UNIVERSIDADE D
COIMBRA

João André Marques

NAVEGAÇÃO ROBUSTA DE UM ROBÔ MÓVEL EM AMBIENTES DINÂMICOS ESTRUTURADOS

Dissertação de Mestrado em Engenharia Eletrotécnica e de Computadores orientada pelo Professor Doutor Rui Paulo Pinto da Rocha e apresentada à Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

Setembro de 2019

1 2  9 0

UNIVERSIDADE D
COIMBRA

Navegação robusta de um robô móvel em ambientes dinâmicos estruturados

João André Marques

Setembro de 2019

Navegação robusta de um robô móvel em ambientes dinâmicos estruturados

Orientador:

Professor Doutor Rui Paulo Pinto da Rocha

Júri:

Prof. Doutor Rui Alexandre de Matos Araújo

Doutor Micael Santos Couceiro

Prof. Doutor Rui Paulo Pinto da Rocha

Dissertação de Mestrado em Engenharia Electrotécnica e de Computadores, orientada pelo Professor Doutor Rui Paulo Pinto da Rocha e apresentada à Faculdade de Ciências e Tecnologia da Universidade de Coimbra

Setembro de 2019

Agradecimentos

Em primeiro lugar gostaria de agradecer aos meus pais e irmãos por todo o seu apoio e carinho ao longo de todos estes anos e porque sem eles este trabalho não era possível.

Queria agradecer também ao amigos que encontrei durante o meu percurso académico por tornarem esta viagem mais divertida.

Finalmente queria agradecer ao meu orientador, Professor Rui Rocha, por ter concedido esta oportunidade e pelo aconselhamento durante este trabalho.

Resumo

Os robôs móveis, industriais e de serviço, têm vindo a ser desenvolvidos para uma grande diversidade de aplicações e utilizações. Prevê-se que os robôs móveis se irão tornar quase omnipresentes nos locais de trabalho e nas nossas casas durante as próximas décadas. Para que um robô móvel possa navegar autonomamente, de forma segura e robusta em ambientes estruturados e dinâmicos, o robô móvel precisa de conseguir determinar a sua localização em relação a um mapa, ter a capacidade de percecionar o ambiente em seu redor e planear trajetórias para navegar entre diferentes pontos do ambiente. Neste trabalho foi feito um estudo das técnicas de navegação implementadas no Robot Operating System (ROS), mais concretamente implementadas no pacote de navegação do ROS *move_base*. O objetivo foi o de descobrir e resolver situações críticas que possam por em causa o bom comportamento do robô. Durante o estudo do *move_base* observou-se como uma configuração descuidada para o *move_base* dá origem a comportamentos de risco (e.g. colidir com um obstáculo). Uma afinação dos parâmetros do *move_base* permite a que o robô seja capaz de gerar uma trajetória mais segura e robusta. Foi descoberto que em algumas situações o *move_base* perde a capacidade de manter uma representação correta do mundo. Esta perda pode ser atribuída à incapacidade do *move_base* de utilizar certos tipos de mensagem para limpar do mapa objetos que ja não existam em cena. Estas limitações podem ser ultrapassadas fazendo a conversão dos dados do sensor para um tipo de dados que o *move_base* consiga usar. No final estudou-se como o *move_base* reage na presença de obstáculos dinâmicos, (pessoas e outros robôs). Verificou-se que o *move_base* tem dificuldade em interagir com estes obstáculos. As camadas sociais do ROS tentam dar resposta ao problema da interação. Os estudos realizados revelaram que as camadas sociais não uma boa resposta apresentando muitas limitações.

Abstract

Mobile, industrial and service robots have been developed for a wide range of applications and uses. Mobile robots are expected to become almost ubiquitous at workplaces and in our homes over the next few decades. In order for a mobile robot to be able to navigate autonomously in a secure and robust way in dynamic structured environments, the mobile robot needs to be able to determine its location in relation to a map, have the ability to perceive the surrounding environment and plan Trajectories to navigate between different points of the environment. In this work, a study of the navigation techniques implemented in the Robot Operating System (ROS) was done, more precisely the techniques implemented in the ROS navigation package *move_base*. The objective consists on a study of challenging situations where the ROS navigation stack presents limitations then device and test solutions for such situations. During the study it was observed how a careless configuration for *move_base* gives rise to risky behaviors (e.g. colliding with an obstacle). A tuning of the *move_base* parameters allows the robot to be able to generate a safer and more robust trajectory. It has been found that in some situations *move_base* loses the ability to maintain a correct representation of the world. This loss can be attributed to *move_base*'s inability to use certain message types to clear objects that no longer exist on the map. These limitations can be overcome by converting the sensor data to a data type that *move_base* can use. At the end it was studied how the *move_base* reacts in the presence of dynamic obstacles, (people and other robots). It has been found that *move_base* has difficulty interacting with these obstacles. The social layers of ROS try to address the problem of robot interaction. Studies have shown that social layers are not a good response presenting many limitations.

"The secret to creativity is knowing how to hide your sources."

— Albert Einstein,

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Motivação

Um robô móvel é um robô que tem capacidade de locomoção, por exemplo com rodas. Ao contrário dos robôs manipuladores, que tipicamente estão fixos num local do ambiente, os robôs móveis podem mover-se no ambiente, o que os torna mais flexíveis e lhes permite realizar tarefas em diferentes locais ou que impliquem a movimentação de materiais.

Os robôs móveis industriais têm vindo a ser desenvolvidos para uma grande diversidade de aplicações e utilizações. Atualmente é possível encontrar robôs móveis em muitas indústrias, a executarem tarefas de transporte de materiais entre as várias etapas de processamento de produto; costumam ser designados por *automatic guided vehicles* (AGVs).

Outro importante domínio de aplicação da robótica móvel tem a ver com robôs sociais e robôs de serviços. Um robô social ou um robô de serviço é um robô que opera, interage e comunica com seres humanos. Estes robôs servem para realizar tarefas do dia-a-dia que melhoram a qualidade de vida da pessoas, tais como robôs de companhia para idosos cuja função é estimular um idoso mentalmente e fisicamente, robôs guia em museus que permitem fazer visitas guiadas quando um guia humano não está presente, robôs tarefeiros em escritórios e enfermarias, robôs de vigilância, etc. Prevê-se que os robôs móveis de serviços se irão tornar quase omnipresentes em locais públicos (e.g. centros comerciais, museus, hospitais, etc.), nos locais de trabalho e nas nossas casas durante as próximas décadas.

1.2 Navegação segura e autónoma

Para que um robô móvel industrial ou um robô móvel de serviços possa navegar autonomamente, de forma segura e robusta em ambientes estruturados e dinâmicos, o robô móvel precisa de:

- Conseguir determinar a sua localização em relação a um mapa ou modelo do seu espaço de trabalho;
- Ter a capacidade de percecionar o ambiente em seu redor, o que inclui a deteção e evitação de obstáculos;
- Planear trajetórias para navegar entre diferentes pontos do ambiente e reagir a situações imprevistas.

1.2.1 Localização

Planear uma trajetória implica uma pose inicial e uma pose final, pelo que é fundamental o robô conhecer a sua pose em relação a um mapa ou modelo do ambiente em cada momento. A pose inclui a localização e a orientação do robô. Num ambiente 3D, podemos ter 3 graus de liberdade para a posição (x, y, z) e 3 graus de liberdade para a orientação (*yaw, pich, roll*). Num ambiente estruturado em que o movimento do robô seja confinado a um plano, como acontece tipicamente em robôs de serviços que operam em edifícios, 2 graus de liberdade para posição (x, y) e um grau de liberdade para orientação (*yaw*, orientação ou azimute) poderão ser suficientes para modelar a pose do robô. A localização é um dos tópicos da robótica móvel que recebeu grande atenção na últimas décadas ([12], [7], [16]). Se se imaginar um robô a navegar num certo ambiente partindo de uma posição inicial conhecida, o robô consegue atualizar a sua pose com base nos seus sensores interoceptivos, e.g. usando odometria ou sensores inerciais. Devido ao erro cumulativo destes sensores, a incerteza ou erro da estimativa da pose do robô vai aumentando ao longo do tempo. Para se limitar este erro e localizar o robô de forma aceitável durante longos períodos de tempo, usa-se sensores exteroceptivos que permitem estimar a pose do robô em relação a pontos de referência no ambiente conhecido a priori e modelado através de um mapa (e.g. sensores "laser range finder" ou de visão) ou em relação a constelações de satélites cuja posição é conhecida usando trilateração [16], i.e. GPS. No primeiro caso, a fusão sensorial entre os dois tipos de sensores permite localizar o robô móvel no mapa do ambiente, usando um filtro

estocástico de estimação da pose, e.g. um filtro de Kalman [12] ou um filtro de partículas [7] . Atualmente, as técnicas mais utilizadas baseiam-se no conceito de mapas probabilísticos [17] que estimam uma dada probabilidade para cada pose possível do robô.

1.2.2 Planeamento, execução de trajetórias e desvio de obstáculos

Dado um mapa do ambiente, a localização atual do robô e uma pose de destino, o robô deve ser capaz de planear e executar uma trajetória para atingir a pose pretendida, i.e. para navegar até à pose pretendida. À medida que o robô avança e os sensores recolhem novas informações, são detetadas alterações ao ambiente. Por exemplo, num ambiente tipo escritório, as cadeiras são objetos que podem não ser corretamente representados no mapa (e.g. podem ser mudadas de sítio) e as pessoas que coexistem com o robô no ambiente, condicionam a navegação do robô. Os objetos ou pessoas em redor do robô representam, respetivamente, obstáculos estáticos ou dinâmicos em relação aos quais o robô deve reagir e ter a capacidade de os contornar, de forma a conseguir alcançar a sua pose destino e ao mesmo tempo garantir a segurança as pessoas, do espaço de trabalho e do próprio robô (e.g. não colidir).

Para resolver este problema, as soluções atuais baseiam-se nos conceitos de planeador global e planeador local. O planeador global tem a tarefa de encontrar o caminho mais curto que o robô consegue efetuar tendo como base a informação proveniente do mapa. O planeador local tem a tarefa de evitar obstáculos e fazer com que o robô cumpra a sua trajetória em segurança.

1.3 Objetivos

O principal objetivo desta dissertação é analisar situações críticas para as quais as técnicas de navegação estado-da-arte não permitem obter um comportamento suficientemente robusto ou seguro por parte do robô móvel. Esta análise será feita com o auxílio dos pacotes e ferramentas de software disponíveis no ROS – Robot Operating System [1] e o simulador Stage [19]. Outro objetivo é a concepção e teste de melhoramentos introduzidos na utilização do software de navegação do ROS para se obter uma navegação mais segura e robusta naquelas situações.

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

1.4 Estrutura da dissertação

No primeiro capitulo, é feita a introdução ao tema, onde se apresenta a pertinência de se estudar este tema. É feita também uma pequena introdução aos principais conceitos no que diz respeito à locomoção de um robô móvel, localização e planeamento de trajetórias, e por fim são apresentados os objetivos para esta dissertação.

No capítulo dois, são explicadas as técnicas atuais usadas para a navegação de um robô móvel, dando especial ênfase às técnicas de localização, planeamento, e de estruturação do mapa do ambiente conhecido *a priori*.

No capitulo três é explicado o *middleware* ROS e é feita uma breve descrição dos pacotes ROS necessários para a navegação e dos principais problemas que afetam a pilha de navegação do ROS.

No capitulo quatro é feita uma análise detalhada da pilha de navegação do ROS, onde se irá aprender como configurar os parâmetros do move_base de forma a se obter um melhor desempenho do move_base. Depois são apresentadas algumas técnicas utilizadas para combater os problemas apresentados no capítulo anterior.

No capitulo cinco são discutidas as experiências realizadas com um robô real.

Por fim no capitulo seis é feita uma conclusão sobre o trabalho realizado.

Capítulo 2

Mapas, localização, planeamento e navegação

2.1 Mapas

Um robô móvel autónomo deve ser dotado da capacidade de criar um modelo do espaço de trabalho em que opera, a partir de informação proveniente dos seus sensores, que representa a forma como o robô perceciona o seu espaço envolvente. A construção deste modelo permite ao robô percecionar o ambiente em seu redor, localizar-se em relação a esse modelo e tomar decisões corretas no que diz respeito à execução de tarefas, evitação de obstáculos e navegação entre pontos arbitrariamente distantes entre si.

2.1.1 Mapas métricos e grelhas de ocupação

A grelha de ocupação [4] é provavelmente o modelo mais utilizado para a representação do espaço de trabalho de um robô. Consiste na decomposição do espaço em células, em que cada célula modela a probabilidade de ocupação da área ou volume elementar que representa no mundo real. A grelha é continuamente atualizada usando a informação dos sensores.

Um mapa de custos [5] é uma extensão ao conceito de grelha de ocupação. Em vez de um mapa com valores binários, livre ou ocupado, cada célula pode ter um valor de custo que varia entre livre e letal onde letal significa que existe um objeto na secção do mapa. Os valores de custo não letais são usados para representar restrições suaves, i.e. áreas do mapa que, apesar de estarem livres, podem ser indesejáveis para a localização do robô. A estas áreas pode-se atribuir um custo elevado de forma a evitar que o robô circule nestas. Escolher o caminho mais curto nem sempre é o mais indicado; um caminho mais longo pode

ser preferível se evitar áreas com elevada densidade de objetos.

A grelha de ocupação apenas informa o robô da presença de um obstáculo conhecido *a priori*. Qualquer informação relativa à natureza desse obstáculo é desconhecida. Se o robô navegar à distância de poucos centímetros (e.g. 20cm) de uma mesa isso será aceitável, mas não o será se navegar a essa distância de uma pessoa. Para contornar este problema usa-se um mapa de custos organizado por camadas [13]. Cada camada é usada para representar objetos de natureza diferente. Isto permite aplicar regras diferentes na presença de objetos diferentes. Um robô que deteta pessoas pode criar criar uma camada onde apenas obstáculos identificados como pessoas são representados e depois inflacionar os custos em redor deste obstáculo, mais do que em relação a obstáculos estáticos, de forma a ter um comportamento mais aceitável perto das pessoas. A informação de cada camada é agregada num único mapa de custos mestre e é depois neste mapa mestre que os algoritmos de planeamento de trajetória são executados.

Apesar de os mapas métricos conseguirem representar corretamente o meio localmente, erros acumulados durante o processo de criação do mapa tendem a distorcer a representação ao longo de áreas de grande escala. Outro problema que acontece em grandes espaços é que a grelha de ocupação tende a ocupar demasiada memória e ter um custo de processamento proibitivo. Os mapas topológicos permitem colmatar estas desvantagens.

2.1.2 Mapas topológicos

Uma alternativa ao mapas métricos são os mapas topológicos. Ao contrário das abordagens métricas que se baseiam nas características geométricas do ambiente, as abordagens topológicas tiram partido das outras características não geométricas existentes no ambiente que são relevantes para a localização.

Um mapa topológico consiste num grafo não dirigido constituído por dois elementos: nós e arestas. Nós são usados para representar regiões no mundo identificadas por marcos cuja pose no mundo é conhecida. Um marco cria uma área, normalmente denominada de "ilha de perfeição". Se o robô estiver dentro desta área, poderá possivelmente detetar e reconhecer o marco de forma a localizar-se no espaço de trabalho com boa precisão. As arestas são usadas para indicar a conectividade entre um par de nós, ou seja quando uma aresta liga dois nós então é possível o robô atravessar de um nó para o outro diretamente.

Geralmente, em zonas onde não existem características salientes, é comum introduzir marcos artificiais no ambiente de forma a impor uma estrutura topológica virtual que ajuda

na criação dos mapas. Exemplos disto são imagens que são colocadas nas paredes ou tetos dos corredores que são facilmente detetadas com uma câmara [11].

Em principio, mapas topológicos deveriam escalar melhor do que mapas métricos no que diz respeito a ambientes de larga escala, mas em prática mapas puramente topológicos não foram aplicados com sucesso neste ambientes. Isto levou ao desenvolvimento de técnicas híbridas que usam vários mapas métricos locais organizados num mapa topológico global [15, 18].

2.2 Localização em relação a um mapa

Se considerarmos um robô a mover-se num ambiente conhecido, seria teoricamente possível estimar a sua pose usando a odometria estimada a partir dos *encoders* do robô se conhecemos a sua pose inicial. Porém, devido a erros sistemáticos que se acumulam durante o movimento do robô (e.g. escorregamento das rodas, imperfeições mecânicas ou do modelo cinemático, resolução finita dos sensores, etc.), após algum tempo, a incerteza associada à pose do robô aumenta demasiado o que pode fazer com que o robô "se perca ". Para limitar o erro de estimação da posição, o robô deve localizar-se no seu ambiente em relação ao mapa usando informação de sensores exteroceptivos (e.g. laser range finders, câmaras, etc.). De uma forma geral, um algoritmo de localização pode ser dividido em duas fases.

Numa primeira fase, é aplicado um modelo de ação. Neste modelo, a pose do robô é atualizada com base na informação da odometria e do estado do robô no instante de tempo anterior. Geralmente este processo introduz incerteza na pose estimada do robô.

Numa segunda fase, é aplicado um modelo sensorial. Neste modelo, a pose calculada pelo modelo de ação é refinada usando informação sensorial que é comparada com o mapa do ambiente.

2.2.1 Algoritmo de localização de Markov

O algoritmo de localização de Markov [6] usa uma função de probabilidade, arbitrária, que indica qual a probabilidade de o robô existir em cada uma das poses possíveis, (x,y,θ) em mapas cartesianos ou em cada um dos nós possíveis em mapas topológicos. O facto de o robô seguir todas as poses possíveis permite ao robô localizar-se a partir de uma pose desconhecida e portanto permite também recuperar de situações ambíguas em que há múltiplas hipóteses igualmente plausíveis para a estimativa de localização, i.e. em que a distribuição

de probabilidade que modela a incerteza da localização é multimodal..

Seja L_t a variável aleatória que representa o espaço de tarefa do robô num instante t. $C(L_t = l)$ indica a confiança que o robô tem que este se encontra na pose l no instante t. Esta confiança, (C), representa uma função de densidade de probabilidade sobre todo o espaço de tarefa L do robô. A função do algoritmo de localização de Markov é atualizar a sua confiança $C(L_t)$ para todas as poses l à medida que o robô se move.

A cada instante t o robô tem um determinado conjunto de dados proveniente das leituras dos sensores exteroceptivos s_t e um conjunto de dados da odometria o_t fornecidos pelos seus sensores proprioceptivos. O algoritmo de Markov tenta, para cada pose l, descobrir qual a sua confiança em l sabendo que os seus sensores lêem s_t e o_t. Nestas condições, a estimação do valor da confiança do robô numa pose l é o mesmo que calcular a probabilidade de o robô ter a pose l sabendo que os seus sensores lêem s_t:

$$C(L_t = l) = P(L_t = l|s_t). \tag{2.1}$$

Aplicando a regra de Bayes à equação 2.1, ficamos com:

$$C(L_t = l) = \frac{P(s_t|L_t = l)P(L_t = l)}{P(s_t)}. \tag{2.2}$$

A equação 2.2 descreve o modelo sensorial aplicado ao algoritmo de localização de Markov. O valor $P(s_t|L_t = l)$ deve ser calculado. Uma estratégia consiste em consultar o mapa e para cada pose possível determinar a probabilidade de as leituras atuais corresponderem a essa pose. O termo $P(L_t = l)$ da equação 2.2 corresponde uma estimativa da probabilidade de ter a pose l. Efetivamente representa a confiança do robô após ser aplicado o modelo de ação. O denominador $P(s_t)$ garante a normalização das probabilidades no intervalo entre 0 e 1, não depende de l, por isso este valor é ignorado até ao final onde a confiança no espaço de tarefa é renormalizada de forma a que o somatório das probabilidades para todas as poses seja igual a um.

À medida que o ambiente do robô aumenta, aumenta também o número de possíveis poses para o robô. Como o algoritmo de localização de Markov implica atualizar a confiança ao longo de todo o espaço de tarefa, para ambientes muito grandes pode demorar muito tempo para ter uma estimativa da pose. Para acelerar o processo de estimação da pose do robô foram desenvolvidas várias técnicas. Estas técnicas, geralmente conhecidas como filtro de partículas [7], consistem numa amostragem do espaço de tarefa. Assim em vez de representar a confiança completa, o robô calcula uma confiança aproximada a partir de um

subconjunto das poses possíveis.

De forma a conseguir uma aproximação à função de densidade, a amostragem é feita de forma a que as amostras fiquem concentradas nas zonas de maior probabilidade para a localização do robô, mas ao mesmo tempo é desejável que algumas poses pouco prováveis sejam também amostradas na eventualidade de o robô se perder. Se nenhuma amostra for gerada perto da pose correta do robô, corre-se o risco de o robô não conseguir relocalizar-se.

Outra otimização possível é usar um número de amostras adaptativo. Quando a pose do robô é desconhecida são necessárias mais amostras para aproximar a função de densidade de probabilidade com rigor [7].

2.2.2 Localização baseada em marcos ou faróis

Geralmente marcos são definidos como objetos passivos no ambiente que fornecem um elevado grau de certeza na localização quando se encontram dentro do campo de visão do robô. Quando um marco está à vista o robô localiza-se frequentemente e com precisão e sem erro acumulado. Mas quando o robô está fora de uma zona de marcos, então apenas a odometria pode acontecer e o robô acumula erros de posição até um novo marco entrar no campo de visão do robô.

Quando o ambiente é muito assético, a falta de características salientes implica a necessidade de se introduzirem marcos artificiais para diferenciar duas localizações idênticas. Outra desvantagem dos sistemas de localização baseada em marcos é que quando a distância entre dois marcos é muito grande a odometria do robô pode não ser suficiente para manter a incerteza associada à pose do robô dentro de limites aceitáveis, o que pode levar ao robô a perder-se.

Outra técnica usada por conseguirem grande fiabilidade e precisão, consiste na introdução de emissores de sinal no espaço de trabalho do robô em posições bem conhecidas no mapa. O robô consegue depois determinar a sua posição com a ajuda e algoritmos de triangulação, trilateração, ou multilateração.

2.3 Planeadores globais e navegação global

A função de um planeador global é a de gerar um caminho que serve como guia ao robô. O plano global permite ao robô executar uma trajetória mais eficiente reduzindo o tempo de viagem do robô e evita que o robô fique preso em mínimos locais. Um dos exemplos mais

famosos de planeadores são os algoritmos de Dijkstra e A*.

2.3.1 Algoritmo de Dijkstra

Publicado em 1959 pelo cientista Edsger W. Dijkstra, o algoritmo de Dijkstra [2] é um algoritmo para encontrar o caminho mais curto entre dois nós num grafo. Uma descrição do algoritmo é apresentada a seguir.

1. Marcar todos os nós como "não visitados". Criar lista todos os nós "não visitados".

2. Atribuir a todos os nós uma distância: ao nó inicial atribuir zero; aos restantes atribuir infinito. Marcar o nó inicial como "atual".

3. Para o nó atual considerar todos os seus vizinhos na lista dos "não visitados"e calcular a distância do nó vizinho através do atual. Se a nova distância for menor do que a anterior atualizar a distância e marcar que o caminho passa por atual.

4. No final de calcular as novas distâncias dos vizinhos do "atual", marcar "atual"como visitado e removê-lo da lista dos "não visitados".

5. Se o nó destino for marcado como visitado acabar. Se não escolher o nó com menor distância, marca-lo como "atual" e saltar para o passo 3.

2.3.2 Algoritmo A* (A star)

O A* [9] é um algoritmo de planeamento que permite encontrar uma trajetória entre dois pontos. Este algoritmo pode ser visto como uma extensão ao algoritmo de Dijkstra, mas em contraste com o algoritmo de Dijkstra, o A* não tenta encontrar uma solução ótima; simplesmente permite obter num tempo computacionalmente razoável uma solução que satisfaça o problema.

O algoritmo A* determina quais os nós a expandir baseando-se no custo do caminho e numa estimativa do custo necessário para percorrer o caminho até o nó final. O algoritmo escolhe o caminho que minimiza a função $f(n) = g(n) + h(n)$ onde n é um nó adjacente ao nó atual, g(n) é o custo do caminho desde do nó inicial até o nó n e h(n) é uma função heurística que estima o custo de n até ao nó final.

1. Criar a lista dos nós visitados e a lista dos nós ativos. Inicialmente a lista dos nós fechados está vazia e a lista dos nós ativos contém apenas o nó inicial.

2. Atribuir os valores g e f iniciais. Para o nó inicial g = 0 e f = h(início), para os restantes nós f e g = infinito.

3. Escolher da lista dos nós ativos o nó com menor valor de f e definir este como atual. Se o nó escolhido for o nó final ou se a lista dos nós ativos estiver vazia podemos acabar.

4. Remover atual da lista dos nós ativos e adicionar atual na lista dos nós visitados.

5. Para cada vizinho do nó atual que não existe na lista dos nós visitados.

   (a) se vizinho não está na lista dos nós abertos, adicionar vizinho na lista dos nós abertos.

   (b) calcular custo g(vizinho) = g(atual) + custo_atual->vizinho

   (c) se novo custo g(vizinho) for menor do que o custo g(vizinho) atual.

   - atualizar o valor de g(vizinho)
   - definir nó anterior do vizinho como atual.
   - atualizar valor de f(vizinho) = g(vizinho) + h(vizinho).

6. Saltar para o ponto 3

2.4 Planeadores locais e navegação local

A função de um planeador local é a de gerar valores de velocidade para o robô de maneira a que este consiga executar o seu percurso de forma segura, ou seja evitar que o robô colida com o seu ambiente.

2.4.1 Dynamic Window Approach

Dynamic window approach [8, 1] é um algoritmo de evitação de obstáculos que incorpora a cinemática do robô para determinar qual a velocidade adequada à situação.

O algoritmo consiste em duas partes: numa primeira parte, é determinado o conjunto de velocidades que o robô pode tomar durante o próximo intervalo; na segunda parte do algoritmo, é feita uma avaliação de forma a determinar qual a velocidade que melhor se adequa às circunstâncias.

PARTE 1: Espaço de pesquisa

A trajetória de um robô síncrono pode ser bem aproximada por uma sequência de arcos circulares (curvaturas). Cada curvaturas é unicamente identificada por um vetor velocidade $[v, \omega]$ em que v corresponde à velocidade de translação e ω à velocidade de rotação. Isto significa que a pesquisa das velocidades é feita num espaço 2D.

Obstáculos próximos do robô impõem restrições de velocidade. Por exemplo, a velocidade máxima admissível numa curvatura depende da distância ao primeiro obstáculo nesta curvatura. Uma velocidade $[v, \omega]$ é considerada admissível se o robô é capaz de parar antes de atingir o obstáculo.

De forma a ter em conta o valor limitado de aceleração exercida pelo motores, o espaço de pesquisa é reduzido a uma janela dinâmica que contém apenas as velocidades que são possíveis de atingir durante o próximo intervalo de tempo. A janela dinâmica é centrada na velocidade atual e o seu tamanho depende das acelerações do robô. Todas as curvaturas fora da janela dinâmica e não alcançáveis no intervalo de tempo não são consideradas para o desvio de obstáculos.

PARTE 2: Função objetivo

O processo de calcular a velocidade ótima consiste em calcular o valor de velocidade que maximiza a função objetivo escrita da seguinte forma:

$$G(v, \omega) = \sigma \cdot (\alpha \cdot dir(v, \omega) + \beta \cdot dist(v, \omega) + \gamma \cdot vel(v, \omega)),$$

em que a direção, $dir(v, \omega)$, mede o alinhamento da direção do robô com a direção pretendida. É dada por $\pi - \theta$, onde θ é o ângulo entre direção do robô e direção pretendida. A função $dist(v, \omega)$ representa a distância ao obstáculo mais próximo que interseta com a curvatura. Se nenhum obstáculo interseta a abertura então este valor é uma constante muito grande. A função $vel(v, \omega)$ é usada para avaliar o progresso do robô na trajetória correspondente. É simplesmente uma projeção da componente escalar no espaço de pesquisa o que leva o robô a preferir altas velocidades.

Capítulo 3

Software de Navegação do ROS

3.1 O que é o ROS?

ROS[1] *(Robot Operating System)* [14] é um *middleware* de código aberto para programação de *software* para robôs e consiste numa coleção de ferramentas, bibliotecas e convenções que permitem simplificar o desenvolvimento de sistemas complexos de forma a conseguir comportamento robusto numa grande variedade de plataformas.

O ROS foi pensado de forma a ser o mais modular possível. Um sistema desenvolvido em ROS consiste num número de programas independentes, normalmente designados por nós, onde cada nó consegue comunicar com os outros nós através de um mecanismo de passagem de mensagens. Por exemplo o *driver* de um determinado sensor é implementado num nó que publica a informação adquirida que é depois processada por um outro nó.

O ROS adquiriu um grande suporte por parte da comunidade robótica e hoje pode ser encontrado em centros de investigação, produtos empresariais e na educação. Este tipo de arquitetura traz muitas vantagens na programação e desenvolvimento de sistemas robóticos, nomeadamente:

- **Computação distribuída:** Com o ROS é possível ter os vários componentes que constituem o sistema espalhados por várias máquinas. Nem sempre é vantajoso ter todo o processamento no robô. A computação distribuída permite a realização de tarefas de elevado custo computacional em máquinas externas e também facilita a criação de sistemas multi-robô cooperativos.

- **Reutilização de software:** Ao dividir o sistema em pequenos programas independentes permite escrever software que pode ser facilmente integrado em vários sistemas.

[1] http://www.ros.org/about-ros/

Assim, ao longo dos anos, foram desenvolvidos vários pacotes para o ROS que implementam uma solução para as tarefas mais comuns, tais como navegação, planeamento, mapeamento, etc. já testados, livres de *bugs* e comprovados no terreno.

- **Facilidade de teste:** O desenvolvimento de software para robôs pode ser uma tarefa demorada, os robôs nem sempre estão disponíveis para serem utilizados, falta de bateria como por exemplo, ou até mesmo testar configurações com múltiplos robôs onde pode haver dezenas de robôs a cooperar. O ROS permite acelerar o tempo de desenvolvimento com recurso a simuladores. Usar simuladores no ROS é bastante simples uma vez que basta substituir o controlador do hardware pelo simulador mantendo toda a lógica de alto nível inalterada. A lista de simuladores possíveis de usar com o ROS inclui STAGE [19], gazebo [10] e o morse [3].

3.2 A pilha de navegação

A pilha de navegação do ROS [2] consiste num conjunto de pacotes de software de navegação para robôs móveis que quando usados em conjunto permitem ler informação dos vários sensores e da odometria e produzir comandos de velocidade para o robô de forma a este conseguir alcançar o seu destino de forma segura. No centro da pilha de navegação encontra-se o *move_base* [3].

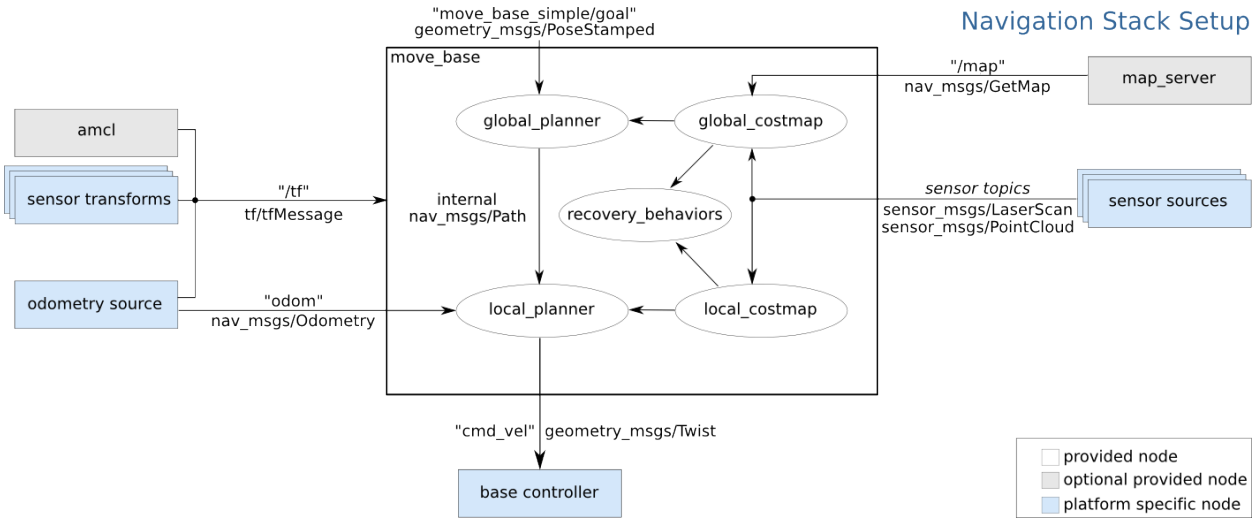


Figura 3.1: Esquemático representativo do funcionamento do move_base. Figura retirada de[wiki.ros.org/move_base].

O *move_base* consiste numa interface para correr, configurar e interagir com a pilha de

[2]http://wiki.ros.org/navigation
[3]https://wiki.ros.org/move_base

navegação do ROS. Na figura 3.1 é possível ver uma visão geral do *move_base* e de como este se integra na pilha de navegação.

Essencialmente o *move_base* permite ligar os vários componentes que constituem o sistema de navegação (sensores, planeamento e robô). Internamente o *move_base* é constituído por um planeador global, um planeador local, um mapa de custos global e um mapa de custos local. Antes de iniciar o *move_base* o utilizador pode escolher qual o planeador (local e global) que quer usar. O utilizador pode também criar *plugins* para implementar alternativas aos planeadores fornecidos pelo *ROS*. Informação sobre como implementar novos planeadores ou sobre os existentes pode ser encontrada no pacote *nav_core* [4]

Resumidamente o funcionamento do *move_base* pode ser explicado da seguinte forma. Quando o robô recebe um novo destino o *move_base* começa por determinar a sua posição atual, geralmente fornecida pelo pacote *AMCL* [5] ou por outro algoritmo de localização do robô em relação a um mapa de grelha de ocupação conhecido a priori. O valor da posição é depois passado ao planeador global que, com base no mapa de custos global, calcula uma trajetória que guia o robô até ao destino. Por fim, o planeador local gera comandos de velocidade para o robô, em função do mapa de custos local e da trajetória obtida pelo planeador global de forma a evitar e contornar obstáculos não previstos pelo planeador global. .

3.3 Análise do *move_base* e das suas limitações

Para cumprir a sua missão é essencial que o robô seja capaz de conseguir planear e executar uma trajetória de forma segura que por sua vez depende da capacidade que o robô tem de observar o mundo e usar essa informação. Nesta secção serão identificados algumas das principais dificuldades que podem surgir na navegação de um robô móvel usando o *move_base*.

Parte do estudo foi realizado com a ajuda de um simulador. O simulador escolhido foi o *Stage*.Foi construída em simulação uma cena que fosse representativa de um espaço do tipo escritório (salas e corredores). A figura 3.2 mostra a planta do ambiente em que foram realizadas as experiências de simulação. As especificações do robô simulado foram escolhidas de forma a serem o mais próximas possível com o robô real usado nas experiências descritas mais à frente nesta dissertação, que consiste num *Pioneer 3DX* equipado com um *laser*

[4] http://wiki.ros.org/nav_core
[5] https://wiki.ros.org/amcl

rangefinder Hokuyo - URG-04LX-UG01 e uma câmara *kinect* 360, ver figura 3.3.

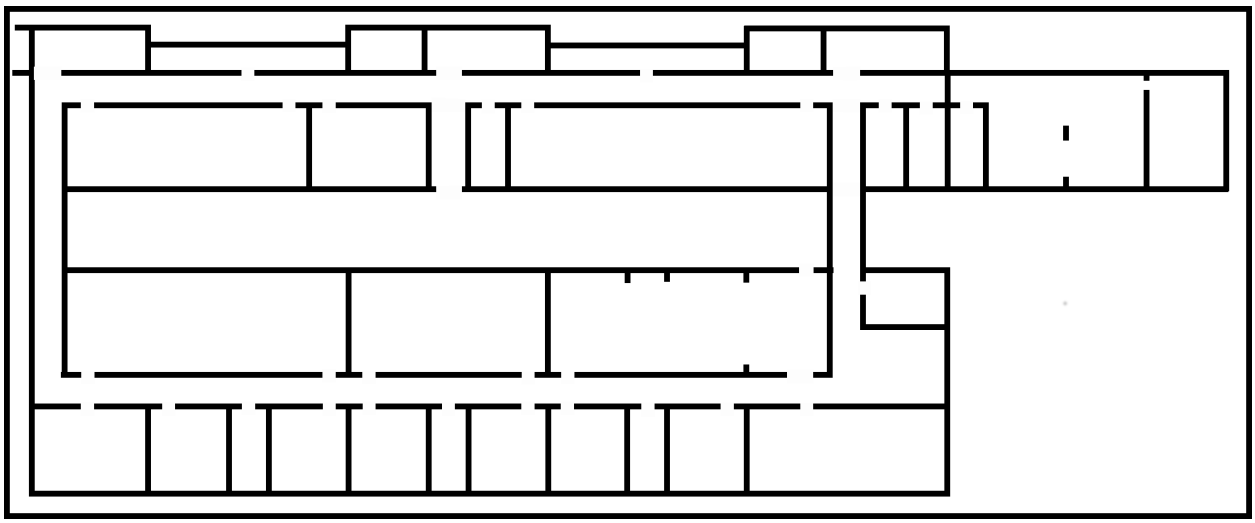


Figura 3.2: Planta de um escritório usado em simulação de dimensões $[85 \times 35]m$.

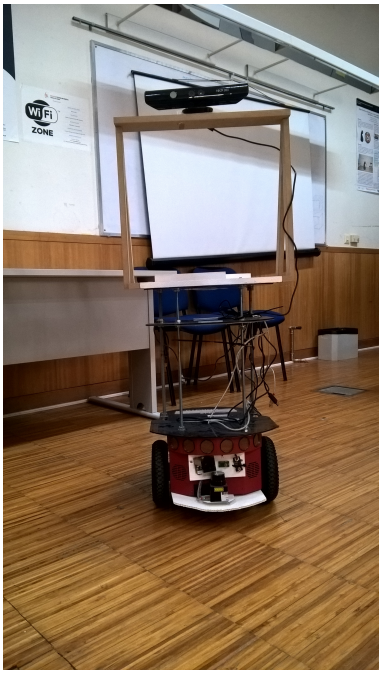


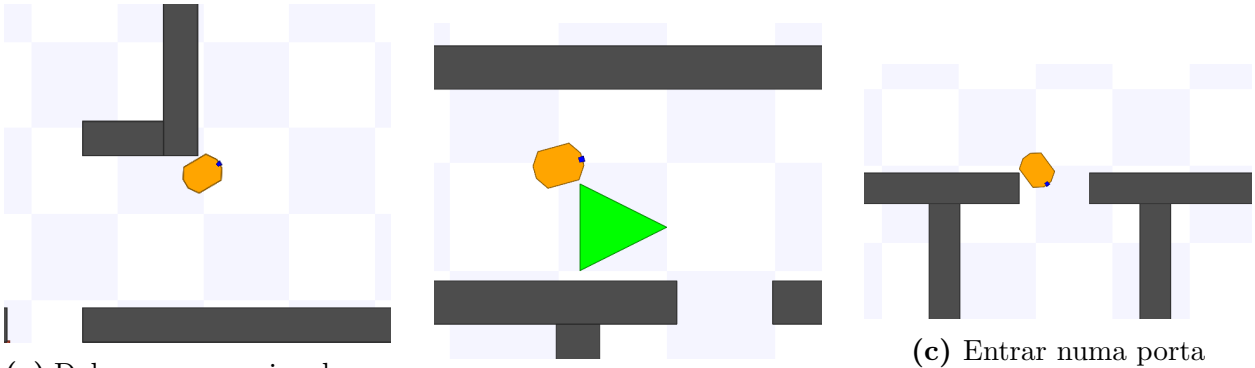
Figura 3.3: Robô *pioneer 3dx* equipado com um *laser rangefinder* Hokuyo - URG-04LX-UG01 e uma câmera *kinect* 360.

Para este projeto os planeadores escolhidos foram o *global_planner* [6] que implementa os algoritmos de Dijkstra e A* para planeamento global e o *dwa_local_planner* [7] que implementa um algoritmo baseado no *Dynamic Window Approach* para planeamento local.

[6] http://wiki.ros.org/global_planner
[7] http://wiki.ros.org/dwa_local_planner

3.3.1 Análise da trajetória com os parâmetros predefinidos

Para começar foi feito um estudo para averiguar o quão aceitável é o comportamento do robô com os parâmetros predefinidos do *move_base* descritos na tabela 4.1. Com estes parâmetros verificou-se que o *move_base*, de uma forma geral, tem um bom desempenho, o robô consegue planear e executar trajetórias com facilidade, incluindo o robô mostrou ser capaz de contornar obstáculos que não estavam representados no mapa.



(a) Dobrar uma esquina de um corredor

(b) Ultrapassar um obstáculo

(c) Entrar numa porta

Figura 3.4: Comportamentos de risco mostrado pelo move_base na sua configuração predefinida.

O principal problema do *move_base* na sua configuração base é que na tentativa de realizar o percurso o mais rápido possível o *move_base* tende a aproximar o robô para perto de paredes e/ou objetos. Este comportamento é bastante visível nas esquinas do corredores ou quando o robô tenta entrar numa divisão (figura 3.4). Navegar perto de obstáculos aumenta o risco de colisão e verificou-se que esta aproximação excessiva leva de facto ao robô a colidir, como mostra a figura 3.5.

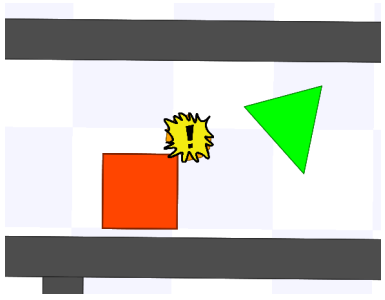


Figura 3.5: Colisão do robô com um obstáculo em simulação.

Também pode acontecer a situação em que o robô deixa de se mover sem razão aparente quando se aproxima de um objeto, como é o caso da figura 3.6. Na consola do programa é visível a mensagem de erro 'Não foi possível obter um plano' o que significa que o *move_base* não encontrou nenhuma trajetória válida que permita mover sem que o robô colida com algo.

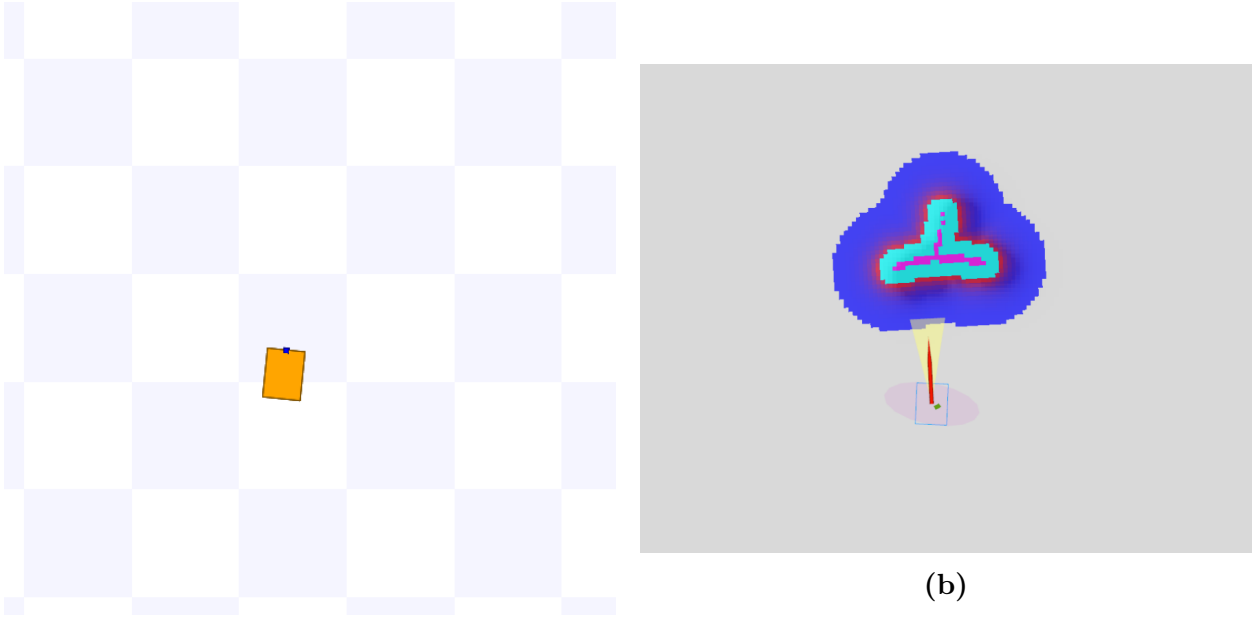
Figura 3.6: Erro no planeamento de trajetória.

3.3.2 Camada de obstáculos

A camada de obstáculos [8] do mapa de custos do ROS é responsável por manter uma representação do mundo a partir da informação proveniente dos sensores. Esta camada é construída a partir de uma grelha de ocupação 3D que é depois projetada no mapa de custos 2D. O *move_base* atualiza os valores da grelha de ocupação a partir da informação proveniente de sensores que disponibilizam os seus dados em formato *LaserScan* usado por sensores como os laser range-finders planares ou então do tipo *PointCloud2* usado por sensores capazes de medir o espaço em três dimensões.

Quando a camada de obstáculos recebe nova informação sobre um obstáculo a célula correspondente da grelha é marcada como ocupada e depois é utilizado um processo de *ray tracing* de forma a limpar as células entre o referencial do sensor até à célula onde foi feita a deteção do obstáculo. Como o processo de limpeza necessita que o robô tenha resposta de um obstáculo, quando o robô se encontra num espaço em que a distância aos objetos é maior do que o alcance do sensor então o robô não consegue limpar objetos marcados no mapa em iterações anteriores tal como acontece na figura 3.7.

[8] http://wiki.ros.org/costmap_2d/hydro/obstacles

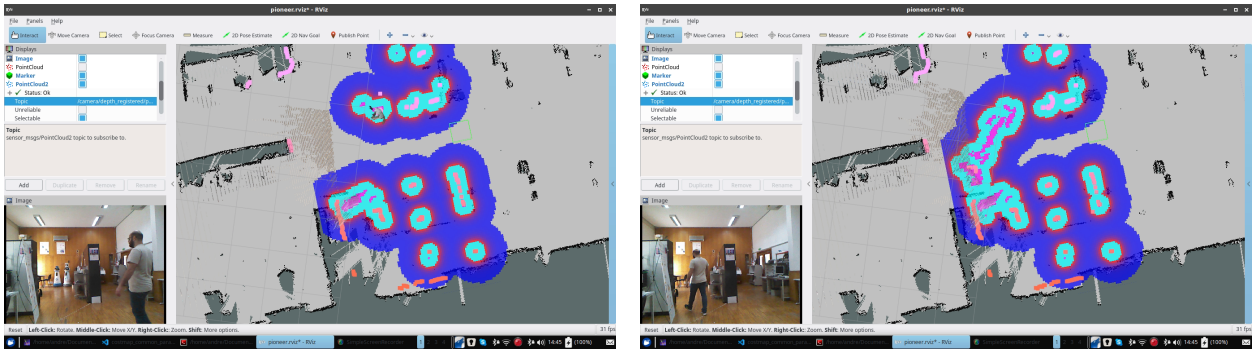


(a) (b)

Figura 3.7: O robô não consegue limpar mapa de custos devido à falta de resposta do ambiente o robô não consegue aplicar o algoritmo de *ray tracing* pare remover o objeto da mapa.

Para evitar esta situação existe um parâmetro da camada de obstáculos, *inf_is_true*, que quando ativo permite limpar o mapa até a um valor estipulado quando o sensor não recebe medidas do ambiente. O problema é que este parâmetro apenas funciona para sensores que fornecem os seus dados no formato *laserScan* e que obedecem à convenção descrita em [9]. Esta convenção convenciona que deteções que estão demasiado perto do sensor para serem quantificadas deveram ser representadas por $-Inf$, medições erróneas devem ser representadas por NaN e finalmente medições fora de alcance devem ser representadas por $+Inf$. Quando o robô perde a habilidade de limpar o mapa corretamente este começa a ganhar "lixo" o que põe em causa a capacidade de navegação. Na figura 3.8 é possível ver um exemplo onde a passagem de uma pessoa em frente do robô provoca a marcação de obstáculos ao longo do percurso da pessoa que se mantiveram no mapas depois da pessoa sair de cena.

[9] http://www.ros.org/reps/rep-0117.html



(a) **(b)**

Figura 3.8: A figura mostra o estado do mapa de custos antes de uma pessoa passar à sua frente (a), e depois da pessoa ter passado (b). Na figura 3.8b é possível ver que medições obtidas da pessoa continuam no mapa de custos.

3.3.3 Navegação com obstáculos dinâmicos

Os robôs de serviços, quando navegam de forma autónoma, estão sujeitos a interagir com objetos que se deslocam no espaço de tarefa do robô, como é o exemplo de pessoas ou até mesmo outros robôs autónomos. A seguir é analisado o *move_base* na mais simples destas situações: um robô a percorrer um corredor ao mesmo tempo que um obstáculo dinâmico faz o mesmo percurso no sentido contrário.

Na figura 3.9 é mostrado o resultado de um robô autónomo a tentar desviar-se de uma pessoa que se desloca em sentido contrário. De forma a simular um ser humano no simulador foi usado um robô tele-operado representado a cor-de-rosa na figura 3.9. Durante a execução da experiência o robô tele-operado não fará qualquer esforço para tentar prevenir uma colisão, ou seja irá seguir em frente com velocidade constante, isto com o objetivo de simular uma pessoa distraída que não repara no robô, portanto o robô autónomo deverá ser capaz por si só de se desviar do obstáculo.

Nesta experiência verificou-se que o *move_base* tem um comportamento reativo, quando na presença de um obstáculo o robô deve simplesmente contornar esse obstáculo. Como o ajuste da trajetória só é efetuado a uma pequena distância do obstáculo o robô pode não ter tempo suficiente para sair do caminho da pessoa e evitar a colisão.

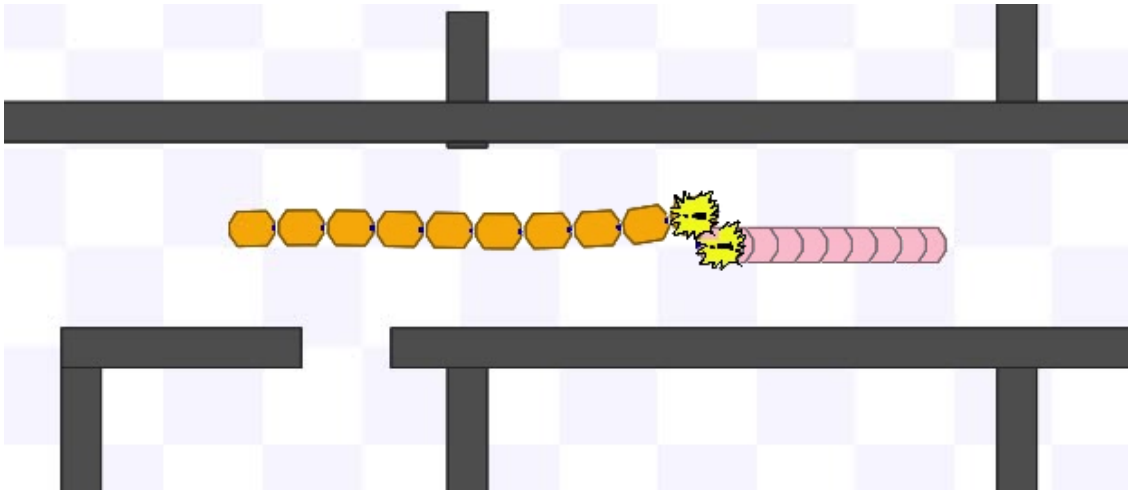


Figura 3.9: Interação entre um robô e uma pessoa.

A figura 3.10 descreve o comportamento de dois robôs autónomos, ambos executando o *move_base*, a deslocarem-se em sentidos opostos e em rota de colisão. Inicialmente os robôs avançam pelo centro do corredor em rota de colisão. Quando chegam perto um do outro ambos tentam desviar-se, mas como eles se deviam para o mesmo lado ambos continuam a bloquear o caminho um ao outro. Depois os robôs começam a rodar sobre si próprios até que um deles, neste caso o robô cor-de-rosa, encontra uma trajetória válida e continua o seu caminho. O robô cor-de-laranja agora sem obstáculos pode então seguir em frente. No final não houve colisão e ambos os robôs conseguiram chegar ao ponto destino com sucesso, mas o processo de negociação de passagem não é perfeito e tende a ser moroso.

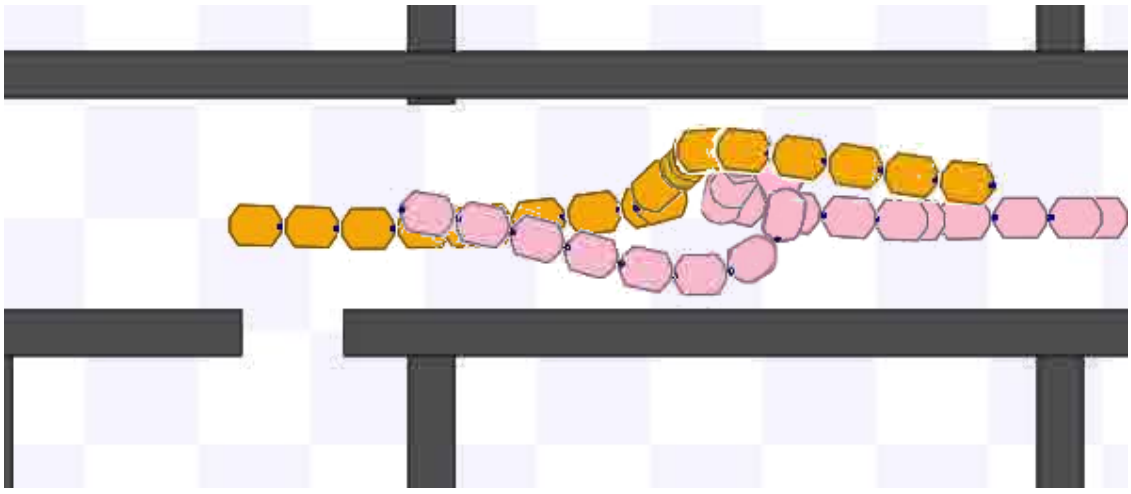


Figura 3.10: Interação entre dois robôs autónomos.

Capítulo 4

Melhoramento do comportamento obtido com o software de navegação do ROS

4.1 Sintonização dos parâmetros do *move_base*

Com o *move_base* é possível configurar uma série de parâmetros que permitem modificar o comportamento do robô. Neta secção será feita uma análise de alguns parâmetros. Esta análise tem como objetivo ajudar o utilizador do *move_base* a perceber como ajustar os parâmetros de forma a obter um comportamento mais robusto e mais aceitável nas situações identificadas no capítulo 3..

4.1.1 Planeador local – *dwa_local_planner*

A equação de classificação de trajetórias

No algoritmo do *Dynamic Window Approach* [8] a escolha da velocidade consiste determinar qual o melhor arco de circunferência que o robô deve tomar fazendo uso de uma função objetivo. No caso do *dwa_local_planner* do ROS consiste em determinar a trajetória que minimiza a função:

$$
\begin{aligned}
\textbf{custo} = \\
&\quad \textbf{\textit{path\_distance\_bias}} \times (\text{distância do caminho global até ao ponto final do arco de circunferência em metros}) \\
&+ \textbf{\textit{goal\_distance\_bias}} \times (\text{distância do objetivo local até ao ponto final do arco de circunferência em metros}) \\
&+ \textbf{\textit{occdist\_scale}} \times (\text{valor de custo máximo ao longo da trajetoria})
\end{aligned}
\tag{4.1}
$$

Nome	Valor predefinido	Gama de teste	Descrição
dwa_local_planner			
path_distance_bias	32.0	6.0 - 64.0	Peso para quanto o robô se deve manter no caminho global
goal_distance_bias	24.0	6.0 - 64.0	Peso para quanto o robô deve tentar chegar ao destino
occdist_scale	0.01	0.01 - 10.0	Peso para quanto o robô deve evitar obstáculos
sim_time	1.7	1.0 - 2.5	Tempo de simulação da trajetória em segundos
Mapa de custos Global			
cost_scaling_factor	10.0	1.0 - 16.0	Taxa de decaimento exponencial do custo de um obstáculo
inflation_radius	0.55	0.5 - 3.0	Distância máxima inflacionada por um obstáculo
Mapa de custos Local			
cost_scaling_factor	10.0	1.0 - 16.0	Taxa de decaimento exponencial do custo de um obstáculo
inflation_radius	0.55	0.50 - 3.0	Distância máxima inflacionada por um obstáculo

Tabela 4.1: Tabela resumo com os parâmetros estudados.

O primeiro termo da soma representa a distância do robô ao caminho global e aumentar o peso *path_distance_bias* faz com que o controlador prefira trajetórias perto do caminho global. Por outro lado o segundo termo representa a distância do robô ao objetivo local e aumentar o valor do peso *goal_distance_bias* faz com que trajetórias que deslocam o robô para o objetivo local sejam escolhidas. As figuras 4.2, 4.3, 4.4 mostram como a variação destes valores afeta o caminho efetuado pelo robô. Se *goal_distance_bias* for maior do que *path_distance_bias* então o robô tem preferência por trajetórias que deslocam o robô rapidamente para o objetivo, caso contrário o robô executa um percurso mais próximo ao caminho dado pelo planeador global. A terceira parcela da equação diz respeito ao maior valor no mapa de custos local encontrado ao longo da trajetória e *occdist_scale* é um peso que diz ao controlador o quanto deve o robô evitar as zonas com custos elevados. Aumentar este valor faz com que o robô tenha mais cuidado e se mantenha mais afastado dos obstáculos como mostra as figuras 4.5 e 4.6. Um efeito secundário desta segurança é que a ultrapassagem do obstáculo é feita a uma menor velocidade aumentado assim o tempo para concluir a trajetória. Para casos em que *occdist_scale* é muito grande existe também a possibilidade de o robô não conseguir atravessar uma porta ficando a rodar à entrada das divisões (ver figura 4.7).

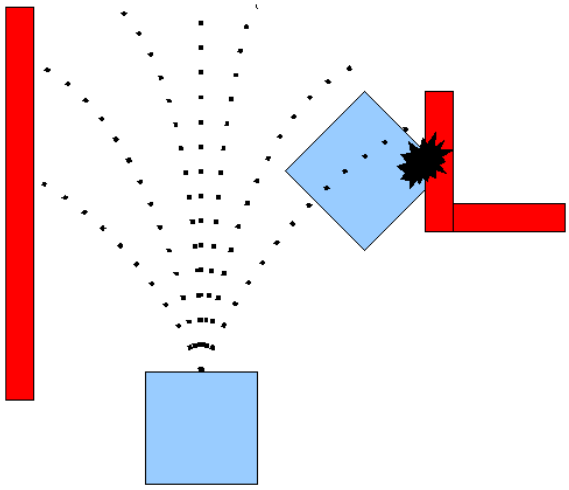
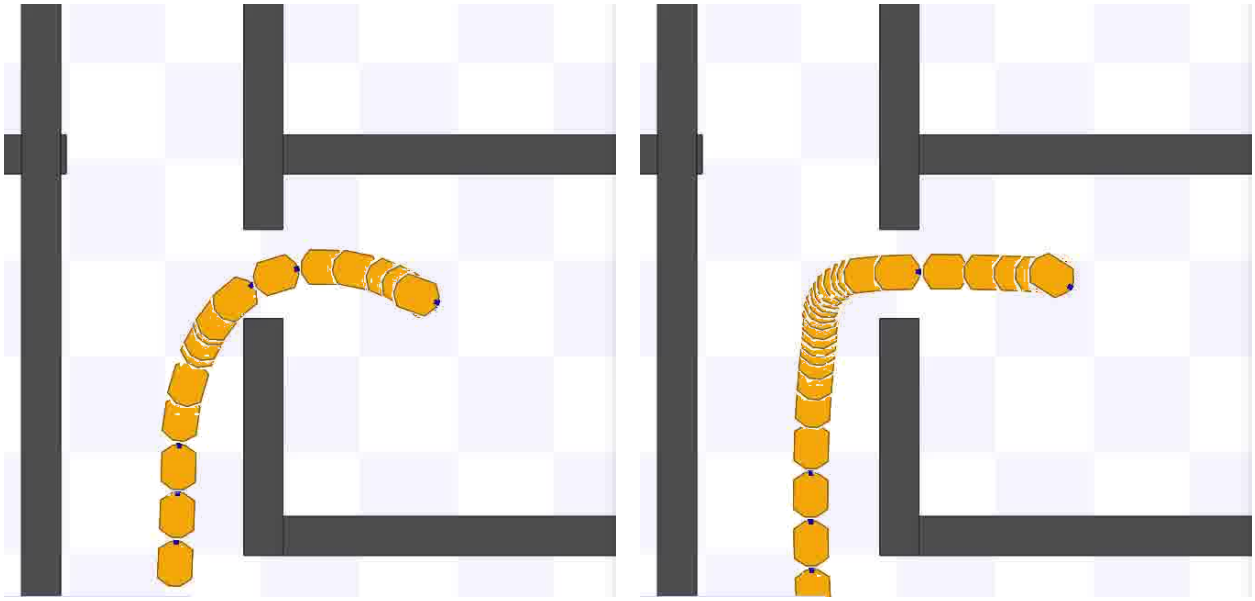


Figura 4.1: Visão geral do *Dynamic Window Approach*. O robô aproxima a sua trajetória a arcos de circunferência. Para cada arco é calculado um custo, e a trajetória com menor custo é escolhida. Imagem retirada de [http://wiki.ros.org/dwa_local_planner].



(a) *occdist_scale* = 0.01 **(b)** *occdist_scale* = 10.0

Figura 4.6: Trajetória realizada pelo robô ao entrar numa sala com dois valores diferentes de *occdist_scale*.

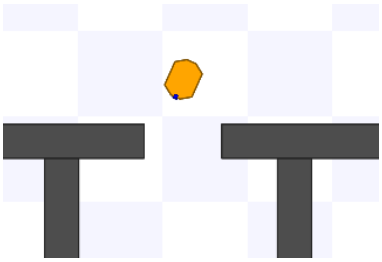


Figura 4.7: Situação em que robô não conseguiu realizar plano. (Porta estreita).

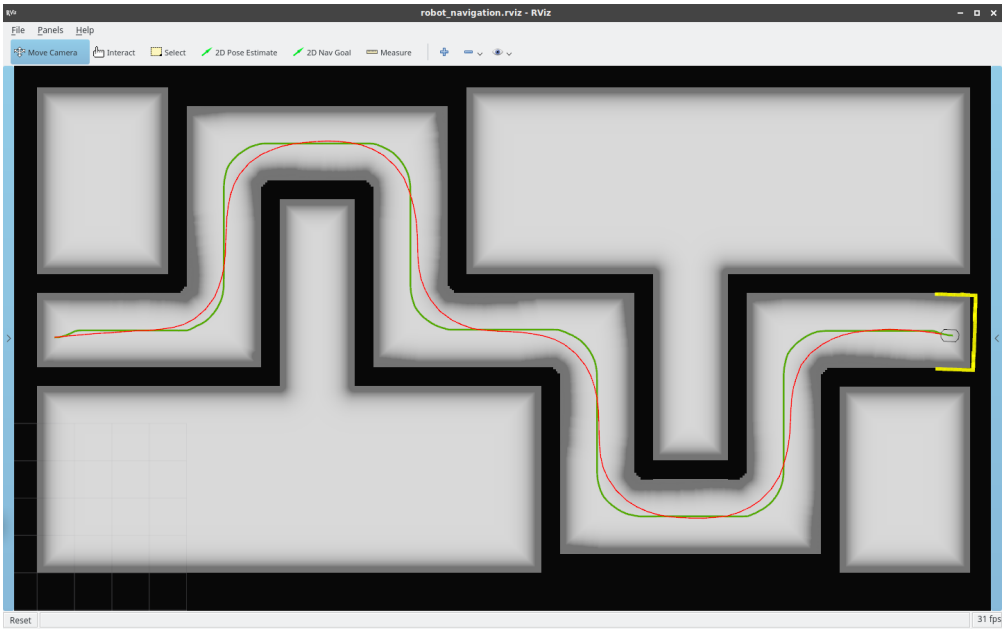
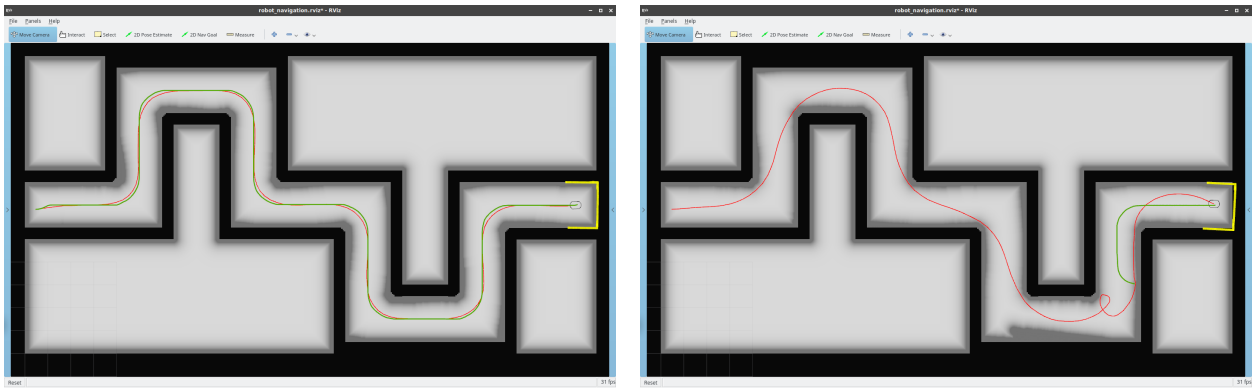


Figura 4.2: Trajetória executada pelo robô num corredor com os parâmetros predefinidos. A cor verde representa o plano global, a vermelho é representado a trajetória efetuada pelo robô.

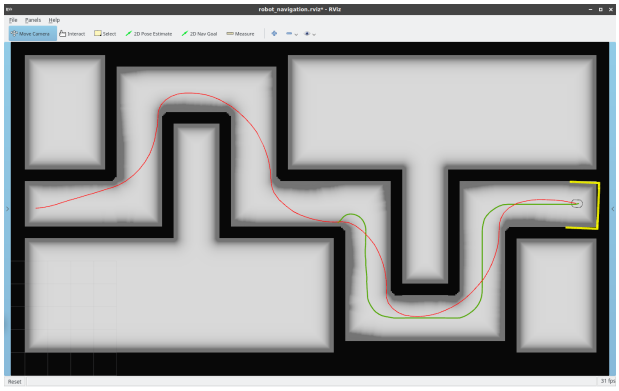
Tempo de simulação da trajetória

O parâmetro *sim_time* corresponde a durante quanto tempo a trajetória é simulada. Valores baixos, (e.g. 1s), fazem com que o robô execute com rigor o plano global. À medida que se aumenta o valor *sim_time* o robô vai ganhando liberdade para poder sair do caminho global e ser mais directo ao objetivo. A figura 4.8 mostra a diferença entre o percurso executado pelo robô com diferentes valores de *sim_time*. Também foi possível identificar que a gama de valores aceitáveis para *sim_time* depende do valor máximo da velocidade do robô. Para valores de velocidade mais baixos é preciso valores mais altos para *sim_time*.

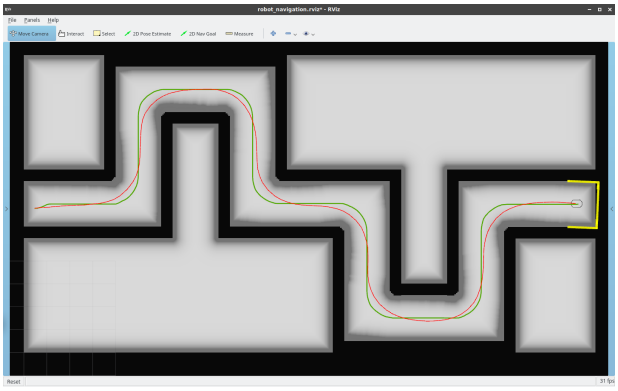


(a) *sim_time* = 1,0 s **(b)** *sim_time* = 2,5 s

Figura 4.8: Efeito do *sim_time* na trajetória de um robô. A cor verde representa o plano global, a vermelho é representado a trajetória efetuada pelo robô. Nota: Durante a execução do percurso na figura 4.4b houve a necessidade de replanear o caminho global.

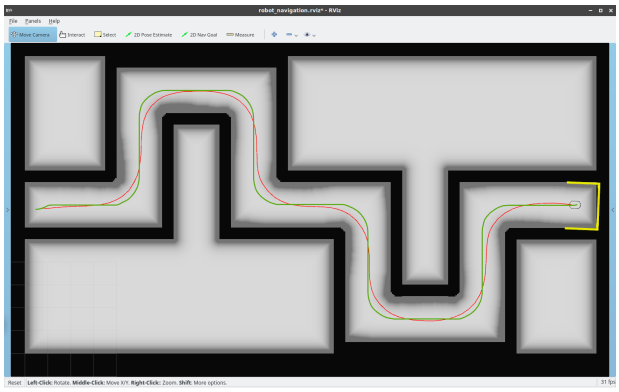


(a) Trajetória efetuada pelo robô após diminuir *path_distance_bias* para 6,0.



(b) Trajetória efetuada pelo robô após aumentar *path_distance_bias* para 64,0.

Figura 4.3: Efeito na trajetória de um robô provocado pela alteração dos valores de *path_distance_bias*. Nota: Durante a execução do percurso na figura 4.3a houve a necessidade de replanear o caminho global.



(a) Trajetória efetuada pelo robô após diminuir *goal_distance_bias* para 6,0.

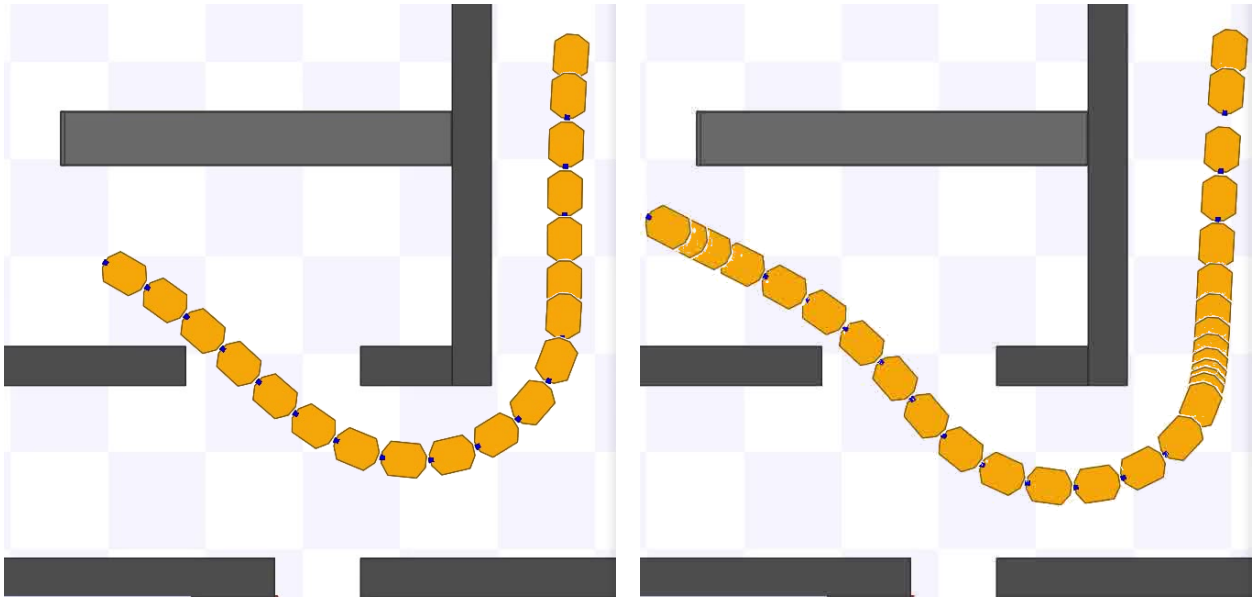
(b) Trajetória efetuada pelo robô após aumentar *goal_distance_bias* para 64,0.

Figura 4.4: Efeito na trajetória de um robô provocado pela alteração dos valores de *goal_distance_bias*. Nota: Durante a execução do percurso na figura 4.4b houve a necessidade de replanear o caminho global.

4.1.2 A camada de inflação do mapa de custos

O mapa de custos do *move_base* é constituído por camadas. Uma das camadas principais é a camada de inflação [1] que tem com função inflacionar os custos das células em redor dos objetos. Esta área de inflação pode ser usada para condicionar o movimento do robô. Para este efeito faz-se uso de duas variáveis: *cost_scaling_factor* e *inflation_radius*.

[1] http://wiki.ros.org/costmap_2d/hydro/inflation



(a) *occdist_scale* = 0.01 **(b)** *occdist_scale* = 10.0

Figura 4.5: Trajetória realizada pelo robô ao contornar uma esquina de um corredor com dois valores diferentes de *occdist_scale*.

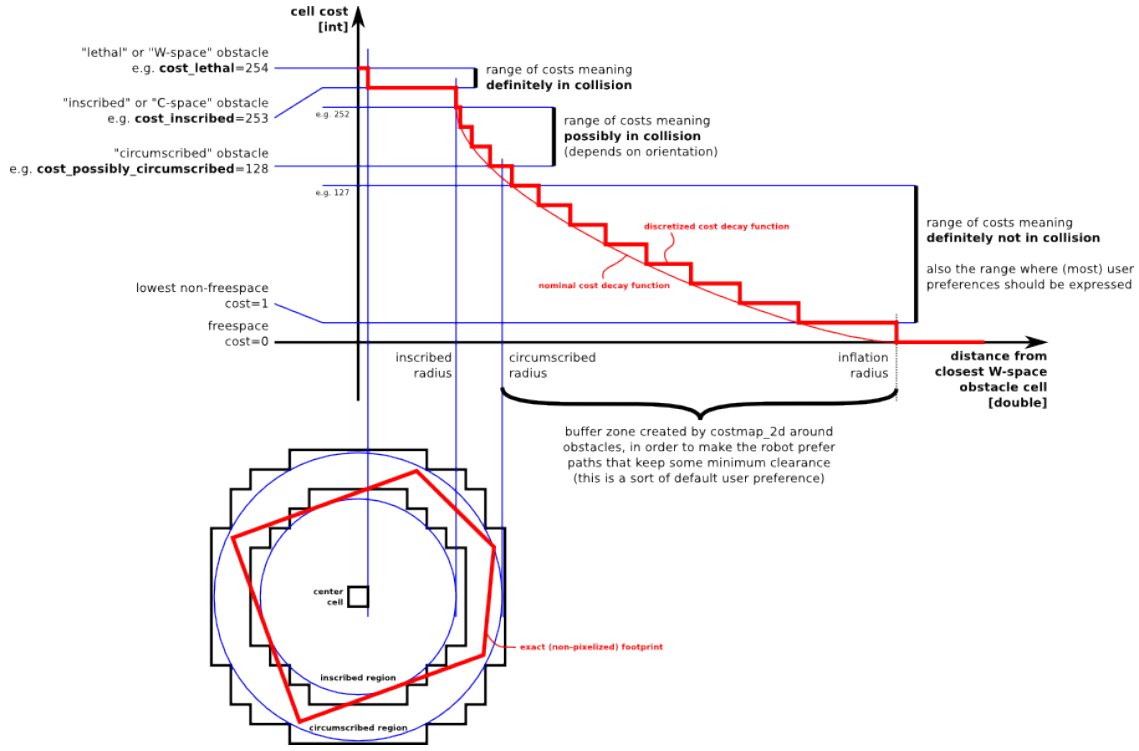
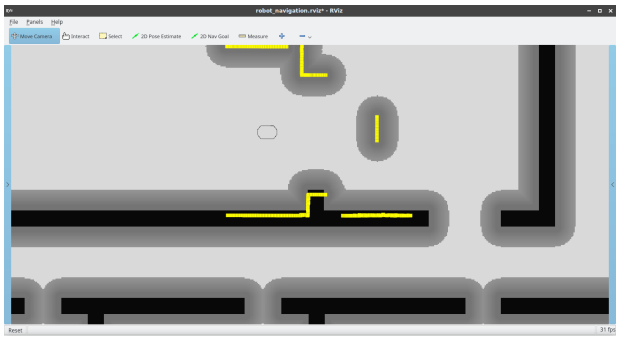


Figura 4.9: Vista geral do funcionamento da Camada de inflação. Imagem retirada de [http://wiki.ros.org/costmap_2d].

A figura 4.9 ilustra como são atribuídos os custos na zona de inflação em função de *cost_scaling_factor* e de *inflation_radius*. Resumidamente *inflation_radius* diz respeito à distancia máxima que deve ser inflacionada em relação a um objeto. Na figura 4.10 é possível observar que à medida que se aumenta *inflation_radius* aumenta a zona de inflação.

cost_scaling_factor diz respeito ao fator de decaimento do valor da inflação em função da distância. Na figura 4.11 é possível observar que para valores altos a zona de inflação tende rapidamente para zero.



(a) *inflation_radius* = 0.5

(b) *inflation_radius* = 1.0

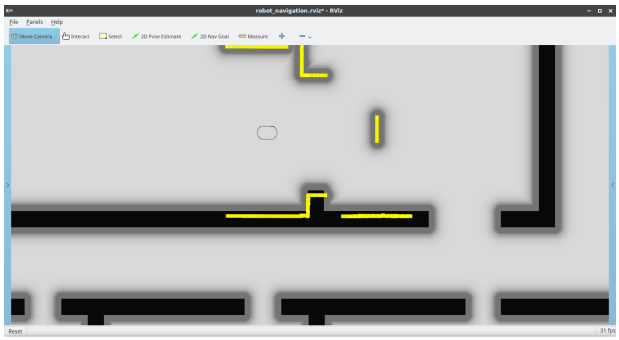
(c) *inflation_radius* = 1.5

(d) *inflation_radius* 2.0

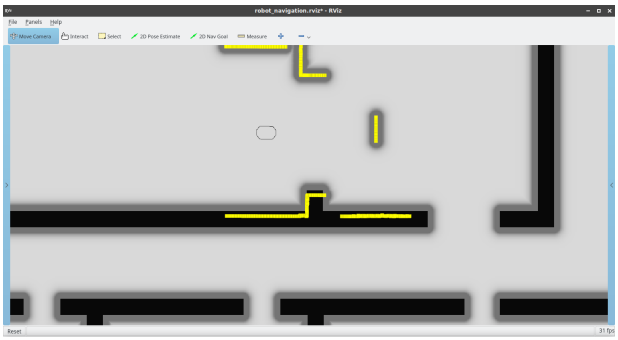
Figura 4.10: Evolução da zona de inflação do mapa de custos à medida que se aumenta o *inflation_radius*, com *cost_scaling_factor* = 1.0.

(a) *cost_scaling_factor* = 2.0

(b) *cost_scaling_factor* = 6.0



(c) *cost_scaling_factor* = 10.0



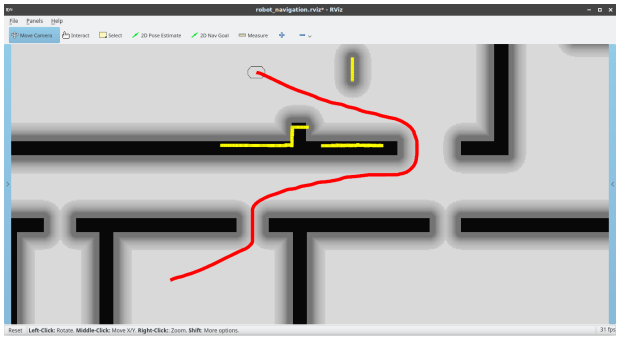
(d) *cost_scaling_factor* = 14.0

Figura 4.11: Evolução da zona de inflação do mapa de custos à medida que se aumenta o *cost_scaling_factor*, com *inflation_radius* = 2.0.

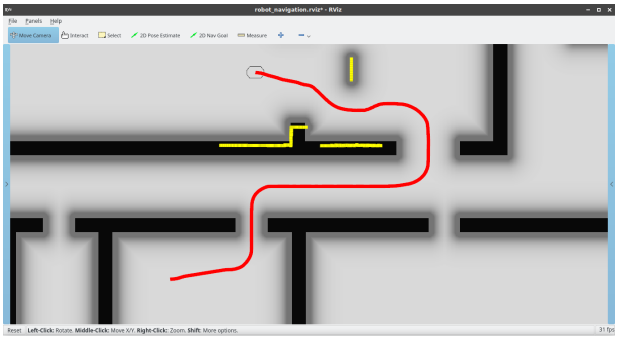
Influencia da camada de inflação no planeador global

O mapa de custos global permite influenciar o caminho criado pelo planeador global. As figuras 4.12 e 4.13 mostram como é possível ajustar o caminho global fazendo uso dos valores de *cost_scaling_factor* e de *inflation_radius*. A figura 4.12 mostra como o caminho global muda quando se aumenta o valor de *inflation_radius* de $0,5m$ até $3.0m$, com um valor para *cost_scaling_factor* constante de 5.0. Como se pode observar aumentar o raio de inflação resulta em caminhos que tendem a afastarem-se das paredes. Para raios superiores a $1.0m$, figuras 4.12b, 4.12c, 4.12d, a diferença entre os raios de inflação parecem não provocar grande alteração no caminho gerado.

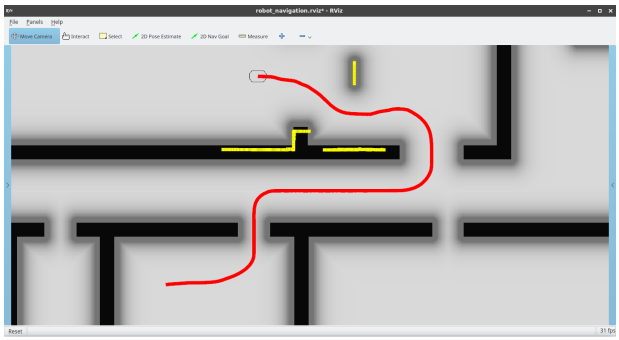
Por outro lado a figura 4.13 mostra os diferentes caminhos quando se varia o valor do fator de decaimento para um valor constante de $2.0m$ para o raio de inflação. Valores baixos de *cost_scaling_factor*, figura 4.13a, resultam em trajetórias desfavoráveis a passar muito perto de paredes. Aumentar um pouco este valor leva a criação de um caminho melhor que aumenta a distância do caminho gerado às paredes, figuras 4.13b e 4.13c, mas a partir de um certo valor a qualidade da trajetória piora voltando a aproximar-se das paredes, figura 4.13d.



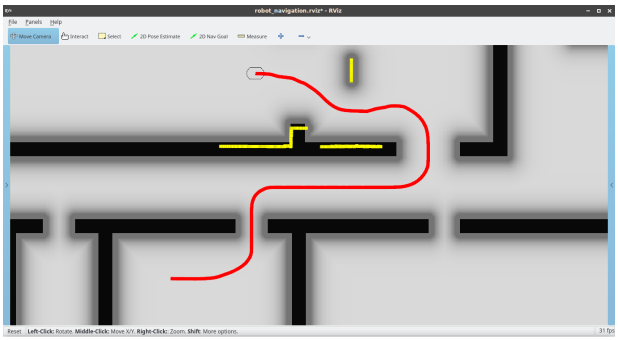
(a) *inflation_radius* = 0.5



(b) *inflation_radius* = 1.0

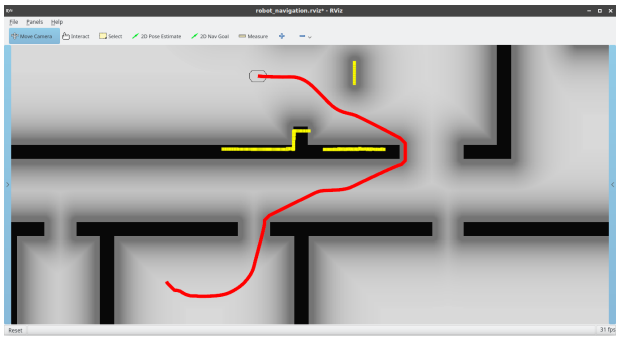


(c) *inflation_radius* = 2.0

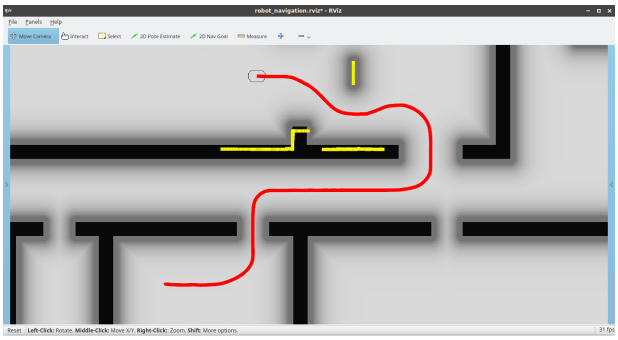


(d) *inflation_radius* = 3.0

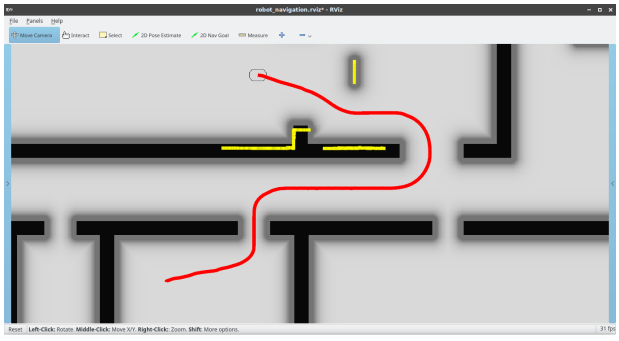
Figura 4.12: Comparação entre os caminhos produzidos com diferentes valores de *inflation_radius* e com *cost_scaling_factor* = 5.0 no mapa de custos global.



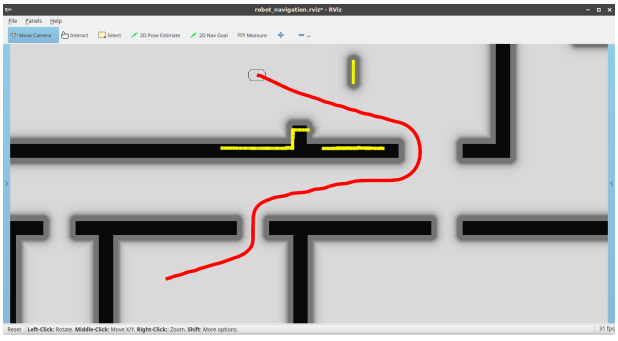
(a) *cost_scaling_factor* = 2.0



(b) *cost_scaling_factor* = 4.0



(c) *cost_scaling_factor* = 7.0



(d) *cost_scaling_factor* = 12.0

Figura 4.13: Comparação entre os caminhos produzidos com diferentes valores de *cost_scaling_factor* e com *inflation_radius* = 2.0 no mapa de custos global.

Influencia da camada de inflação no planeador local

Se relembrarmos a equação 4.1 o terceiro termo da soma é calculado em função dos valores do mapa de custos local. Portanto foi-se averiguar de que forma a variação dos valores de custo afetam a trajetória efetuada pelo robô. De uma forma geral a variação destes valores parece afetar pouco a trajetória do robô. Verificou-se que para valores constantes de *cost_scaling_factor* reduzir o valor de *inflation_radius* faz o robô aproximar das paredes, da mesma forma que para valores constantes de *inflation_radius* aumentar o valor de *cost_scaling_factor* produz o mesmo efeito (ver figura 4.15).

(a) *cost_scaling_factor* = 4.0 **(b)** *cost_scaling_factor* = 10.0

Figura 4.14: Comparação entre os caminhos produzidos, linha vermelha, quando se aumenta *cost_scaling_factor* com um valor constante para *inflation_radius* de 2.0 no mapa de custos local.

(a) *inflation_radius* = 2.5 **(b)** *inflation_radius* = 0.5

Figura 4.15: Comparação entre os caminhos produzidos, linha vermelha, quando se aumenta *inflation_radius* com um valor constante para *cost_scaling_factor* de 5.0 no mapa de custos local.

4.1.3 Uma nova configuração para o *move_base*

Com o objetivo de dar uma maior margem de segurança tentou-se fazer com que o robô durante a navegação maximizasse a distância aos objetos em seu redor. Para isto,

tendo em conta a análise anterior, foram feitas as seguintes alterações: No planeador local reduziu-se o valor *goal_distance_bias* de 24.0 para 8.0 e *sim_time* de 1.7 para 1.0. Estas mudanças resultaram numa trajetória que segue com maior rigor a referência dada pelo planeador global. Por outro lado de forma a que o planeador global gere caminhos mais favoráveis no mapa de custos global alterou-se o valor de *inflation_radius* para 2.0 e o de *cost_scaling_factor* para 10.0. Apesar dos resultados promissores de quando se aumenta *occdist_scale* foi decidido manter não alterar este valor a fim de evitar a situação da figura 4.7.

Obrigar o robô a andar perto do caminho global implica que o robô perde capacidade de ultrapassar obstáculos não previstos. Para resolver este problema usou-se o parâmetro *planner_frequency* do *move_base* que permite chamar o planeador global a uma determinada frequência. Invocar o planeador global periodicamente permite ajustar a trajetória tendo em conta os novos objetos encontrados ao longo do caminho que não não existiam no mapa previamente.

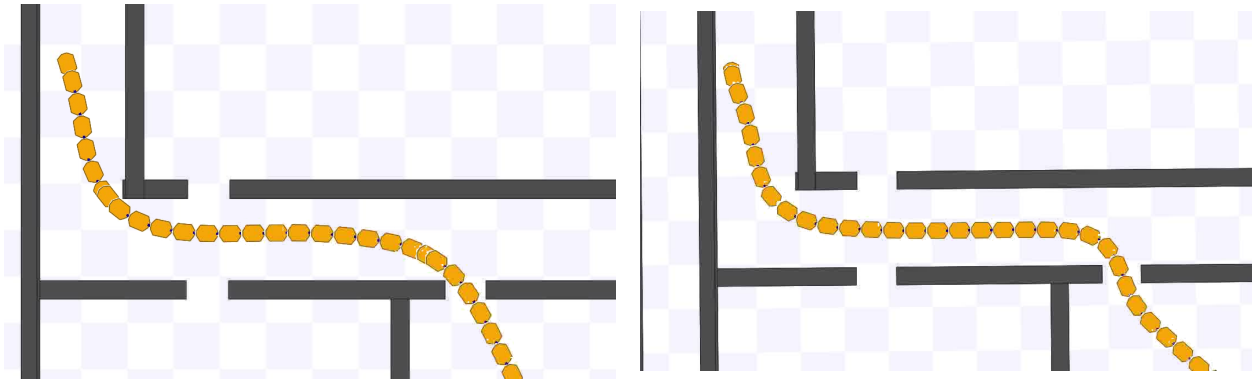
Nome	**Valor predefinido**	**Novo valor**	**Objetivo**
dwa_local_planner			
goal_distance_bias	24.0	8.0	Aproximar a trajetória ao caminho global
sim_time	1.7	1.0	Aproximar a trajetória ao caminho global
Mapa de custos Global			
cost_scaling_factor	10.0	10.0	Gerar percursos mais favoráveis no planeador global
inflation_radius	0.55	2.0	Gerar percursos mais favoráveis no planeador global
move_base			
planner_frequency	0.0	2.0	Atualizar plano global com novos valores do mapa de custos

Tabela 4.2: Tabela resumo com as alterações dos parâmetros do *move_base*.

4.1.4 Validação dos novos valores em simulação

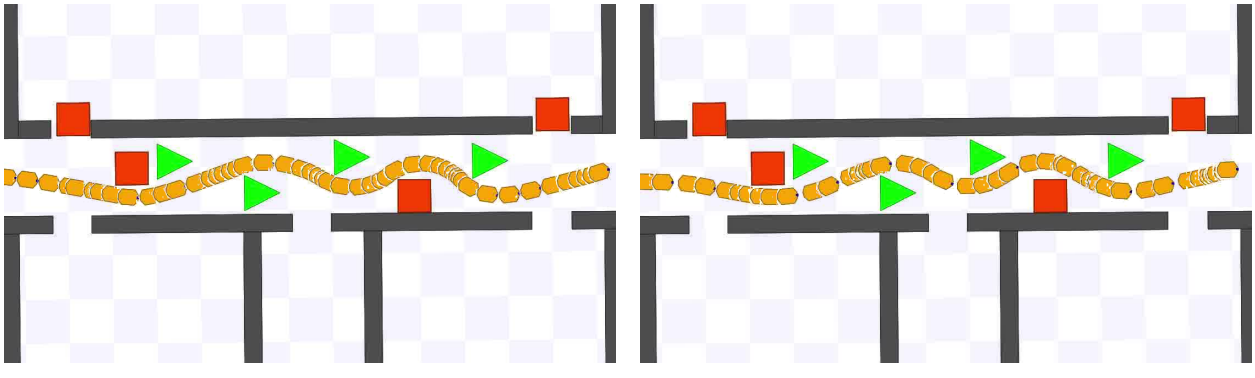
Nas figuras 4.16 e 4.17 são mostradas as trajetórias efetuadas pelo robô durante a execução de duas tarefas simples. Na figura 4.16 foi pedido ao robô que se deslocasse até uma sala e para isso o robô teria que percorrer um corredor dobrar uma esquina e atravessar uma porta. Na figura 4.17 a tarefa consiste em percorrer um corredor cheio de obstáculos desconhecidos que o robô deve ultrapassar.

Na figura 4.16 é visível como a utilização dos novos parâmetros causa o robô a afastar-se das paredes. Na figura 4.17 a diferença entre as trajetórias não percetível devido ao pouco espaço disponível ao robô, no entanto observou-se que com os novos parâmetros o robô é capaz de executar o percurso em menos tempo; no caso da figura 4.17a o robô demorou $\approx 22s$ e no caso da figura 4.17b o robô demorou $\approx 20s$.



(a) **(b)**

Figura 4.16: Comparação entre valores predefinidos (a) e novos valores (b).



(a) **(b)**

Figura 4.17: Comparação entre valores predefinidos (a) e novos valores (b).

4.2 Sensores

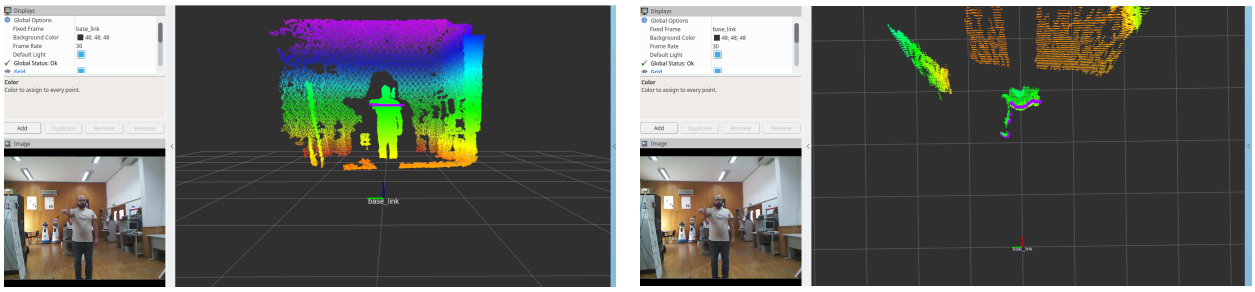
Os nossos sensores apresentam uma série de dificuldades que não nos permite tirar o máximo proveito da camada de obstáculos. Para ultrapassar esta situação foi necessário aplicar medidas adicionais que são descritas a seguir para cada sensor.

4.2.1 Hokuyo - URG-04LX-UG01

No caso do laser rangefinder foram analisados os dados fornecidos pelo nó *urg_node*. Na análise verificou-se que todas as leituras invalidas do sensor, tais como leituras acima do alcance do sensor, são atribuídas um valor NaN. Neste caso a solução passou por criar um outro nó ROS que lê os dados provenientes do sensor e para leituras com o valor de NaN altera para um valor de +inf e depois envia os dados corrigidos para o *move_base*.

4.2.2 Câmara *kinect* 360

Para a câmera *kinect* 360 optou-se por transformar a *PointClould2* em *LaserScan*. A transformação foi feita com a ajuda do pacote ROS *pointcloud_to_laserscan* [2]. O *LaserScan* criado por este pacote obedece à convenção mencionada e portanto consegue-se utilizar a flag *inf_is_true* com um sensor 3d.



(a) **(b)**

Figura 4.18: Geração de um laser virtual, linha a roxo, a partir de uma *PointCloud*. Vista frontal (a) e vista de cima (b)

4.3 Introdução das camadas sociais de navegação

De forma a facilitar a passagem do robô por obstáculos que se movem o robô precisa de um modelo preditivo que permita ao robô estimar a posição do objeto ao longo do tempo de forma a poder tomar uma decisão correta sobre como ultrapassar o obstáculo.As camadas sociais

[2]http://wiki.ros.org/pointcloud_to_laserscan

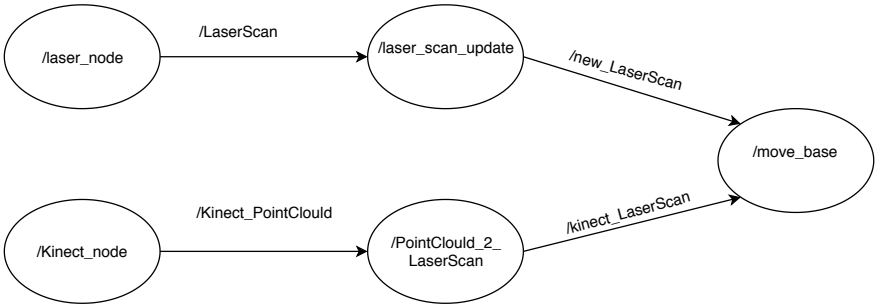
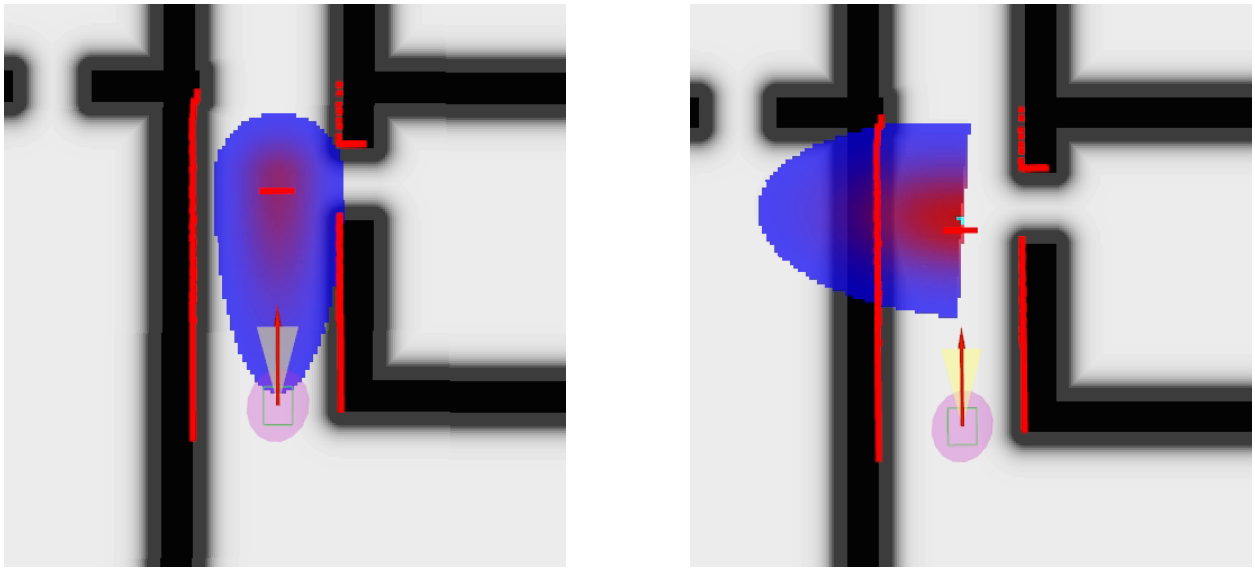


Figura 4.19: Visão geral das ligações entre os nós dos sensores, os nós introduzidos e o *move_base*.

de navegação [3] disponível no ROS oferecem a capacidade de predição ao robô ao inflacionar os valores do mapa de custos. Existem dois tipos de camadas sociais a: *ProxemicLayer* e a *PassingLayer*.

A *ProxemicLayer*, (fig. 4.20a), inflaciona os custos à volta de uma pessoa detetada de acordo com uma distribuição Gaussiana. Quando a pessoa detetada está parada a zona de inflação corresponde a um circulo, mas se a pessoa estiver a movimentar-se então a zona de inflação aumenta na direção do movimento da pessoa. O tamanho da zona de inflacionada em frente da pessoa depende da velocidade.



(a) *ProxemicLayer* **(b)** *PassingLayer*

Figura 4.20: A figura mostra a área inflacionada pela camadas sociais, áreas a azul correspondem a baixos custos e áreas a vermelho correspondem a valores mais altos.

A *PassingLayer*, (fig. 4.20b), serve simplesmente para fazer o robô passar pelo lado esquerdo da pessoa, para isso apenas inflaciona os custos apenas de um lado.

[3] http://wiki.ros.org/social_navigation_layers

4.3.1 Validação em simulação

Primeiro foi analisado a situação de passar por uma pessoa num corredor. Nesta situação foi usada apenas a *ProxemicLayer*. O raciocínio para esta escolha foi o de que se assumirmos que o robô anda pelo centro do corredor então o robô deve escolher o lado que oferece maior distância entre a pessoa e a parede do corredor em vez de tentar obrigar o robô a ultrapassar a pessoa sempre pelo mesmo lado. Por outro lado no caso de dois robôs autónomos o uso da *PassingLayer* é o mais indicado.

Nas figuras 4.21 e 4.22 é mostrado o resultado da introdução da *ProxemicLayer* para facilitar a passagem do robô num corredor onde o robô tem de interagir com um humano a mover-se em sentido contrário. A figura 4.22 é possível ver o momento em que o robô re-planeou a trajetória tendo em conta a nova informação sobre o movimento da pessoa. Graças à *ProxemicLayer* o robô consegue uma trajetória que evita a área prestes a ser ocupada e o desvio do obstáculo é feito de forma a evitar a colisão.

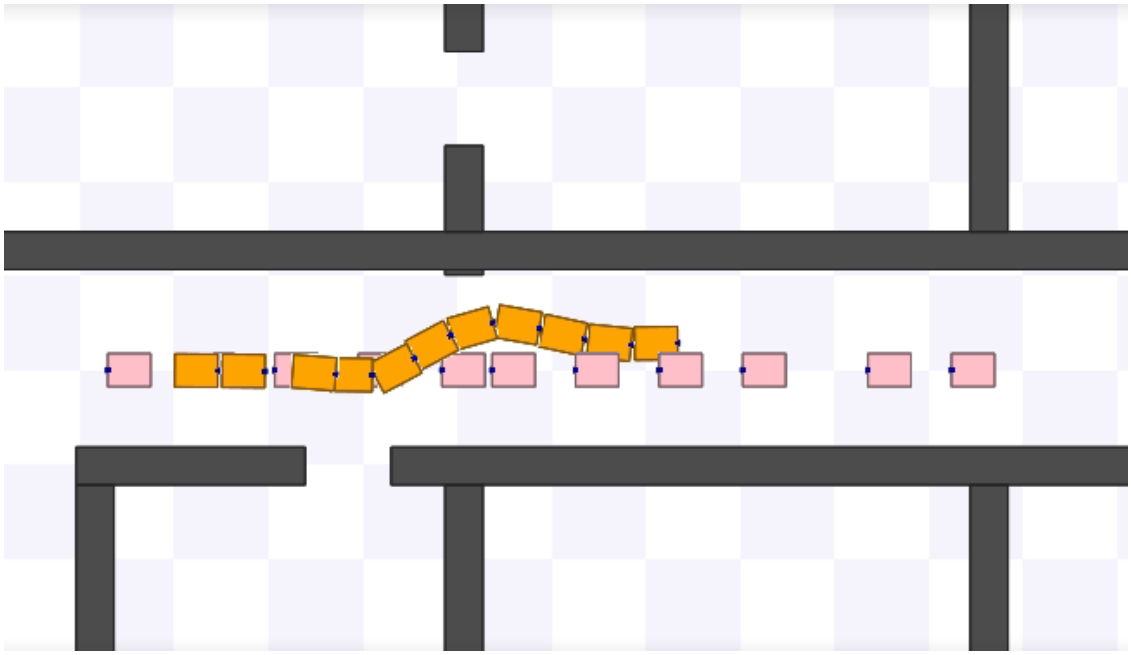


Figura 4.21: Teste de interação robô pessoa no corredor com a *ProxemicLayer*. A figura mostra que o robô foi capaz de se desviar a tempo de evitar a colisão.

Para dois robôs autónomos a introdução da *PassingLayer* permite aos robôs planearem uma trajetória compatível onde cada robô escolhe um lado do corredor por onde passar. Desta forma ambos os robôs conseguem passar sem a necessidade de um movimento de "dança"no centro do corredor. A figura 4.24 mostra a trajetória planeada pelos robôs aquando da presença do outro robô e a figura 4.23 mostra a trajetória realmente executada pelos robôs.

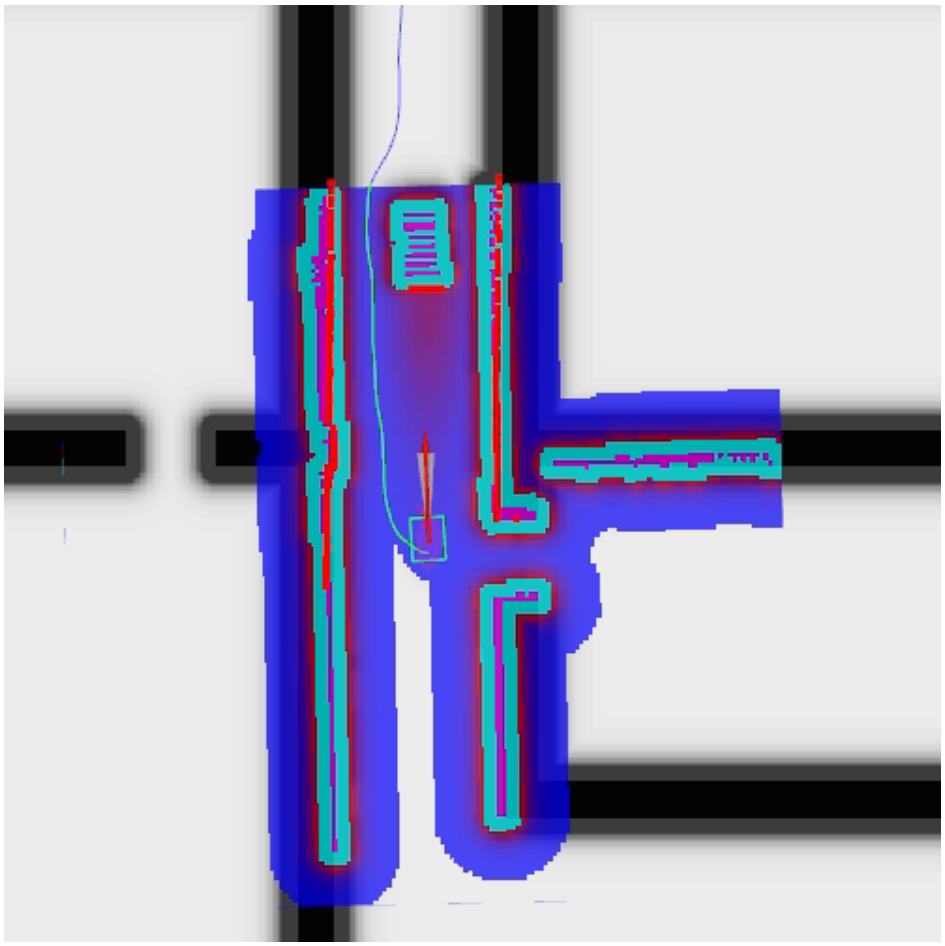


Figura 4.22: Teste de interação robô pessoa no corredor com a *ProxemicLayer*. A figura mostra como a introdução da *ProxemicLayer* altera a trajetória gerada.

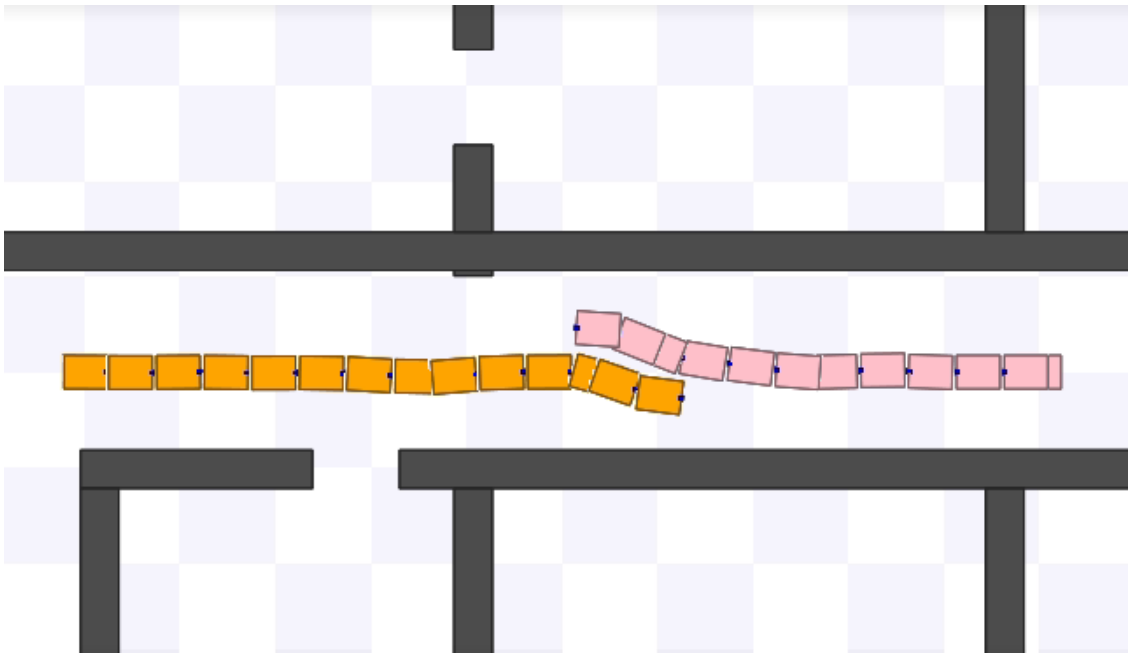
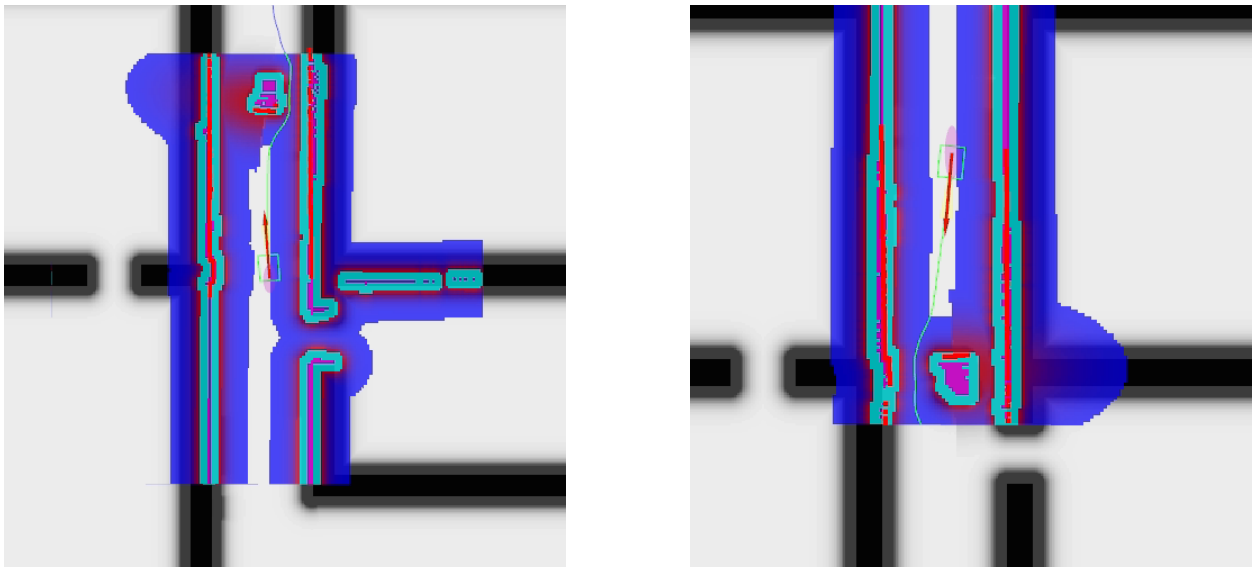


Figura 4.23: Teste de interação entre dois robôs num corredor após aintrodução da *PassingLayer*. A figura mostra os robôs a negociar a passagem de forma a que ambos possam continuar sem problemas.



(a) robô cor-de-laranja **(b)** robô cor-de-rosa

Figura 4.24: Teste de interação entre dois robôs num corredor após a introdução da *PassingLayer*. A figura mostra como a *PassingLayer* obriga a ultrapassar o obstáculo pela direita evitando uma situação de impasse.

Capítulo 5

Testes com um robô Pioneer 3dx

5.1 Patrulhamento de um corredor

O primeiro teste consiste numa missão de patrulhamento nos corredores do piso 0 do ISR. Este corredor é um ambiente simples e amplo o que não deve apresentar grandes dificuldades ao robô. Durante a missão o robô tomou sempre uma posição defensiva maximizando a distâncias às paredes o que resultou numa trajetória pelo centro do corredor, figura 5.1. Durante uma das voltas o robô deparou-se com uma porta que ocupou a maior parte do caminho, figura 5.2. Perante esta situação inesperada o robô foi capaz de re-planear e ultrapassar o desafio apresentado sem grandes dificuldades. No final podemos dizer que o robô comportou-se de forma muito satisfatória e de acordo com as expectativas.

(a) Robô a percorrer um corredor **(b)** Robô a dobrar uma esquina

Figura 5.1: Imagem de um robô real durante a patrulha de um corredor

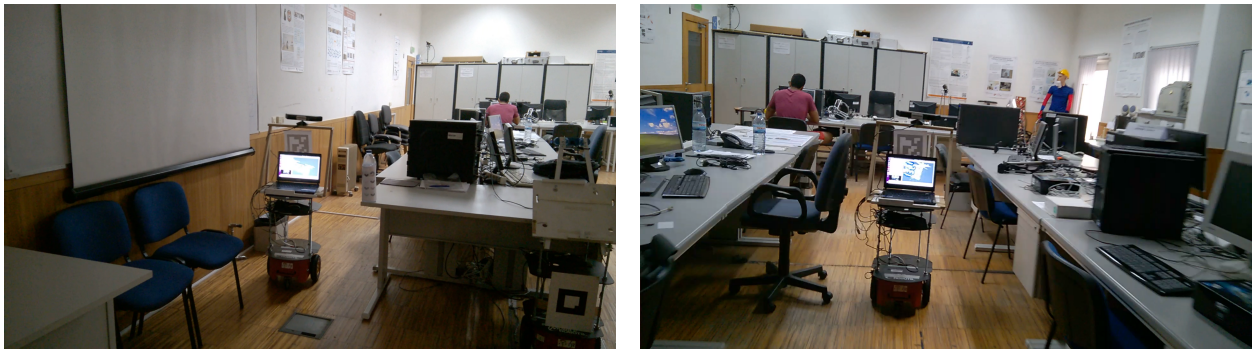
(a) (b) (c)

Figura 5.2: Imagem de um robô real durante a patrulha de um corredor

5.2 Navegação dentro de um escritório

O segundo teste consiste em dar ao robô um conjunto de coordenadas dentro de um escritório. Depois o robô deve ser capaz de alcançar cada um destes objetivos de forma autónoma. O escritório corresponde a um ambiente mais desorganizado com uma grande variedade de obstáculos que podem não ser detetados de forma correta ou fiável pelos sensores do robô, e de passagens estreitas que só permitem a passagem de uma pessoa de cada vez. Estas características tornam o escritório um local ideal para pôr à prova as capacidades do robô.

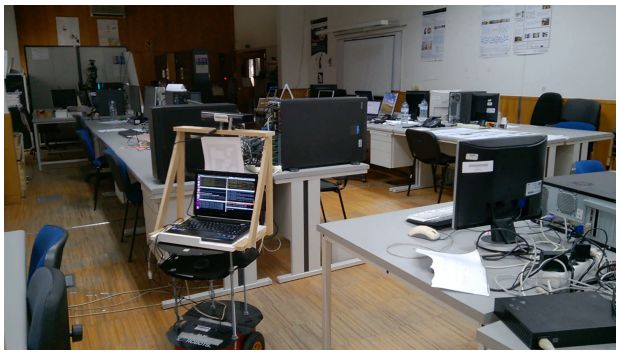
Durante a realização da tarefa provou ser capaz de detetar e representar corretamente os variados objetos que constituem o escritório. Ao longo do percurso realizado foi visível como o robô tenta posicionar-se de maximizando as distâncias ao objetos. Este comportamento é especialmente importante quando à necessidade de realizar uma curva em espaços apartados porque ajuda ao robô evitar poses criticas das quais não consegue recuperar. Na figura 5.3 é mostrado o robô em algumas das situações mais críticas encontradas no seu caminho. No final todos os objetivos foram conseguidos e mais uma vez o robô comportou-se de forma satisfatória e de acordo com as expectativas.



(a) (b)

(c)

(d)



(e)

(f)

Figura 5.3: Imagens de um robô a por entre os corredores de um escritório.

5.3 Navegação robusta na presença de objetos dinâmicos

No capitulo anterior vimos como as camadas sociais do mapa de custos podem ser uma boa opção para lidar com outras entidades com capacidade de locomoção. No entanto para poder usar estas camadas é necessário um mecanismo que permita a identificação e a localização dessas entidades (e.g. pessoas, outros robôs, etc.). Para fazer a deteção de pessoas foram utilizados os seguintes pacotes ROS: *leg_detector* [1] e *people_velocity_tracker* [2]. O nó *leg_detector* usa uma rede neuronal treinada para detetar conjuntos de medições que correspondam a um par de pernas na leitura feita pelo laser rangefinder. Quando um par de pernas é detetado uma mensagem é gerada que contém a posição no mapa da pessoa e essa mensagem é enviada para o nó *people_velocity_tracker*. O nó *people_velocity_tracker* recebe as mensagem com a posição associadas a uma pessoa ao longo do tempo de forma a calcular a velocidade da pessoa e gera a mensagem que é enviada para o *move_base*.

[1] http://wiki.ros.org/leg_detector
[2] http://wiki.ros.org/people_velocity_tracker

Para a deteção de um robô por foi implementada, no nosso sistema, uma solução baseada na biblioteca ArUco do openCV. Um marcador ArUco consiste num marcador sintético composto por uma borda preta e uma matriz binária que determina o id do marcador, figura 5.4. A partir das funções do openCV criou-se um novo nó, *aruco_marker_detetor*. Este nó recebe uma imagem proveniente de uma câmara RGB montada no robô e se existir um marcador na imagem o programa é capaz de o identificar e calcular a pose do marcador em relação à câmara. O valor da pose do marcador é depois usado para criar uma mensagem que é enviada para o nó *people_velocity_tracker*. Como mostra a figura 5.5 um marcador ArUco foi colocado na frente da cada robô para que estes possam ser detetados.

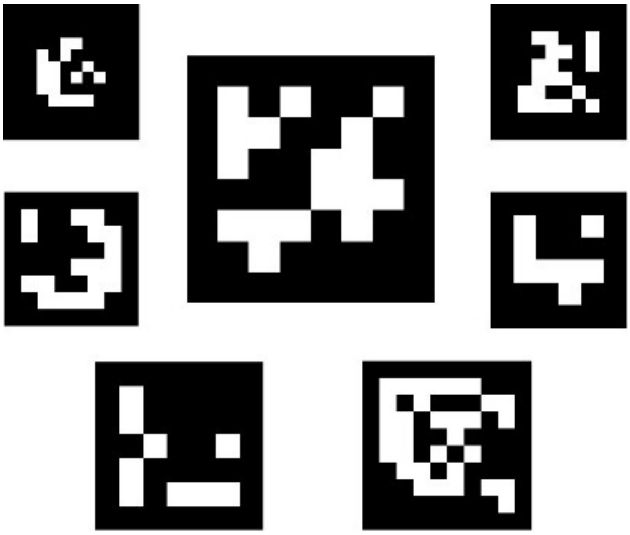


Figura 5.4: Exemplo de marcadores ArUco

Figura 5.5: Robôs com um marcador ArUco para identificação.

5.3.1 Interação robô-Humano

Apesar de em simulação apresentar resultados promissores, as experiências com um robô real revelaram que as camadas sociais estão longe de ser uma solução perfeita. O sistema de deteção de pessoas constituído por *leg_detector* e *people_velocity_tracker* foi capaz de identificar a presença de uma pessoa a mover, mas por vezes o valor de velocidade medido afasta-se muito do valor real. No exemplo da figura 5.6 uma pessoa desloca-se ao longo do corredor, mas o vetor de velocidade calculado por *people_velocity_tracker* tem uma orientação diferente da orientação do vetor de velocidade real da pessoa. A camada de custos inflaciona o valor das células ao longo do vetor de velocidade calculado, esta inflação pode causar problemas na geração de trajetórias e no caso mais grave pode levar ao robô a virar em direção à pessoa o que é exatamente o contrário do que é pretendido. Além desta situação à que notar que, mesmo quando o vetor de velocidade é calculado corretamente, a deteção é feita já muito próxima da pessoa o que leva o desvio do robô seja feito imediatamente à frente da pessoa (ver figura 5.7).

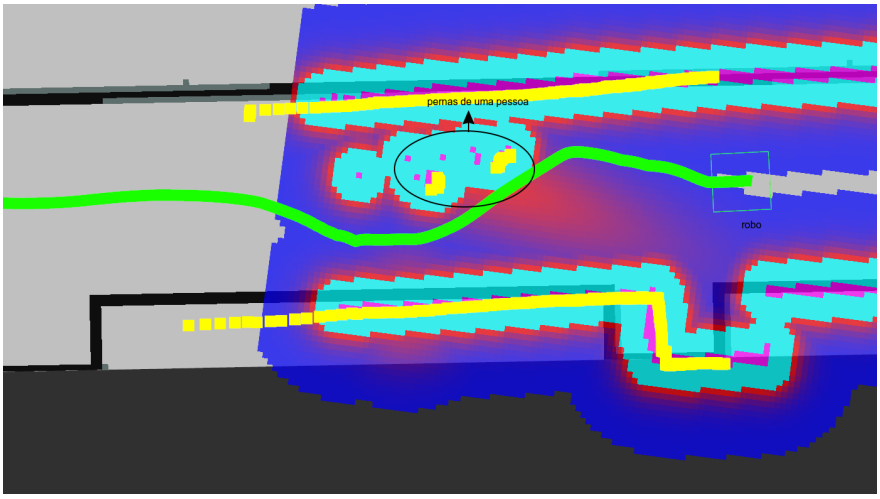
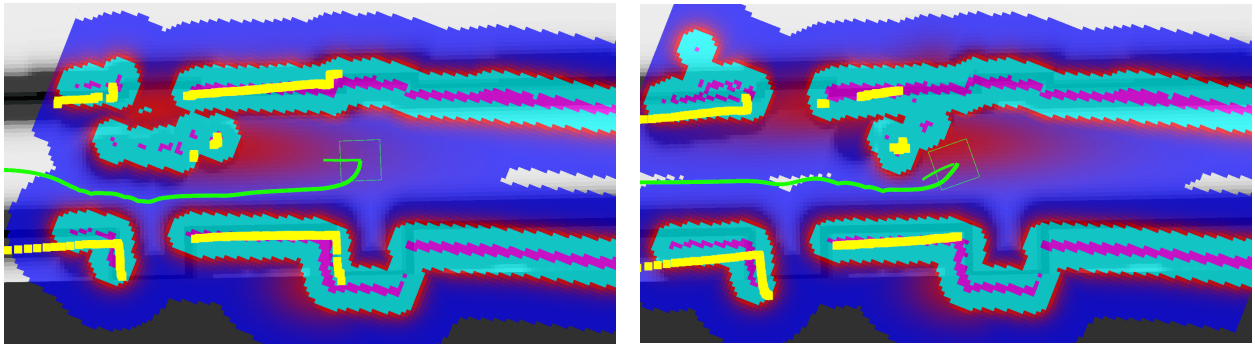


Figura 5.6: Exemplo que mostra como leituras erradas de velocidade pode causar problemas. O quadrado verde representa o robô, a linha verde representa o plano global.

Para detetar a pessoa a uma maior distância usou-se um marcador ArUco para o cálculo da posição da pessoa onde a pessoa segura o marcador à sua frente à medida que se desloca como mostra a figura 5.8.

Com o marcador ArUco o robô consegue ver a pessoa mais longe mas a velocidade calculada, em muita ocasiões, continua a ter erro. Para colmatar os erros da orientação da velocidade criou-se um nó que recebe as mensagens vindas de *people_velocity_tracker* e projeta o vetor da velocidade no vetor da posição entre a pessoa e o robô. A introdução deste nó no sistema resultou numa melhoria no desempenho. O replaneamento da trajetória

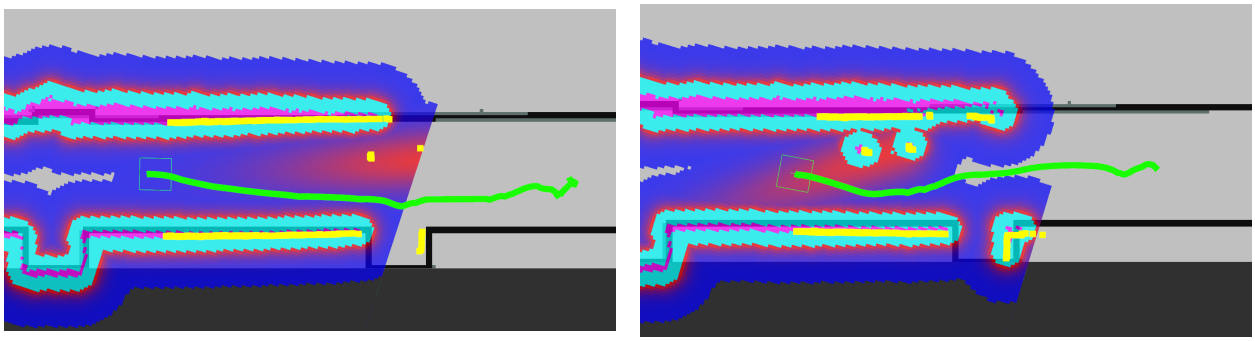


(a) (b)

Figura 5.7: A figura 5.7a mostra o momento em que o robô detetou a pessoa e replaneou a trajetória,($dist. \approx 2.0m$). A figura 5.7b mostra o momento em que o robô começou a virar para se desviar do obstáculo ($dist. \approx 0.1m$).

Figura 5.8: Pessoa a segurar um marcador arUco durante a realização da experiência.

acontece mais cedo e o robô mantém uma maior distância em relação à pessoa , figura 5.9a. Apesar das melhorias concedera-se que as distâncias mencionadas continuam a ser desconfortáveis para uma pessoa, para além fo facto de que em alguns dos exercícios o robô teve um comportamento semelhante ao mencionado na figura 5.6 onde o robô se aproxima demasiado da pessoa.



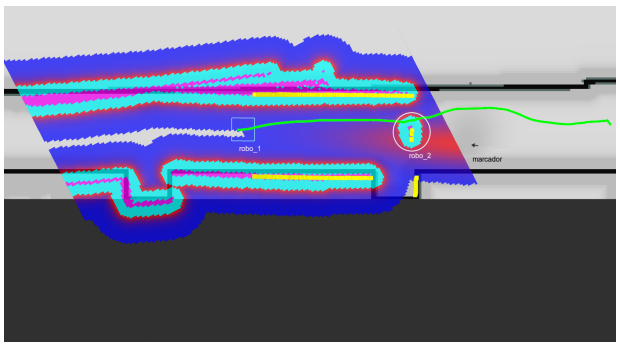
(a) (b)

Figura 5.9: A figura 5.9a mostra o momento em que o robô detetou a pessoa e replaneou a trajetória,($dist. \approx 4.0m$). A figura 5.9b mostra o momento em que o robô começou a virar para se desviar do obstáculo ($dist. \approx 0.5m$).

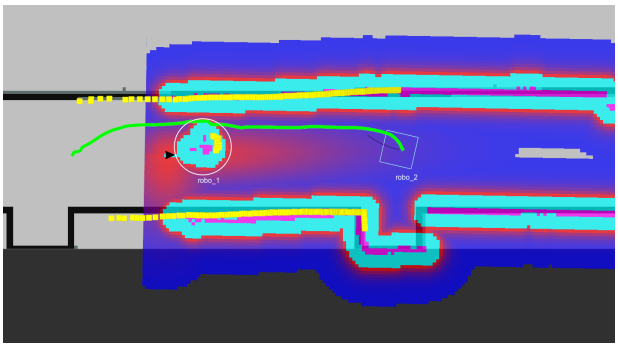
5.3.2 Interação robô-robô

Para facilitar a negociação da passagem entre dois robôs, robo_1 e robo_2, num corredor, são usadas as camadas sociais com a ajuda de marcadores ArUco para deteção do robô. As experiências realizadas mostraram que o robô não consegue detetar a pose do marcador ArUco com elevada precisão, especialmente para distâncias maiores do que $5m$. A inflação causada pelas camadas sociais do ROS ,que resultam de medidas erradas de posição, podem levar a trajetórias que diferem do objetivo inicial, a figura 5.10 ilustra um exemplo desta situação. As figuras do lado esquerdo, figuras 5.10a, 5.10c, 5.10e e 5.10g, mostram o ponto de vista do robo_1 durante o teste. As figuras do lado direito, figuras 5.10b, 5.10d, 5.10f e 5.10h, o ponto de vista do robô2.

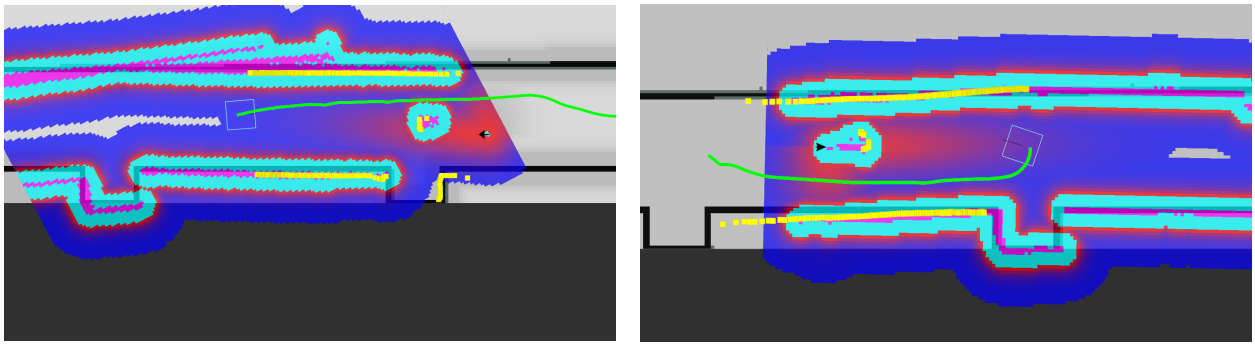
Idealmente as camadas sociais deveriam fazer o robo_1 passar à esquerda do robo_2 e vice-versa, de outra forma, nas condições da figura 5.10 o robo_1 deveria em direção à parde de baixo eo robo_2 em direção à parede de cima. . As figuras 5.10a e 5.10b mostram o momento em que os dois robôs se detetam. Na figura 5.10a é visível que a deteção do marcador coloca, erradamente, o robo_2 junto à parede de baixo o que causa com que o robo_1 vire em direção à parte de cima da parede, ao mesmo tempo o robo_2 deteta o robo_1 e calcula uma trajetória a passar também pela parte de cima do corredor. Na figura 5.10d o robo_2 é forçado a recalcular a sua trajetória devido à presença do robo_1 no seu caminho que planeaou erradamente a sua trajetória. Na figura 5.10f o robo_2 desvia-se para a sua esquerda abrindo espaço para que o robo_1 possa continuar a sua trajetória junto à parede de cima, 5.10e. No final, figuras 5.10g e 5.10h, o robo_1 passou à direita do robo_2 o que é o oposto do comportamento desejado.



(a)

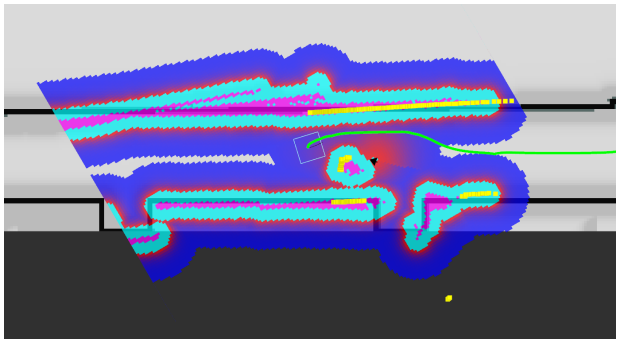


(b)

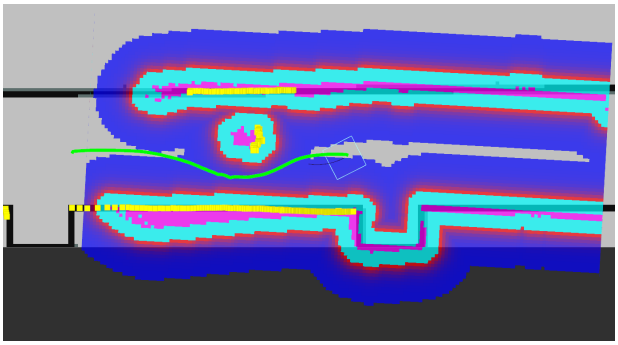


(c)

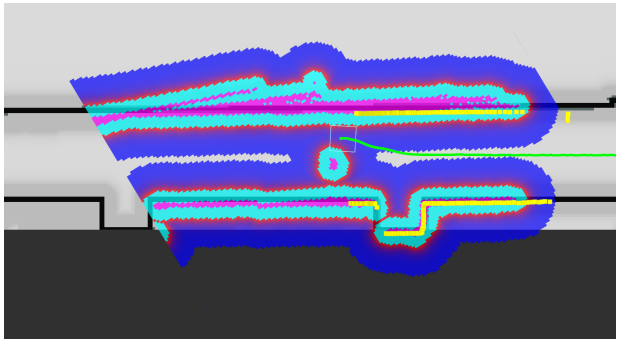
(d)



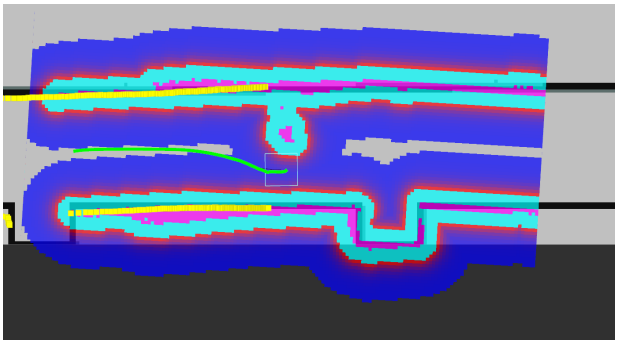
(e)



(f)



(g)



(h)

Figura 5.10: Negociação da passagem entre dois robôs num corredor. Uma seta preta é utilizada para representar a localização do marcador medida.

Capítulo 6

Conclusão

Durante a realização deste trabalho foram postas à prova as capacidades do *move_base* de planear trajetórias, de recolher informação sobre o seu meio envolvente e a capacidade de negociação de passagem com outras entidades que possam existir no mundo.

Os planeadores do *move_base*, global e local, expõem ao utilizador um conjunto de parâmetros que permitem variar o comportamento do robô. Provou-se que com uma afinação cuidadosa é possível produzir e executar trajetórias em diversas situações, incluindo áreas com pouco espaço de manobra onde o robô facilmente pode colidir.

A camada de obstáculos do *move_base* mostrou ser eficaz para representar o mundo conseguindo mapear objetos com grande detalhe. No entanto foi identificado que sensores existentes podem não conseguir usar ao máximo a camada de obstáculos sendo necessário implementar técnicas adicionais de forma a tentar colmatar as incompatibilidades entre o sensor e a camada de obstáculos.

A camada social do *move_base* tenta ser uma ferramenta que permite facilitar a interação entre o robô e um objeto dinâmico. Para esta ser eficaz é necessário que a deteção seja efetuada a uma grande distância e com grande precisão. A capacidade de deteção do robô usado nas experiências realizadas é muito reduzida e é afetada por erros de medição. O fraco desempenho do sistema de deteção não permite um uso eficaz das camadas sociais do ROS o que impede o robô de negociar a passagem com os obstáculos corretamente.

Para um robô que navegue sozinho andar pelo centro de um corredor é um comportamento de segurança, mas se esse robô tiver que partilhar o seu espaço com outros robôs ou pessoas andar pelo centro do corredor pode não ser o mais correto especialmente se o corredor for largo o suficiente para duas entidades navegarem em paralelo. Ao ocupar uma posição central, o robô efetivamente está a minimizar a distância disponível a uma outra entidade

para se desviar do robô. Durante a realização das experiências no corredor foi observado que quando uma pessoa que se desloca ao longo do corredor com o robô, a pessoa tende a encostar-se a uma parede e a tomar cuidado extra em relação ao robô devido à pequena distancia entre o robô e a parede.

Uma solução possível consiste em: quando um robô determina que se encontra num corredor, o robô analisa o corredor e se verificar que existe espaço suficiente então deve instruir o planeador local a encostar, por exemplo, à direita. Esta solução seria interessante porque para o caso de dois robôs a deslocarem-se em sentido oposto eliminava o problema uma vez que os robôs estariam a navegar em faixas diferentes. No caso da interação com humanos também trazia vantagens. Ao mover-se para um dos lados o robô está a deixar um espaço onde o humano possa passar confortavelmente tornando a interação mais agradável para a pessoa.

Bibliografia

[1] Oliver Brock and Oussama Khatib. High-speed navigation using the global dynamic window approach. In *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, volume 1, pages 341–346. IEEE, 1999.

[2] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[3] Gilberto Echeverria, Séverin Lemaignan, Arnaud Degroote, Simon Lacroix, Michael Karg, Pierrick Koch, Charles Lesire, and Serge Stinckwich. Simulating complex robotic scenarios with morse. In *SIMPAR*, pages 197–208, 2012.

[4] Alberto Elfes. Occupancy grids: A stochastic spatial representation for active robot perception. In *Proceedings of the Sixth Conference on Uncertainty in AI*, volume 2929, page 6, 1990.

[5] Dave Ferguson and Maxim Likhachev. Efficiently using cost maps for planning complex maneuvers. *Lab Papers (GRASP)*, page 20, 2008.

[6] Dieter Fox. *Markov localization: A probabilistic framework for mobile robot localization and navigation*. PhD thesis, Citeseer, 1998.

[7] Dieter Fox, Wolfram Burgard, Frank Dellaert, and Sebastian Thrun. Monte carlo localization: Efficient position estimation for mobile robots. *AAAI/IAAI*, 1999(343-349):2–2, 1999.

[8] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1):23–33, 1997.

[9] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[10] Nathan Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 3, pages 2149–2154. IEEE.

[11] Anthony Lazanas and Jean-Claude Latombe. Motion planning with uncertainty: a landmark approach. *Artificial intelligence*, 76(1-2):287–317, 1995.

[12] John J Leonard and Hugh F Durrant-Whyte. Mobile robot localization by tracking geometric beacons. *IEEE Transactions on robotics and Automation*, 7(3):376–382, 1991.

[13] David V Lu, Dave Hershberger, and William D Smart. Layered costmaps for context-sensitive navigation. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 709–715. IEEE, 2014.

[14] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[15] Saul Simhon and Gregory Dudek. A global topological map formed by local metric maps. In *Intelligent Robots and Systems, 1998. Proceedings., 1998 IEEE/RSJ International Conference on*, volume 3, pages 1708–1714. IEEE, 1998.

[16] Federico Thomas and Lluís Ros. Revisiting trilateration for robot localization. *IEEE Transactions on robotics*, 21(1):93–101, 2005.

[17] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.

[18] Sebastian Thrun, Jens-Steffen Gutmann, Dieter Fox, Wolfram Burgard, Benjamin Kuipers, et al. Integrating topological and metric maps for mobile robot navigation: A statistical approach. In *AAAI/IAAI*, pages 989–995, 1998.

[19] Richard Vaughan. Massively multi-robot simulation in stage. *Swarm Intelligence*, 2(2):189–208, Dec 2008.