma orientação aos pontos-chave

São utilizados filtros, chamados *Haar wavelets*, em conjunção com imagens integrais. Os filtros, *Haar wavelets*, podem ser usados para encontrar os gradientes na direção x e y.

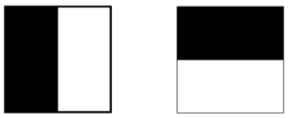

Figura 3.17: Filtros *Haar wavelet*, para calcular as respostas em x(esquerda), dx, e y(direita), dy. Em combinação com imagens integrais são necessárias apenas seis operações para calcular a resposta em cada sentido. [20]

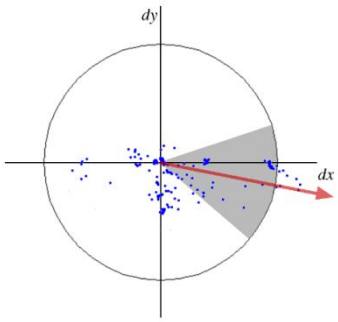

Figura 3.18: Determinação da orientação de um ponto-chave[20].

Para atribuir uma orientação aos pontos-chave é primeiro calculada, e ponderada, para cada ponto/pixel, a resposta ao filtro *Haar wavelet*, na direção x e y, dentro de uma região circular à volta do ponto-chave. Cada ponto é ponderado, baseado na sua distância ao ponto-chave, utilizando uma função gaussiana. Depois de calculadas, é estimada a orientação dominante, somando todas as respostas em x e y, dentro de uma janela de orientação deslizante de tamanho $\pi/3$. As respostas em x e y dentro de cada segmento são somadas, e formam um novo vetor. O vetor mais longo torna-se a orientação do ponto-chave. Este processo está ilustrado na Figura 3.19.

3.4.2.7 Descrição dos pontos-chave

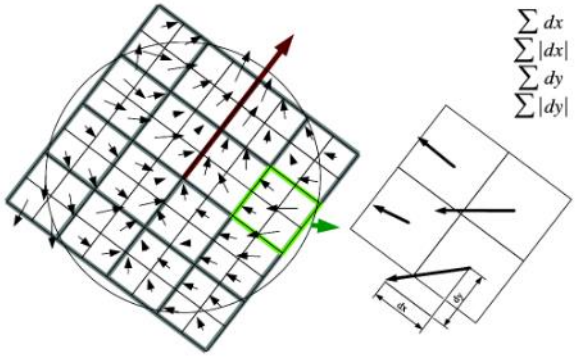

Figura 3.19: Descrição de um ponto-chave [20].

Para descrever os pontos-chave, é criada uma janela 4x4 centrada no ponto-chave e orientada ao longo da orientação selecionada no passo anterior. Para cada um dos quadrados desta janela, são calculadas as respostas ao filtro *Haar wavelet* em 25 pontos igualmente espaçados. As subdivisões 2x2 de cada quadrado, correspondem às somas dx, $|dx|$, dy e $|dy|$. Juntando estas quatro somas para todas as sub-regiões 4x4, resulta num vetor descritor de tamanho 64. Processo ilustrado na Figura 3.19.

3.5 Transformação entre Imagens

Depois de determinadas as características de uma imagem, é necessário acompanhá-las ao longo do movimento da câmara, permitindo saber a posição da mesma em cada instante. Isto é feito associando-as entre imagens capturadas. No entanto, o processo de corresponder diversas características não é direto, dado que a sua descrição não é exatamente a mesma entre duas imagens diferentes, pelo movimento da câmara e ruído presente na imagem. Posto isto, é necessária a utilização de técnicas e algoritmos, que eliminem más correspondências entre características com um bom grau de confiança. São chamados de *inliers*, às correspondências que se enquadram num modelo e *outliers*, às que são eliminadas por não se encaixarem no modelo.

3.5.1 RANSAC

O método RANSAC, abreviatura de "*RANdom SAmple Consensus*", foi primeiro apresentado por *Fischler* e *Bolles* [21] em 1981. É um método iterativo utilizado para estimar os parâmetros de um modelo matemático, de um conjunto de dados observado que contém *outliers*. A ideia é encontrar os parâmetros que são válidos para a maioria dos pontos, estabelecendo um consenso, descartando os pontos presentes de ruído.

O Algoritmo 4 descreve este método.

Algoritmo 4 RANSAC

Input: Conjunto de pontos,*points*; número de iterações, *K*; valor de *threshold*, *t*.

```
1   bestModel, bestInliers ←Ø;
2   for iteration = 1 to K do
3   samples ← RandomPoints(points);
4   currentModel ← Model(samples);
5   inliers ← Ø;
6       for all pt ∈ points do
7           error ← EvaluateFromModel(pt, currentModel);
8           if error < t then
9               inliers ← inliers + pt;
10          end if
11      end for
12      computeMeanError (inliers.Size());
13      if currentModel.Valid(inliers.Size(), inliers.Ratio()) then
14          currentModel ← Model(inliers);
15          if currentModel .Better(bestModel) then
16              bestModel ← currentModel;
17              bestInliers ← inliers;
18          end if
19      end if
20  end for
21  return bestInliers, bestModel
```

Primeiro é selecionado um conjunto de pontos, *N*, aleatoriamente (Linha 3), e a partir destes, é parametrizado um modelo, que formula uma hipótese (Linha 4). Para cada um dos pontos, são adicionados comos *inliers*, aqueles que fazem parte do modelo com um erro menor que *t* (Linhas 6-9). Se for considerando que o número de *inliers*, bem como a razão entre o número de *inliers* e o número total de pontos, for maior que um valor pré-definido, ou seja, se o modelo atual for considerado válido, é recalculado a partir dos *inliers* atuais (Linha 13-14). Se este modelo for melhor que o modelo até então (Linha 15), o modelo e *inliers* atuais tornam-se nos respetivos melhores (Linhas 16-17).

O algoritmo retorna os melhores *inliers* e modelo, assim que terminar o número de iterações estabelecidas (Linha 21).

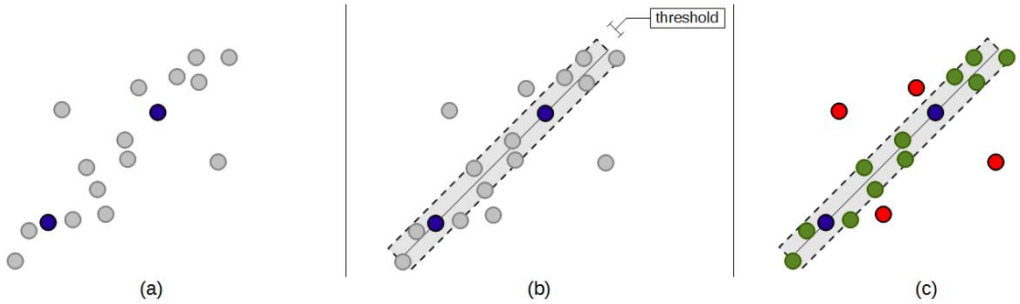

Figura 3.20: Ilustração de uma iteração do método RANSAC

Na Figura 3.20 está exemplificada uma iteração do método RANSAC, para um conjunto de pontos em duas dimensões. São primeiro selecionados *k* itens (neste caso *K*=2, o necessário para descrever uma linha) de um conjunto (Figura 3.20a). A partir destes dois itens, é desenhada a linha que passa pelos mesmos, com uma área de validação que representa o limite máximo (*threshold*), a que os restantes itens devem distar para poderem ser considerados *inliers* (Figura 3.20b). O modelo é avaliado medindo o erro para cada ponto, neste caso, calculando a distância à linha. Os pontos que tiverem um erro maior que o *threshold* pré definido são considerados *outliers*. O número total de *inliers*, bem como a razão do número de *inliers* relativamente ao número total de pontos, pode dar a validação do modelo (Figura 3.20c).

3.5.2 ICP (Iterative Closest Point)

O algoritmo Iterative Closest Point [22], pode ser empregue para minimizar a diferença entre duas nuvens de pontos, determinando iterativamente a transformação (rotação e translação) necessária para que tal aconteça. É geralmente utilizada para reconstruir superfícies 2D ou 3D de diferentes amostras, podendo ainda ser utilizada para localizar robôs bem como otimizar o planeamento de trajetória [23] (especialmente quando são utilizadas técnicas de odometria, e a informação é incerta devido a terreno escorregadio). Neste trabalho pode ser utilizada de maneira a reduzir a distância entre características correspondidas entre duas imagens diferentes, aumentando a qualidade do mapa construído.

Dados dois conjuntos de pontos:

$$X = \{x_1, \dots, x_n\} \tag{3.14}$$

e,

$$P = \{p_1, \dots, p_n\} \tag{3.15}$$

é pretendido encontrar a translação t e rotação R que minimiza a soma dos erros quadráticos entre os pontos:

$$E(R,t) = \frac{1}{N_p}\sum_{i=1}^{N_p} \left\| x_i - R_{p_i} - t \right\|^2 \tag{3.16}$$

em que x_i e p_i são pontos correspondentes.

O algoritmo ICP é aplicável quando se tem uma transformação relativamente boa antecipadamente. Uma possibilidade para melhorar as correspondências é utilizar primeiro o algoritmo RANSAC, para determinar uma boa aproximação da transformação e usá-la como primeira suposição no algoritmo ICP.

Capítulo 4

Construção do Mapa

Neste capítulo serão apresentados as técnicas e métodos utilizados, bem como os passos seguidos que permitem a um robô construir um mapa enquanto simultaneamente se localiza, resolvendo o problema SLAM (*Simultaneous Localization and Mapping*). Entenda-se por mapa a um conjunto de poses com capturas RGB e de profundidade associadas.

A solução encontrada e utilizada neste projeto passa pelos seguintes passos:

- Observar o ambiente;
- Detetar pontos de interesse no ambiente observado em capturas RGB (apenas em 2D);
- Associar os pontos de interesse com pontos de interesse anteriormente observados;
- Converter os pontos associados em 3D.
- Estimar o movimento do robô, ou a transformação 3D, a partir da correspondência entre pontos de interesse associados;
- Corrigir a posição dos pontos de interesse, bem como do ambiente observado a que estão associados, a partir da transformação estimada;
- Adicionar os pontos de interesse, bem como o ambiente observado a que estão associados, ao mapa.

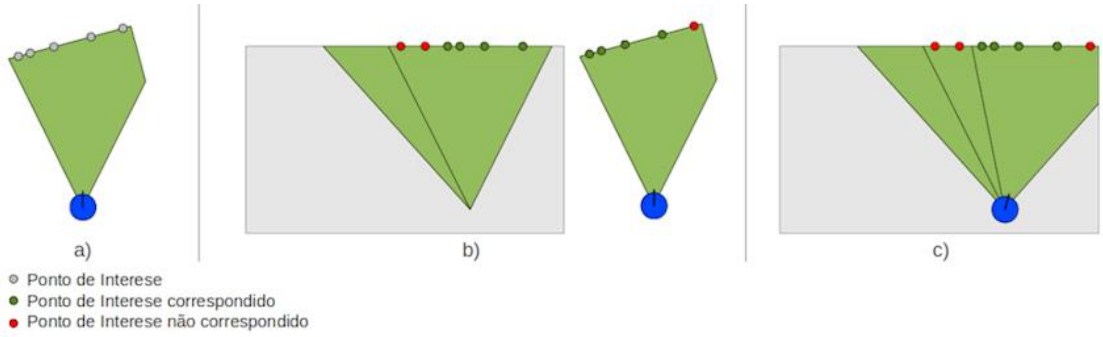

Figura 4.1: Ilustração da resolução proposta do problema SLAM.

A Figura 4.1 ilustra o processo para a resolução do problema SLAM. O robô é movimentado e são detetados pontos de interesse no ambiente observado (Figura 4.1a).Os novos pontos de interesse detetados são correspondidos aos anteriormente capturados (Figura

4.1b). Os pontos correspondidos são convertidos em 3D. É estimado o movimento do robô a partir da correspondência anterior, e é corrigida a posição dos mesmos, bem como do ambiente observado associado, a partir da transformação estimada (Figura 4.1c).

4.1 Extração e correspondência de características

Como apresentado anteriormente, é necessário detetar pontos de interesse (Figura 4.1a) para depois os acompanhar ao longo do movimento da câmara/robô.

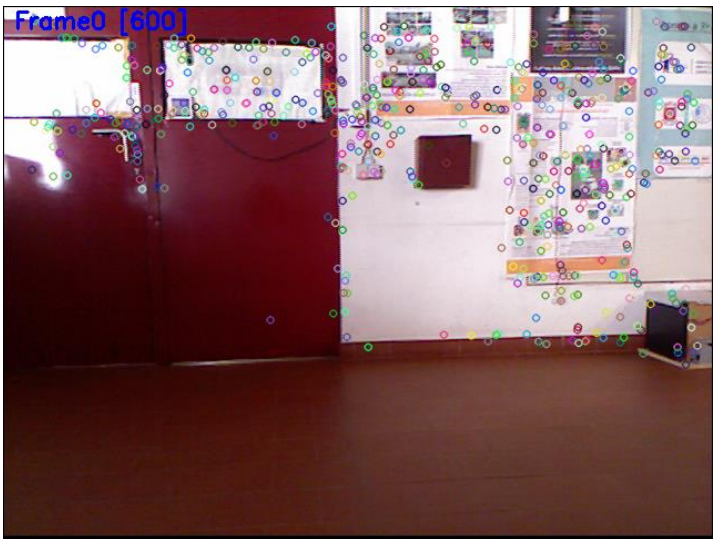

Figura 4.2: Pontos-chave (ou pontos de interesse) detetados numa imagem RGB capturada pelo sensor *Microsft Kinect*, utilizando o algoritmo SURF.

Depois de detetados estes pontos, é possível encontrar a sua correspondência entre duas imagens capturadas (Figura 4.1b).

O melhor candidato a uma correspondência para cada ponto-chave é encontrado, identificando o vizinho mais próximo, *nearest neighbor search*, no conjunto de pontos-chave da imagem alvo.

Os vizinhos mais próximos são definidos como os pontos-chave, descritos por um vetor (vetor descritor), com menor distância euclidiana.

Considerando o método SURF, a distância euclidiana, d, entre dois vetores característicos de duas imagens diferentes, p e q (Figura 3.19), de tamanho 64, com um índice i, pode ser dada por,

$$d = \sum_{i=1}^{64} \sqrt{(p_i - q_i)^2} \tag{4.1}$$

Neste projeto foi utilizada a biblioteca OpenCV[2] (*Open Source Computer Vision Library)*. Numa primeira fase para a extração de características, ou seja, deteção dos pontos-chave (Figura 4.2) bem como dos seus vetores descritores. Numa segunda fase, para a associação dos pontos entre imagens utilizando a interface da biblioteca OpenCV para a biblioteca FLANN [24] (Figura 4.3). A biblioteca FLANN (*Fast Library for Approximate Nearest Neighbors*) contém uma coleção de algoritmos otimizados para pesquisa rápida do vizinho mais próximo.

Ao longo do trabalho, foi utilizada a linguagem de programação c++ e ainda, a coleção de bibliotecas multiplataforma Boost[3], para gestão de ficheiros e diretorias; criação e controlo de *threads* e comunicação via porta série.

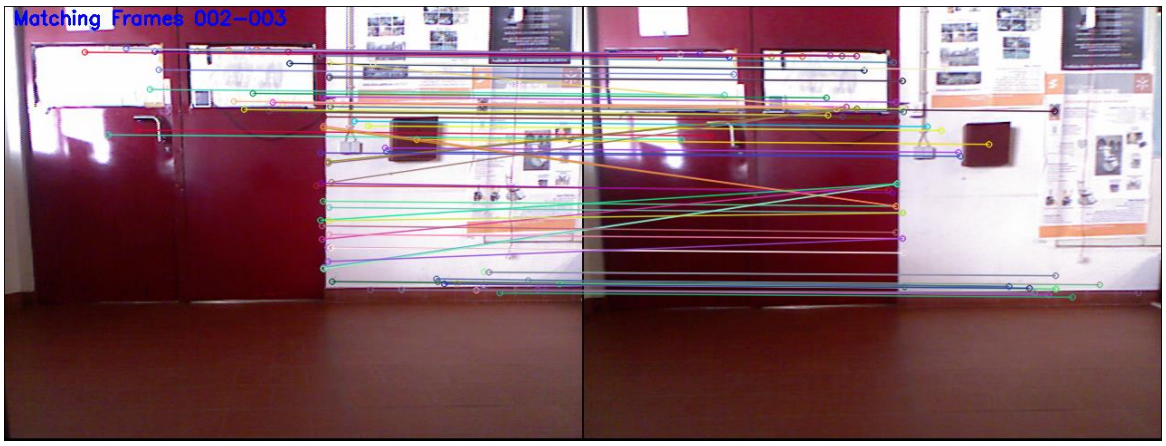

Figura 4.3: Correspondência inicial entre características extraídas de duas capturas consecutivas.

[2] http://opencv.org/
[3] http://www.boost.org/

4.2 Determinação da transformação 3D

Depois de obtidas as correspondências iniciais, entre características, em duas capturas diferentes, é possível encontrar a transformação 3D que une os dois conjuntos de pontos, isto é, a operação que projeta cada um dos pontos na imagem fonte para o ponto correspondente na imagem alvo (Figura 4.1c). Neste caso, a transformação pode ser dada por uma matriz quadrada de dimensão 4 com 6 graus de liberdade, composta por uma matriz de rotação e um vetor de translação em 3 dimensões.

Um ponto $P = (x, y, z, 1)^T$, pode ser projetado aplicando uma matriz de transformação T,

$$P' = T \times P \tag{4.2}$$

É importante referir que a extração de características no método SURF foi feita nas capturas RGB em 2D e portanto, os pontos dados pela correspondência inicial (Figura 4.3), precisam de ser convertidos em 3 dimensões. Foi primeiro definido um sistema de coordenadas, da seguinte maneira,

$$\begin{cases} x: profundidade \\ y: altura \\ z: largura \end{cases} \tag{4.3}$$

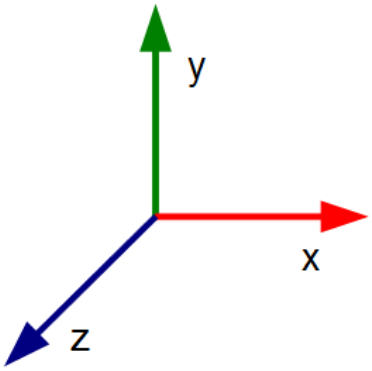

Figura 4.4: Sistema de coordenadas utilizado, x: profundidade, representado a vermelho; y: altura, representado a verde e z: largura, representado a azul.

Foi utilizado o suporte da biblioteca OpenCV a sensores compatíveis com a *framework* OpenNi, para aquisição de dados RGB-D do sensor *Microsoft Kinect*..

Um passo importante na utilização de sensores como o *Microsoft Kinect*, é assegurar que os geradores de imagem RGB e de profundidade utilizam o mesmo ponto de vista. No *Kinect*, o mapa de profundidade é derivado do seu sensor infravermelho, que está posicionado a alguns milímetros de distância da câmara RGB, e por isso possui um ponto de vista diferente.

O suporte da biblioteca OpenCV à *framework* OpenNI permite corrigir isto, alterando propriedades da câmara, tal como o registo das imagens RGB-D. Este processo resulta em imagens alinhadas ao píxel, ou seja, um píxel da imagem RGB com as coordenadas (u, v), possui a sua informação de profundidade correspondente, num pixel com as mesmas coordenadas (u, v), no mapa de profundidade.

Conhecendo os valores do mapa de profundidade, obtidos em milímetros pela biblioteca OpenCV, para um ponto de coordenadas (u, v), numa imagem de 640x480 pixéis de resolução, e a distância focal do sensor, f, é possível obter o ponto em 3D, P, correspondente, a partir da geometria baseada no modelo de câmara *pinhole* (Figura 4.5).

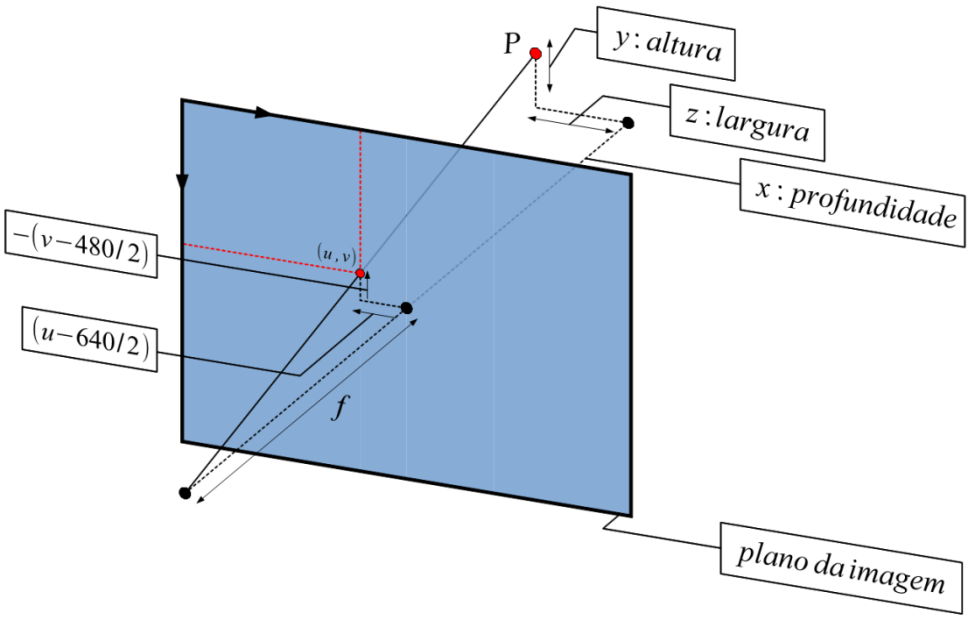

Figura 4.5: Geometria baseada no modelo da câmara *pinhole* para obtenção das coordenadas 3D, correspondentes a um pixel de coordenadas (u, v).

Posto isto, os pontos dados pela correspondência inicial podem ser convertidos em 3 dimensões, da seguinte maneira,

$$P(x,y,z)\begin{cases} x = profundidade \\ y = -(v - 480/2) \times profundidade/f \\ z = (u - 640/2) \times profundidade/f \end{cases} \tag{4.4}$$

Depois de convertidos em 3D, o próximo passo é encontrar a matriz de transformação que resulta numa projeção satisfatória para a maioria destes pontos. Idealmente seria possível determinar a matriz de transformação através do ajuste de dados pelos mínimos quadrados, para todos os pontos. O número mínimo de pares necessário seria três, com cada par a definir uma equação,

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{4.5}$$

Supondo-se que existem n pares de pontos correspondentes P'_i e P_i,

$$\begin{aligned} x'_i &= a_{11}x_i + a_{12}y_i + a_{13}z_i + a_{14} \\ y'_i &= a_{21}x_i + a_{22}y_i + a_{23}z_i + a_{24} \\ z'_i &= a_{31}x_i + a_{32}y_i + a_{33}z_i + a_{34} \end{aligned} \tag{4.6}$$

Resultando em três conjuntos de equações lineares na forma de,

$$Ma = b \tag{4.7}$$

Primeiro conjunto:

$$\begin{bmatrix} x_1 & y_1 & z_1 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ x_n & y_n & z_n & 1 \end{bmatrix} \times \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \\ a_{14} \end{bmatrix} = \begin{bmatrix} x'_1 \\ \vdots \\ x'_n \end{bmatrix} \tag{4.8}$$

Segundo conjunto:

$$\begin{bmatrix} x_1 & y_1 & z_1 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ x_n & y_n & z_n & 1 \end{bmatrix} \times \begin{bmatrix} a_{21} \\ a_{22} \\ a_{23} \\ a_{24} \end{bmatrix} = \begin{bmatrix} y'_1 \\ \vdots \\ y'_n \end{bmatrix} \tag{4.9}$$

Terceiro conjunto:

$$\begin{bmatrix} x_1 & y_1 & z_1 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ x_n & y_n & z_n & 1 \end{bmatrix} \times \begin{bmatrix} a_{31} \\ a_{32} \\ a_{33} \\ a_{34} \end{bmatrix} = \begin{bmatrix} z'_1 \\ \vdots \\ z'_n \end{bmatrix} \tag{4.10}$$

Na equação (4.7), M não é uma matriz quadrada e por isso não tem inversa, mas $M^T M$ é quadrada e tipicamente tem inversa, por isso, a solução do problema de mínimos quadrados linear é dada por,

$$M^T M a = M^T b \tag{4.11}$$

$$a = (M^T M)^{-1} M^T b \tag{4.12}$$

Em que, $(M^T M)^{-1} M^T$ é a pseudoinversa de M.

No entanto, um único ponto cuja localização foi incorretamente estimada, pelo ruído do sensor e pela própria correspondência entre características pode levar a um erro significante na transformação final. Por isto, o método RANSAC, descrito anteriormente, pode ser usado iterativamente para achar uma transformação satisfatória, eliminando os pontos que não fazem parte do modelo (*outliers*).

Considerando o conjunto de par de pontos da correspondência inicial, cada par contendo um ponto da imagem fonte, *origin*, associado a um da imagem alvo, *destination*, o algoritmo RANSAC, neste contexto, pode ser descrito da seguinte forma,

Algoritmo 5 Transformação 3D com RANSAC

Input: Pares de pontos 3D, *pairPoints* (*origin*, *destination*); número de iterações, *K*; valor de *threshold*, *t*.

```
1   bestTransform, bestInliers ←Ø;
2   for iteration = 1 to K do
3   samples ← RandomPairPoints(pairPoints);
4   currentTransform ← computeTransform(samples);
5   inliers ← Ø;
6       for all pair ∈ pairPoints do
7           projected_i ← projectPoint(origin_i , currentTransform);
8           error ← computeDistance(projected_i , destination_i );
9           if error < t then
10              inliers ← inliers + pair_i ;
11          end if
12      end for
13      computeMeanError (inliers.Size());
14      if currentTransform.Valid(inliers.Size(), inliers.Ratio()) then
15          currentTransform ← computeTransform (inliers);
16          if currentTransform.Better(bestTransform) then
17              bestTransform ← currentTransform;
18              bestInliers ← inliers;
19          end if
20      end if
21  end for
22  return bestInliers, bestTransform
```

Primeiro é selecionada aleatoriamente uma amostra de três pares de pontos, *N*, a partir das correspondências iniciais (Linha 3), e é calculada uma transformação, que formula uma hipótese, a partir desta amostra (Linha 4). A transformação estimada é utilizada para projetar cada um dos pontos da imagem fonte (Linha 7). Depois, é calculada a distância, ou erro, entre cada um dos pontos projetados, *projected*, e os pontos correspondentes da imagem alvo, *destination* (Linha 8). Se o erro ou distância for menor que *t* (Linha 9), o par é adicionado como *inlier* (Linha 10). Depois de terem sido verificados todos os pares, é calculada a média do erro para que mais tarde possa ser utilizada para caracterizar a transformação (Linha 13). Se for considerando que o número de *inliers*, bem como a razão entre o número de *inliers* e o número total de correspondências, for maior que um valor pré-definido, ou seja, se a transformação atual for considerada válida, é recalculada a partir dos *inliers* atuais (Linha 14- 15). Se esta

transformação for melhor que a transformação até então (Linha 16), a transformação e *inliers* atuais tornam-se nos respetivos melhores (Linhas 17-18). O algoritmo retorna os melhores *inliers* e melhor transformação, T, assim que terminar o número de iterações estabelecidas (Linha 22).

Figura 4.6: Correspondência entre imagens depois de aplicado o algoritmo RANSAC.

Na Figura 4.6 está presente o resultado depois de aplicado o algoritmo RANSAC, nos pontos da correspondência inicial (Figura 4.3), depois de convertidos em 3D, com algumas das associações rejeitadas. Apenas os *inliers* estão representados.

Uma transformação pode ser caracterizada por:

- **Erro médio:** A norma do vetor do erro.
- **Número de *inliers:*** O número total de *inliers*.
- **Razão de *inliers:*** Razão entre o número de *inliers* e o número total de pares da correspondência inicial.

Sabendo que quanto maior o número total de *inliers*, maior a razão de *inliers*, e menor o erro médio, é possível ajustar os limites destes parâmetros para caracterizar uma transformação como boa ou má.

Depois de estimada a transformação pelo método RANSAC, é possível aperfeiçoá-la utilizando o algoritmo ICP (Secção 3.5.2). Este algoritmo utiliza a transformação estimada pelo método RANSAC como suposição inicial, para depois minimizar a diferença entre os pontos da imagem fonte, projetados pela transformação estimada (suposição inicial), e os pontos da imagem alvo correspondentes.

O algoritmo ICP foi implementado por intermédio da biblioteca PCL[4] (*Point Cloud Library*) [25]

4.3 Sucessão de Poses

A partir de cada uma das transformações 3D determinadas entre cada duas capturas, e sabendo a pose da câmara/robô inicial (a referência) é possível estimar qualquer pose depois de uma sucessão de transformações. Considerando uma sequência finita de capturas, $[captura_0; captura_N]$, a pose P_k, de índice $k \in [0; N]$ pode ser dada por uma matriz quadrada de dimensão 4 com 6 graus de liberdade, composta por uma matriz de rotação, R e um vetor de translação em 3 dimensões, t.

$$P_k = \begin{bmatrix} & & & t_x \\ & R & & t_y \\ & & & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.13}$$

Para $i \in [1; N]$, existe uma matriz de transformação 4x4, T_{i-1}^{i}, estimada pelo método RANSAC e aperfeiçoada pelo algoritmo ICP, que une a posição P_{i-1}, à posição P_i.

Sabendo a posição inicial, P_0, é possível determinar a pose P_k, de índice $k \in [0; N]$, combinando as matrizes de transformação da seguinte maneira,

$$P_k = \prod_{i=k}^{1} T_{i-1}^{i} \, P_0 \tag{4.14}$$

A cada pose está associada a captura de índice k correspondente.

[4] http://pointclouds.org/

4.3.1 Pose inicial

A pose inicial define a orientação da câmara para cada eixo, dada pela matriz R_0, e a translação inicial pelo vetor $t_0 = (x_0, y_0, z_0)^T$, originando a matriz seguinte,

$$P_0 = \begin{bmatrix} & R_0 & & \begin{matrix} x_0 \\ y_0 \\ z_0 \end{matrix} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.15}$$

Foi utilizado um acelerómetro (ADXL335), por cima do sensor *Kinect*, de modo a obter a orientação da câmara em relação ao solo para que posteriormente, no planeamento de trajetórias, possa ser removido o chão.

Considerando o sistema de coordenadas da Figura 4.4 e as acelerações lidas pelo acelerómetro, G_x, G_y, G_z, sobre os eixos xx, yy e zz, respetivamente, é possível determinar a rotação sobre o eixo dos xx, por um ângulo α, e sobre o eixo dos zz, por um ângulo γ a partir das seguintes expressões [26],

$$\alpha = tan^{-1}\left(\frac{-G_z}{\sqrt{{G_x}^2 + {G_y}^2}}\right) \tag{4.16}$$

$$\gamma = tan^{-1}\left(\frac{G_x}{G_y}\right) \tag{4.17}$$

Figura 4.7: Acelerómetro de três eixos (ADXL335), montado sobre o sensor *Kinect*.

De modo a filtrar as rotações obtidas (contaminadas com ruído e outras incertezas) foi utilizado um filtro de Kalman [27], para estimar resultados que se aproximam dos valores reais dos ângulos α e γ, medidos.

Com $x = [\alpha \quad \gamma]^T$, o modelo para o filtro de Kalman, assume que o estado real no tempo k é obtido através do estado no tempo, $(k-1)$, de acordo com,

$$x_k = Ax_{k-1} + Bu_k + w_k \tag{4.18}$$

em que,

- A_k, é o modelo de transição de estados, aplicado no estado anterior $x_{(k-1)}$;
- B_k, é o modelo das entradas de controlo, aplicado no vetor de entradas de controlo, u_k;
- w_k, é o ruído do processo, assumido como sendo amostrado de uma distribuição normal multivariada de média zero e covariância Q_k. $w_k \sim N(0, Q_k)$;

No tempo k, uma medição z_k, no estado real x_k, é feita de acordo com a expressão,

$$z_k = H_k x_k + v_k \tag{4.19}$$

Em que,

- H_k é o modelo de observação, que mapeia o espaço de estados real no espaço de estados observados;
- v_k, é o ruído da medição, assumido como sendo um ruído branco gaussiano de média zero e covariância, R_k. $v_k \sim N(0, R_k)$;

O ciclo do filtro de Kalman é definido pelas seguintes expressões:

Previsão do estado,

$$x_k = Ax_{k-1} + Bu_k \tag{4.20}$$

Em que,

- O modelo de transição de estados, $A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$;
- $B = 0$.

Previsão da matriz de covariância P,

$$P_k = AP_{k-1}A^T + Q \tag{4.21}$$

Com,

- $Q = \begin{bmatrix} q1 & 0 \\ 0 & q2 \end{bmatrix}$.

Renovação, que compara os estados previstos a partir do modelo, x_k, com aquilo que é efetivamente medido, z_k,

$$y_k = z_k - Hx_k \tag{4.22}$$

Em que,

- $H = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$;
- $z = [\alpha \quad \gamma]^T$, representa as rotações, sobre os eixos longitudinal e transversal, obtidas a partir dos valores lidos pelo acelerómetro nas equações (4.16) e (4.17).

Renovação da covariância,

$$S_k = HP_kH^T + R \tag{4.23}$$

em que,

- $R = \begin{bmatrix} r1 & 0 \\ 0 & r2 \end{bmatrix}$.

Ganho de Kalman,

$$K_k = P_k H^T S_k^{-1} \tag{4.24}$$

Atualização do estado,

$$x_k = x_k + K_k y \tag{4.25}$$

Atualização da covariância,

$$P_k = (I - K_k H) P_k \tag{4.26}$$

O filtro de kalman é normalmente representado em duas etapas: **previsão** e **atualização**.

A primeira etapa, **previsão**, é composta pelas equações (4.20) e (4.21).

A segunda etapa, **atualização**, é composta pelas equações, (4.22), (4.23), (4.24), (4.25) e (4.26).

O filtro pode ser ajustado, alterando as matrizes de covariância Q e R, mais concretamente $q1$, $q2$, $r1$ e $r2$. Neste projeto estes valores foram ajustados empiricamente.

Depois de filtrados os ângulos, obtidos a partir dos dados medidos pelo acelerómetro, é possível calcular a matriz de rotação inicial, R_0, da seguinte maneira,

$$R_0 = R_x R_y R_z \tag{4.27}$$

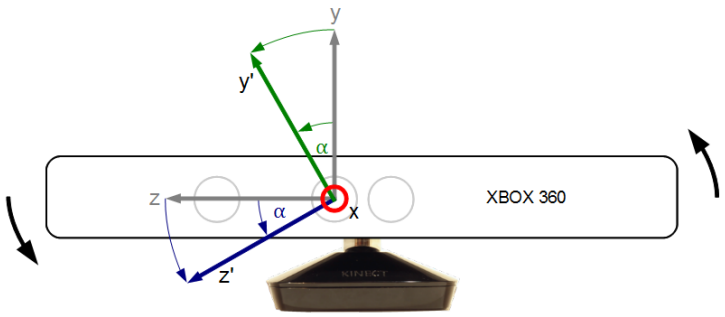

Figura 4.8: Rotação sobre o eixo longitudinal.

em que, R_x é a matriz de rotação sobre o eixo dos xx, por um ângulo α (Figura 4.8),

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\,\alpha & -sin\,\alpha \\ 0 & sin\,\alpha & cos\,\alpha \end{bmatrix} \tag{4.28}$$

R_y é a matriz de rotação sobre o eixo dos yy, por um ângulo β,

$$R_y = \begin{bmatrix} cos\,\beta & 0 & sin\,\beta \\ 0 & 1 & 0 \\ -sin\,\beta & 0 & cos\,\beta \end{bmatrix} \tag{4.29}$$

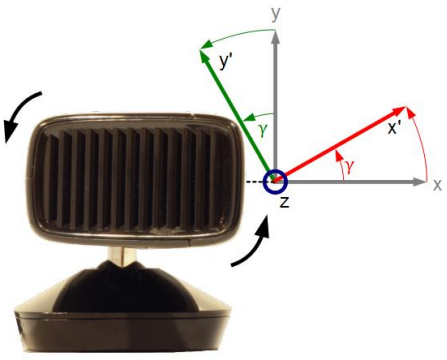

Figura 4.9: Rotação sobre o eixo transversal.

e R_z é a matriz de rotação sobre o eixo dos zz, por um ângulo γ (Figura 4.9),

$$R_z = \begin{bmatrix} cos\,\gamma & -sin\,\gamma & 0 \\ sin\,\gamma & cos\,\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.30}$$

$$R_0 = \begin{bmatrix} cos\,\beta\,cos\,\gamma & -cos\,\beta\,sin\,\gamma & sin\,\beta \\ sin\,\alpha\,sin\,\beta\,cos\,\gamma + cos\,\alpha\,sin\,\gamma & -sin\,\alpha\,sin\,\beta\,sin\,\gamma + cos\,\alpha\,cos\,\gamma & -sin\,\alpha\,cos\,\beta \\ -cos\,\alpha\,sin\,\beta\,cos\,\gamma + sin\,\alpha\,sin\,\gamma & cos\,\alpha\,sin\,\beta\,sin\,\gamma + sin\,\alpha\,cos\,\gamma & cos\,\alpha\,cos\,\beta \end{bmatrix} \tag{4.31}$$

Atribuindo um valor a β, por exemplo, $\beta = 0$, basta substituir os valores, α e γ, depois de filtrados pelo filtro de Kalman, para obter a rotação inicial R_0. A translação inicial pode ser dada na origem, $t_0 = (0,0,0)^T$, ou com quaisquer outros valores de x_0, y_0 e z_0, se pretendido.

4.4 Grafo de poses

O problema SLAM pode ser resolvido com uma otimização pelo método dos mínimos quadrados, depois de ter sido descrito por um grafo, composto por vértices e arestas. Cada vértice representa uma pose/posição da câmara/robô, e uma aresta representa uma transformação ou restrição espacial entre vértices. A representação por um grafo não só providencia uma representação intuitiva e compacta do problema, mas é também a base para a otimização matemática do problema, em que se pretende encontrar a configuração mais provável de vértices.

A equação (4.14) dá uma estimativa da pose da câmara/robô que pode ser inserida num grafo utilizando a *framework* [5] g^2o [28]. No entanto, uma sequência de capturas pode conter um número elevado de poses parecidas, especialmente quando o robô está parado.

Acumulando o movimento do robô entre cada duas capturas, $(i-1)$ e i, a partir da transformação 3D que as une, T_{i-1}^{i}, é possível saber quanto o robô se movimentou em qualquer captura de uma dada sequência. Posto isto, apenas são inseridas no grafo novas poses, assim que o robô se movimente acima de um determinado limite estabelecido de rotação ou

[5] http://openslam.org/g2o.html

translação, nos três eixos acumulados. Estas poses são chamadas de poses-chave para se diferenciarem das outras. [29]

Para cada pose-chave, um vértice é adicionado ao grafo com uma aresta que o liga à pose-chave anterior.

Por exemplo, considerando 6 capturas sequenciais de imagens; um limite mínimo estabelecido de 20 graus de rotação, e 0.5 metros de translação para que uma pose possa ser considerada de pose-cave. Considerando também que apenas nas capturas 3 e 6, o robô se movimentou acima do limite estabelecido de rotação ou translação,

Os vértices, V_j e arestas, E_j, de índice j, para a captura 3 (Pose 3, P_3) e captura 6 (Pose 6, P_6), bem como para a pose inicial P_0, podem ser adicionados da seguinte maneira (Figura 4.10),

Para P_0,

$$V_0 = P_0 \tag{4.32}$$

Para P_3,

$$P_3 = T_2^3 T_1^2 T_0^1 P_0 \tag{4.33}$$

$$V_1 = P_3 \tag{4.34}$$

$$E_0 = T_2^3 T_1^2 T_0^1 \tag{4.35}$$

Para P_6,

$$P_6 = T_5^6 T_4^5 T_3^4 T_2^3 T_1^2 T_0^1 P_0 \tag{4.36}$$

$$V_2 = P_6 \tag{4.37}$$

$$E_1 = T_5^6 T_4^5 T_3^4 \tag{4.38}$$

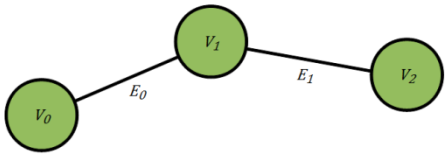

Figura 4.10: Grafo composto por vértices e arestas.

4.5 Fecho do *Loop*

Ao longo do tempo e movimento, o erro do mapa é acumulado, já que as poses atuais são calculadas a partir das anteriores. Por exemplo, a pose, $P_6 = T_5^6 T_4^5 T_3^4 T_2^3 T_1^2 T_0^1 P_0$, ou $P_6 = T_5^6 P_5$, acumula qualquer erro que tenha existido até P_5, mais um possível erro que exista na transformação 3D, entre as capturas 5 e 6, T_5^6.

A deteção de fecho do *loop*, ou *loop closure detection*, em inglês, ocorre quando o robô se move para uma posição que já tinha sido visitada no passado. Se a pose-chave atual, contiver similaridades suficientes com uma outra passada, significa que a posição atual tem uma grande probabilidade de já ter sido visitada, e uma nova restrição espacial pode ser introduzida no grafo.

Estas novas restrições permitem a otimização e atualização do grafo minimizando o erro que tinha sido acumulado. [29]

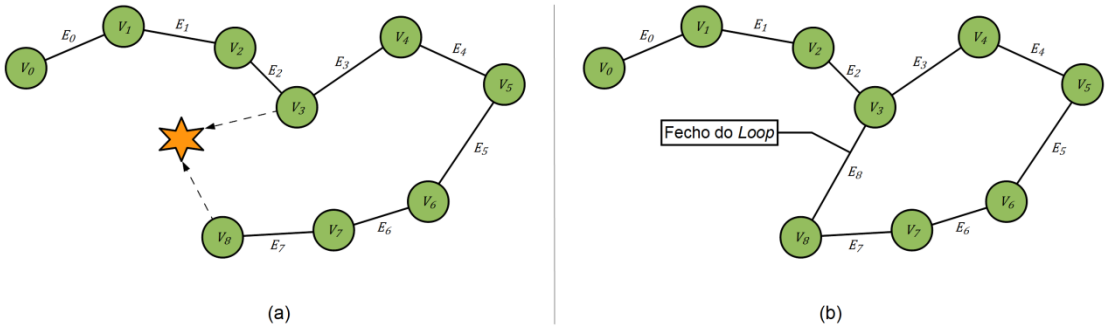

Figura 4.11: Representação do fecho do *Loop*

A Figura 4.11 exemplifica este processo. Foi identificado que a pose-chave atual possui semelhanças significativas com uma anterior (Figura 4.11a), e por isso uma nova aresta que as une é adicionada ao grafo (Figura 4.11b).

A deteção de poses semelhantes é efetuada comparando a pose atual com poses anteriores seguindo a ordem da sequência. A comparação é feita através da transformação 3D

entre duas poses (Secção 4.2). Primeiro estabelece-se um valor mínimo para a razão de *inliers* da transformação (no âmbito deste projeto foi estabelecido um valor de 50%, determinado empiricamente). Se a razão de *inliers* da transformação determinada for maior que este valor significa que as duas poses são semelhantes e uma nova aresta é adicionada ao grafo.

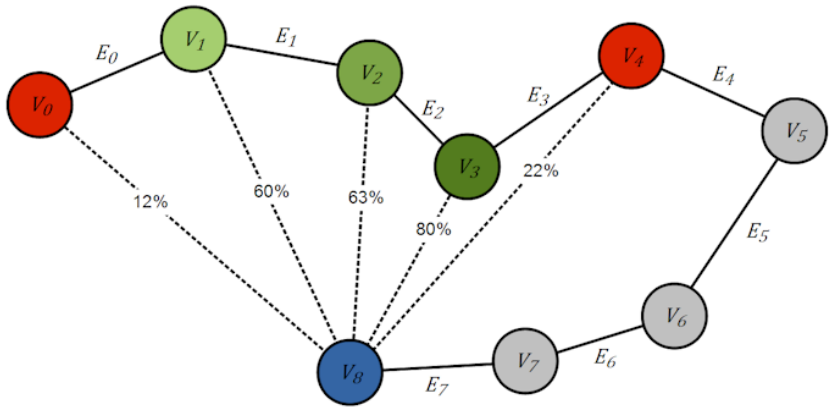

Figura 4.12: Comparação da pose-chave atual com anteriores.

A Figura 4.12 ilustra a comparação da pose/vértice atual, V_8, com poses passada. A comparação inicia-se entre pose-chave atual e a inicial, V_0. O processo é repetido para as poses seguintes, seguindo a ordem da sequência. Assim que a razão de *inliers* apresentar um valor superior a um valor estabelecido, é efetuada a transformação com poses seguintes, para averiguar se existe uma transformação que apresente melhores resultados. Assim que a razão de *inliers* decrescer, o processo termina, e uma restrição espacial entre a pose-chave atual e a que obteve maior razão de *inliers* (entre V_8 e V_3 na Figura 4.12), pode ser adicionada ao grafo, a partir da *framework* g^2o.

Ao invés de comparar a pose-chave atual com todas as outras anteriores, foi reduzida a lista de candidatos, de modo a reduzir o número de comparações, e aumentar o desempenho. Para isso foi utilizada uma janela de tamanho fixo k, que contém poses selecionadas de uma lista de F capturas, composto por todas as poses-chave, exceto as mais recentes. A seleção é feita, dando maior probabilidade às temporalmente mais distantes, já que o fecho do *loop* é de mais interesse se for detetado em capturas mais antigas.

O número de capturas recentes ignoradas, bem como o tamanho da janela podem ser regulados por parâmetros.

Considerando o número de capturas F, a função que retorna a probabilidade para cada uma das poses-chave, de índice i, dando maior prioridade às mais antigas, pode ser dada por,

$$p(i) = \frac{i}{S} \tag{4.39}$$

Com,

$$S = \frac{F \times (F+1)}{2} \tag{4.40}$$

Por exemplo, de um conjunto de 10 poses-chave, um tamanho da janela de 4 $(k = 4)$, e que se pretendem ignorar as 4 poses-chave mais recentes, o tamanho de F, pode ser dado pela diferença entre as poses-chave existentes e o número de poses-chave recentes que se pretendem ignorar, ou seja,

$$F = 10 - 4 = 6$$
$$S = 21$$

Portanto, olhando para a equação (4.39), a mais recente pose do conjunto, tem uma probabilidade de $1/21$ de ser escolhida, a segunda de $2/21$, e as seguintes por ordem crescente de probabilidade e de antiguidade $3/21$, $4/21$, e $5/21$ e $6/21$.

O primeiro elemento da janela de tamanho 4 $(k = 4)$, deve ser selecionado segundo estas condições.

Na segunda iteração, sabendo que as poses da janela não devem ser repetidas, o tamanho de F deverá ser igual ao conjunto da primeira iteração exceto a pose escolhida.

Assim sendo, o segundo elemento da janela deve ser selecionado segundo as probabilidades: $1/15$ para a mais recente pose do conjunto F, e, $2/15$, $3/15$, $4/15$ e $5/15$, para as seguintes por ordem crescente de antiguidade.

O processo deve ser repetido até a janela ficar completa. Depois, deve-se efetuar a transformação 3D entre a pose-chave atual e todos os elementos da janela, e é adicionada ao grafo, uma restrição espacial entre a pose-chave atual e a que obteve maior razão de *inliers*, desde que seja superior ao valor mínimo estabelecido.

4.6 Otimização do Grafo

Depois de inseridas as poses no grafo, bem como restrições espaciais obtidas a partir da deteção do fecho do *loop* (Figura 4.13a), é possível otimizá-lo através da *framework* g^2o. Posteriormente, os vértices são corrigidos (Figura 4.13b), e é possível extrair a estimativa das poses do robô/câmara na forma de matrizes 4x4.

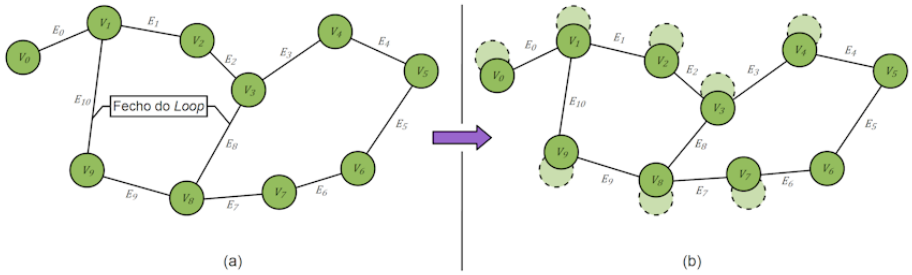

Figura 4.13: Otimização do Grafo.

4.7 Reconstrução 3D

Finalmente, tendo um mapa constituído pelas poses do robô/câmara corrigidas, é possível efetuar a sua reconstrução 3D. Para cada uma das poses é possível gerar uma nuvem de pontos a partir das capturas RGB e de profundidade correspondentes (Figura 4.14). Contudo para gerar um ambiente 3D completo, as capturas têm de ser combinadas num único ponto de vista. A este processo chama-se registo em que cada nuvem de pontos é transformada, projetando todos os seus pontos de acordo com a pose associada, dada na equação (4.14). Depois disto, as nuvens de pontos podem ser encadeadas, resultando num mapa 3D completo (Figura 4.15).

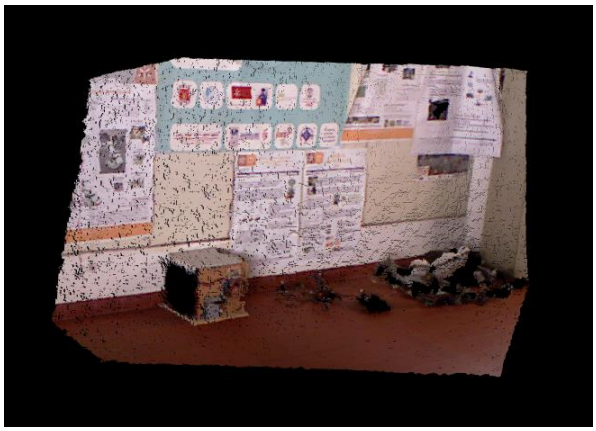

Figura 4.14: Visualização de uma nuvem de pontos, por intermédio da biblioteca *Point Cloud Library*.

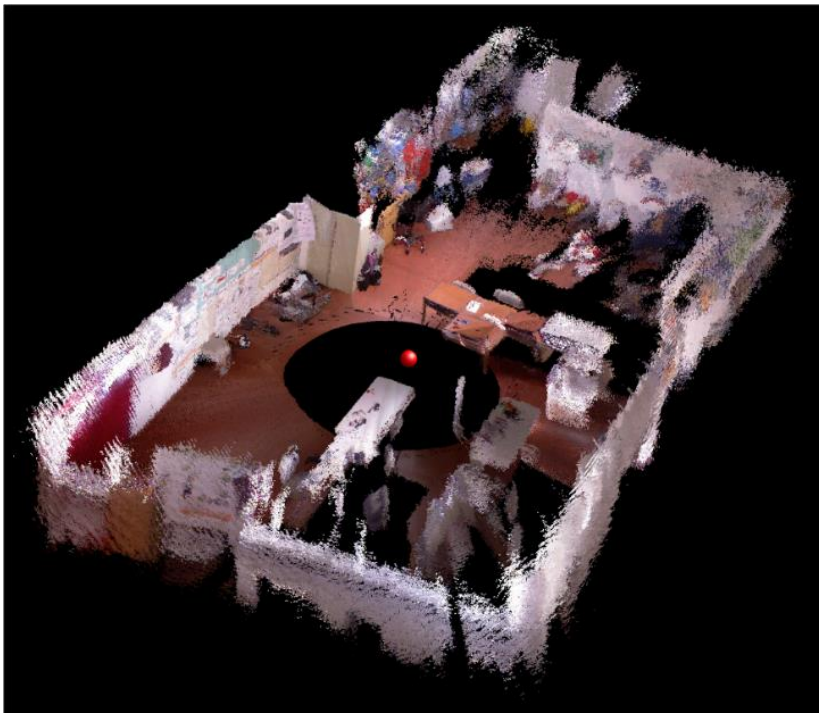

Figura 4.15: Mapa 3D construído depois do robô ter rodado 360 graus sobre o seu eixo vertical, com a localização da câmara representada pelo círculo vermelho.

4.8 Resumo

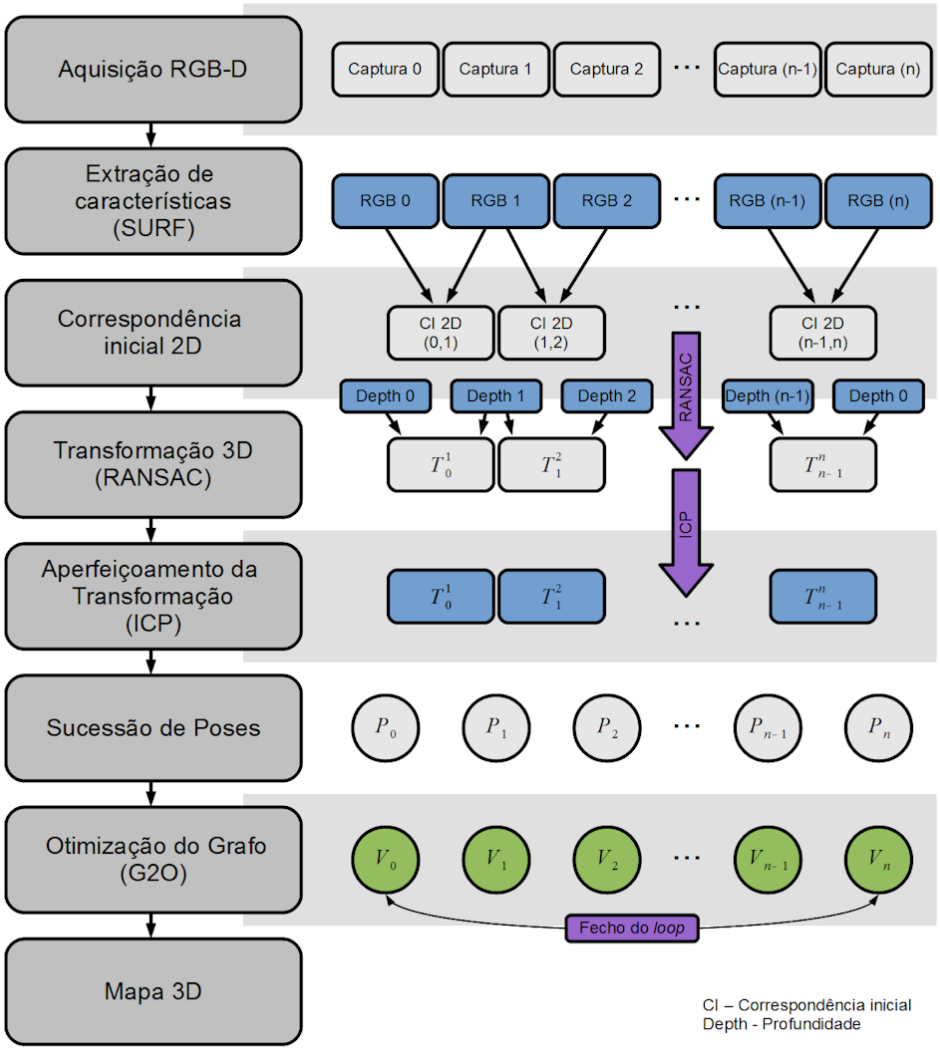

Figura 4.16: Visão geral da construção do mapa 3D.

A Figura 4.16 é uma representação geral dos passos necessários para a criação de um mapa 3D com o sensor *Kinect*, que podem ser descritos da seguinte forma:

- Aquisição de capturas RGB-D a partir do *Kinect;*
- Extração de características (SURF) na imagem RGB;
- Correspondência inicial, em 2D, entre pares de imagens RGB;
- Conversão dos pontos da correspondência inicial, de 2D para 3D, e determinação da transformação 3D utilizando o algoritmo RANSAC;
- Aperfeiçoamento da transformação utilizando o algoritmo ICP;

- Determinação da sucessão de poses da câmara/robô, a partir das transformações 3D calculadas;
- Deteção de fecho de *loop*, nas poses da câmara, representadas por um grafo, e inserção das arestas correspondentes ao mesmo;
- Otimização do grafo e extração das poses câmara atualizadas;
- Reconstrução global 3D.

Capítulo 5

Localização Global

A técnica SLAM, por si só já permite ao robô que se localize enquanto, simultaneamente constrói o mapa do ambiente observado. No entanto a localização é sempre relativa à pose inicial. A partir do momento em que o robô "acorda" numa posição desconhecida e possui, em memória, o mapa do local em que se encontra, pretende-se que este se localize relativamente ao mesmo. É possível completar essa tarefa, adquirindo uma imagem RGB-D, e compará-la com todas as capturas associadas a cada uma das poses que fazem parte do mapa. A comparação é feita, determinando a transformação 3D (capítulo 4.2) entre cada uma das poses e a captura atual. A transformação que apresentar maior razão de *inliers,* desde que válida, é aceite como aquela que une a posição atual, l, à pose n, que apresentou melhores resultados. Entenda-se por transformação válida, aquela que possui um número de *inliers*, bem como a sua razão em relação ao número total de pares da correspondência inicial, superior a um valor mínimo estabelecido.

A pose atual, pode então ser dada por,

$$P_l = T_n^l P_n \tag{5.1}$$

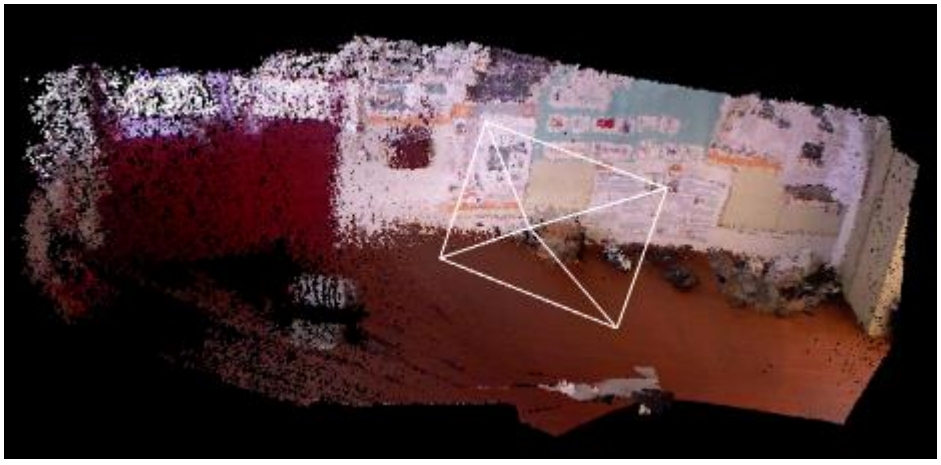

Figura 5.1: Localização do Robô relativamente ao mapa construído.

A Figura 5.1 ilustra a localização do robô, bem-sucedida, relativamente a um mapa previamente conhecido, depois de "acordar" numa posição desconhecida. A pirâmide presente na imagem representa a localização, e orientação da câmara.

Capítulo 6

Robô MARy

O trabalho apresentado neste documento foi efetuado com a intenção de dotar um robô móvel de um sistema que lhe permitisse construir um mapa 3D, localizar-se e planear trajetórias, tendo em conta o mapa elaborado. O robô MARy foi escolhido, um robô desenvolvido pela equipa *MINHO@home* [30], tendo em vista a entrada na competição *Robocup@*home[4]. Possui uma base omnidirecional, um braço robótico articulado com três graus de liberdade, e também uma garra manipuladora de dois graus de liberdade na extremidade do braço. O sistema de visão é composto pelo sensor *Kinect,* utilizada para cumprir os objetivos propostos.

Figura 6.1: Robô MARy.

6.1 Plataforma Omnidirecional

O robô MARy engloba um sistema de locomoção omnidirecional, desenvolvido por Júlio Rodrigues [5]. Esta plataforma é composta por três rodas omnidirecionais, ou suecas,

desfasadas 120 graus entre elas, podendo-se mover lateralmente devido ao conjunto de pequenos cilindros que se situam ao redor da roda.

A comunicação entre o PC e a base é do tipo série com o protocolo RS232. Os dados enviados pelo computador incluem:

- Ângulo de deslocação da plataforma;
- Velocidade linear;
- Velocidade angular.

A equação (6.1) providencia um exemplo de comando que pode ser enviado composto por três conjuntos de dígitos separados por ponto e vírgula. O primeiro representa a velocidade linear em centímetros por segundo, o segundo, o ângulo em graus de deslocação da plataforma e o terceiro, o valor da velocidade angular, em radianos por segundo, multiplicado por cem. O carácter final, f, indica o final do comando.

$$10; 12; 10f \tag{6.1}$$

Este tipo de configuração permite a simplificação do problema de planeamento de trajetória, já que o robô pode rodar sobre um eixo, podendo movimentar-se em qualquer direção sem restrições, tornando-o num robô holonómico. Como consequência, consegue executar qualquer trajeto composto por segmentos de reta (Figura 6.2).

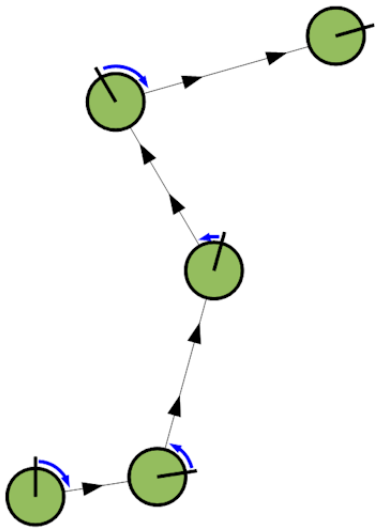

Figura 6.2: Exemplo de execução de um trajeto composto por segmentos de reta.

6.2 Controlo por *Gamepad*

Foi ainda implementado o controlo do robô a partir do gamepad sem fios, Logitech Cordless Rumblepad 2 [31].

Esta solução permite, por exemplo controlar o robô remotamente, enquanto constrói ao mesmo tempo, o mapa do ambiente observado. Para isso foram utilizadas duas *threads* ou fios de execução. A primeira trata do controlo remoto do robô por *gamepad*, enquanto na outra é feita a construção do mapa. Foi utilizada a biblioteca Boost.Thread, pertencente à coleção de bibliotecas multiplataforma Boost. Esta biblioteca, possibilita o uso de múltiplas *threads* ou linhas de execução, com dados partilhados. Providencia classes e funções para administração de *threads,* sincronização de dados, entre outras funcionalidades.

Figura 6.3: Logitech Cordless Rumblepad 2 [31], utilizado para controlar o robô MARy.

Capítulo 7

Planeamento de Trajetória e Contorno de Obstáculos

A tarefa de planeamento de trajetória é geralmente realizada com conhecimento prévio do ambiente. O mapa 3D construído (Capítulo 4.7) serve para este propósito, no entanto é necessário efetuar a sua modelação. Em geral, há duas grandes abordagens para modelar um ambiente em aplicações robóticas: representação contínua, e representação discreta, sendo que a primeira descreve o ambiente com maior precisão.

Na representação discreta, o ambiente é descrito por células que formam uma grelha. Cada célula representa uma área quadrada do ambiente, e armazena um valor que indica o estado de ocupação para esta área. Usualmente as células são classificadas como "ocupada", "livre" ou "desconhecida", ou com um valor que representa a probabilidade da célula estar ou não ocupada. O algoritmo A* (Secção 3.2.1) é um algoritmo que pode utilizar este tipo de representação para planear trajetórias.

Em contraste, existem outros algoritmos, como o RRT (Secção 3.2.2) e RRT* (Secção 3.2.3) que lidam com a abordagem contínua. Uma representação contínua requer a definição dos limites interiores (obstáculos) e exteriores, tipicamente sob a forma de polígonos.

É ainda necessário decidir se o mundo do robô irá ser modulado em 2D ou 3D. O mundo em que vivemos é 3D (na perspetiva do ser humano), tornando-se num fato tentador para modelar o ambiente em três dimensões, no entanto há razões que levam a evitá-lo. À medida que o número de dimensões aumenta, a quantidade de memória e processamento necessários também aumentam. Outra razão prende-se com o fato do robô se movimentar num ambiente interior plano. A modulação 2D é computacionalmente menos dispendiosa e ao mesmo tempo um modelo adequado para o robô planear trajetórias. [32]

Neste trabalho, o mapa 3D construído foi modulado em duas dimensões, e foi utilizado o algoritmo RRT* (Capítulo 3.2.3), por intermédio da biblioteca multiplataforma OMPL[6] (Open Motion Planning Library) [33], que contém implementações de vários algoritmos de planeamento

[6] http://ompl.kavrakilab.org/

estado-da-arte. Sabendo que o robô MARy é holonómico, não foram consideradas restrições no planeamento.

7.1 Mapa 2D

O mapa 3D (Secção 4.7) pode ser projetado em duas dimensões. Primeiro, sabendo que o mapa está referenciado em relação à pose inicial (Capítulo 4.3.1), é possível remover o chão, que não pode ser considerado um obstáculo, mantendo todos os pontos que estão entre a altura mínima e máxima do robô e eliminando todos os restantes.

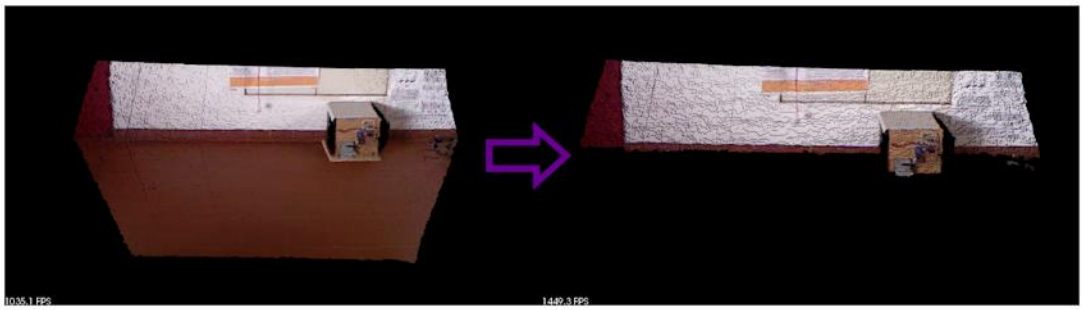

Figura 7.1: Remoção do chão, eliminando todos os pontos que estão abaixo da altura mínima do robô.

Dadas as coordenadas x, y e z de cada um dos pontos do mapa 3D, o conjunto de todos os pontos no plano, P_{2D}, que formam o novo mapa 2D, pode ser dada pela seguinte equação,

$$P_{2D} = \{(x, z) | h_{min} < y < h_{max}\} \tag{7.1}$$

Em que h_{min} e h_{max} são a altura mínima e máxima do robô, respetivamente,

A ilustração seguinte (Figura 7.2) mostra o conjunto de pontos que formam o mapa 3D da Figura 4.15, exceto aqueles que estão abaixo da mínima e acima da máxima altura do robô, projetados sobre o plano xz.

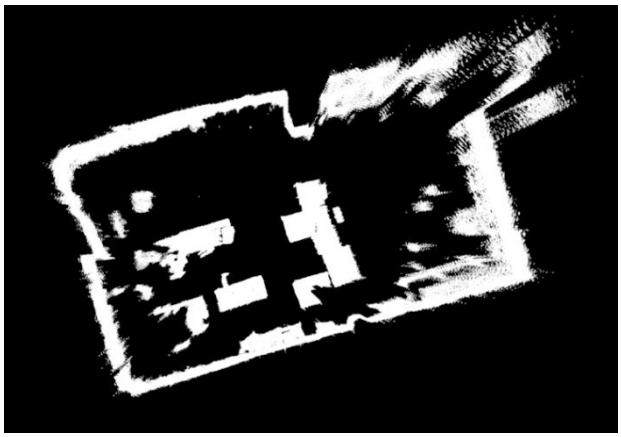

Figura 7.2: Mapa 2D.

7.2 Configuração do espaço

Um passo essencial para simplificar significativamente o planeamento de trajetórias, é a transformação do espaço numa representação em que o robô é visto como um ponto 2D, e o crescimento dos obstáculos de acordo com a forma do robô. Esta representação é chamada Espaço de Configuração, ou C-Space (Configuration Space).

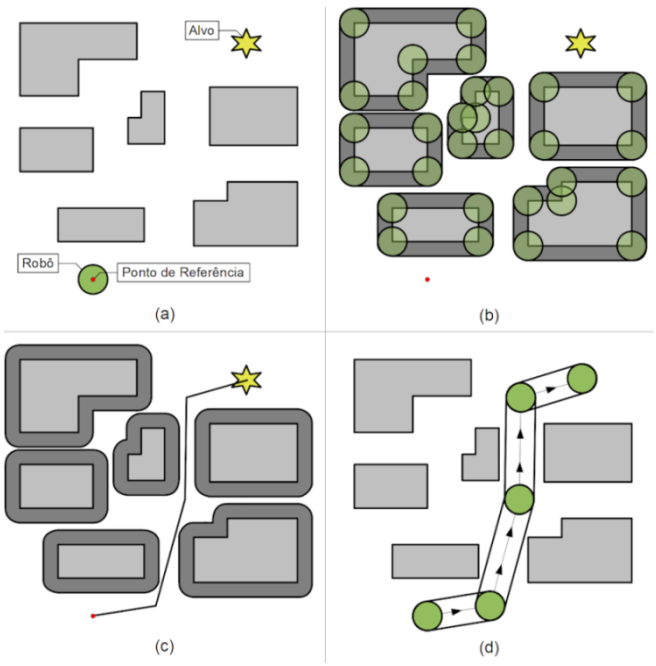

Figura 7.3: Exemplo de criação do Espaço de Configuração.

A Figura 7.3 exemplifica a criação do espaço de configuração, dado um robô holonómico, e um conjunto de obstáculos, representados a cinza claro. Primeiro, é necessário selecionar um ponto dentro ou na borda da figura geométrica que representa a forma do robô (Figura 7.3a). A seguir, são desenhadas novas curvas, conduzindo a forma do robô à volta dos obstáculos, passando por todos os vértices (Figura 7.3b). Estas curvas representam os novos limites dos obstáculos no espaço de configuração, em que o robô é representado pelo ponto de referência.

Por intermédio da biblioteca OpenCV, são determinados os contornos do mapa projetado no plano. Os contornos definidos são compostos por vértices que possibilitam a criação do espaço de configuração do robô.

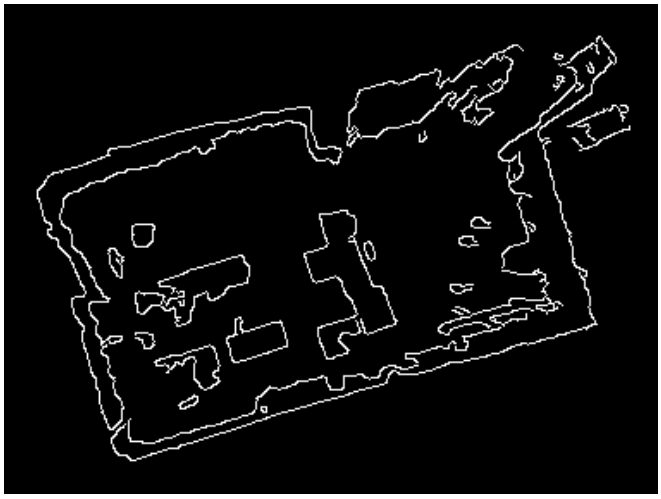

Figura 7.4: Contornos do mapa 2D

A projeção do robô MARy sobre o plano xz, pode ser aproximada a um círculo com 23 centímetros de raio. Assim sendo, é possível formar a configuração do espaço do mapa da Figura 7.4, ou de um outro qualquer, selecionando o centro do círculo que representa o robô, e conduzi-lo à volta dos vértices que constituem os obstáculos (exemplificado na Figura 7.3b), para obter o espaço de configuração da Figura 7.5.

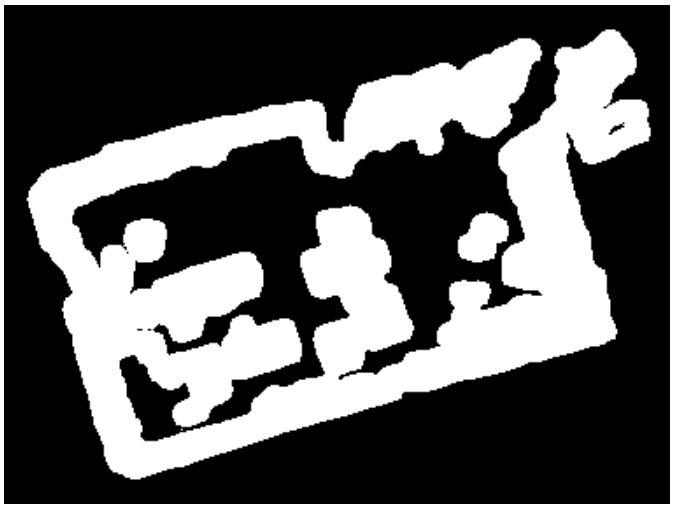

Figura 7.5: Espaço de Configuração.

7.3 Deteção de Colisões no Planeamento

A biblioteca OMPL é flexível e aplicável a uma vasta variedade de sistemas robóticos. Consequentemente, não possui deteção de colisões. Isto é deliberado, já que definir esta noção depende do tipo de problemas a resolver. Por exemplo, a interface gráfica para a biblioteca OMPL, chamada OMPL.app, define a validação de um estado em termos de colisão entre modelos CAD (*Computer Aided Design*). Como resultado, o utilizador deve selecionar uma representação computacional para o robô e providenciar um método explícito de validação/colisão. [34] Neste projeto o robô foi representado por um ponto, e o espaço de configuração criado (Figura 7.5) permite averiguar a validade de um estado. Se um determinado estado do robô, representado por um ponto, estiver dentro de uma região definida como obstáculo (a branco, na Figura 7.5), então o estado é considerado não-válido, se estiver numa região livre (a preto, na Figura 7.5), é considerado válido.

Capítulo 8

Análise de Resultados

Ao longo deste capítulo são apresentados os resultados mais significativos que se obtiveram no decorrer de uma série de experiências, nomeadamente no mapa 3D e respetiva otimização, na localização e no planeamento de trajetória.

A implementação do projeto foi realizada no robô MARy (Capítulo 6) equipado apenas com o sensor Microsoft Kinect (Capítulo 3.3) como ilustra a Figura 6.1 e um acelerómetro colado no topo (Figura 4.7) para obter a pose inicial.

O programa foi executado num PC equipado com um Intel Core i7-3630QM (2.4GHz, 6MB cache), 6GB de memória RAM, a correr no Linux Ubuntu 12.10 (Kernel 3.5.0-27)

8.1 Resultados do Mapa 3D

De modo a verificar a qualidade do mapa foram realizadas algumas experiências. Durante as experiências foram definidos os valores mínimos de 10 centímetros de translação e 5 graus de rotação nos três eixos acumulados para que uma pose pudesse ser inserida no grafo e fazer parte do mapa (pose-chave).

Numa primeira experiência foi movimentado o sensor Kinect à mão, obtendo-se o mapa da Figura 8.1. A segunda experiência consistiu no sensor Kinect colocado por cima do robô MARy que rodou sobre o seu eixo vertical a uma velocidade de 10 graus por segundo, construindo o mapa 3D do Laboratório de Automação e Robótica da Universidade do Minho, com dimensões de aproximadamente 12 metros de comprimento e 7 metros de largura, em 36 segundos (Figura 8.2).

O sistema foi capaz de processar entre 9 e 14 capturas por segundo ao longo da construção do mapa. Mais especificamente, em relação aos algoritmos do sistema, obtiveram-se os seguintes tempos, por captura, durante a elaboração dos mapas:

- Cerca de 50 milissegundos para a deteção e caracterização de pontos-chave, pelo método SURF;

- Entre 2 e 10 milissegundos para o algoritmo RANSAC;
- Cerca de 8 milissegundos para a correspondência inicial entre características extraídas de duas capturas consecutivas;
- Entre 1 e 5 milissegundos para o algoritmo ICP.

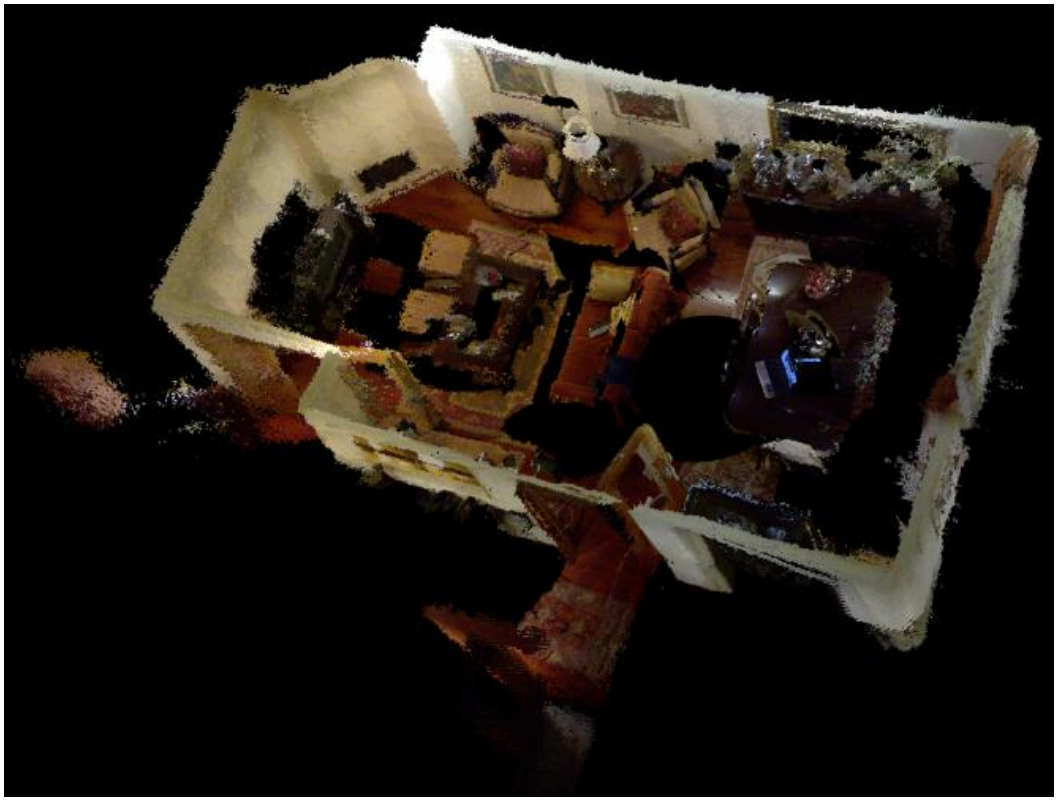

Figura 8.1: Mapa 3D de uma sala, constituído por 95 poses-chave.

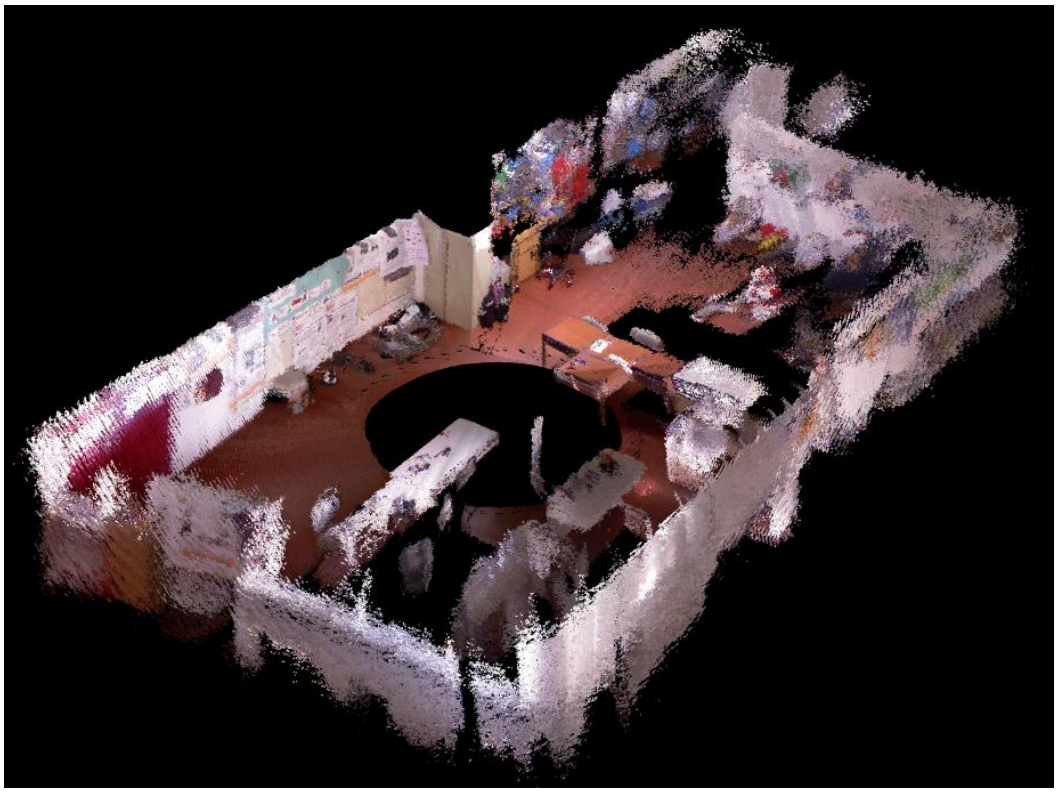

Figura 8.2: Mapa 3D do Laboratório de Automação e Robótica (LAR) da Universidade do Minho, constituído por 60 poses-chave.

As figuras seguintes ilustram alguns mapas antes e depois de otimizados, por consequência da deteção de fecho do *loop* com uma janela de tamanho 10 e excluindo as 5 poses-chave mais recentes.

Figura 8.3: Mapa não otimizado do Laboratório de Automação e Robótica.

Figura 8.4: Mapa otimizado do Laboratório de Automação e Robótica.

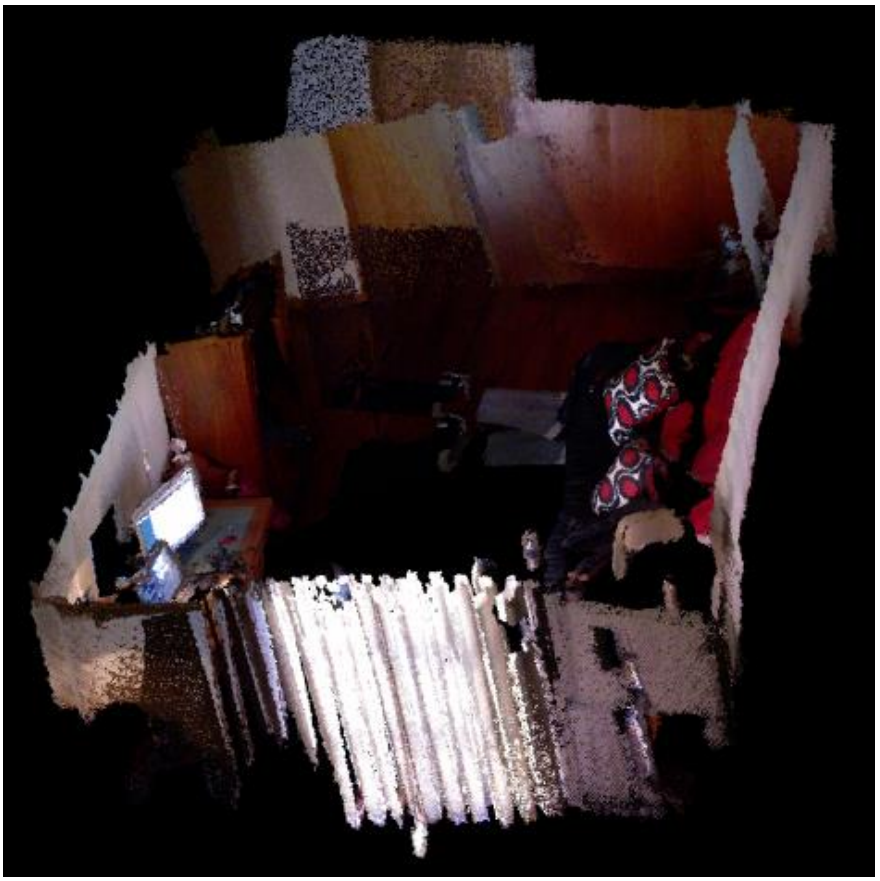

Figura 8.5: Mapa não otimizado de um quarto.

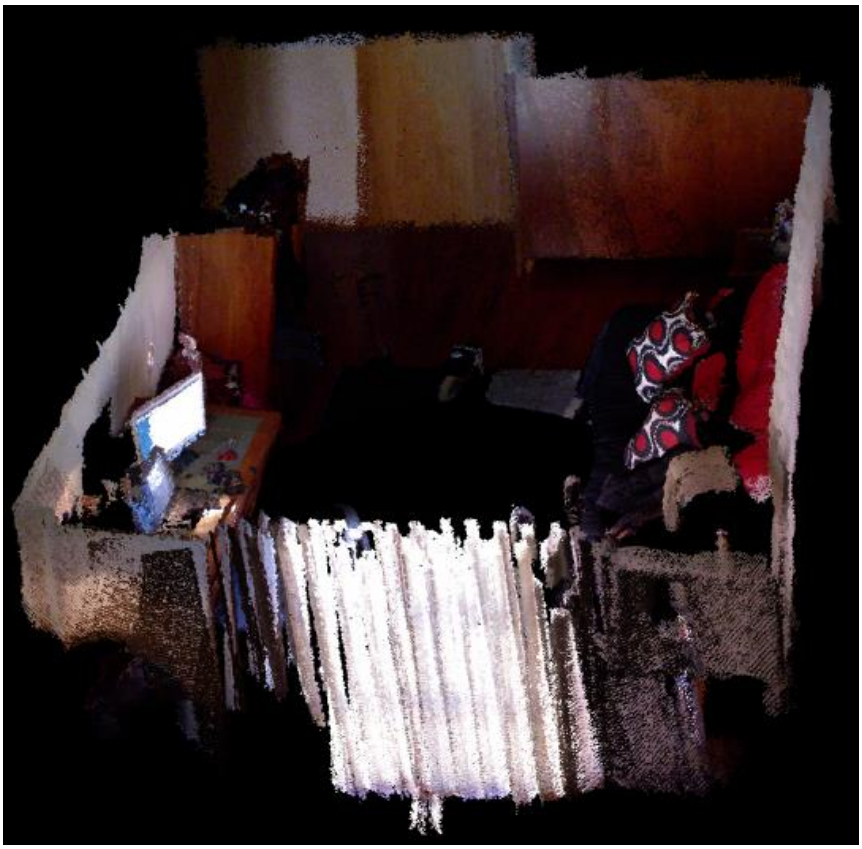

Figura 8.6: Mapa otimizado de um quarto.

Figura 8.7: Mapa não otimizado de uma sala.

Figura 8.8: Mapa otimizado de uma sala.

É possível observar que os mapas não otimizados (Figura 8.3, Figura 8.5 e Figura 8.7) apresentam partes duplicadas e desviadas da posição correta. A otimização do grafo permitiu corrigir estes desvios (Figura 8.4, Figura 8.6 e Figura 8.8).

8.2 Resultados de Localização

Para averiguar a qualidade da localização foi utilizada a seguinte metodologia. Primeiro foi selecionado o mapa sobre o qual se pretendeu localizar a câmara, neste caso, o mapa ilustrado na Figura 8.8. De seguida, foi efetuado o processo de localização (Capítulo 5), colocando a câmara em diferentes posições.

O desfasamento entre a captura atual, projetada pela pose dada na equação (5.1) e o mapa, permite qualificar a transformação e consequentemente a posição da câmara. O alinhamento perfeito entre os dois revela uma localização correta.

Nas figuras seguintes estão presentes os resultados alcançados. Cada figura é constituída pela captura ou perspetiva da câmara no momento em que se efetuou a localização; pela captura atual projetada e mapa combinados, representados a diferentes cores para se tornarem distinguíveis, e pela mesma combinação, desta vez com as cores originais dadas pela câmara RGB.

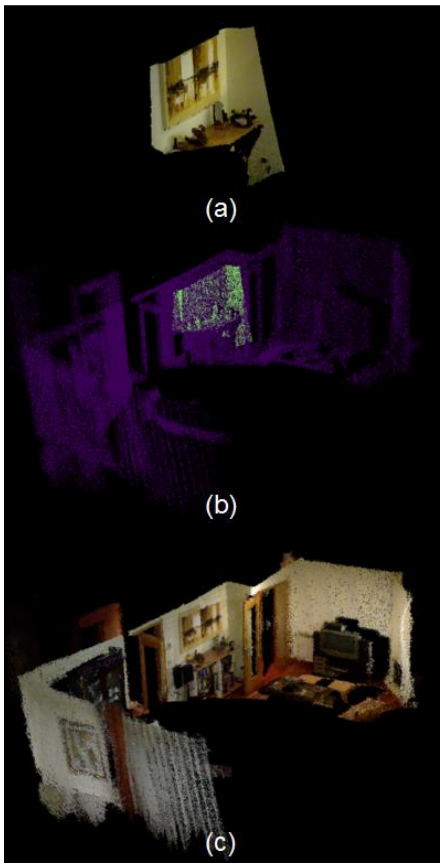

Figura 8.9: Primeiro teste de localização.

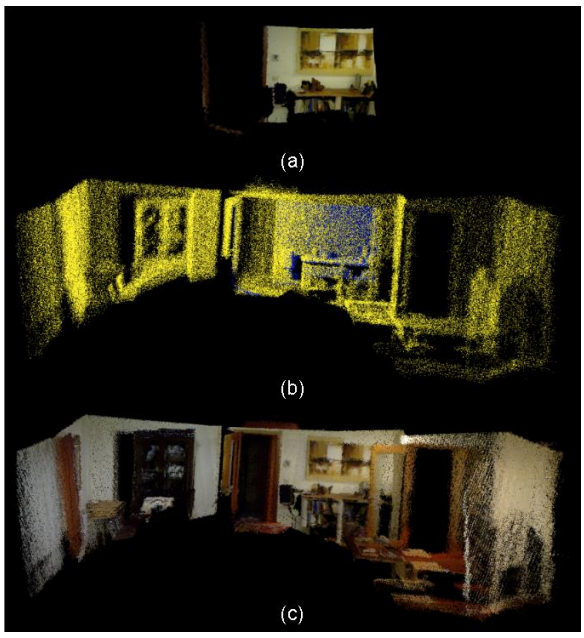

Figura 8.10: Segundo teste de localização.

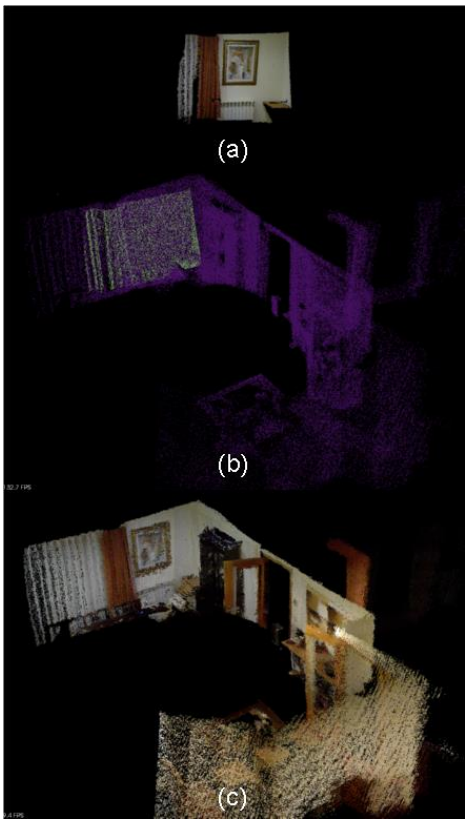

Figura 8.11: Terceiro teste de localização.

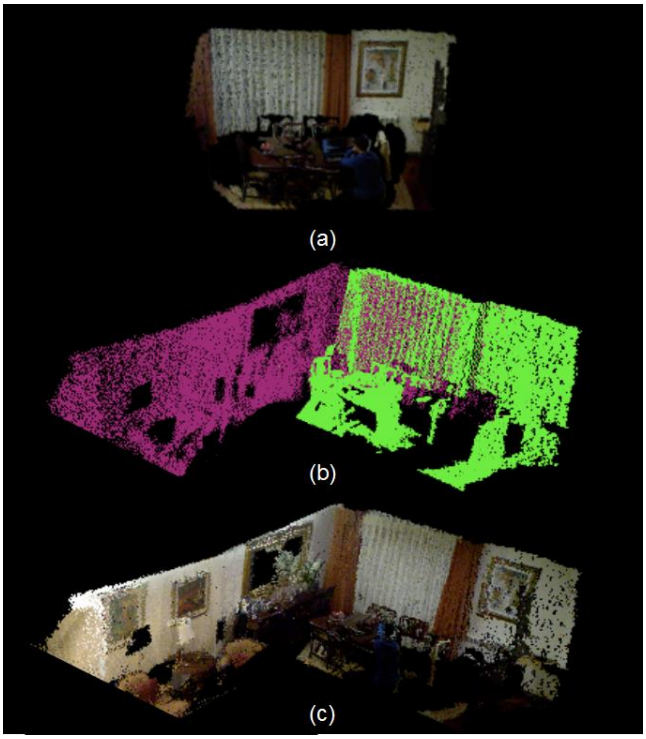

Figura 8.12: Quarto teste de localização.

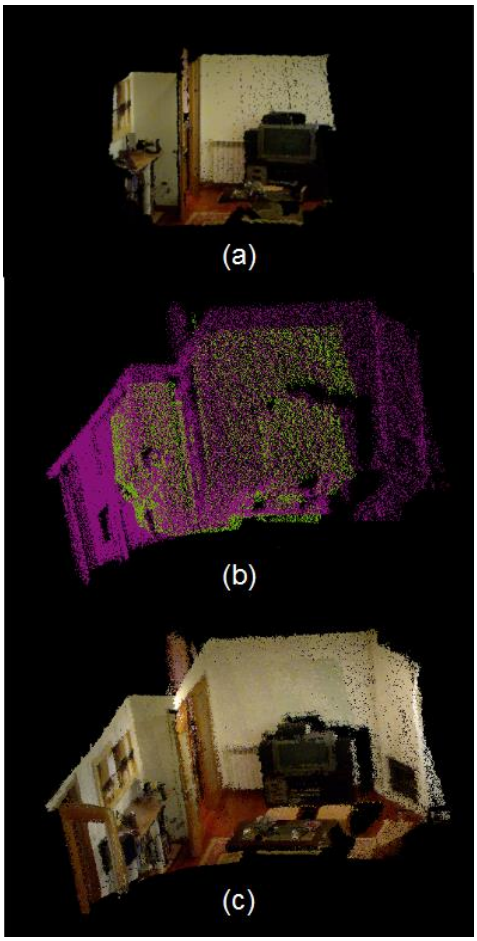

Figura 8.13: Quinto teste de localização.

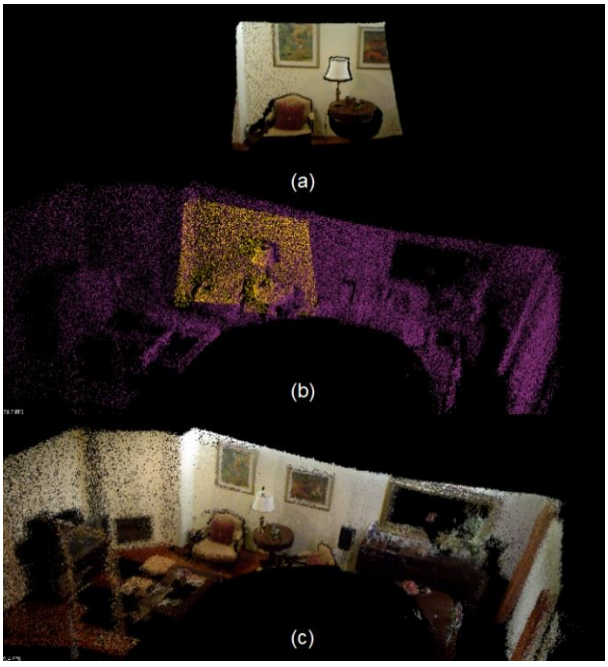

Figura 8.14: Sexto teste de localização

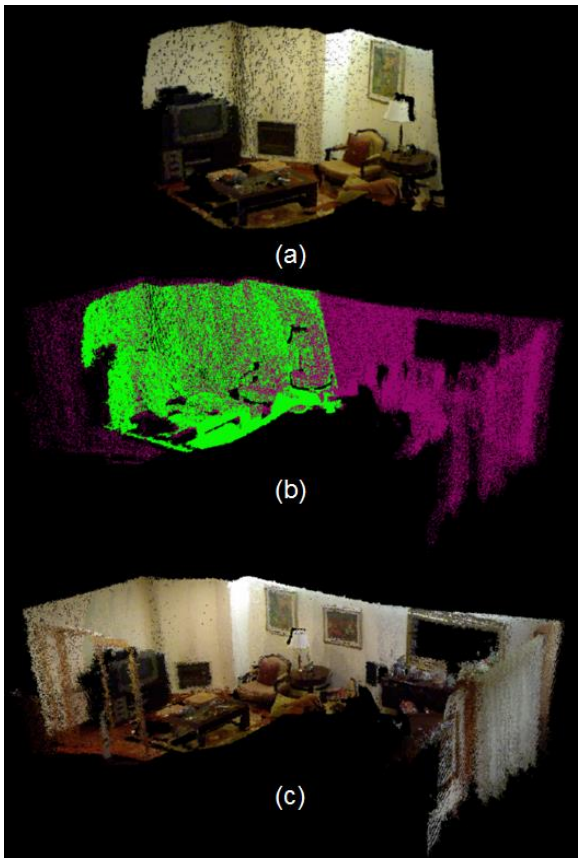

Figura 8.15: Sétimo teste de localização.

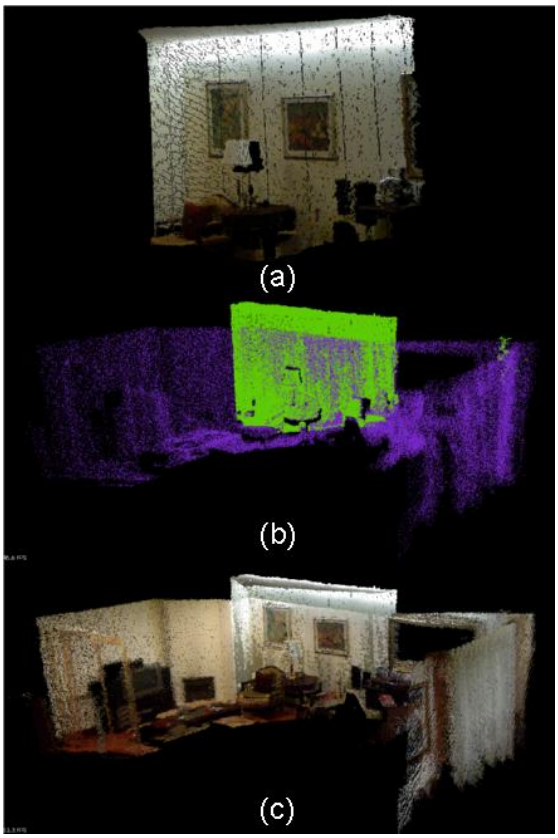

Figura 8.16: Oitavo teste de localização.

As figuras seguintes ilustram casos em que não foi possível determinar a localização da câmara em relação ao mapa construído,

Figura 8.17: Nono teste de localização.

Figura 8.18: Décimo teste de localização.

Figura 8.19: Décimo primeiro teste de localização.

Dos testes realizados, verificou-se que a projeção das capturas no momento da localização (da Figura 8.9(a) à Figura 8.16(a)), a partir das transformações determinadas (quando válidas), resultaram num alinhamento quase perfeito relativamente ao mapa construído (Figura 8.9(b) à Figura 8.16(b)) tornando-se praticamente indistinguíveis (Figura 8.9(c) à Figura 8.16(c)).

No entanto, existem situações em que não é possível obter uma transformação válida, nomeadamente, quando a câmara se encontra em frente a obstáculos sem elementos que o possam caracterizar (Figura 8.17) ou quando a perspetiva da câmara é substancialmente diferente de qualquer uma das perspetivas associadas às poses que fazem parte do mapa (Figura 8.18 e Figura 8.19).

8.3 Resultados do Planeamento de Trajetória

Como referido anteriormente foi utilizada a biblioteca OMPL para o planeamento de trajetórias e o algoritmo utilizado foi o RRT*. De seguida apresentam-se dois trajetos planeados com o algoritmo escolhido.

Os trajetos ilustrados na Figura 8.20 e Figura 8.21 demoraram 0,001767 e 0,120753 segundos a calcular, respetivamente

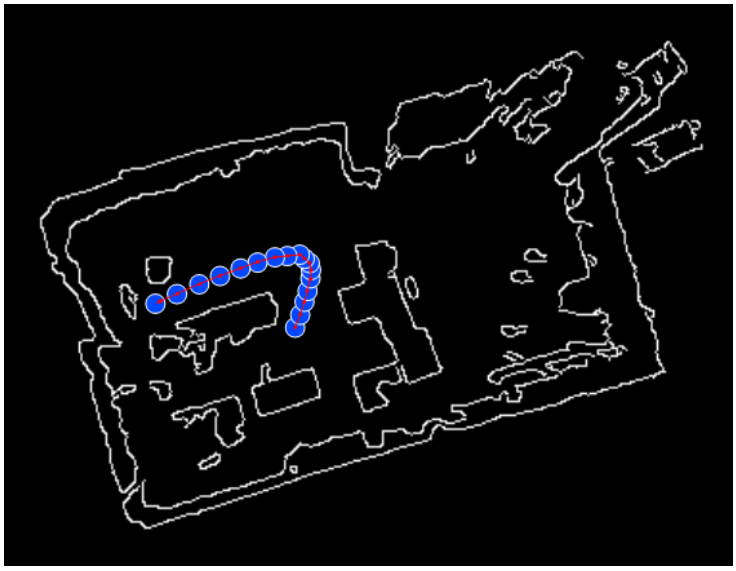

Figura 8.20: Exemplo de trajetória determinada.

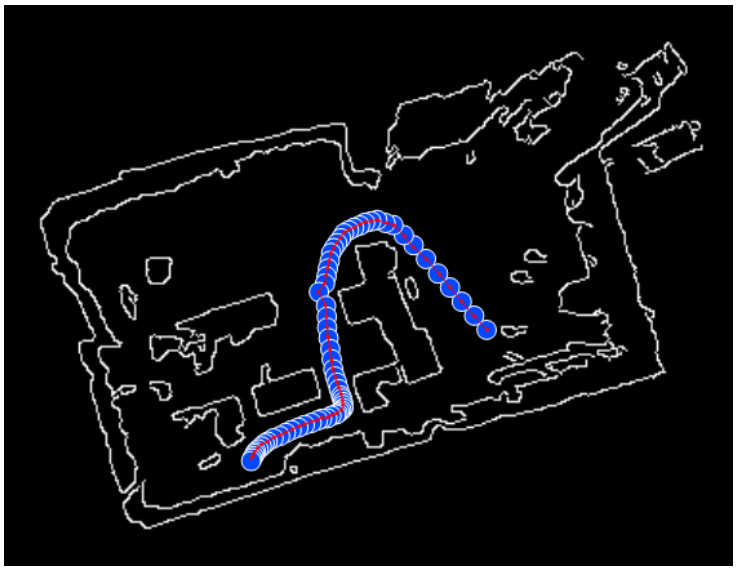

Figura 8.21: Exemplo de trajetória determinada.

Capítulo 9

Conclusões e Trabalho Futuro

9.1 Conclusões

O principal objetivo do trabalho apresentado nesta dissertação residiu no desenvolvimento de um sistema de visão robótico, capaz de construir um modelo do ambiente envolvente enquanto se localiza, e no planeamento de caminhos livres de obstáculos. Para tal, foram estudadas e desenvolvidas soluções para resolver o problema SLAM com o sensor Microsoft Kinect, e ainda de planeamento de trajetória.

Os elementos necessários ao sistema foram reunidos e integrados numa aplicação informática.

As experiências feitas revelam que é possível **construir mapas 3D** e ainda otimizá-los com este sistema. Ainda que os resultados tenham sido satisfatórios para muitos dos casos, existem situações em que não é possível construir um mapa, nomeadamente pelas limitações de profundidade do sensor Kinect, quando se encontra sob condições de luminosidade desfavoráveis e também quando o número de características extraídas pelo método SURF é insuficiente para uma correspondência entre capturas satisfatória, por exemplo quando a câmara se encontra em frente a uma parede, porta ou outro obstáculo sem elementos que o possam caracterizar.

Na **localização**, existiram maiores dificuldades em obter a pose da câmara em relação a um mapa que possui em memória, quando a sua perspetiva é substancialmente diferente daquela, ou daquelas, caso se tenha movimentado a câmara, em que foi construído o mapa. No entanto este aspeto poderá ser aperfeiçoado, construindo o mapa sobre múltiplas perspetivas.

No que diz respeito ao **planeamento de trajetórias**, o algoritmo revelou-se eficiente e foi sempre capaz de encontrar um trajeto para o alvo pretendido.

A **implementação** deste sistema tirou proveito da linguagem de programação c++, orientada a objetos, para que intuitivamente se possam integrar diversas funcionalidades do sistema. Por exemplo, pode ser facilmente implementada uma verificação de colisão, que consiste em determinar se algum dos pontos do mapa que está a ser construído interseta o caminho que entretanto foi planeado. Embora esta solução tenha sido implementada, não foram

verificados resultados satisfatórios pelas limitações enunciadas acerca da construção do mapa 3D. Por exemplo, o trajeto determinado pelo robô pode passar por zonas em que a construção do mapa seja impossível ou pouco adequada.

9.2 Trabalho Futuro

Ainda que se tenham tentado examinar e implementar todas as ideias que surgiram ao longo do projeto, o tempo impediu de as realizar a todas. Por isto, é de interesse apresentar estas ideias como trabalho que se possa realizar no futuro.

Em relação à construção do mapa 3D e simultânea localização, seria vantajoso integrar e fundir uma unidade inercial de medida (IMU), principalmente quando as transformações 3D determinadas por este método são menos fiáveis ou impossíveis de determinar.

A inclusão de sensores do tipo LRF no robô MARy constituiria também uma mais-valia. Os três robôs finalistas do evento Robocup@home, edições de 2011 e 2012 utilizam este tipo de sensores para elaborar um mapa 2D geral do ambiente, e ainda para localização. Esta solução poderia ser implementada no robô MARy. Pelo seu longo alcance e campo de visão, os sensores LRF fornecem uma boa visão geral, do mapa envolvente. Esta informação poderia ser adicionada à informação 3D obtida pelo sensor Kinect para uma perceção mais completa do ambiente envolvente.

Seria também importante dotar o robô MARy de um sistema de odometria. Não envolve custos elevados, pode fornecer informações importantes na localização e conceder dados que podem ser integrados no sistema para aumentar a robustez da técnica SLAM implementada neste projeto.

Referências

[1] H. Durrant-Whyte and T. Bailey, "Simultaneous Localization and Mapping: Part I," *IEEE Robot. Autom. Mag.*, vol. 13, no. 2, pp. 99–110, Jun. 2006.

[2] T. Emter and A. Stein, "Simultaneous Localization and Mapping with the Kinect sensor," in *Robotik 2012, 7th German Conference on Robotics*, pp. 1–6.

[3] J. E. Young, R. Hawkins, E. Sharlin, and T. Igarashi, "Toward Acceptable Domestic Robots: Applying Insights from Social Psychology," *Int. J. Soc. Robot.*, vol. 1, no. 1, pp. 95–108, Nov. 2008.

[4] "Robocup." [Online]. Available: http://www.robocup.org/.

[5] J. Rodrigues, "Plataforma Omnidireccional para Robô de Serviços em Casa," Universidade do Minho, 2010.

[6] D. Oliveira, "Braço robótico manipulador para aplicação em robô de serviços," Universidade do Minho, 2010.

[7] R. Dwiputra, M. Füller, F. Hegger, N. Hochgeschwender, J. Paulus, I. Awaad, S. Schneider, A. Bierbrauer, E. Shpieva, I. Ivanovska, N. V Deshpande, A. Hagg, J. M. Sánches, A. Y. Ozhigov, P. G. Ploeger, and G. K. Kraetzschmar, "The b-it-bots RoboCup @ Home 2013 Team Description Paper," 2013.

[8] X. Chen, F. Wang, H. Sun, J. Xie, M. Cheng, and K. Chen, "KeJia : The Integrated Intelligent Robot for RoboCup @ Home 2013," 2013.

[9] I. Badami, D. Droeschel, K. Gr, D. Holz, M. Mcelhone, M. Nieuwenhuisen, M. Schreiber, M. Schwarz, and S. Behnke, "NimbRo @ Home : Winning Team of the RoboCup @ Home Competition 2012," Bonn, Germany, 2012.

[10] K. Gr, D. Holz, M. Schreiber, and S. Behnke, "NimbRo @ Home 2011 Team Description," Bonn, Germany, 2011.

[11] H. Okada, T. Omori, N. Watanabe, T. Shimotomai, N. Iwahashi, K. Sugiura, T. Nagai, and T. Nakamura, "Team eR@sers 2012 in the @ Home League Team Description Paper," 2012.

[12] L. Ziegler, J. Wittrowski, M. Sch, F. Siepmann, and S. Wachsmuth, "ToBI - Team of Bielefeld : The Human-Robot Interaction System for RoboCup @ Home 2013," 2013.

[13] R. González, F. Rodríguez, J. L. Guzmán, and M. Berenguel, "Comparative Study of Localization Techniques for Mobile Robots based on Indirect Kalman Filter," 2009, pp. 253–258.

[14] D. Ferguson, M. Likhachev, and A. Stentz, “A Guide to Heuristicbased Path Planning,” in *Proceedings of the Workshop on Planning under Uncertainty for Autonomous Systems, Int. Conf. on Automated Planning and Scheduling (ICAPS’05)*, 2005.

[15] S. M.MasudurRahmanAl-Arif, A. H. M. Iftekharul Ferdous, and S. Hassan Nijami, “Comparative Study of Different Path Planning Algorithms: A Water based Rescue System,” *Int. J. Comput. Appl.*, vol. 39, no. 5, pp. 25–29, Feb. 2012.

[16] S. M. Lavalle, “Rapidly-Exploring Random Trees: A New Tool for Path Planning,” 1998.

[17] S. Karaman and E. Frazzoli, “Sampling-based algorithms for optimal motion planning,” *Int. J. Rob. Res.*, vol. 30, no. 7, pp. 846–894, Jun. 2011.

[18] D. G. Lowe, “Distinctive Image Features from Scale-Invariant Keypoints,” *Int. J. Comput. Vis.*, vol. 60, no. 2, pp. 91–110, Nov. 2004.

[19] T. Lindeberg, “Scale-Space Theory in Computer Vision,” Jan. 1994.

[20] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, “Speeded-Up Robust Features (SURF),” *Comput. Vis. Image Underst.*, vol. 110, no. 3, pp. 346–359, Jun. 2008.

[21] M. A. Fischler and R. C. Bolles, “Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography,” *Commun. ACM*, vol. 24, no. 6, pp. 381–395, Jun. 1981.

[22] Z. Zhang, “Iterative point matching for registration of free-form curves and surfaces,” *Int. J. Comput. Vis.*, vol. 13, no. 2, pp. 119–152, Oct. 1994.

[23] M. A. Akhloufi, “A 2D-3D Hybrid Vision System for Robotic Manipulation of Randomly Oriented Objects,” *World Acad. Sci. Eng. Technol.*, vol. 6, no. 71, pp. 396 – 404, 2012.

[24] D. G. L. Marius Muja, “Fast approximate nearest neighbors with automatic algorithm configuration,” in *International Conference on Computer Vision Theory and Application (VISSAPP’09)*, 2009, pp. 331–340.

[25] R. B. Rusu and S. Cousins, “3D is here: Point Cloud Library (PCL),” in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 1–4.

[26] M. Pedley, “Tilt Sensing Using Linear Accelerometers.” Freescale Semiconductor, 2013.

[27] R. Kalman, “A New Approach to Linear Filtering and Prediction Problems,” *Trans. ASME – J. Basic Eng.*, no. 82 (Series D), pp. 35 – 45, 1960.

[28] R. Kummerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, “g2o: A general framework for graph optimization,” in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 3607–3613.

[29] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox, “RGB-D Mapping: Using depth cameras for dense 3D modeling of indoor environments,” 2010.

[30] F. Ribeiro, G. Lopes, D. Oliveira, F. Gonçalves, and J. Rodrigues, “Minho@home,” Guimarães.

[31] “Logitech.” [Online]. Available: http://www.logitech.com/.

[32] R. Mojtahedzadeh, “Robot Obstacle Avoidance using the Kinect,” Royal Institute of Technology, 2011.

[33] I. Sucan, M. Moll, and L. Kavraki, “The Open Motion Planning Library,” *IEEE Robot. Autom. Mag.*, vol. 19, no. 4, pp. 72–82, Dec. 2012.

[34] Kavraki Lab, “Open Motion Planning Library: A Primer,” 2013.