Universidade Federal da Paraíba

Centro de Informática

Programa de Pós-graduação em Informática

Um Sistema Anti-colisão 3D baseado no método de Campo Potencial Artificial para um robô móvel

Carlos Eduardo Silva Morais

João Pessoa, Paraíba, Brasil

Fevereiro de 2017

Carlos Eduardo Silva Morais

Um Sistema Anti-colisão 3D baseado no método de Campo Potencial Artificial para um robô móvel

Dissertação submetida à Cordenação do Curso de Pós-Graduação em Informática do Centro de Informática, da Universidade Federal da Paraíba, como parte dos requisitos necessários para obtenção do grau de Mestre em Informática.

Área de Concentração: Ciência da Computação

Linha de Pesquisa: Sinais, Sistemas e Gráficos

Orientador: Tiago Pereira do Nascimento

Co-orientador: Alisson Vasconcelos de Brito

João Pessoa, Fevereiro de 2017

M827s Morais, Carlos Eduardo Silva.
Um sistema anti-colisão 3D baseado no método de Campo Potencial Artificial para um robô móvel / Carlos Eduardo Silva Morais.- João Pessoa, 2017.
91 f. : il. -

Orientador: Tiago Pereira do Nascimento.
Coorientador: Alisson Vasconcelos de Brito.
Dissertação (Mestrado) – UFPB/CI

1. Sistema Anti-Colisão 3D. 2. Campo Potencial Artificial. 3. Robô Móvel. 4. Kinect. I. Título.

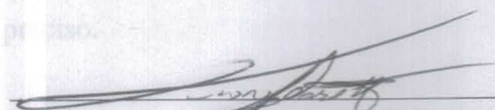
UFPB/BC CDU: 004(043)

Centro de Informática

Universidade Federal da Paraíba

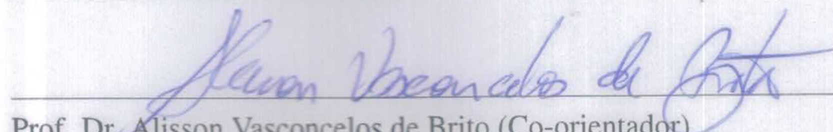
Programa de Pós-Gradução em Informática

Dissertação do Curso de Pós-Graduação em Informática intitulado ***Um Sistema Anti-colisão 3D baseado no método de Campo Potencial Artificial para um robô móvel*** de autoria de Carlos Eduardo Silva Morais, aprovada pela banca examinadora constituída pelos seguintes professores:



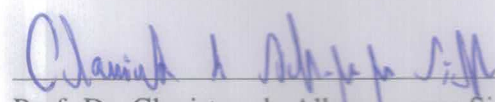
Prof. Dr. Tiago Pereira do Nascimento (Orientador)

Universidade Federal da Paraiba



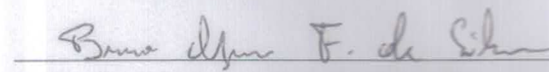
Prof. Dr. Alisson Vasconcelos de Brito (Co-orientador)

Universidade Federal da Paraiba



Prof. Dr. Clauirton de Alburquerque Siebra

Universidade Federal da Paraíba



Prof. Dr. Bruno Marques Ferreira da Silva

Universidade Federal do Rio Grande do Norte

Vice-Coordenador(a) do Programa

Prof. Dr. Ruy Alberto Pisani Altafim CI/UFPB

João Pessoa, 16 de Fevereiro de 2017.

Centro de Informática, Universidade Federal da Paraíba

Rua dos Escoteiros, Mangabeira VII, João Pessoa, Paraíba, Brasil CEP: 58058-600

Fone: +55 (83) 3216 7093 / Fax: +55 (83) 3216 7117

Dedicatória

Dedico este trabalho a minha mãe, dona Fabiana Targino da Silva Morais, e ao meu pai, o senhor Carlos Antônio Borges de Morais, que nunca mediram esforços para garantir que eu e minhas irmãs tivéssemos a oportunidade de dar continuidade aos nossos estudos.

Também dedico este trabalho ao grande amigo Dr. Alexander Fink, que há muitos anos se faz presente, tanto na minha vida pessoal quanto profissional, acompanhando meus estudos e esforços na área de empreendedorismo inovador e tecnológico, me ajudando sempre que preciso.

Agradecimentos

Agradeço a minha família, que apesar de dispor de poucos recursos financeiros, puderam e foram a base por trás da realização de mais este sonho (o qual compartilhado com todos), me incentivando e apoiando integralmente. Agradeço, em especial, ao meu Professor e Orientador, o PhD Eng. Tiago Pereira do Nascimento, pela atenção e paciência para comigo, e pela oportunidade a mim concedida de participar desse projeto. Também quero agradecer ao amigo Dr. Gabriel Basso pela ajuda ao longo do desenvolvimento desse trabalho e ao amigo André Castro por todo apoio, companheirismo e atenção. Obrigado à todos que fizeram parte, de forma direta ou indiretamente, dessa conquista. Que Deus abençoe à todos!

"A palavra convence, o exemplo arrasta."

(Lair Ribeiro)

Resumo

Sistemas anti-colisão são baseados na percepção e estimação da pose do robô móvel (coordenadas e orientação), em referência ao ambiente em que ele se encontra. A detecção de obstáculos, planejamento de caminhos e estimação da pose são fundamentais para assegurar a autonomia e a segurança do robô, no intuito de reduzir o risco de colisão com objetos ou pessoas que dividem o mesmo espaço. Para isso, o uso de sensores RGB-Ds, tal como o Microsoft Kinect, vem se tornando popular nos últimos anos, por serem sensores relativamente precisos e de baixo custo. Nesse trabalho nós propomos um novo algoritmo anti-colisão 3D baseado no método de Campo Potencial Artificial, que é capaz de fazer um robô móvel passar em espaços estreitos, entre obstáculos, minimizando as oscilações, que é uma característica comum desse tipo de método, durante seu deslocamento. Foi desenvolvido um sistema para plataforma robótica 'Turtlebot 2', o qual foi utilizado para realizar todos os experimentos.

Palavras-chave: Sistema anti-colisão 3D, campo potencial artificial, robô móvel, kinect.

Abstract

Anti-collision systems are based on sensing and estimating the mobile robot pose (coordinates and orientation), with respect to its environment. Obstacles detection, path planning and pose estimation are primordial to ensure the autonomy and safety of the robot, in order to reduce the risk of collision with objects and living beings that share the same space. For this, the use of RGB-D sensors, such as the Microsoft Kinect, has become popular in the last years, for being relative accurate and low cost sensors. In this work we propose a new 3D anti-collision algorithm based on Artificial Potential Field method, that is able to make a mobile robot pass between closely spaced obstacles, minimizing the oscillations during the cross. We develop our Unmanned Ground Vehicles (UGV) system on a 'Turtlebot 2' platform, with which we perform the experiments.

Keywords: 3D anti-collision system, artificial potential field, mobile robot, Kinect.

Sumário

1 Introdução **1**

1.1 Contexto e Motivação . . . 1

1.2 Objetivos . . . 4

1.2.1 Objetivo Geral . . . 4

1.2.2 Objetivos Específicos . . . 4

1.3 Metodologia . . . 5

1.4 Publicações Relacionadas . . . 5

1.5 Estrutura da Dissertação . . . 6

2 Fundamentação Teórica **7**

2.1 Sistemas anti-colisão . . . 7

2.2 Planejamento de rotas . . . 8

2.2.1 Campos Potenciais Artificias . . . 9

2.3 Localização de robôs móveis . . . 13

2.3.1 Odometria mecânica . . . 14

2.4 ROS - Robot Operating System . . . 16

2.5 Considerações Finais . . . 19

3 Trabalhos Relacionados **21**

3.1 Trabalhos Relacionados . . . 21

3.2 Considerações Finais . . . 30

4 Metodologia **31**

4.1 Campo Potencial Artifical 3D . . . 31

4.1.1 Campo e força atrativa . . . 32

4.1.2 Campo e Força Repulsiva 33
4.1.3 Força Atrativa e Repulsiva 35
4.2 Sistema anti-colisão implementado 36
4.3 O Sensor RGB-D Kinect 37
4.4 Framework de desenvolvimento e bibliotecas 39
4.4.1 PCL e Freenect usando ROS 39
4.4.2 Pré-processamento 41
4.4.3 RANSAC 42
4.4.4 K-d Tree 43
4.4.5 Euclidean Extraction Cluster 44
4.5 Considerações Finais 45

5 Avaliação Experimental **46**
5.1 Estudo de Caso 46
5.1.1 Ferramentas e Tecnologias 46
5.1.2 Requisitos 47
5.2 Experimento 47
5.2.1 Execução do Experimento 47
5.2.2 Análise do Experimento 53
5.3 Considerações Finais 58

6 Conclusão **59**

Referências **66**

A Código **67**

B Log **77**

Lista de Figuras

1.1 Turtlebot 2. 3

2.1 Gradiente do potencial de repulsão. 10

2.2 Gradiente do potencial de atração. 11

2.3 Gradiente do potencial resultante. 11

2.4 Exemplo de gradiente descendente. 12

2.5 Odômetro real (a) e seu esquema interno (b). 15

2.6 Funcionamento do odômetro. 15

2.7 Método de localização relativa. 16

2.8 Mapa da comunidade ROS. 18

2.9 Robôs com repositórios no ROS. 19

2.10 Rviz - ferramenta de visualização do ROS. 20

4.1 Apresenta o mapa local do 'Turtlebot 2' (a) e o sistema de coordenadas do Kinect (b) 32

4.2 Arquitetura do sistema anti-colisão implementado. 37

4.3 Sensor RGB-D Kinect. 38

4.4 Range do Kinect. 39

4.5 Fluxo dos algoritmos implementados da PCL. 40

4.6 Exemplo de segmentação de plano com RANSAC. 42

4.7 Estrutura KdTree. 44

5.1 Vídeo apresentando os 4 tipos de experimentos realizados. 49

5.2 Imagem RGB do primeiro experimento. 50

5.3 Nuvem de pontos da PCL. A profundidade é representada pelo um range de cores, quanto mais próximo o objeto do sensor, mais quente são as cores na imagem. As cores variam entre azul marinho, que define os obstáculos mais distantes, e vermelho, que representa os obstáculos mais próximos. A região em preto é uma área desconhecida, ou seja, não há dados sobre ela. 51
5.4 Filtro *passThrough* aplicado aos eixos 'x' e 'y' da nuvem de pontos. 51
5.5 Filtro VoxelGrid aplicado ao eixo 'z' do nuvem de pontos. 52
5.6 Resultado final após filtragem com Passthrough e Voxelgrid. 52
5.7 Foto do primeiro experimento: desvio de obstáculo. 54
5.8 Análise do caminho percorrido pelo robô no primeiro experimento. 54
5.9 Foto do segundo experimento: trafegando em passagem estreita. 55
5.10 Análise do caminho percorrido pelo robô no segundo experimento. 55
5.11 Foto do terceiro experimento: passando debaixo da mesa. 56
5.12 Análise do caminho percorrido pelo robô no terceiro experimento. 56
5.13 Foto do quarto experimento: contorno de mesa cuja a altura (baixa) impede a passagem do robô 57
5.14 Análise do caminho percorrido pelo robô no quarto experimento. 57

Lista de Tabelas

3.1 Listagem de algoritmos 29

5.1 Requisitos - especificações técnicas 48
5.2 Experimentos realizados. 49

Lista de Códigos Fonte

A.1 node de visão - manhattan.cpp 67

A.2 node de movimento - moving.py 72

Capítulo 1

Introdução

Neste capítulo é introduzido a motivação por trás desse trabalho sobre o desenvolvimento de um sistema anti-colisão utilizando um método de campos potenciais artificiais 3D. Este capítulo está organizado da seguinte maneira: a Seção 1.1 contextualiza e apresenta a motivação do trabalho descrito nesse documento, a Seção 1.2 mostra o objetivo geral e os objetivos específicos que guiaram este trabalho, a Seção 1.3 apresenta a metodologia utilizada, a Seção 1.4 aborda a publicação em conferência do trabalho desenvolvido, e por fim, a Seção 1.5 apresenta a estrutura desse documento como um todo.

1.1 Contexto e Motivação

Atualmente, robôs móveis têm sido desenvolvidos para diferentes aplicações [1] [2] [3] [4] [5], tais como: vigilância, defesa, resgate, *health care* móvel, controle de fronteiras, suporte a produção agrícola, entretenimento, entre outras. Entretanto, existe uma variedade de potenciais aplicações ainda inexploradas. Um dos principais problemas referente a robôs móveis autônomos é a capacidade de mover-se de forma segura, entre obstáculos, evitando a colisão com qualquer tipo de objeto ou pessoas, tal desafio envolve outros sub-problemas como detecção de objetos, estimação da pose (coordenadas e orientação) do robô ao longo do tempo e planejamento de caminhos.

Quanto a detecção de obstáculos em um ambiente onde o robô atua, existem duas diferentes abordagens que são comumente adotadas [6] [7] [8]: uma é acoplar o sensor ao robô, num esquema de detecção embarcada, e a outra alternativa de abordagem é usar um ou

mais sensores espalhados dentro do próprio ambiente ou um sistema de percepção híbrida citenascimento2013 [9] [10]. Para detecção embarcada, existe um range de sensores (como sensores RGB-Ds - exemplo: Kinect; câmeras, lasers, LiDARs, sonares, etc.) adequados para dar ao robô alguma informação sobre o ambiente. Um outro desafio fundamental para os robôs autônomos é conseguir estimar a sua pose no espaço, periodicamente. Isso é imprescindível para que o robô saiba sua posição atual, e assim, estimar a distância dele em relação ao alvo (local de destino) e qual caminho tomar. Isso pode ser obtido com um esquema de detecção embarcada através do método *dead reckoning* [6] que utiliza os dados dos odômetros (*encoders*) ou do sensor Kinect (através do método odometria visual) [11].

Há uma grande variedade de métodos que podem ser adotados para o desenvolvimento de sistemas anti-colisão, em conjunto com planejadores de caminho, que são explorados na literatura. Como exemplos, podemos citar o escape tangencial [9], detecção de cantos [12], grade de ocupação [13], campo potencial artificial [14; 15; 16; 10], e Força Virtual e Campo Virtual [13].

Em especial, o método de campo potencial artificial (CPA) se destaca pelo fato de ser relativamente simples de ser implementado, eficiente, rápido e preciso para a maioria das aplicações relacionadas à sistemas anti-colisão. Contudo, o método tradicional de campos potenciais normalmente sofre com problemas associados a mínimos locais, os quais geram restrições, como:

1. Alvo inacessível;

2. Impedimento de passagem entre obstáculos próximos;

3. Oscilações na presença de obstáculos;

4. Oscilações em passagens estreitas.

Para melhor entendimento, consideraremos o alvo como destino final do trajeto do robô, e quando tratamos de oscilações em passagens estreitas, podemos tomar como exemplo o caso em que o robô precise passar por um corredor estreito.

Esse trabalho apresenta um sistema anti-colisão 3D baseado num método de campos potenciais artificiais modificado, o qual soluciona três problemas típicos encontrados no método tradicional de CPA: Impedimento de passagem entre obstáculos próximos, oscilações

na presença de obstáculos, e oscilações em passagens estreitas. A detecção de obstáculos é feita via informação visual, imagens RGB-Ds (RGB - informação de cores, mais 'D' do inglês *depth*, referente à informação de profundidade obtida pelo sensor de infra-vermelho embarcado no Kinect) capturadas e processadas em forma de nuvem de pontos utilizando a biblioteca *Point Cloud Library (PCL)*. Entre os diversos algoritmos da PCL, para esse trabalho usamos Os filtros *VoxelGrid* e *Passthrough*, o que nos permite diminuir o custo computacional e limitar o campo de visão do Kinect apenas às dimensões de nosso interesse, respectivamente.

Foi implementado o método proposto nesse trabalho e realizado todos os experimentos utilizando um robô móvel terrestre chamado 'Turtlebot 2', o qual é uma plataforma de baixo custo – mostrado na Figura 1.1. Para que o robô possa perceber o ambiente, em que ele está inserido, e detectar obstáculos, o 'Turtlebot 2' tem um sensor Kinect acoplado à sua estrutura, permitindo assim que ele calcule sua distância até os obstáculos encontrados na cena. Para calcular sua pose utilizamos apenas os dados advindos dos odômetros (acoplados as suas rodas) e bússola interna, o que nos possibilita a estimação de seu deslocamento ao longo do tempo através do método *dead reckoning* [6].

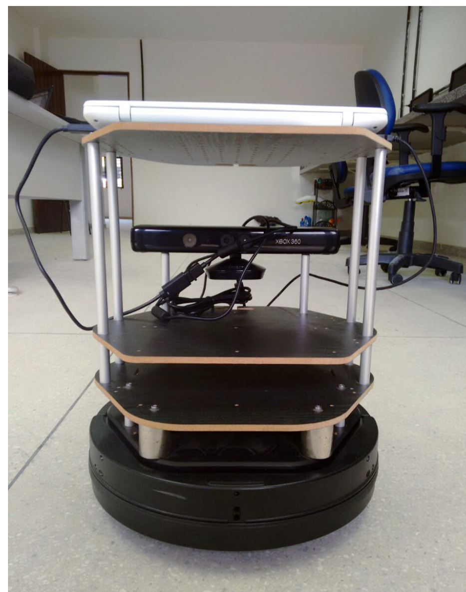


Figura 1.1: Turtlebot 2.

A ideia de criar e aperfeiçoar máquinas capazes de substituir os seres humanos em trabalhos complexos, já não é uma exclusividade dos filmes de ficção. Sistemas robóticos

autônomos são desenvolvidos para realizar tarefas de alta periculosidade, que exijam executar movimentos repetitivos, sem sofrer com fadiga, ou um alto grau de precisão. Contudo, tais sistemas robóticos devem ser capazes de executar tarefas sem oferecer perigo ao ambiente em que estão inseridos, nem a objetos e/ou pessoas. Por isso, é muito importante o desenvolvimento de sistemas anti-colisão mais eficazes, seja qual for a área de interesse em que esses robôs são empregados.

Entre os métodos de planejamento de rotas existentes, o método de campos potenciais artificiais é utilizado por ser uma estrutura simples, de baixo custo computacional (o que é uma restrição de sistemas de tempo real) e de fácil implementação, além de obter resultados satisfatórios. Porém, esse método sofre com algumas limitações, como: 1) impedimento de passagem entre obstáculos próximos; 2) oscilações na presença de obstáculos e 3) oscilações em passagens estreitas. Por este motivo, nesse trabalho, propomos um método de campos potenciais artificiais modificado que ataca os três problemas mencionados.

Para tal, utilizamos uma nuvem de pontos ao invés de um mapa de profundidade, por apresentar por inteiro as informações do campo de visão do sensor Kinect. Para o nosso algoritmo é fundamental obter apenas as informações pertinentes de uma região do espaço delineada, criando uma espécie de "foco". O intuito é diminuir a quantidade de pontos da nuvem, sem interferir no funcionamento do algoritmo de campos potenciais artificiais.

1.2 Objetivos

1.2.1 Objetivo Geral

Diante do contexto apresentado, o objetivo desse trabalho é apresentar o desenvolvimento de um sistema anti-colisão baseado no método de campos potenciais artificiais modificado para auxiliar a navegação robótica em ambientes internos, que seja capaz de evitar a colisão do robô com quaisquer obstáculos (estáticos e/ou dinâmicos) apartir da detecção dos mesmos em uma nuvem de pontos obtida em tempo real através do sensor RGB-D Kinect.

1.2.2 Objetivos Específicos

Para alcançar o objetivo geral descrito, alguns objetivos específicos precisaram ser atingidos:

1. Propor um campo potencial artificial modificado capaz de solucionar os problemas usuais de CPA, como oscilações em passagens estreitas, oscilações na presença de obstáculos, e impedimento de passagem entre obstáculos próximos;

2. Implementar a metodologia de CPA proposta, utilizando odometria mecânica (método 'dead reckoning') e detecção visual de obstáculos;

3. Validar a metodologia de CPA proposta de forma extensiva, porém não exaustiva, em cenários tipicamente problemáticos para os CPA usuais.

4. Fomentar ferramentas de código aberto.

1.3 Metodologia

A metodologia do presente projeto de mestrado propõe desenvolver uma pesquisa composta de uma revisão bibliográfica sobre o tema de pesquisa, a qual visa explorar a proposição e evolução de novos métodos e análises que contribuíram para o desenvolvimento científico e acadêmico num importante campo do conhecimento que, embora seja desenvolvido, carece de melhores resultados, no que se trata de manipulação de dados visuais 3D. Os fundamentos teóricos são de grande importância pois deve-se possuir um forte embasamento para a pesquisa se desenvolver de forma satisfatória. Os resultados dessa pesquisa foram obtidos através de experimentos realizados em um ambiente real e *indoor* visando a identificação, através da prática, de possíveis problemas que dificultam o sucesso da missão do robô - de chegar ao alvo (destino) sem colidir com quaisquer obstáculos da cena. A obtenção do modelo matemático foi baseada na revisão bibliográfica e na adequação do tema considerado, e a análise dos dados foi feita com base nos *log*s do robô 'Turtlebot 2'.

1.4 Publicações Relacionadas

Esse trabalho foi aceito como "A 3D anti-collision system based on Artificial Potential Field Method for a mobile robot"na ICAART 2017 - 9a Conferência Internacional de Agentes e Inteligência Artificial, que acontecerá entre os dias 24 e 26 de fevereiro de 2017 na cidade de Porto, em Portugal.

1.5 Estrutura da Dissertação

A organização dos capítulos segue uma ordem lógica, o que permite uma melhor compreensão de cada assunto apresentado aqui. Para tanto, o trabalho está organizado da seguinte forma:

No Capítulo 2 é apresentado alguns dos algoritmos de planejamento de rotas e seu funcionamento.

No Capítulo 3 é abordado os trabalhos relacionados ao tema.

No Capítulo 4 mostramos a metodologia proposta para construção do sistema anti-colisão com base no método de campos potenciais artificiais.

No Capítulo 5 é apresentado os experimentos realizados e é feita uma análise dos resultados obtidos.

Por fim, no Capítulo 6 é apresentado a conclusão deste trabalho juntamente com as perspectivas para trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Este capítulo apresenta os principais conceitos relacionados a aos métodos utilizados para construção de um sistema anti-colisão. Métodos de localização de robôs não são o foco deste trabalho, porém, para realizar os testes, admite-se que o método de odometria mecânica não sofre com ruídos, as rodas do robô conseguem executar movimentos circulares de raio r perfeitos, que não sofrem com derrapagens, resultando num movimento sem quaisquer interferências. Neste capítulo, a Seção 2.1 apresenta uma síntese sobre a composição de um sistema anti-colisão e quais as possibilidades existentes, a Seção 2.2 aprofunda-se na discussão sobre métodos de planejamento de rotas e aborda o funcionamento do método de campos potenciais artificias, enquanto a Seção 2.3 apresenta o método de localização do robô necessário para monitorar seu deslocamento, por fim, na Seção 2.4 faz-se uma introdução ao framework ROS, base pra o desenvolvimento do sistema anti-colisão proposto nesse trabalho.

2.1 Sistemas anti-colisão

Um sistema anti-colisão baseia-se em duas principais frentes: planejamento de rotas e localização do robô [7] [9] [17]. O foco desse trabalho está na discussão sobre planejamento de rotas, embora a localização do robô através do método de localização relativa (do inglês - *dead reckoning*), chamado odometria mecânica, seja necessária para realizar os experimentos reais, além de também ser abordado nesse capítulo.

Como citado anteriormente, há diversos métodos na literatura, que tratam do tema pro-

posto nesse trabalho, como por exemplo: escape tangencial [9], detecção de cantos [12], grade de ocupação [13; 18], $A*$ [19; 20; 21; 22], campo potencial artificial [14; 15; 16; 10], Força e Campo Virtuais [13], entre outros.

A decisão de qual método adotar para o desenvolvimento do sistema anti-colisão depende da topologia do espaço de configuração, ou seja, da dimensão em que o robô atua (2D ou 3D), da dinamicidade do ambiente, de quais tipos de obstáculos podem ser encontrados na cena, e até mesmo de como está montado o sistema de sensoriamento (embarcado no robô e/ou externo).

2.2 Planejamento de rotas

Em sistemas anti-colisão, o planejamento de rotas é essencial na tomada de decisão do robô quanto a direção à ser seguida, de acordo com o alvo (destino) que ele está buscando.

Alguns trabalhos divergem sobre a interpretação das nomenclaturas, como: planejamento de rota / planejamento de caminho, e planejamento de trajetória. O planejamento de rota ou caminho preocupa-se exclusivamente sobre qual caminho tomar, desviando dos obstáculos mas procurando o alvo, enquanto o planejamento de trajetória também considera o tempo gasto durante o deslocamento do robô para realizar o cálculo.

Para fins de melhor compreensão, iremos utilizar a nomenclatura "planejamento de rota", pois aqui não levamos em conta o tempo para realização de tarefas.

Os algoritmos de planejamento de rotas podem ser divididos em duas principais categorias []: 1) de questionamento único e 2) de múltiplos questionamento. A diferença entre eles é que o primeiro tipo não faz nenhum pré-processamento do espaço de configurações e a cada requisição o planejador faz a representação necessária do caminho para decidir por onde seguir. Ao contrário disso, os algoritmos de múltiplo-questionamento trabalham com uma representação global do espaço de configurações.

As principais vantagens e desvantagens atreladas as duas categorias mencionadas, são:

- o tipo questionamento único, embora leve mais tempo para processar e tomar uma decisão sempre que há um questionamento, permite uma melhor adaptação à ambientes dinâmicos.

- múltiplo questionamento - precisa de mais tempo para criar a representação global inicial, o que, por outro lado, diminui o tempo de processamento de questionamento posteriores que são rapidamente solucionado, tipicamente adotado para o desenvolvimento de sistemas que atuem em ambientes de trabalho estáticos.

Uma outra classificação dos métodos de planejamento de rotas ainda pode ser feita no que se diz respeito à utilidade, em determinístico ou probabilístico. Dado a configuração inicial e final, os planejadores determinísticos retornam a mesma rota, enquanto os planejadores probabilísticos podem retornar rotas diferentes [23].

Cada tipo apresenta característica que são positivas e outras negativas, exemplo:

- planejadores determinísticos exigem um esforço computacional maior, porém tem resultados e comportamento previsíveis, além de serem de fácil depuração.
- planejadores probabilísticos são mais eficientes por tratarem parte do espaço de configuração como amostras, o que diminui o tempo de execução, mas não se pode prever que rota o robô tomará.

O algoritmo A* é um exemplo de planejador de rotas determinístico. Enquanto, os algoritmos de Grade de Ocupação [13; 18], Lógica Fuzzy [7; 24] são do tipo probabilístico.

Alguns trabalhos envolvem a construção de algoritmos híbridos baseado nos tradicionais planejadores de rotas já existentes, como A* e Campos Potenciais. Nesse caso, a ideia principal por trás disso é encontrar uma solução capaz de garantir que o robô atinja o mínimo global sem aumentar o custo computacional.

2.2.1 Campos Potenciais Artificias

O método de campos potenciais artificias (CPA) foi inicialmente proposto por Khatib [2], desenvolvido originalmente para contorno de obstáculos, aplicável quando não se dispõe de um modelo *a priori* dos mesmos, mas são percebidos *on-line* [23].

Algoritmos que utilizam esse tipo de método (campos potenciais artificiais) são de questionamento único, determinístico, ideais para aplicações de tempo real em ambientes dinâmicos.

Algoritmos de CPA representam o robô como sendo uma partícula que sofre influência de forças que as guiam automaticamente até o alvo, ou seja o destino final. Os obstáculos encontrados no ambiente inferem uma força repulsiva (ver Figura 2.1), afastando o robô, enquanto o alvo induz uma força atrativa (ver Figura 2.2), puxando o robô em sua direção até sua posição. A força resultante é quem desenha o caminho por onde o robô irá passar, ou seja, determina a direção e também a velocidade do robô (ver Figura 2.3).

Dado o espaço de configuração tridimensional do robô composta pelas coordenadas 'x' e 'y' (da posição do robô) mais a informação de theta (sua orientação), obtemos a chamada *pose*. Sabendo qual a pose inicial q e final q_fim da rota desejada para o robô, temos definido a função potencial como sendo:

$$U(q) = U_{att}(q) + U_{rep}(q) \tag{2.1}$$

onde $U_{att}(q)$ representa o potencial atrativo e $U_{rep}(q)$ o potencial negativo. Assim, o princípio do planejador de rotas baseado em funções potencias consiste em calcular o movimento do robô a partir das forças de atração e repulsão que nele atuam [23].

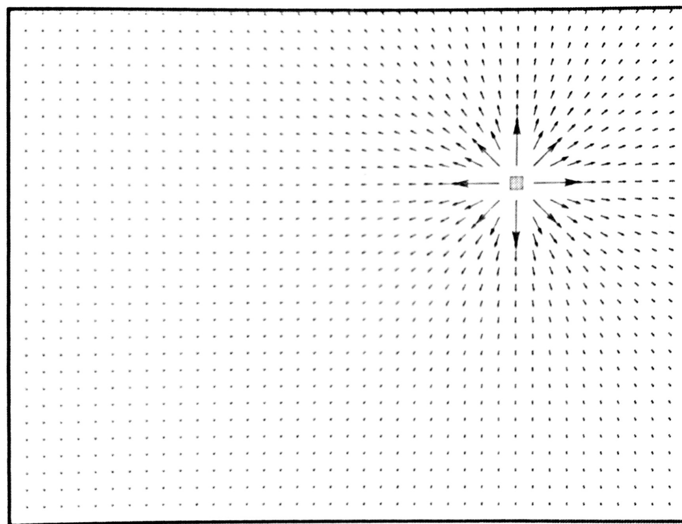


Figura 2.1: Gradiente do potencial de repulsão.

As forças de atração e repulsão são responsáveis pela construção de um gradiente descendente, essas forças atuam como forças gravitacionais que fazem com que o robô se desloque numa velocidade variante, como uma bola arremessada dentro de uma "tigela"(como mostra

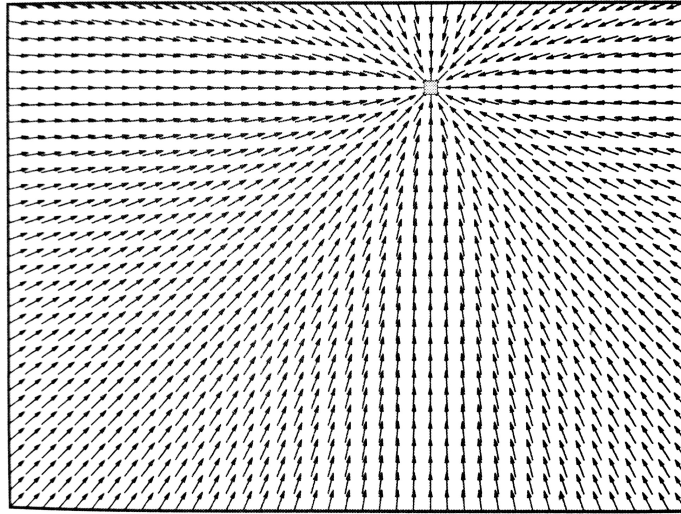


Figura 2.2: Gradiente do potencial de atração.

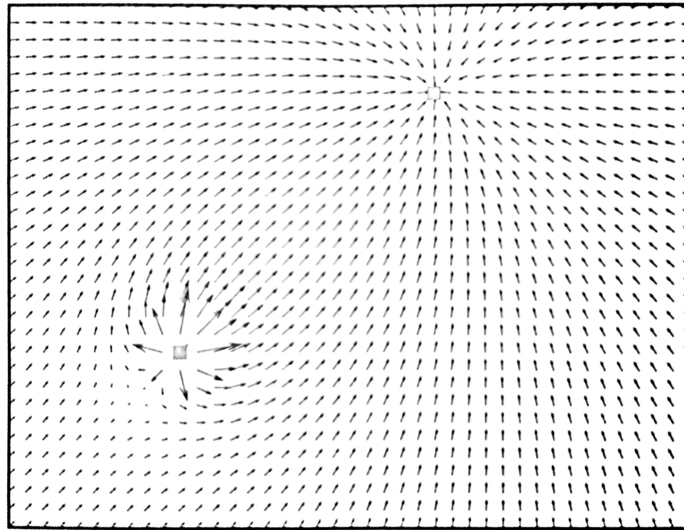


Figura 2.3: Gradiente do potencial resultante.

a Figura 2.4). A força resultante, para construção de tal gradiente, é calculada da seguinte maneira:

$$f = f_{att} + f_{rep} \tag{2.2}$$

A Figura 2.4, extraída do vídeo "Artificial Potential Field Approach in Robot Motion Planning"[25] ilustra o campo potencial resultante da soma dos campos atrativo e repulsivo. Esse campo resultante gera uma força que guia o robô pelo espaço, fazendo-o movimentar-se

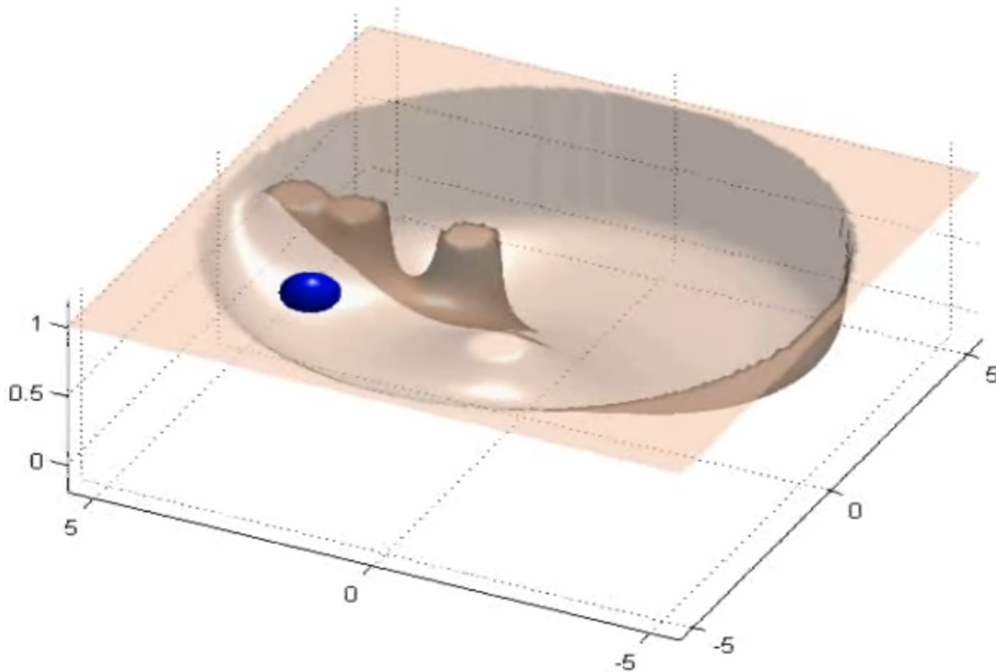


Figura 2.4: Exemplo de gradiente descendente.

entre os obstáculos em direção ao alvo enquanto a força resultante for maior que zero ou um limiar específico. Geralmente é definido um limiar no cálculo da força resultante para evitar problemas quanto a discretização. Esse método é capaz de evitar a colisão do robô com os obstáculos, além de possibilitar um deslocamento suave do robô [23].

A grande maioria dos trabalhos adotam a distância euclidiana para calcular o quão distante está o robô dos obstáculos. No entanto, para este trabalho foi utilizado o cálculo da distância rectilínea, popularmente conhecida por distância de manhantan.

Como será demonstrado por essa dissertação, essa mudança, aparentemente simples, faz com que o robô consiga passar por espaços estreitos, em corredores formados por obstáculos próximos, e até mesmo por baixo de mesas que tenham sua altura maior do que a do robô.

No cálculo da força repulsiva, quanto mais próximo o obstáculo estiver do robô, maior será a sua participação na força resultante. Por isso, o algoritmo possibilita o tráfego do robô sem problemas com oscilações entre obstáculos num ambiente desorganizado, ou seja, repleto de obstáculos. A adoção do método da distância de manhattan é uma alternativa ao

tradicional método euclideano utilizado para calcular a distância entre o robô e os obstáculos.

Neste trabalho apresentamos uma abordagem que, diferentemente dos trabalhos relacionados ao tema aqui abordado, baseia-se no uso de informações 3D, sem que seja necessário qualquer conversão desses dados para dimensão 2D, o que geralmente é feito com intuito de diminuir o custo computacional, e assim, obter uma resposta em tempo real que permita o robô móvel deslocar-se sem colidir com os obstáculos encontrados na cena.

Ao invés disso, aplicamos o método de campo potencial artificial sobre uma nuvem de pontos 3D obtida pela PCL (Point Cloud Library). Porém, o uso desses dados tridimensionais requer um elevado custo computacional, o que impossibilita o sistema responder ... processamento.

A solução é aplicar filtros que possam diminuir o número de pontos da nuvem e aplicar métodos que melhor definam quais pontos tem maiores chances de representar algum obstáculo diante do robô. Isso faz com que...

e a distância de manhattan...

Contudo, a força repulsiva desses pontos será alta.

Contudo, existem algumas limitações atreladas ao tradicional método de campos potenciais que acabam não garantindo o êxito da missão - do robô chegar ao destino final sem colidir com quaisquer obstáculos ao longo do percurso. Esses problemas ocorrem quando, por exemplo, o robô se encontra na cavidade de algum obstáculo não convexo ou outras situações que façam com que a força resultante seja igual a zero (nula). Enfim, o método de campos potenciais sofre com a possibilidade de ficar preso a mínimos locais.

Para contornar os problemas comuns desse tipo de método, alguma técnicas são normalmente utilizadas, como: fazer passeios aleatórios (do robô) até encontrar uma saída, uso de campos harmônicos [26] [27], ou ainda encontrar funções potenciais que resultam em um único mínimo global [10], ou seja, na configuração final.

2.3 Localização de robôs móveis

O problema da localização de um robô móvel consiste basicamente em estimar sua pose atual (coordenadas de posição e orientação) e comparar isto à um referencial fixo, geralmente o ponto de partida (inicialização do sistema). Os métodos de localização existentes podem ser

classificados em três categorias: Localização Relativa (do inglês - *dead reckoning*), Localização Absoluta e Fusão de Multi-sensores que seria nada mais que a junção de métodos das duas primeiras categorias [6].

A Localização Relativa baseia-se na informação sobre localizações em instantes anteriores, onde as medidas se deduzem por integração, e pode ser feita de duas maneiras: através do método de Odometria (Mecânica e/ou Visual) ou Navegação Inercial. O método de odometria determina a localização pela integração incremental do movimento das rodas através de sensores embarcados nelas chamados odômetros (do inglês - *encoders*). Com isso, é possível estimar o deslocamento linear e a orientação do robô, enquanto a navegação inercial baseia-se na informação oriunda apenas de giroscópios e acelerômetros - que são capazes de medir a orientação e a aceleração [6].

Diferentemente dos métodos de Localização Relativa, a Localização Absoluta não depende de localizações previamente calculadas derivadas de integrações, mas sim, deriva de uma medição direta, e pode ser obtida de três formas: Faróis Ativos, Marcas no ambiente (naturais ou artificiais) e Mapas [28]. Como exemplo de Faróis Ativos podemos citar dispositivos que utilizam um processo de triangulação para inferir uma localização, o Global System Position (GPS) é um caso. Já o método de "Marcas do ambiente"trabalha com marcas que podem ser naturais (portas, janelas, quinas, luz no teto, etc.) ou artificiais (QR-Code, figuras geométricas coloridas, etc.). Por fim, um método popular para inferir sobre a localização do robô é o "Mapas", onde medições obtidas de dados dos sensores são comparados com os dados de um mapa prévio [6].

Para esse trabalho foi adotado o método de odometria mecânica por ser de fácil e rápida implementação utilizando o 'Turtlebot 2' para realizar os experimentos do sistema anticolisão baseado em campos potenciais artificiais.

2.3.1 Odometria mecânica

A Odometria Mecânica funciona com base na integração incremental dos dados advindos dos sensores de odometria (*encoders*) que são acoplados as rodas do robô, com isso, é possível calcular o seu deslocamento ao longo do tempo.

O princípio do funcionamento dos odômetros é a transmissão de luz, de um LED para um foto sensor, através de um disco perfurado que, por meio da frequência de pulsos gerados, é

possível calcular a velocidade de rotação da roda, e assim, medir o deslocamento do robô no espaço [6]. A Figura 2.5 apresenta um odômetro real (a) e seu esquema interno (b) extraída de [29] e [30] respectivamente. A Figura 2.6, extraída de [6], ilustra de forma simples como é composto o mecanismo desse sensor.

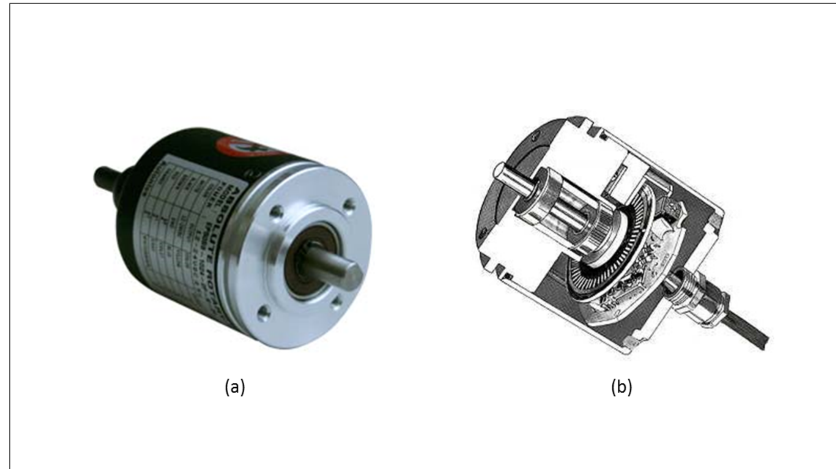


Figura 2.5: Odômetro real (a) e seu esquema interno (b).

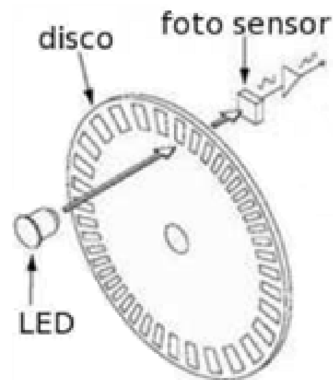


Figura 2.6: Funcionamento do odômetro.

Embora esse tipo de sensor sofra com ruídos, ele é amplamente usado em pesquisas relacionadas à trajetória de robôs móveis, e possibilita a obtenção de resultados satisfatórios para o nosso trabalho.

Para ilustrar melhor como esse método funciona na prática, sabendo que para se estimar a localização de um robô com base na odometria é necessário integrar os deslocamentos incrementais das rodas, tendo um referencial fixo como ponto de partida, consideremos um

robô que se desloca de forma linear em seu ambiente conforme certa trajetória (ver Figura 2.7, extraída de [6]), então teríamos:

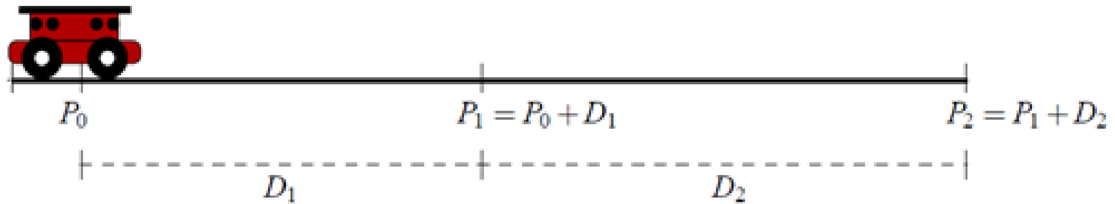


Figura 2.7: Método de localização relativa.

1. P_0 – ponto inicial em que o robô está localizado;
2. P_1 – um ponto qualquer seguinte ao P_0;
3. P_2 – a posição final do robô em sua trajetória;
4. D_1 – a distância percorrida do ponto inicial (P_0) ao ponto P_1;
5. D_2 – a distância percorrida do ponto P_1 ao ponto final, P_2.

Como mostra a Figura 2.7, o robô sai do ponto inicial (de partida) P_0, depois ele chega ao ponto P_1 em sua trajetória. O deslocamento realizado P_1 pode ser deduzido através da soma de P_0 e D_1. Continuando sua trajetória, o robô chega ao ponto P_2, o qual pode ser calculado somando-se P_1 ao novo deslocamento D_2. Com isso, a localização final é igual aos deslocamentos realizados em relação à localização inicial [6].

2.4 ROS - Robot Operating System

O ROS - 'Robot Operating System' é um framework de desenvolvimento ágil para sistemas robóticos que visa abstrair o hardware ao máximo, trabalhar com uma coleção de estruturas de softwares e aumentar a eficiência do processo de desenvolvimento [31]. Embora seja chamado de "sistema operacional", o ROS não trabalha com escalonamento, gerenciamento

de memória, janela ou processos, entre outras características comuns atribuídas aos Sistemas Operacionais que conhecemos, como Windows, Ubuntu ou Mac OS.

Assim como ROS, existe outros projetos de framework/middleware com o mesmo propósito, como o YARD, Orocos e o RD. Hammer [32] faz uma comparação interessante dessas ferramentas, no entanto o ROS é o mais popular entre elas.

O ROS é um projeto *open source* criado pela empresa e incubadora Willow Garage em 2007, oriundo de diversos trabalhos desempenhados por cientistas da Universidade de Stanford em meados do ano 2000 e vem se popularizando rapidamente. A comunidade ROS conta com cerca de 1.500 participantes em sua mailing list, 5.700 usuários no fórum que serve para esclarecer dúvidas à respeito do desenvolvimento com ROS (com taxa de resposta acima dos 70 porcento), mais de 3.300 usuários participam da construção de conteúdo na sua Wiki, além de contar com mais de 22.000 páginas e cerca de 30 edições por dia [33].

No GoogleMaps encontramos o mapeamento de 156 instituições espalhadas pelo mundo que contribuíram ou contribuem com o ROS (Figura 2.8) fazendo o *upload* dos seus repositórios de projetos de pesquisa pela plataforma online do ROS e compartilhando seus resultados para que outras pessoas possam utilizar e acrescentar algo, sempre construindo novos sistemas mais complexos sobre o que já foi desenvolvido, abstraindo o funcionamento de módulos feitos no passado. No Brasil, temos apenas dois registros de colaboradores, um no Rio de Janeiro e outro em São Paulo, como mostra o mapa abaixo. [34].

Além disso, é possível encontrar boa parte da documentação de projetos desenvolvidos com ROS para robôs específicos, os quais são listados no site [35]. Entre tantas plataformas robóticas, faz-se necessário destacar alguns exemplos: Robonaut - da NASA, PR2 - criado pela Willow Garage e doado, 11 unidades, para os principais centros de pesquisa que desenvolveram um projeto piloto com diferentes aplicações domésticas utilizando esta plataforma robótica, Turtlebot, NAO, Lego NXT, entre outros [1].

O ROS é distribuído sob a licença BSD [36], o que permite reutilizar projetos abertos em iniciativas privadas, tendo a liberdade de transformar código aberto em proprietário, por exemplo, para construção de novos produtos.

Entre os diversos recursos disponíveis pelo ROS, podemos destacar alguns conjuntos de estruturas de software utilizados para aumentar a eficiência do desenvolvimento de sistemas robóticos: catkin - um sistema de montagem de software, um compilador; tf - que permite

Figura 2.8: Mapa da comunidade ROS.

transformar coordenadas de frames diferentes; rosparam - que monitora os parâmetros do servidor; rviz - que permite visualizar nuvem de pontos, imagens RGB-D, robôs simulados, etc (ver Figura 2.10); rqt - uma interface gráfica (GUI) que permite alterar os parâmetros do software em tempo real; entre diversos outros que podem ser encontrados no site do projeto [37]. Tudo isso para ajudar no desenvolvimento de aplicações nas áreas de navegação, simulação, planejamento de rotas, monitoramento, percepção, controle, etc.

A estrutura funcional do ROS está baseada na construção de Nodes, que nada mais são que programas executáveis individuais, que podem publicar (divulgar) ou subescrever (assinar) para receber informações de um determinado tópico. O tópico é um *stream* de mensagem de um tipo definido (inteiro, float, string, etc.), e dessa forma os nodes podem comunicar-se um com os outros através de um modelo *broadcasting*, ou seja, *1 - to - N*, um node pode publicar um tópico que pode ter vários outros nodes subescritos.

Uma outra forma de permitir a comunicação entre nodes, é usando 'serviços'. Essa estrutura garante transações inter-node síncronas (RPC), como estrutura Cliente / Servidor, no modelo de comunicação '*1 - to - 1*', requisição - resposta. Porém, neste trabalho, adotamos apenas tópicos como meio de comunicação entre os nossos nodes. O tópico publicado pelo node "manhattan"chama-se "distance", já o tópico publicado pelo node "moving"é do tipo Twist, chamado "move cmd"[38].

Figura 2.9: Robôs com repositórios no ROS.

O ROS permite a programação de robôs usando as linguagens C++ ou Python. O node deve ser escrito com apenas uma dessas linguagens, porém nada impede que nodes de linguagens diferentes possam se comunicar, pois a saída é publicada como tópico.

Dois nodes (ROS) principais foram desenvolvidos para a nossa aplicação. Um node chamado "manhattan"é responsável pela visão computacional, captura da imagem RGB-D (usando Kinect) para transformá-las em nuvem de pontos com ajuda da biblioteca PCL, fazer filtragem da nuvem, clusterização, e por fim, cálculo da força repulsiva atrelada a cada ponto dos obstáculos detectados pelo sistema. Enquanto o node "moving"é responsável por definir os parâmetros de velocidade do robô, rastrear a posição (pré-definida) do alvo com base nos dados advindos dos odômetros, além de ser subescrito ao node "manhattan"para calcular a força resultante considerando a informação sobre obstáculos e calcular a força resultante.

2.5 Considerações Finais

Este capítulo tratou dos principais conceitos relacionados a análise dos métodos escolhidos para realizar as tarefas de planejamento de rota e localização do robô, usando o framework

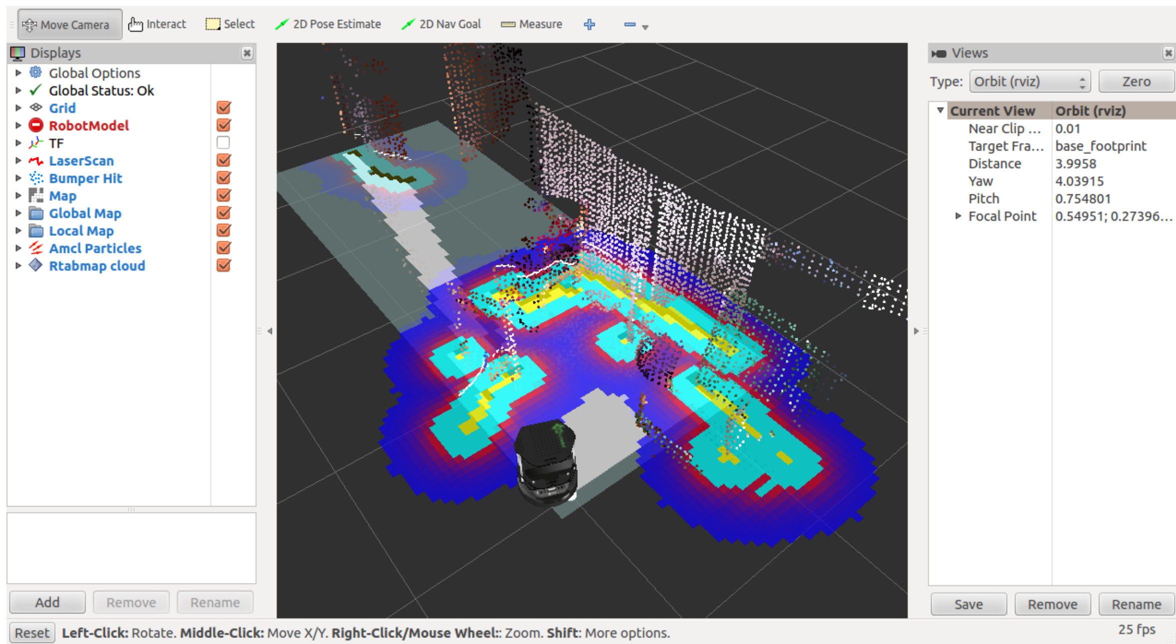


Figura 2.10: Rviz - ferramenta de visualização do ROS.

ROS, que são necessárias para o desenvolvimento de um sistema anti-colisão. Foi considerado um estudo dos conceitos e funcionamento para a escolha dos algoritmos para nossa aplicação. No próximo capítulo será explanado os trabalhos relacionados ao tema dessa pesquisa.

Capítulo 3

Trabalhos Relacionados

Neste capítulo apresentamos uma revisão dos principais trabalhos do estado-da-arte relacionados ao tema "sistemas anti-colisão"aplicados a robôs móveis terrestres ou aéreos utilizados em ambientes *indoor*. Em seguida, com base nessa revisão, estado-da-arte, para embasar nossa escolha pelo método de campos potenciais artificias.

3.1 Trabalhos Relacionados

Para evitar colisões, Tarazona [7] apresenta um sistema anti-colisão para um pequeno quadricoptero, um veículo aéreo não-tripulado - VANT (ou do inglês, *unmanned aerial vehicle - UAV*), capaz de navegar *indoor* usando um controlador baseado em lógica fuzzy. O sistema é composto por sete sensores, entre os quais estão sensores de ultrassom (sonares) do tipo HC-SR04 para detectar obstáculos. A localização relativa do robô é obtida através do método de navegação inercial com base nos dados oriundos do conjunto de sensores composto por: acelerômetro , giroscópio e sonares. Com os dados obtidos pelo acelerômetro é possível medir a aceleração nos três eixos (x, y, z) que indicam a inclinação do robô. Essa informação é utilizada para controlar a velocidade desejada para os quatro atuadores, de acordo com a distância em que se encontram os obstáculos, sendo que dois atuadores giram em sentido horário e os outros dois no sentido anti-contrário, necessários para manter a estabilidade do robô aéreo que é de pequeno porte. O sistema também utiliza um filtro para tratar dados ruidosos, chamado filtro de Kalman. Assim, é possível obter resultados mais precisos, inclusive, o filtro de Kalman é utilizado neste trabalho para identificar qual sensor, dentre o

conjunto de sensores do VANT, tem maior acurácia. Isso é percebido quando o sistema sofre algum tipo de interferência externa.

Santos [9] propõe um estudo de caso de uma estratégia anti-colisão baseada em uma abordagem de escape tangencial, onde um VANT, Ar Drone 2.0 da Parrot, pode tomar um dos vários caminhos possíveis para escapar do perigo eminente de uma colisão. Esse "escape"pode ser lateral e/ou vertical ou tangentes. Também é apresentado um ambiente virtual que simula o deslocamento do robô enquanto um VANT real executa o algoritmo simultaneamente. Para rastrear o robô, usa-se um sensor RGB-D Xtion Pro Live da Asus instalado no ambiente *indoor*, um IMU (*Inertial Measurement Unit*), um laser Sick L200 e um sonar HC-SR04 *onboad* embarcados no quadricoptero. Os dados de cada sensor passa por um Filtro de Informação Local (FIL), o que é essencialmente um Filtro de Kalman (FK) que pode estimar a posição dos obstáculos dinâmicos e do próprio robô. Depois, esses dados passam através de um novo filtro, um Filtro Decentralizado de Informação (FDI), que é responsável por fundir todos os dados do sistema de sensores (todos os sensores) para obter um resultado com maior acurácia. A localização do robô é feita por um sistema supervisor, que utiliza um controlador de posição não-linear com o objetivo de guiar o robô com base em sua localização atual, tendo como referência os dados obtidos pelo sensor RGB-D Pro Live instalado no teto do ambiente, o que garante que o robô alcance o mínimo global.

O trabalho de Elfes [13], engenheiro brasileiro que trabalha no laboratório de propulsão a jato da NASA, apresenta um método chamado "grade de ocupação"capaz de construir um mapa 2D do ambiente com base nas informações obtidas através de um conjunto de sensores de ultrassom (sonares). Uma grande vantagem de usar esse tipo de sensor é que ultrassons requerem um baixo custo computacional para construir mapas densos capazes de prover informações seguras que auxiliam o sistema de navegação do robô. O método de grade de ocupação aplica uma grade virtual sobre uma determinada região, visível ao sensor, dividindo a mesma em células num formato de cone (na frente do sensor) e depois calcula a probabilidade de cada célula estar ocupada ou não por qualquer obstáculo. O sistema retorna um valor que representa o *status* da célula, que podem ser: ocupada ou vazia. O resultado é um mapa bi-dimensional do ambiente, um bitmap, que permite que o robô se mova de forma segura sem colisões. Esse é um método geralmente utilizado para fins de mapeamento e navegação de robôs que atuem em ambientes desconhecidos e desestruturados.

Nesse mesmo sentido, Hornung [1] propõe uma combinação de representações 2D e 3D utilizando uma estrutura de dados *octree* da PCL para desenhar uma grade de ocupação e assim criar um novo método para robôs humanoides, o qual é chamado de '2.5D'. O foco desse trabalho é apresentar uma nova solução que ataque o problema comum aos sistemas de navegação tradicionais baseados em um mapa de grade de ocupação 2D projetado a partir de uma representação 3D do ambiente, por causa da eficiência. Essa solução impede a navegação do robô próximo a objetos em situações onde as configurações 3D projetadas estão setadas como "área de colisão"dentro de uma grade de ocupação (mapa de ocupação) 2D, mesmo não havendo risco real de colisão do robô com qualquer obstáculo no ambiente 3D. O método híbrido (2D e 3D) permite que o robô PR2 se desloque num espaço desorganizado sem colidir sua base ou manipuladores com quaisquer obstáculos, além de garantir a manipulação de objetos sobre uma mesa tendo a real ideia de espaço em que pode atuar.

Souza [18] defende uma abordagem baseada num sistema stereo de câmeras (duas câmeras) que geram uma grade de ocupação considerando a elevação da superfície (*Occupancy-Elevation Grid* - OEG), a partir da percepção do mundo real 3D, para construção de um mapa 2D. A estrutura resultante da aplicação do método é chama-se "2 e meio D", que consiste numa grade de ocupação 2D mais o dado sobre a elevação do terreno e sua variação dentro da célula. Tal sistema de percepção auxilia o sistema de navegação do robô móvel Pioneer 3-AT. Esse método permite que o robô possa tomar uma decisão mais acertada quanto à tentar atravessar ou não uma região da superfície mais elevada, uma solução interessante para permitir o tráfego de robôs em ambientes externos, desestruturados e imprevisíveis, assim como em explorações planetárias.

Já o trabalho de Hart [39] de 1968 propõe um algoritmo, hoje conhecido por, A*. Tal algoritmo é capaz de encontrar a rota de menor custo e de forma ótima. Ele percorre um grafo de vértices do inicio ao fim e retorna a melhor rota possível. Esse método surgiu através de uma combinação de aproximações heurísticas como do algoritmos *Best-first* e da formalidade do *Algoritmo de Dijsktra*. No entanto, essa solução tem um custo computacional relativo elevado, já que se faz necessário ter informações prévias sobre o ambiente para permitir ao algoritmo calcular todas as possíveis rotas e retornar o melhor resultado.

Uma das primeiras abordagens sobre algoritmos de planejamento de rotas foi proposta por khatib [2], baseada no método de campo potencial artificial (CPA), empregado em sis-

temas anti-colisão para manipuladores e robôs móveis. Basicamente a ideia é utilizar uma função potencial similar ao conceito físico de potencial eletrostático. O método de CPA é simples e eficaz, consiste de forças de atração e repulsão que fazem o robô se mover no espaço. O alvo (destino final) do robô exerce uma força de atração, enquanto os obstáculos exercem uma força repulsiva. Assim, a força resultante (força atrativa + força repulsiva) guia o robô até seu destino final contornando os obstáculos que obstruem o caminho de forma automática. O método foi implementado no sistema COSMOS para um robô manipulador Puma 560 usando sensoriamento visual. O grande desafio superado por Khatib foi implementar um algoritmo capaz de realizar tarefas em tempo real, evitar a colisão com obstáculos estáticos e dinâmicos.

Similar a isso, Hwang [14] atribui uma função potencial à cada obstáculo para identificar a chance de colisão com um robô móvel, onde obstáculos podem ser representados por superfície ou volume. Para evitar a colisão, um descritor global é usado com intuito de ajudar o robô à encontrar o melhor caminho, sem ou com o mínimo de obstáculos possível, dado a trajetória baseada no mínimo potencial. Então, um descritor local modifica a saída do descritor global, com o objetivo de encontrar a melhor rota e orientação. Caso o descritor local falhe, tendo informado um caminho que é obstruído por algum obstáculo, o sistema volta a chamar o descritor global para refazer a rota, e todo processo é repetido.

Khuswendi [3] apresenta um conjunto de algoritmos derivados dos tradicionais métodos de campo potencial artificial e A*. Um ambiente virtual permite simular o comportamento de quatro algoritmos: campo potencial artificial, A* 3D, A* 3D hierárquico e o método de escape horizontal A* 3D. O objetivo é avaliar cada método e escolher o melhor para implementar em um VANT de asa-fixa chamado Puna Sriti. Além disso, o trabalho mostra que o algoritmo A* 3D garante um tempo de resposta menor para encontrar a melhor rota, quando utiliza uma estratégia de escape horizontal, o que diminui o custo relacionado a distância à ser percorrida. O sistema simulado é composto por vários nodes (*waypoints*) de uma rota que inferem uma força de atração. Assim, um campo potencial U_0 representa os obstáculos, e um outro campo U_n, representa todos os pontos (posições).

Bentes [4] propôs um planejador global A* para encontrar uma rota livre ou com o menor número de obstáculos possível, além de usar o método de campos potenciais para controlar a formação de enxames de VANTs autônomos. O algoritmo é testado num sistema de simu-

lação de ambientes 2D e 3D. Um novo parâmetro referente ao cálculo do campo repulsivo é atualizado durante o voo do VANT que refaz a rota, de acordo com os dados de sensores virtuais espalhados pela formação. A simulação de ambientes 3D com obstáculos possibilita realizar testes da eficácia do método anti-colisão proposto, assistir as possíveis rotas de escapes verticais dos VANTs e a reorganização da formação do enxame.

Enquanto, Chen [40] apresenta, em um ambiente virtual, um algoritmo 3D de planejamento de caminho para um VANT utilizando um campo potencial artificial melhorado capaz de evitar a colisão em ambientes dinâmicos, que demonstrou ser rápido quanto à resposta e de alta precisão. O algoritmo sana em parte o problema do tradicional método CPA de não conseguir alcançar o alvo (destino final), quando este está próximo à obstáculos. Um limiar variável na função da força repulsiva é acrescentado. Quando a distância entre o robô e o alvo é pequena, o limiar faz com que qualquer força repulsiva seja diminuída ou eliminada. Porém, não há nenhuma garantia de sucesso, caso isso aconteça. Além disso, o algoritmo também é capaz de planejar a rota evitando regiões de picos. Quando obstáculos altos são encontrados, o planejamento da rota é refeito evitando tais regiões.

Kundu [41] propõe o uso de campos potenciais para controlar um robô móvel terrestre WMR não-holonômico num ambiente real e também modelado no Simulink de acordo com suas características físicas. O sistema rastreia o robô real através de uma câmera web HD fixada num ambiente *indoor*, onde há apenas obstáculos estáticos, enquanto simula, simultaneamente, o deslocamento do robô num ambiente virtual. Para localizar o robô no ambiente real é empregado uma técnica de realidade aumentada que projeta um objeto virtual (marca artificial) sobre o centro do robô.

Já no trabalho de Liang [24] é simulado um método de fluídos mecânicos para fazer um VANT voar suavemente, contornando obstáculos em formato de esferas, e em seguida insere uma interpolação para evitar a colisão do VANT com múltiplos obstáculos, além de também ser usado um algoritmo chamado "rede neural competitiva Fuzzy generalizada"(do inglês, *Generalized Fuzzy Competitive Neural Network*, G-FCNN) para mensurar as rotas possíveis para voos. Essa solução é sugerida como uma alternativa à outros métodos tradicionais de planejamento de caminhos, como por exemplo, A* e diagrama de Voronoi, que quando trabalham com dados tri-dimensionais, o custo computacional aumenta consideravelmente.

Kong [8] propôs uma solução baseada num sistema monocular (com uma única câmera)

para um sistema anti-colisão de "Micro VANT"(*Micro Air Vehicle* - MAV) Ar Drone 2.0 da Parrot para evitar a colisão com obstáculos em ambientes *indoor*. Por causa do problema de *payload*, é imprescindível ter um sistema de visão computacional que funcione bem utilizando um pequeno sensor, neste caso, a própria câmera do Micro VANT. Foi realizada uma análise de 9 algoritmos diferentes de planejamento de rotas e, em seguida, combinaram 5 deles: *Thresholding,Blurring, Edge detection, Hough transform* e *Contour detection*, usando a biblioteca de visão computacional 'OpenCV' para criar um novo algoritmo, mais eficiente e mais rápido.

Nascimento [16] apresenta uma abordagem anti-colisão para um sistema multi-robô que usa campo potencial como um termo da função de custo para a formação de modelos de controle preditivo não-linear *Nonlinear Model Predictive Formation Control* (NMPFC). A abordagem emprega um controlador local em cada robô que varia a velocidade com base num controle PID (*Proportional-Integral-Derivative*) que, por sua vez, usa uma função custo para cada robô (jogador de futebol) ser penalizado de acordo com a geometria da formação ou o desvio do alvo, ou seja, as ações de cada robô influenciam no comportamento do grupo. Para garantir a convergência do grupo de acordo com o objetivo do jogo e superar os problemas inerentes ao gradiente descendente puro, foi usado o método de Propagação Resiliente (do inglês: *resilient propagation* - RPORP) com intuito de ajustar os pesos da função de erro internamente, em cada robô.

No trabalho de Zhu [17] é proposto um algoritmo baseado num campo potencial artificial modificado, chamado 'MAPF', que é capaz de decompor a força total e estimar as barreiras físicas, em formas de cilindro, no ambiente 3D. Para construção do MAPF, considera-se o número de obstáculos, velocidade e posição do robô, além da posição do alvo. Em seguida é gerada a força resultante que infere na navegação de um VANT real e simulado, simultaneamente. O objetivo é ter um VANT que possa responder rapidamente a incertezas de um ambiente dinâmico e atualizar sua rota online, em tempo real. Zhu ainda elenca dois principais problemas comuns ao tradicional método de CPA: problema de Deadlock e o problema de Oscilação. O primeiro problema trata-se da anulação da força atrativa quando o alvo está próximo de obstáculos, pois a força repulsiva aplicada sobre o VANT é igual a um ângulo de 180 graus. O segundo problema refere-se a passagens estreitas onde um conjunto de obstáculos pode gerar uma força repulsiva que faça o robô se afastar de forma grosseira, por

causa do ângulo mencionado acima. Algumas restrições físicas (parâmetros) são consideradas importantes para fazer com que o VANT tenha um melhor desempenho segundo Zhu. Os parâmetros usados pelo sistema são: velocidade máxima de voo, raio mínimo de curva horizontal e raio mínimo de curva vertical. Todos esses parâmetros são levados em conta no cálculo do novo algoritmo proposto, MAPF. O algoritmo gera uma rota que e consulta os parâmetros de restrições físicas. Caso a rota não esteja de acordo com essas restrições, é gerada uma nova rota para o VANT.

Enquanto, Hameed [5] propôs um novo algoritmo baseado em campos potenciais artificiais aplicado ao planejamento de caminho para tratores-robôs utilizados no setor agrícola, visando reduzir a sobreposição de áreas já exploradas pelo robô. Foi desenvolvida uma abordagem cilíndrica para estimar as os desvios e sobreposições entre faixas necessárias em cada região devido a natureza da topologia gráfica do terreno. Uma análise de outros algoritmos para o mesmo propósito (áreas sobrepostas) é feita através de uma abordagem numérica. Experimentos usando um robô real e simulado num ambiente virtual são realizados para comprovar a eficiência do algoritmo. Os resultados apontam uma economia de 2 à 14% do percurso que seria feito pela máquina.

Finalmente, Mac [10] enumera alguns problemas associados ao tradicional método de campo potencial e foca no problema do robô não conseguir atingir o alvo, chegar ao destino final, quando o alvo está próximo a obstáculos que inferem uma força repulsiva anulando ou diminuindo drasticamente a força atrativa que o alvo exerce. Esse problema ocorre sempre que o alvo está muito próximo de obstáculos, e a proposta aqui é um método de campo potencial modificado (do inglês, *modified potential field method* - MPFM) baseado na compensação da força repulsiva adicionando a distância euclideana relacionada à força atrativa ao cálculo da força repulsiva. Isso faz com que a força repulsiva caia e a força atrativa aumente permitindo que o robô consiga atingir o alvo.

A localização do robô aéreo, usado na realização dos experimentos, funciona com base numa grade física de marcas artificiais espalhadas pelo chão que é "lida"por uma câmera acoplada abaixo do Ar Drone 2.0 da Parrot, que conta também com uma câmera frontal para detectar os obstáculos. O computador calcula a rota e faz o upload de uma lista de *way-points* (sequência de pontos de uma rota) para que um controlador PD embarcado no Drone possa assegurar que a rota será realizada com segurança, mesmo perdendo a comunicação com a

base de comando terrestre, e tudo é monitorado através de um ambiente virtual em tempo real.

Uma comparação entre os principais métodos de planejamento de rotas aplicado à robótica móvel pode ser encontrado no trabalho de [42], porém o centro da discussão aqui são os métodos derivados do tradicional algoritmo A*. Já no trabalho de [43], mais recente, há uma comparação experimental desses algoritmos de planejamento de rotas aplicados à micro agentes magnéticos, objeto de investigação científica na área biomédica para condução de fármacos.

Contudo, cada algoritmo tem suas vantagens e desvantagens. O tradicional método de Campo Potencial Artificial (CPA) é o mais rápido, ou seja, de menor tempo de resposta segundo o trabalho de [43]. Porém, CPA não é um algoritmo ótimo (não há garantia de que ele alcançara o alvo). Diversos trabalhos abordam CPAs modificados para contornar os problemas comuns à este método. Já o algoritmo A* e seus derivados [43] é do tipo ótimo, retorna a melhor rota, porém não é tão rápido quanto o algoritmo de CPA. A grade de ocupação, por exemplo, é um método probabilístico que pode retornar diferentes rotas para um mesmo cenário. Embora seja um método capaz de se adaptar à imprevistos, há uma representação obsoleta da ocupação do espaço, podendo não condizer com a realidade.

Uma listagem dos principais algoritmos de planejamento de rotas, com base na literatura [42; 43], é apresentada na Tabela 3.1.

Com base no estudo dos algoritmos de planejamento de rotas citados aqui, para o desenvolvimento desse trabalho optamos pelo uso do método de campos potenciais artificias por ser de resposta rápida (baixo custo computacional), flexível, não depende da densidade de ocupação, é de fácil implementação, estrutura simples e eficiente para a maioria das aplicações [41; 23; 10].

No entanto, nossa proposta se difere do tradicional método de campos potenciais artificiais e de outros mencionados acima, por se tratar de um novo algoritmo de CPA que funciona com base na manipulação de informações 3D obtidas através de uma nuvem de pontos (da PCL), sem converter esses dados para um sistema bidimensional - o que geralmente é feito para diminuir o custo computacional e obter respostas mais rápidas do sistema quanto à detecção de obstáculos. Porém, esse tipo de solução impede a navegação do robô próximo à objetos, quando o sistema, erroneamente, admite como verdade um falso risco de colisão.

Tabela 3.1: Listagem de algoritmos

Algoritmos	***Forças***	***Fraquezas***
Lógica Fuzzy [7]	Rapidez	Baixa representatividade do ambiente 3D
CPA + A* [4]	melhor rota para sistemas multi-robô (eficiência)	simulação, custo computacional maior que CPA
Grade de ocupação [13]	Rápido e eficaz	+ Informação resumida do ambiente 3D, representado em um plano 2D
CPA 2.5D [1]	Os manipuladores trabalham com coordenadas 3D	Parte do ambiente é representado em 2D
OEG [18]	É possível identificar relevos que obstruam a passagem do robô	Não permite que o robô passe por debaixo de mesas
A* [39]	Retorna melhor rota	Elevado custo computacional
CPA [2]	Rapidez e eficácia	Problema dos mínimos locais
G-FCNN [24]	Rapidez e eficácia	- simulação, ambiente discreto
NMPFC [16]	controle de sistemas multi-robôs	oscilação em espaços estreitos
MAPF [17]	adaptação a ambientes dinâmicos	oscilação em ambientes desorganizados (cheios de objetos)
MPFM [10]	Garante chegar ao mínimo global	Depende de sensores externos (inseridos no ambiente)
CPA proposto	**Não sofre com mínimos locais**	**Risco de não-detectar objetos à distâncias menores que 40 cm (limitação do Kinect)**

Com isso, queremos atacar os seguintes problemas comuns atrelados à algoritmos de campos potenciais artificiais:

1. Alvo inacessível;
2. Impedimento de passagem entre obstáculos próximos;
3. Oscilações na presença de obstáculos;
4. Oscilações em passagens estreitas.

O sistema implementado neste trabalho não depende de sensores externos (inseridos no ambiente) para que o robô tenha sucesso em sua missão (de encontrar o alvo - destino final) e utiliza a biblioteca de visão computacional *Point Cloud Library* (PCL) para criar uma nuvem de pontos composta por dados 3D que é a base de informação de entrada para o algoritmo de campo potencial artificial modificado proposto nesse trabalho.

3.2 Considerações Finais

Este capítulo apresentou uma revisão de trabalhos sobre sistemas anti-colisão aplicados à robôs móveis terrestres e aéreos.

Foi considerado um estudo dos conceitos e funcionamento, além de apresentar o algoritmo escolhido para o desenvolvimento de nossa aplicação e o porquê. A seguir, é apresentada a metodologia utilizada nesse trabalho de forma detalhada.

Capítulo 4

Metodologia

Neste capítulo apresentamos uma metodologia de trabalho para evitar a colisão de um robô móvel com obstáculos encontrados em ambientes *indoor*. A Seção 4.1 apresenta o método Campo Potencial Artificial Modificado, a Seção sec:sistemaimplementado apresenta a arquitetura do sistema implementado, a Secão sec:kinect aborda detalhes do sensor Kinect utilizado para percepção do ambiente e a Seção sec:frameworks introduz aos frameworks e tecnologias usadas no desenvolvimento do sistema anti-colisão aqui proposto.

4.1 Campo Potencial Artifical 3D

A ideia por trás do método de Campo Potencial Artificial (CPA) vem do conceito físico de campos, construção matemática que tem um valor numérico em cada ponto no espaço e tempo, a qual resulta num gradiente que representam forças repulsivas e atrativas. Na robótica, esse método é aplicado para solucionar o problema de planejamento de rotas. Para tal, a configuração final do robô (pose e orientação) é configurado como uma "região de atração", ou seja, o alvo (destino final) exerce uma força atrativa, um potencial negativo, sobre o robô, enquanto os obstáculos repelem o robô com seu potencial positivo. Devido a natureza dos campos, o campo resultante é exatamente a soma de todos os campos existentes, e o caminho ótimo é um que minimize o trabalho equivalente, ou seja, a força resultante não pode ser igual a zero enquanto o robô não atingir o alvo [23].

4.1.1 Campo e força atrativa

Nosso sistema foi desenvolvido utilizando como referência o mapa local e a informação de orientação baseadas no sistema de coordenadas do robô *Turtebot 2*. O mapa local é apresentado pela Figura 4.1(a). Nesse mapa, no eixo x está a abscissas, a distância à frente do robô, e no eixo y está a coordenada ordenada no plano. Essas coordenadas são definidas pelo *Turtebot* na inicialização e são usadas como uma referência padrão para o método de campos potenciais artificiais. A orientação do *Turtebot* é estimada a partir da informação de sua bússola que está embarcada no próprio robô.

Diferentemente do sistema de coordenadas do mapa local do robô *Turtlebot*, que é um plano (2D), o sistema de coordenadas do sensor RGB-D Kinect trabalha com a dimensão 3D e seus eixos são diferentes, como mostra a Figura 4.1(b), o qual é explanado em detalhe na próxima sessão.

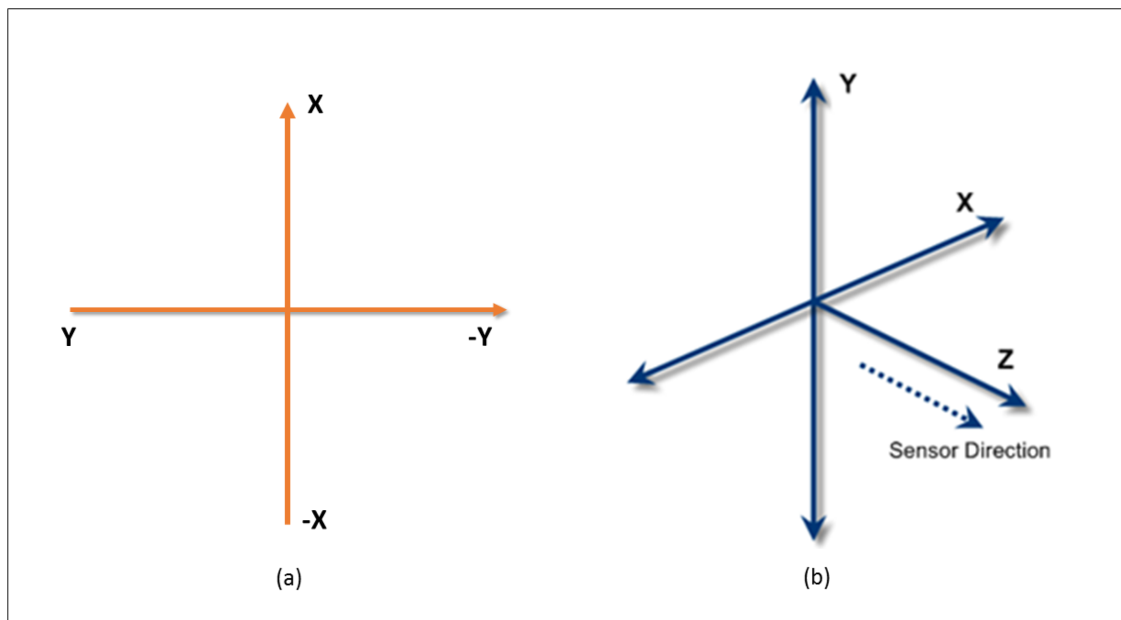


Figura 4.1: Apresenta o mapa local do ’Turtlebot 2’ (a) e o sistema de coordenadas do Kinect (b)

4.1.2 Campo e Força Repulsiva

Inicialmente, os dados obtidos pelo sensor Kinect são processados utilizando a biblioteca de visão computacional *Point Cloud Library* - PCL para trabalhar com uma nuvem de pontos. Nosso algoritmo utiliza várias etapas de processamento de imagens para obter a nuvem resultante esperada, basicamente funciona da seguinte maneira: dois filtros chamados *passthrough* (passa-baixa) são aplicados à nuvem de pontos de entrada para reduzir a área (altura e largura) e, posteriormente, é aplicado outro filtro, chamado *voxel-grid*, para reduzir o número de pontos que devem ser processados, economizando tempo, diminuindo o custo computacional e aumentando a eficiência do sistema.

Primeiro, o filtro *passthough* é aplicado a cada coordenada, definindo um box, como um foco da nuvem de pontos, que descarta o que estiver fora dessa área. Depois, um filtro *voxel-grid* reduz o número de pontos em cada *voxel* de um grid virtual 3D que é aplicado sobre a nuvem. Esse filtro (*voxel-grid*) define que quaisquer pontos (dentro do box) deverão ter uma distância mínima de 0.01 m entre si, isso aumenta a eficiência do algoritmo, pois, com a diminuição do número de pontos, menos informações sobre o ambiente precisam serem processadas.

Trabalhar com dados tridimensionais obtidos através de imagens capturadas pelo sensor kinect, sem fazer qualquer conversão para 2D, por exemplo, é um desafio que exige que apenas dados relevantes sejam considerados como entradas possíveis para a computação, já que o custo computacional é elevado e um sistema anti-colisão precisa funcionar em tempo-real.

A nuvem original obtida através do uso da PCL, sem qualquer filtragem, impossibilita que uma CPU comum, como a utilizada nos experimentos deste trabalho (core i5 - 3ª geração) consiga responder em tempo hábil se o robô precisa desviar ou não de um obstáculo detectado à frente, aumentando o risco de colisão. Dessa forma, o sistema é incapaz de conseguir êxito, ou seja, evitar colisão. Por este motivo, é imprescindível que seja aplicado filtros que diminuam o número de pontos, além disso, algoritmos de segmentação como o RANSAC e o *Euclidean Cluster Extraction* ajudam a detectar, com maior precisão, conjuntos de pontos (dentro do box) que representarem obstáculos reais.

As dimensões especificadas pelo filtro *passthrough* devem ser menores que as dimensões da nuvem de pontos original e maiores que o tamanho do robô. O intuito disso é diminuir

o número de pontos que são processados, e assim diminuir o tempo de resposta do sistema anti-colisão, além de permitir que o robô transite por corredores estreitos e passe por debaixo de obstáculos suspensos que tem altura maior que a sua.

O campo repulsivo é extraído dos dados do Kinect, que por conseguinte identificam os clusters por suas coordenadas x, y, z do Kinect, as quais são diferentes das coordenadas do mapa local, como mostra a Figura 4.1.

Para calcular a intensidade do campo repulsivo, nós calculamos a distância de Manhattan, sobre o plano, entre o robô e cada ponto do cluster final. Assim, para cada ponto i do cluster, a distância de Manhattan é igual a:

$$d_i = |x_i^{(k)}| + |z_i^{(k)}| \tag{4.1}$$

Nós adotamos um potencial linear, ou seja, um potencial que varia proporcionalmente com a distância, na forma:

$$V_i = \lambda\{1 - d_i/D_{max}\}, \tag{4.2}$$

onde D_{max} é a maior distância considerada pelo box que é delineado pelos filtros do tipo *passthrough* e λ é uma constante. Para calcular os módulos da força atribuída à esse campo potencial, o que nós precisamos fazer é tomar a derivada:

$$|F_i| = -\lambda/D_{max}, \tag{4.3}$$

a qual é uma constante para todos os pontos da nuvem. Além disso, é preciso saber qual a direção dessa força, já que isso fará com que o robô desvie dos obstáculos, o que é definido pelo cálculo do ângulo:

$$\theta_i = \arctan\left(\frac{x_i^{(k)}}{z_i^{(k)}}\right), \tag{4.4}$$

o qual implica na força:

$$F_i^{(\hat{x}^{(k)})} = -\sin(\theta) \times \lambda/D_{max} \tag{4.5}$$

$$F_i^{(\hat{z}^{(k)})} = -\cos(\theta) \times \lambda/D_{max} \tag{4.6}$$

Com isso, a força repulsiva resultante é a soma das forças de cada ponto identificado em cada cluster:

$$F_{rep}^{(\hat{x}^{(k)})} = \lambda\ /D_{max} \sum_i \sin(\theta), \tag{4.7}$$

$$F_{rep}^{(\hat{z}^{(k)})} = \lambda\ /D_{max} \sum_i \cos(\theta), \tag{4.8}$$

com um módulo:

$$|F_{rep}| = \sqrt{(F_{rep}^{(\hat{x})})^2 + (F_{rep}^{(\hat{z})})^2} \tag{4.9}$$

É importante dizer que essa força é representada sobre o sistema de coordenadas do sensor Kinect, que é acoplado ao robô, e por isso pode se deslocar-se de forma translacional ou rotacional em relação as coordenadas do mapa local, à qualquer momento. Considerando que o robô é rotacionado à um ângulo φ em relação as coordenadas do mapa local, a força repulsiva resultante pode ser descrita em coordenadas locais como sendo:

$$\begin{aligned} F_{rep}^{(\hat{x})} &= \sin(\varphi)\lambda\ /D_{max} \sum_i \sin(\theta) + \\ &\quad + \cos(\varphi)\lambda\ /D_{max} \sum_i \cos(\theta), \end{aligned} \tag{4.10}$$

$$\begin{aligned} F_{rep}^{(\hat{y})} &= -\cos(\varphi)\lambda\ /D_{max} \sum_i \sin(\theta) + \\ &\quad + \sin(\varphi)\lambda\ /D_{max} \sum_i \cos(\theta). \end{aligned} \tag{4.11}$$

4.1.3 Força Atrativa e Repulsiva

O potencial atrativo é responsável por direcionar o robô até o alvo (destino final) e a dedução do deslocamento do robô é feita através do cálculo da distância euclideana, baseado nos dados obtidos pelos *encoders* ao longo do tempo. Consideramos que o alvo exerce uma força atrativa, do tipo gravitacional, sobre o sistema, que pode ser descrita da seguinte maneira:

$$|F_{att}| = \frac{\Lambda}{\sqrt{(x_a)^2 + (y_a)^2}}, \tag{4.12}$$

onde Λ é uma constante de intensidade e (x_a, y_a) são distâncias $\hat{x}$ e $\hat{y}$, respectivamente, do robô até o alvo, calculadas sobre o sistema de coordenadas do mapa local.

O ângulo da direção do alvo é dado por:

$$\phi = \arctan\left(\frac{y_a}{x_a}\right), \tag{4.13}$$

o qual implica nas componentes:

$$F_{att}^{(\hat{x})} = \frac{\cos(\phi)\Lambda}{\sqrt{(x_a)^2 + (y_a)^2}}, \tag{4.14}$$

$$F_{att}^{(\hat{y})} = \frac{\sin(\phi)\Lambda}{\sqrt{(x_a)^2 + (y_a)^2}}. \tag{4.15}$$

Portanto, a força resultante é a soma das forças atrativas e repulsivas:

$$F_{res}^{(\hat{x})} = F_{att}^{(\hat{x})} + F_{rep}^{(\hat{x})}, \tag{4.16}$$

$$F_{res}^{(\hat{y})} = F_{att}^{(\hat{y})} + F_{rep}^{(\hat{y})}, \tag{4.17}$$

Agora é preciso calcular o módulo e a direção da força resultante, respectivamente:

$$|F_{res}| = \sqrt{(F_{res}^{(\hat{x})})^2 + (F_{res}^{(\hat{y})})^2} \tag{4.18}$$

$$\psi = \arctan\left(\frac{F_{res}^{(\hat{y})}}{F_{res}^{(\hat{x})}}\right) \tag{4.19}$$

Lembrando que as coordenadas do mapa local (2D) do 'Turtlebot 2' difere do sistema de coordenadas (3D) do sensor Kinect, como apresentado na Figura 4.1.

4.2 Sistema anti-colisão implementado

A Figura 4.2 apresenta a arquitetura do sistema anti-colisão implementado e mostra o funcionamento principal do sistema.

Nesse trabalho, a nuvem de pontos é utilizada apenas para detectar obstáculos, de qualquer natureza (sejam móveis ou estáticos), e com isso, calcular a força de repulsão sobre cada ponto que estiver numa região de nosso interesse que podemos chamar de foco.

A nuvem de pontos é obtida com base nos dados do sensor de infravermelho simultaneamente à captura de uma imagem 2D pela câmera VGA, ambos embarcados no Kinect.

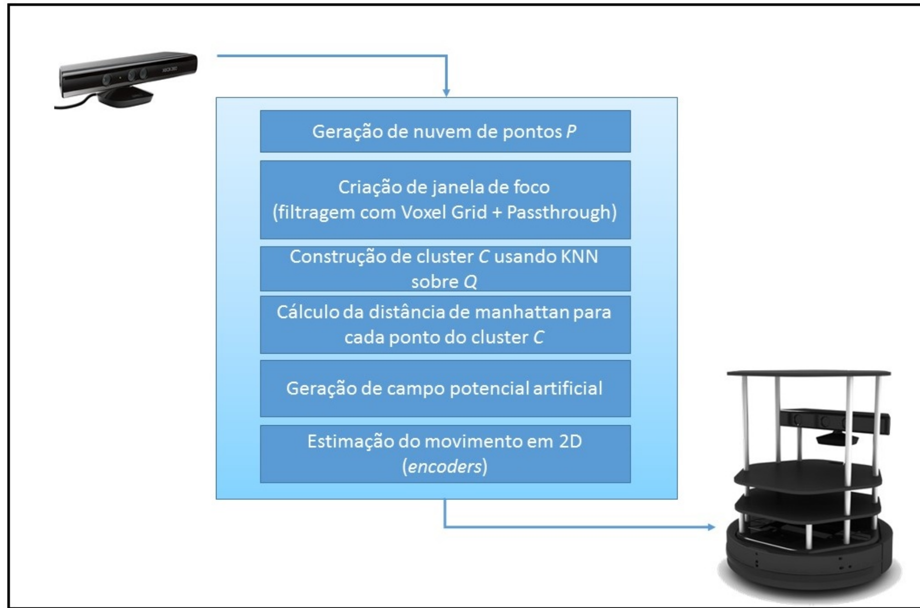


Figura 4.2: Arquitetura do sistema anti-colisão implementado.

Após aplicarmos os filtros *passthrough* e *voxel-grid*, é aplicado os métodos RANSAC, para segmentação dos objetos, e *Euclidean Cluster Extraction* para a formação de um único cluster utilizando uma estrutura Kd-tree [44] para encontrar os vizinhos mais próximos com o algoritmo de *k-nearest neighbors* (K-NN). Por fim, o método de campo potencial artificial é aplicado a nuvem resultante, que na verdade é um conjunto de clusters, para obtermos a força repulsiva total e o seu ângulo.

Depois, seguimos com o cálculo da força repulsiva sobre as coordenadas 3D de cada um dos pontos que compõem o cluster final (a nova nuvem de pontos resultante). Com essas coordenadas, faz-se a estimação da distância, com base no cálculo da distância de manhattan, do robô em relação aos obstáculos.

4.3 O Sensor RGB-D Kinect

Para realizar este trabalho, foi utilizado o Kinect, que é um sensor RGB-D robusto e que tem melhor desempenho do que câmeras em ambientes sem boa iluminação [45], além de permitir a obtenção de ricas informações das cenas, por meio de uma câmera VGA e um sensor infravermelho embarcados. Este sensor vem acoplado ao robô 'Turtlebot 2'. Pelo fator custo x benefício ser um atrativo, o Kinect vem sendo adotado em diversas aplicações

científicas [18] [46] [10] [47].

A Figura 4.3 (a) e (b) extraídas de [48] [49] representa o sensor RGB-D Kinect, que possui especificações:

Figura 4.3: Sensor RGB-D Kinect.

- Ângulo do campo de visão é de 43º na vertical e 57º na horizontal;
- Intervalo máximo de inclinação de ± 27º;
- 30 quadros por segundo;
- Uso de 4 microfones embutidos para captura de áudio no formato de 16 kHz, trabalhando com 24-bit para modulação por código de pulso (PCM);
- Um acelerômetro 8G configurado para a faixa de 2G, com limite máximo de 1º de precisão;
- Intervalo padrão de distância em que o infravermelho pode identificar objetos é de 0,8 a 4 metros.

Embora o sensor Kinect seja uma alternativa interessante para fazer pesquisa na área de visão computacional e de baixo custo, há algumas limitações de distância em que esse sensor RGB-D pode ou não detectar obstáculos, como mostra a Figura 4.4 extraída de [50]

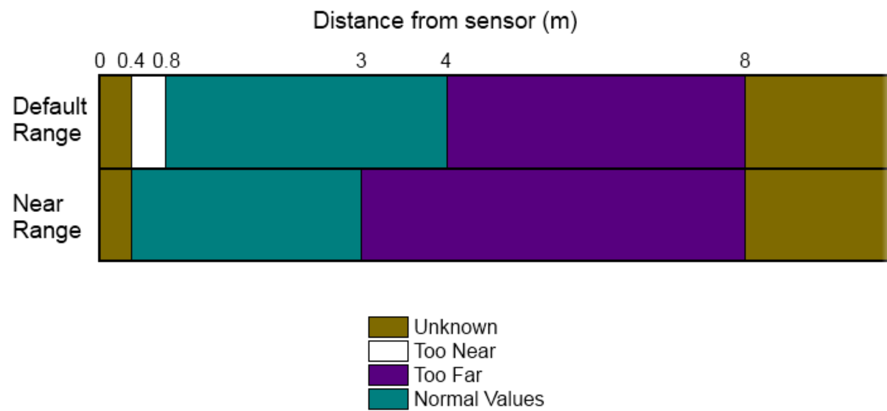


Figura 4.4: Range do Kinect.

4.4 Framework de desenvolvimento e bibliotecas

4.4.1 PCL e Freenect usando ROS

A biblioteca de visão computacional *Point Cloud Library* (PCL) é um projeto *open source* e muito utilizada para o desenvolvimento de aplicações que trabalhem com processamento de imagens 2D e 3D, e principalmente, com nuvem de pontos [51].

Assim como ROS, a PCL também é um framework distribuído sob a licença BSD [36] que conta com diversos algoritmos do estado-arte da área de visão computacional que podem ser comercializados ou utilizados para fins de pesquisa científica. Alguns exemplos dos módulos PCL mais importantes, são: filtros, *features*, pontos-chaves, registro, kd-tree, octree, segmentação, entre outros. Todos os códigos exemplos e explicações sobre os mesmos podem ser encontrados na documentação disposta no site do próprio projeto [52].

A Figura 4.5 apresenta os algoritmos PCL e o fluxo em que foram implementados para a solução de percepção 3D proposta neste trabalho.

Na PCL, cada ponto da nuvem é denotado pelas coordenadas XYZ no espaço tridimensional. Essa nuvem pode ser filtrada, diminuindo o número de pontos ou até mesmo criando um foco na área de interesse do sistema, durante a fase de pré-processamento, onde utilizamos os filtros *passthrough* e *voxel-grid*, como citado anteriormente.

Vale salientar que o ROS permite alterar os parâmetros de todos os filtros em tempo real através de uma GUI de desenvolvimento para ROS chamada 'rqt'.

Obtendo a nuvem de pontos já filtrada, aplica-se um modelo de segmentação de plano

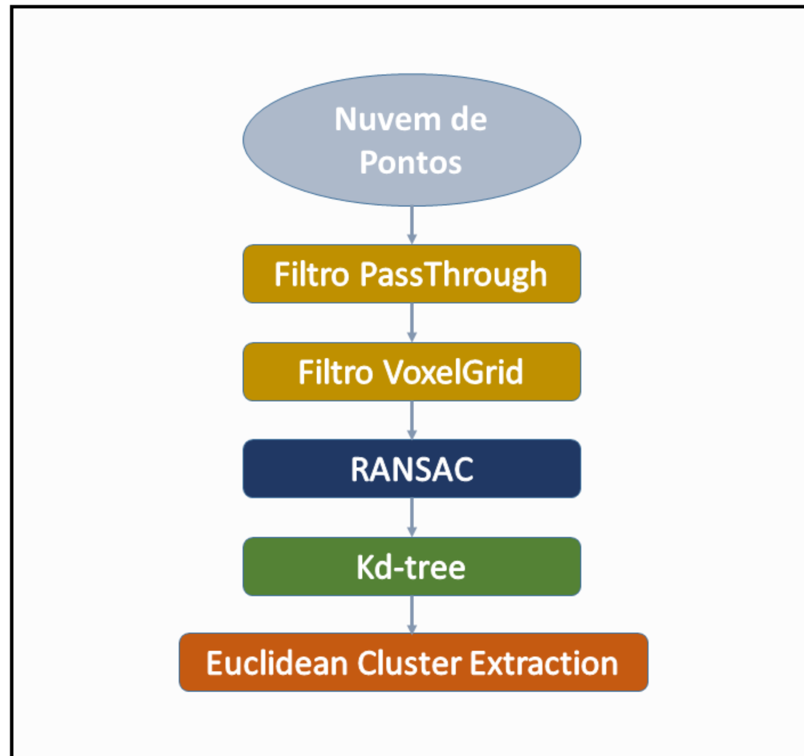


Figura 4.5: Fluxo dos algoritmos implementados da PCL.

baseado no algoritmo RANSAC [53] que é capaz de detectar o plano principal onde se encontram os objetos. Mais detalhes sobre o funcionamento dos algoritmos *passThrough*, *voxel-grid*, e RANSAC podem ser encontrados nas subseções 4.4.2 e 4.4.3.

No nosso trabalho adotamos o tipo de dado (mensagem ROS) *PointCloud2* por se tratar da estrutura padrão utilizada pelo ROS, além de ser a versão mais recente, e também dispor de suporte pela comunidade.

A principal vantagem em utilizar o ROS no desenvolvimento deste trabalho é ter a liberdade de desenvolver um sistema modular, com nodes independentes que se comunicam entre si através de tópicos, independente de qual seja a linguagem de programação utilizada (C++ ou Python), além de facilitar a compilação do código e possibilitar o uso de *launchs* (scritps ROS) que acessam outros recursos como a biblioteca 'freenect' capaz de acionar o Kinect. Assim, o ROS acaba abstraindo parte do desenvolvimento oferecendo programas auxiliares que podem ser chamados por uma simples linha de comando no shell.

A biblioteca Freenect foi desenvolvida, e é atualmente suportada, pelo projeto *open source* 'OpenKinect' que dispõe de um conjunto de drives para o sensor Kinect, chamado

'libfreenect'.

4.4.2 Pré-processamento

A nuvem de pontos obtida pelo Kinect pode conter mais de 300.000 pontos de entrada, como mostra o trabalho de Junemann [54]. Para construir um sistema de percepção robusto é preciso diminuir o tempo de resposta. Na área de robótica isso é fundamental, já que se trata de um sistema de tempo real que interage com ambientes que podem ser dinâmicos e isso oferecer riscos a integridade física, tanto do robô, quanto dos objetos e pessoas que estejam compartilhando o mesmo espaço.

Preocupando-se com isto, tem-se a necessidade de realizar um pré-processamento através de uma filtragem que seja capaz de diminuir o tempo de resposta do sistema de percepção às transformações inesperadas do ambiente.

O pré-processamento de uma nuvem de pontos pode ser feita através de filtros que diminuam a área da nuvem (criando um foco) ou o número total de pontos (*downsampling*).

O filtro *Passthrough* (filtro passa baixa) é utilizado para delimitar uma "área de interesse"de atuação do algoritmo. Os limites dessa área são definidos pelo programador e podem ser facilmente alterados em tempo real através da ferramenta 'rqt', como mencionado anteriormente. Esse filtro pode atuar sobre uma nuvem de pontos em quaisquer das coordenadas do sensor Kinect [55].

Simbolicamente, os parâmetros do filtro *PassThrough* utilizando a informação de profundidade da coordenada 'z' como exemplo, definem o intervalo de valores desejado, como $l_{min} < zl_{max}$.

Junto a isso, o filtro *voxel-grid* é utilizado para diminuir o número de pontos da nuvem. Esse processo é chamado de *downsampling*. O filtro VoxelGrid cria uma espécie de grade tridimensional (*voxel grid*) sobre a nuvem de pontos, como organizasse a nuvem de pontos em subconjuntos dentro de cubos (como "pequenas caixas de espaço"). Então, todos os pontos de uma região que cabem num cubo são aproximados ao centroides do próprio cubo.

Embora essa abordagem seja mais lenta comparada a hipotese de aproximar os pontos da nuvem ao centroide do voxel, ela representa melhor a superfície subjacente [56].

O tamanho do cubo é definido pelo programador através do vetor tridimensional *leaf size*. Neste trabalho, o tamanho utilizado da "caixa"foi de 0,01 m x 0,01 m x 0,01 m.

4.4.3 RANSAC

O RANdom SAmple Consensus - RANSAC é um método iterativo utilizado para estimar os parâmetros de um modelo matemático, proposto por Fischler [53] em 1981. Esse método assume que todo conjunto de dados é composto por *inliers* e *outliers*.

Podemos definir *Inliers* como sendo dados que compõe um conjunto de dados com características semelhantes. Enquanto, *outliers* são dados distantes de semelhar-se à maioria dos dados de um mesmo conjunto. Um exemplo prático disso seria detectar pontos dentro um plano de uma mesa real (*inliers*) e outros pontos fora da mesa (*outliers*). O resultado seria similar à este da Figura 4.6 extraída de [57].

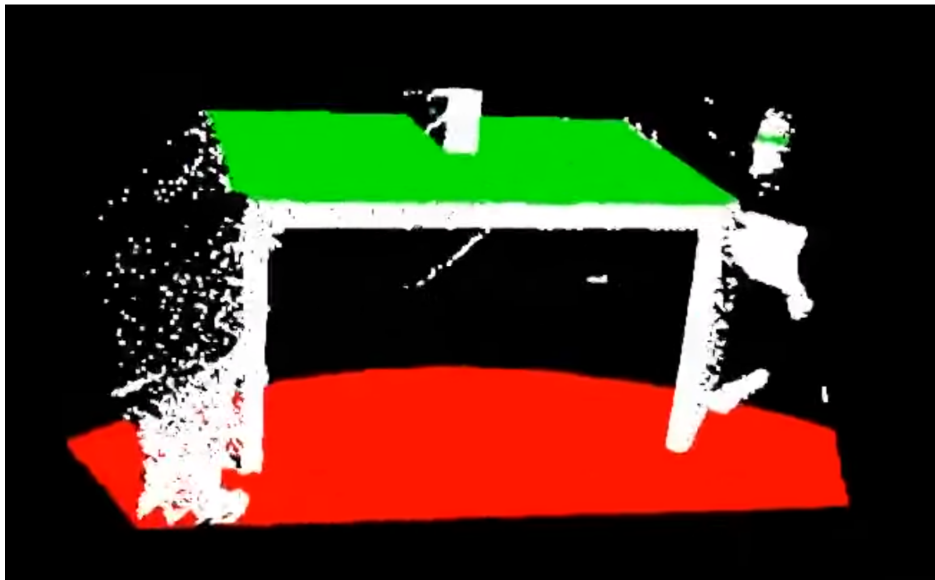


Figura 4.6: Exemplo de segmentação de plano com RANSAC.

Ao contrário das técnicas convencionais de amostragem como *least-median squares* [58] e *M-estimators* [59] encontradas na literatura de estatística, RANSAC usa o menor conjunto de amostras possível e prossegue aumentando com dados relevantes [53]. Assim, o método RANSAC pode estimar os parâmetros de um modelo escolhido a partir dos dados disponíveis [60].

O funcionamento do algoritmo RANSAC é apresentado com clareza no trabalho de Castro [61] e reproduzido à seguir com algumas adaptações, incluindo os parâmetros encontrados para o trabalho aqui proposto. Podemos resumir o funcionamento do algoritmo de RANSAC nos seguintes passos:

1. Obter uma nuvem de pontos $\boldsymbol{N}$ e um modelo $\boldsymbol{M}$ de um plano.
2. Escolher aleatoriamente 3 pontos contidos em $\boldsymbol{N}$, os quais serão chamados de *inliers* hipotéticos. Se esses 3 pontos forem colineares (plano ambíguo), selecionar outros 3 pontos aleatoriamente. Incluir esses 3 pontos em um subconjunto $\boldsymbol{S}$ da nuvem $\boldsymbol{N}$.
3. A partir desses 3 pontos, estimar o modelo $\boldsymbol{M}$, calculando os parâmetros $\boldsymbol{a}$,$\boldsymbol{b}$,$\boldsymbol{c}$ e $\boldsymbol{d}$ da equação do plano: $\boldsymbol{ax + by + cz + d = 0}$
4. Usando o modelo estimado, calcular quantos pontos da nuvem se encaixam bem ao modelo e incluí-los em $\boldsymbol{S}$. Um ponto é considerado um *inlier* se a distância para o plano não exceder um determinado limiar (neste trabalho, usou-se 2 cm).
5. Se uma quantidade definida de pontos for classificados como *inliers* hipotéticos, então o modelo $\boldsymbol{M}$ é suficientemente bom. Pular para o último passo.
6. Repetir o passo 3 usando os novos *inliers* hipotéticos.
7. Avaliar o modelo $\boldsymbol{M}$, estimando os erros dos pontos de $\boldsymbol{S}$ em relação a $\boldsymbol{M}$.
8. Salvar o modelo que tenha menor erro.
9. Voltar para o passo 2 um certo número $\boldsymbol{k}$ de vezes (foi utilizado $\boldsymbol{k} = 100$).
10. Retornar o modelo $\boldsymbol{M}$ junto com todos os *inliers*.

Mais detalhes sobre o método RANSAC pode ser encontrado em [53] [54] [61].

4.4.4 K-d Tree

K-d tree, ou k-dimensional tree, é uma estrutura de dados (árvore binária) que serve para efetuar operações de decomposição espacial [61]. K-d trees são muito úteis para procura de vizinhos mais próximos. Sua estrutura é composta por níveis que dividem todos os filhos numa dimensão específica usando um hiperplano que é perpendicular ao eixo correspondente [44]. Essa estrutura é comumente usada pela maioria dos algoritmos da PCL.

A divisão de todos os filhos dar-se a partir da primeira dimensão, ou seja, da raiz da árvore. O nível inferior ao da raiz dividi-se na próxima dimensão e assim por diante. Geralmente esta divisão é efetuada com a média da dimensão com o maior variância no conjunto

dos dados. Por exemplo, se a dimensão x é escolhida, todos os pontos do espaço com valores menores irão aparecer do lado esquerdo da árvore, enquanto os pontos com valores maiores que x irão aparecer do lado direito [61].

A Figura 4.7 extraída de [62] apresenta um desenho de uma estrutura KdTree. A divisão no primeiro nível (nó raiz) é representada pelo plano vertical vermelho. Em seguida Os retângulos verdes representam o segundo nível da árvore. E, por fim, o terceiro nível é representado pelos retângulos azuis da imagem.

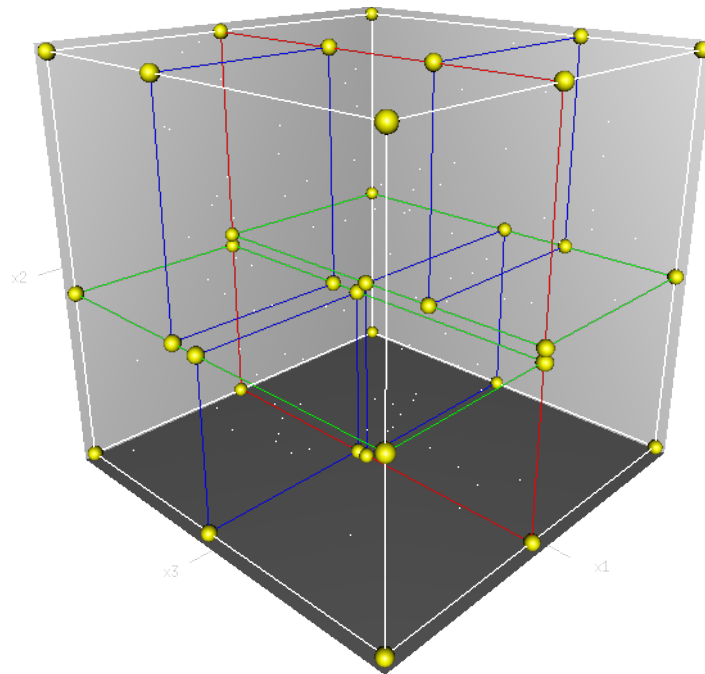


Figura 4.7: Estrutura KdTree.

4.4.5 Euclidean Extraction Cluster

Tendo a nuvem de pontos já filtrada P, nós aplicamos o método chamado "Cluster Euclideano"(do inglês: *"Euclidean Extract Cluster"*) da PCL para construir o cluster que queremos. Esse método divide a nuvem desorganizada P em partes menores (sub-nuvens), que reduzem o custo computacional referente ao tempo de processamento para P [63]. O método de extração do Cluster Euclideano consiste de quatro etapas básicas:

1. Criar uma representação Kd-tree para nuvem de entrada P [44];

2. Iniciar um cluster vazio C que será preenchido por uma fila de pontos à serem checados Q;

3. Então, para todo ponto $p_i \in P$, é necessário:

   - adicionar p_i à fila atual Q;
   - para todo ponto $p_i \in Q$, procure para o conjunto de pontos $P_k i$ os pontos vizinhos de p_i encontrados na esfera de tamanho (de raio) $r < d_(th)$;
   - para todo vizinho $P_i k \in P_i k$ cheque se o ponto já foi processado, se não, adicione-o a fila Q.
   - quando todos os pontos $p_i \in P$ tiverem sido processados, iremos obter os sub-clusters (que representam todos os obstáculos detectados na cena) projetados.

Por fim, os sub-clusters são transformados em único cluster, ou seja, uma nova nuvem de pontos. Essa nova nuvem de pontos é utilizada para calcular a força repulsiva do sistema.

4.5 Considerações Finais

Este capítulo apresentou uma metodologia de trabalho para evitar a colisão de um robô móvel com obstáculos encontrados em ambientes *indoor*. No próximo capítulo é apresentado uma avaliação experimental sobre o estudo de caso.

Capítulo 5

Avaliação Experimental

Neste capítulo apresentamos todas as ferramentas e tecnologias necessárias para o desenvolvimento do sistema anti-colisão proposto, assim como os requisitos mínimos e os resultados obtidos através dos experimentos realizados no laboratório LASER da UFPB. A Seção 5.1 apresenta as ferramentas e tecnologias, além dos requisitos mínimos de hardware e software, por fim, a Seção 5.2 explana sobre os experimentos realizados e os resultados obtidos, assim como faz uma análise da performance do sistema anti-colisão desenvolvido.

5.1 Estudo de Caso

5.1.1 Ferramentas e Tecnologias

As ferramentas e tecnologias adotadas para o desenvolvimento desse trabalho foram escolhidas com base em uma criteriosa pesquisa do estado da arte. Para conseguirmos testar o nosso algoritmo de campos potenciais artificiais 3D num ambiente real, utilizamos as seguintes ferramentas e tecnologias:

1. um robô móvel 'Turtlebot 2' que vem com um sensor RGB-D Kinect acoplado;

2. um notebook HP (descrito em detalhes na próxima subsessão);

3. biblioteca de processamento de imagens, *Point Cloud Library* (PCL);

4. biblioteca de acesso ao hardware Kinect, chamada Freenect;

5. e, por fim, o framework de desenvolvimento de softwares para robôs, *Robot Operating System* (ROS).

A seguir, apresentamos os requisitos mínimos para a construção do sistema implementado neste trabalho.

5.1.2 Requisitos

A Tabela 5.1 mostra os demais equipamentos de hardware e ferramentas de software utilizados para o desenvolvimento da aplicação, juntamente com suas especificações.

5.2 Experimento

5.2.1 Execução do Experimento

Os experimentos foram realizados no espaço do Laboratório de Sistemas Embarcados e Robótica (LASER) da Universidade Federal da Paraíba (UFPB) usando uma plataforma robótica *Turtlebot 2* com 60 cm de altura e 48 cm de largura (já com o laptop acoplado).

Nós não transformamos a nuvem de pontos (3D) numa mapa de profundidade (2D) porque a nuvem de pontos apresenta o campo visual por inteiro, considerando que admitimos apenas uma região limitada do espaço para o cálculo dos campos potenciais. Por isso, é fundamental, para o nosso algoritmo, ter o maior número de informações possível. Ao invés de um mapa de profundidade, nós utilizamos uma nuvem de pontos que passa por um pré-processamento antes de ser aplicarmos algoritmos de segmentação, como mencionado anteriormente.

Durante nossa pesquisa encontramos um *launch* do ROS que chama a Freenect, capaz de alinhar automaticamente os dados do infravermelho com os da câmera VGA sem precisar fazer qualquer tipo de calibração prévia dos sensores (embarcados no Kinect) [64].

Foram realizados quatro tipos de experimentos, listados na tabela 5.2, que utilizamos como base para provar a eficiência do nosso algoritmo. A Tabela 5.2 apresenta os detalhes sobre as medidas do espaço (metragem) onde o robô *Turtlebot 2* trafegou durante a realização de cada teste.

Tabela 5.1: Requisitos - especificações técnicas

Hardware	***Detalhes do Hard.***	***Software***	***Detalhes de Soft.***
Kinect	30 frames por segundo; Resolução máxima: 1280 x 960; Sensores: CMOS e Infravermelho	Sistema Operacional Linux	Distribuição Ubuntu 14.04 LTS (Trusty)
Robô Turtlebot 2	Base Kobuki; Netbook (Compatível com ROS); Sensor Kinect; Estrutura do TurtleBot; Placa do módulo TurtleBot com padrão de furo de espaçamento de 1 polegada	Framework ROS (Robot Operating System)	Versão ROS: Indigo LTS
Notebook HP Pavilion G4	Modelo TPN-Q109. Configuração: Core i5-3210M (3ª geração) 2.50 GHz; Memória RAM: 6 GB DDR3 SDRAM; Placa de Video: Intel Graphics Media Accelerator 4500MHD; Disco rígido (HD): 512 GB 5400 RPM	Biblioteca PCL (Point Cloud Library)	Versão PCL: 1.7.0
-	-	Biblioteca Freenect	Versão Freenect: 1.7.0

Tabela 5.2: Experimentos realizados.

Experimentos	Desafio	Alvo	Passagens estreitas
1	evitando obstáculos	4.0 m	-
2	passando entre barreiras	3.5 m	80 cm (largura)
3	passando debaixo da mesa	4.0 m	73 cm (altura)
4	contornando a mesa (baixa)	4.0 m	40 cm (altura)

A Figura 5.1 apresenta cenas do vídeo sobre a execução dos 4 tipos de experimentos. Todos os testes foram feitos num ambiente *indoor*, dentro do Laboratório de Sistemas Embarcados e Robótica (LASER). Foram utilizadas a sala dos mestrandos, o *lounge* do piso superior, além da bancada de experimentos que fica no centro do prédio.

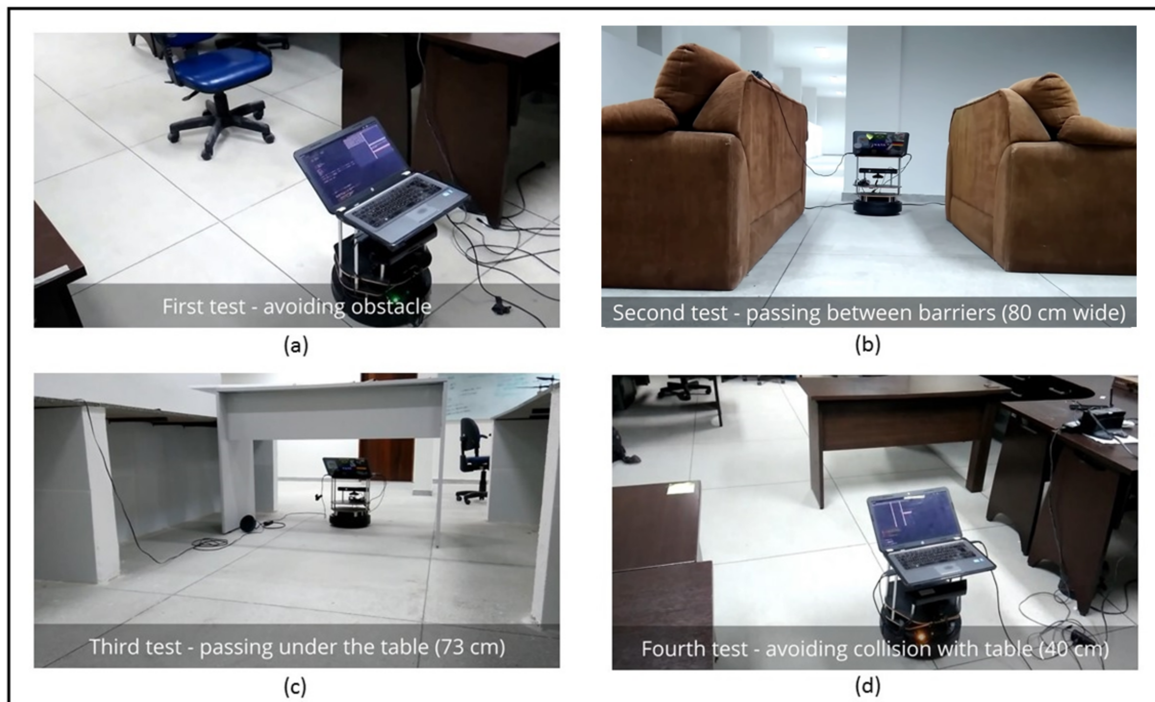


Figura 5.1: Vídeo apresentando os 4 tipos de experimentos realizados.

No nosso código, o filtro *voxel-grid* considera apenas os pontos com, no mínimo, 1 cm de distância (euclideana) entre si. Em seguida, os filtros *passthrough* reduzem a área para $1.8m$ em cada direção de coordenadas do Kinect, $\hat{x}$, $-0.1m$ para $2.5m$ em $\hat{y}$ e $0.0m$ para $6.0m$ em $\hat{z}$, . Para construir uma Kd-tree, com o método de extração de clusters da PCL, a distância euclideana foi definida como 2 cm.

Para melhor entendimento de cada etapa do sistema, tomemos como exemplo o primeiro experimento da tabela 5.2 como base.

A imagem RGB, ou seja a imagem colorida da câmera VGA que é embarcada no Kinect, mostra o que o robô consegue ver da cena (ver Figura 5.2). Simultaneamente, o Kinect obtém a imagem de profundidade que é transformada em nuvem de pontos automaticamente pela PCL, o resultado é apresentado na Figura 5.3. Em seguida aplicamos os filtros *Passthrough* para os eixos 'x' e 'y' (Figura 5.2).

Figura 5.2: Imagem RGB do primeiro experimento.

A mesma imagem filtrada pelo algoritmo 'VoxelGrid' , que é utilizado para fazer downsampling (diminuir o número de pontos da nuvem) pode ser vista na Figura 5.5.

O resultado final, quando aplicamos tanto os filtros 'Passthrough' quanto o 'VoxelGrid', pode ser visto na Figura 5.6. Esse é a nuvem de pontos filtrada que posteriormente é usada como entrada para o algoritmo de clusterização baseado no método de *K-NN* [63].

Após uma bateria de testes, foram encontrados os melhores parâmetros para superar os desafios propostos pelos experimentos, especificados a seguir:

- $\Lambda = 2.7$
- $\lambda = 0.4$

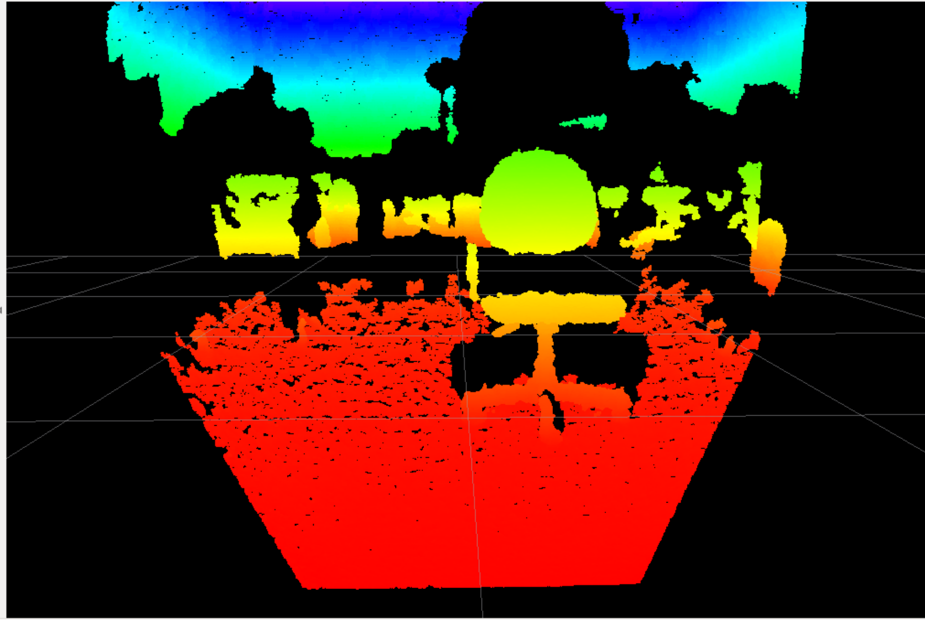


Figura 5.3: Nuvem de pontos da PCL. A profundidade é representada pelo um range de cores, quanto mais próximo o objeto do sensor, mais quente são as cores na imagem. As cores variam entre azul marinho, que define os obstáculos mais distantes, e vermelho, que representa os obstáculos mais próximos. A região em preto é uma área desconhecida, ou seja, não há dados sobre ela.

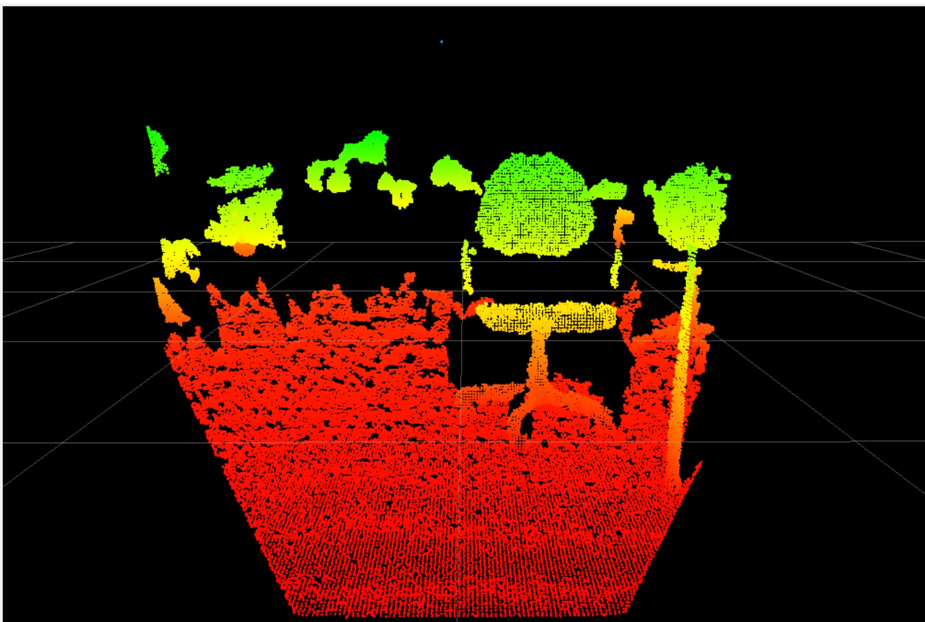


Figura 5.4: Filtro *passThrough* aplicado aos eixos 'x' e 'y' da nuvem de pontos.

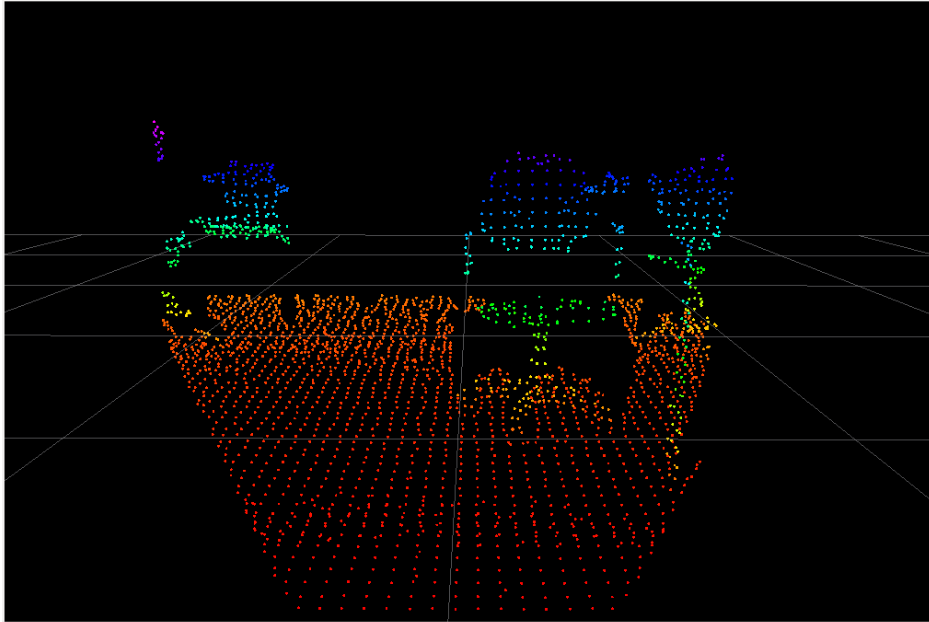


Figura 5.5: Filtro VoxelGrid aplicado ao eixo 'z' do nuvem de pontos.

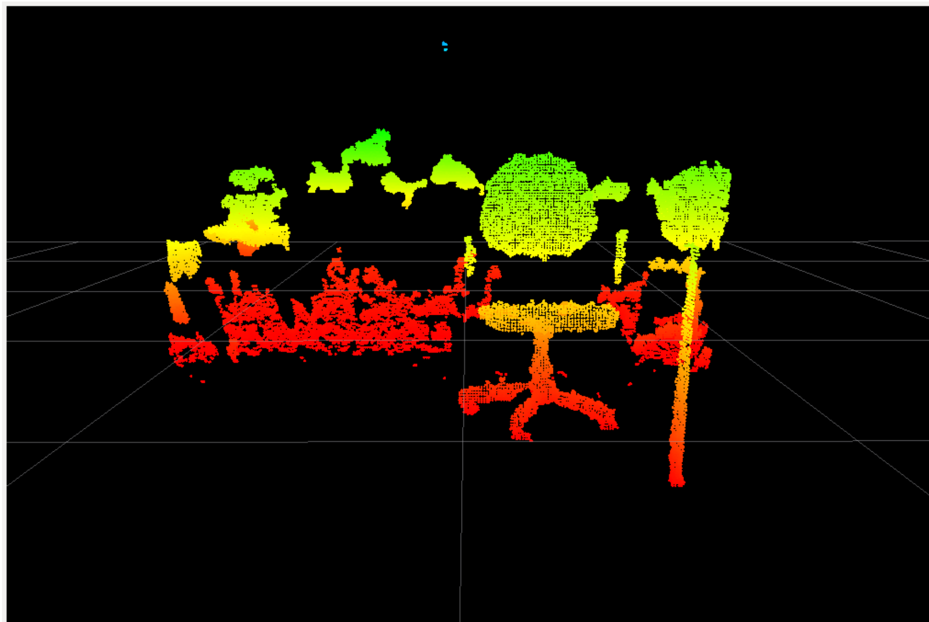


Figura 5.6: Resultado final após filtragem com Passthrough e Voxelgrid.

- $D_{max} = 1.5$
- Velocidade linear: min = 0.0, max = 0.2
- Velocidade angular: min = 0.0, max = 0.2

O alvo, o qual infere uma força atrativa sobre o 'Turtlebot 2', nos experimentos 1, 3 e

4, foi definido com uma distância em linha reta de $4m$, e para o experimento 2, o alvo foi definido como $3.5m$ por causa das restrições do espaço físico. A distância em relação ao alvo era atualizada frequentemente através dos dados advindos dos odômetros (acoplados as rodas) do robô e sua bússola. Esse dados permitiram localizar o 'Turtlebot 2' retornando sua pose (posição e orientação).

Com os parâmetros descritos acima, o sistema foi capaz de evitar a colisão com sucesso de obstáculos encontrados na cena, além de ser capaz de movimentar-se por entre esses obstáculos, em caminhos estreitos, ou por baixo de obstáculos "suspensos"(como uma mesa) com altura maior do que a plataforma robótica, sem que houvesse quaisquer oscilações que inferissem um campo potencial repulsivo à ponto de atrapalhar sua passagem segura rumo ao destino do alvo definido com base na odometria mecânica.

Um vídeo demonstrando os resultados pode ser encontrado no youtube, intitulado "Artificial Potential Field Algorithm", através do link `https://www.youtube.com/watch?edit=vd&v=mkdBEWM07YI`. Esse vídeo mostra um teste de cada experimento, o qual demonstra os resultados positivos obtidos usando o algoritmo proposto.

O código implementado desse projeto pode ser visto no apêndice A desse documento.

5.2.2 Análise do Experimento

Nesta sessão fazemos uma análise sobre os desafios propostos, o comportamento do robô e os resultados obtidos. As figuras de análises das trajetórias foram geradas com base nos logs de cada um desses testes. Os logs estão disponíveis no Apêndice B desse documento. Para mais detalhes sobre os experimentos realizados, consulte a tabela 5.2.

O sistema funcionou de forma satisfatória diante dos desafios propostos neste trabalho, para os quatro diferentes tipos de experimentos. Contudo, o sistema encontrou dificuldade em encontrar um caminho de escape que pudesse evitar o perigo eminente de colisão em casos em que o obstáculo se encontrava muito próximo ao robô, por conta das limitações do sensor RGB-D Kinect.

Um outro problema enfrentado durante a realização dos experimentos foi o deslizamento do robô por conta do piso de granizo do laboratório, pois o 'Turtlebot 2' não foi desenvolvido para ser utilizado sobre esse tipo de piso. Além disso, o robô apresenta alguns problemas mecânicos, por conta do longo tempo de uso, o que também influencia na forma como o

mesmo consegue se mover.

O primeiro experimento "desvio de obstáculo (cadeira)"consta no desafio de fazer o robô seguir até o alvo, 4m à sua frente, sem colidir com quaisquer obstáculos, incluindo a cadeira que obstruía seu caminho, além de testar o comportamento do robô, para qual lado o robô iria virar quando precisasse desviar da cadeira. A Figura 5.7 mostra o cenário do experimento proposto e a Figura 5.8 apresenta o trajeto percorrido pelo robô nesse experimento.

Figura 5.7: Foto do primeiro experimento: desvio de obstáculo.

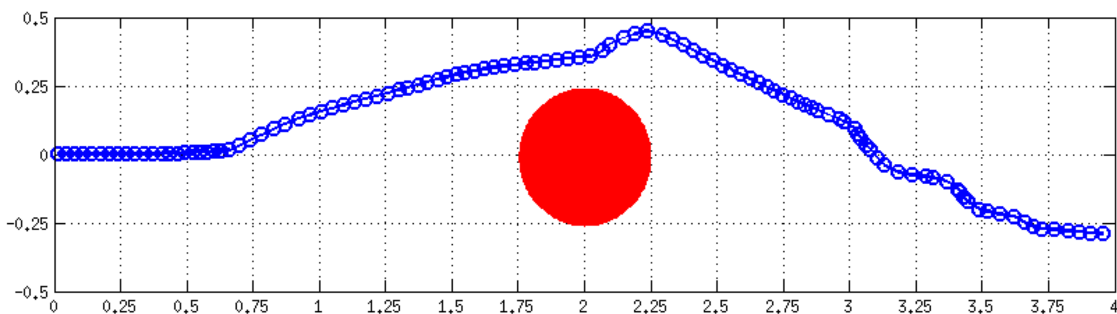


Figura 5.8: Análise do caminho percorrido pelo robô no primeiro experimento.

O experimento 2 "deslocamento do robô em passagem estreita"trata-se de fazer com que

o robô consiga trafegar por passagens estreitas, ou seja, com barreiras laterais próximas a ele, como mostra a Figura 5.9. A distância entre os sofás é de exatamente 80 cm e o alvo se encontra à uma distância de 3.5 m.

A mudança da distância do alvo setada para o experimento 2 (em comparação com os outros experimentos realizados) se dá pelas restrições do espaço físico onde tínhamos os sofás dispostos.

Figura 5.9: Foto do segundo experimento: trafegando em passagem estreita.

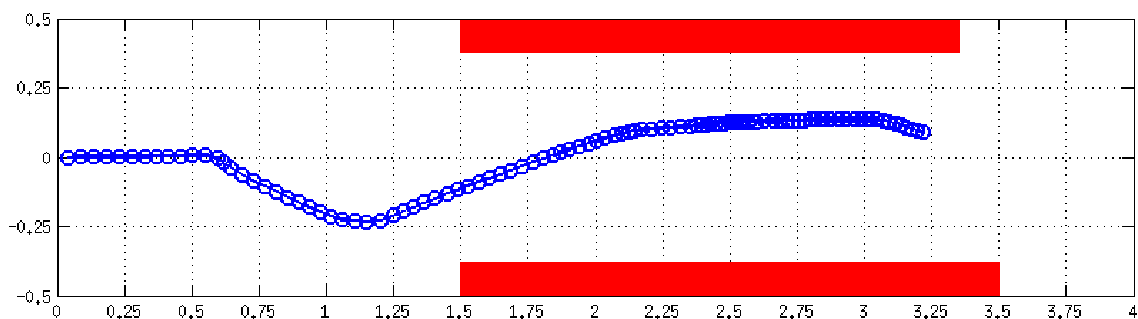


Figura 5.10: Análise do caminho percorrido pelo robô no segundo experimento.

O experimento 3 "passando debaixo da mesa"testa a percepção do robô, de que o obstáculo está acima dele e não oferece perigo de colisão. A área livre por debaixo da mesa é de

73 cm. A Figura 5.11 mostra o cenário da área de testes, e a Figura 5.12 apresenta a trajetória do robô.

Figura 5.11: Foto do terceiro experimento: passando debaixo da mesa.

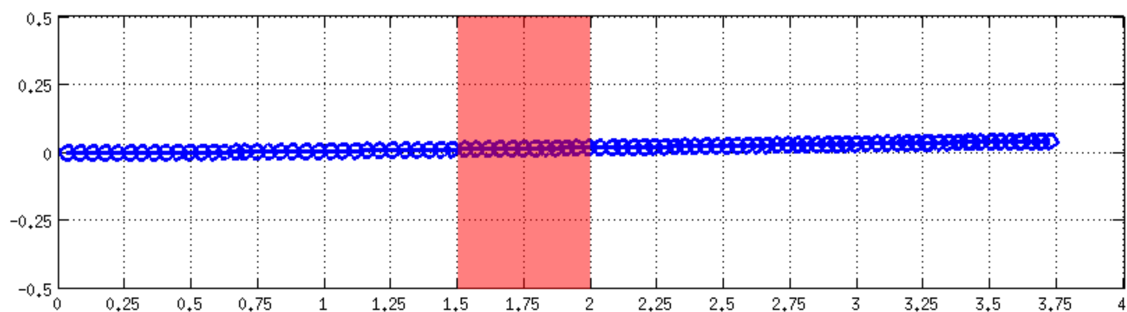


Figura 5.12: Análise do caminho percorrido pelo robô no terceiro experimento.

Um problema percebido durante a execução de vários testes do tipo experimento 4, foi que o sistema de visão precisa identificar um número razoável de pontos da parte superior da mesa (acima do robô) para que a força repulsiva resultante seja suficiente para afastar o robô do obstáculo e evitar a colisão. Isso significa que mesas que a parte horizontal (superior) seja de largura pequena, o sistema não terá êxito. No entanto, para as mesas dispostas no ambiente onde o robô se encontra, o algoritmo teve um desempenho satisfatório, já que a parte inferior da mesa tem uma largura e altura consideráveis, o que infere num maior

número de pontos, aumentando a força de repulsão e fazendo com que o robô não colida.

Também percebemos que o cálculo da força repulsiva leva mais tempo para ser feito quando se trata de obstáculos largos, como as mesas do tipo da Figura 5.13. A mesa tem 1.5m de largura e 1.2m de altura. Por questão de segurança, diminuímos a velocidade máxima de 0.2 para 0.1, o que possibilitou o sistema de visão (node "manhattan") ter tempo suficiente para construir a árvore Kd-tree necessária para depois construir o cluster que, por último, é estimado a força repulsiva.

Figura 5.13: Foto do quarto experimento: contorno de mesa cuja a altura (baixa) impede a passagem do robô

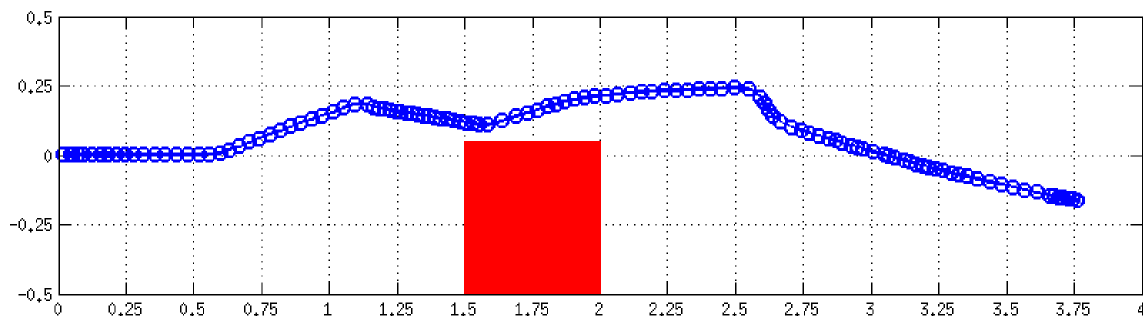


Figura 5.14: Análise do caminho percorrido pelo robô no quarto experimento.

A confiança sobre os dados obtidos pelo sensor Kinect limita-se a um intervalo de distância maior igual a 0,4 m e menor igual a 3,5 m. Os detalhes sobre as restrições do sensor de profundidade do Kinect podem ser encontrados em [50]. Tais restrições tornam-se um

problema na prática quando o robô atua em ambientes *indoor*, principalmente em ambientes desorganizados (repletos de obstáculos). Uma possível solução para este problema, seria adicionar novos sensores, como ultrassons, à plataforma 'Turltebot 2'.

5.3 Considerações Finais

Este capítulo apresentou os experimentos e resultados obtidos através de testes realizados no âmbito do Laboratório de Sistemas Embarcados e Robótica (LASER) da Universidade Federal da Paraíba. Imagens dos testes foram apresentadas, além de gráficos da trajetória do robô para cada experimento com base nos dados do log do robô. Além de abordarmos os principais problemas encontrados na aplicação que será propostos como trabalhos futuros.

Capítulo 6

Conclusão

Como mostra os resultados obtidos através dos experimentos realizados, nosso método de campo potencial artificial modificado consegue superar todos os três desafios propostos. O Turtlebot foi capaz de evitar prováveis colisões com obstáculos estáticos num ambiente controlado, um laboratório de informática, repleto de móveis, também conseguiu encontrar passagens estreitas entre obstáculos que estavam próximos um do outro, além de distinguir "obstáculos suspensos"entre os que poderiam impedir o robô de trafegar, como a parte superior de uma mesa baixa por exemplo, ou o contrário - uma mesa alta o suficiente, permitindo que o robô fizesse uma passagem segura por baixo da mesma. Todos os movimentos realizados de forma suave e livre de oscilações.

Acreditamos que, a partir desse trabalho, há um vasto número de possíveis melhorias que podem ser feitas sobre a nossa técnica, especialmente se o propósito for aplicar essa solução à ambientes dinâmicos e complexos (com tráfego de pessoas). Nesse contexto, acreditamos que sejam necessário melhorias quanto a performance da solução aqui proposta.

Como trabalhos futuros, sugerimos a adoção de outros sensores que trabalhem com uma distância menor que 80 cm, como ultrassons, por exemplo, para depois fundir os dados desses sensores com os dados obtidos pelo Kinect. Esperamos que isso contribua para aumentar a eficiência do algoritmo e diminua o tempo de resposta, já que a aplicação reduziria a dependência dos dados oriundos do sensor Kinect. Além de implementar um sistema de odometria visual que trabalharia em conjunto com a odometria mecânica para amenizar as incertezas quanto a localização do alvo.

Referências Bibliográficas

[1] Armin Hornung, Mike Phillips, Edward Gil Jones, Maren Bennewitz, Maxim Likhachev, and Sachin Chitta. Navigation in three-dimensional cluttered environments for mobile manipulation. In *ICRA*, pages 423–429. IEEE, 2012.

[2] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Robotics and Automation. Proceedings. 1985 IEEE International Conference on*, volume 2, pages 500–505, Mar 1985.

[3] T. Khuswendi, H. Hindersah, and W. Adiprawita. Uav path planning using potential field and modified receding horizon a* 3d algorithm. In *Electrical Engineering and Informatics (ICEEI), 2011 International Conference on*, pages 1–6, July 2011.

[4] C. Bentes and O. Saotome. Dynamic swarm formation with potential fields and a* path planning in 3d environment. In *Robotics Symposium and Latin American Robotics Symposium (SBR-LARS), 2012 Brazilian*, pages 74–78, Oct 2012.

[5] I.A. Hameed, A. la Cour-Harbo, and O.L. Osen. Side-to-side 3d coverage path planning approach for agricultural robots to minimize skip/overlap areas between swaths. *Robotics and Autonomous Systems*, 76:36 – 45, 2016.

[6] Anderson Abner de S. Souza and Luiz M. Garcia Gonçalves. Mapeamento com sonar usando grade de ocupação baseado em modelagem probabilística, 2008.

[7] R.D. Fonnegra Tarazona, F. Rios Lopera, and G.-D. Goez Sanchez. Anti-collision system for navigation inside an uav using fuzzy controllers and range sensors. In *Image, Signal Processing and Artificial Vision (STSIVA), 2014 XIX Symposium on*, pages 1–5, Sept 2014.

[8] Lim-Kwan Kong, Jie Sheng, and A. Teredesai. Basic micro-aerial vehicles (mavs) obstacles avoidance using monocular computer vision. In *Control Automation Robotics Vision (ICARCV), 2014 13th International Conference on*, pages 1051–1056, Dec 2014.

[9] M.C.P. Santos, L.V. Santana, A.S. Brandao, and M. Sarcinelli-Filho. Uav obstacle avoidance using rgb-d system. In *Unmanned Aircraft Systems (ICUAS), 2015 International Conference on*, pages 312–319, June 2015.

[10] T. T. Mac, C. Copot, A. Hernandez, and R. De Keyser. Improved potential field method for unknown obstacle avoidance using uav in indoor environment. In *2016 IEEE 14th International Symposium on Applied Machine Intelligence and Informatics (SAMI)*, pages 345–350, Jan 2016.

[11] D. Scaramuzza and F. Fraundorfer. Visual odometry [tutorial]. *Robotics Automation Magazine, IEEE*, 18(4):80–92, Dec 2011.

[12] J. Borenstein and Y. Koren. Obstacle avoidance with ultrasonic sensors. *Robotics and Automation, IEEE Journal of*, 4(2):213–218, Apr 1988.

[13] A. Elfes. Sonar-based real-world mapping and navigation. *Robotics and Automation, IEEE Journal of*, 3(3):249–265, June 1987.

[14] Y.K. Hwang and N. Ahuja. A potential field approach to path planning. *Robotics and Automation, IEEE Transactions on*, 8(1):23–32, Feb 1992.

[15] Fusheng Tan, Jun Yang, Jianming Huang, Tinggang Jia, Weidong Chen, and Jingchuan Wang. A navigation system for family indoor monitor mobile robot. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 5978–5983, Oct 2010.

[16] Tiago P. Nascimento, António Paulo Moreira, and André G. Scolari Conceição. Multi-robot nonlinear model predictive formation control: Moving target and target absence. *Robotics and Autonomous Systems*, 61(12):1502 – 1515, 2013.

[17] Lihua Zhu, Xianghong Cheng, and Fuh-Gwo Yuan. A 3d collision avoidance strategy for {UAV} with physical constraints. *Measurement*, 77:40 – 49, 2016.

[18] A.A.S. Souza and R.S. Maia. Occupancy-elevation grid mapping from stereo vision. In *Robotics Symposium and Competition (LARS/LARC), 2013 Latin American*, pages 49–54, Oct 2013.

[19] J. Yao, C. Lin, X. Xie, A. J. Wang, and C. C. Hung. Path planning for virtual human motion using improved a* star algorithm. In *2010 Seventh International Conference on Information Technology: New Generations*, pages 1154–1158, April 2010.

[20] L. Cheng, C. Liu, and B. Yan. Improved hierarchical a-star algorithm for optimal parking path planning of the large parking lot. In *2014 IEEE International Conference on Information and Automation (ICIA)*, pages 695–698, July 2014.

[21] F. H. Tseng, T. T. Liang, C. H. Lee, L. D. Chou, and H. C. Chao. A star search algorithm for civil uav path planning with 3g communication. In *2014 Tenth International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, pages 942–945, Aug 2014.

[22] C. Wang, L. Wang, J. Qin, Z. Wu, L. Duan, Z. Li, M. Cao, X. Ou, X. Su, W. Li, Z. Lu, M. Li, Y. Wang, J. Long, M. Huang, Y. Li, and Q. Wang. Path planning of automated guided vehicles based on improved a-star algorithm. In *2015 IEEE International Conference on Information and Automation*, pages 2071–2076, Aug 2015.

[23] Bruno Vilhena Adorno and Geovany AraUjo Broges. Planejamento de rotas. In Roseli Aparecida F. Romero, Edson Prestes, Fernando Osório, and Denis Wolf, editors, *Robótica Móvel*, chapter 6, pages 84–112. GEN LTC, Rio de Janeiro, 2014.

[24] Xiao Liang, Honglun Wang, Dawei Li, and Chang Liu. Three-dimensional path planning for unmanned aerial vehicles based on fluid flow. In *Aerospace Conference, 2014 IEEE*, pages 1–13, March 2014.

[25] Artificial potential field approach in robot motion planning. `https://www.youtube.com/watch?v=r9FD7P76zJs&t=41s`. acessado em 01/02/2017.

[26] A. A. Masoud. Decentralized self-organizing potential field-based control for individually motivated mobile agents in a cluttered environment: A vector-harmonic potential

field approach. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 37(3):372–390, May 2007.

[27] K. Motonaka, K. Watanabe, and S. Maeyama. 3-dimensional kinodynamic motion planning for an x4-flyer using 2-dimensional harmonic potential fields. In *2014 14th International Conference on Control, Automation and Systems (ICCAS 2014)*, pages 1181–1184, Oct 2014.

[28] Rudy Negenborn. Robot localization and kalman filters: On finding your positions in a noisy world, 2003.

[29] Contemp. `https://www.google.com.br/search?q=encoders&espv=2&source=lnms&tbm=isch&sa=X&ved=0ahUKEwjnpeyKlYLSAhVHHZAKHaFWDiYQ_AUICCgB&biw=1366&bih=613#imgrc=o0kOOXNYnZ9CtM:`. acessado em 02/02/2017.

[30] Grupo giga manutenção. `http://www.grupogigamanutencao.com.br/informacoes/manutencao-de-encoder`. acessado em 02/02/2017.

[31] Lentin Joseph. *Mastering ROS for Robotics Programming*. Packt Publishing Ltd., 35 Livery Street, Birmigham B3 2PB, UK, december 2015.

[32] Tobias Hammer and Berthold Bäuml. The communication layer of the ardx software framework: Highly performant and realtime deterministic. *Journal of Intelligent & Robotic Systems*, 77(1):171–185, 2015.

[33] Turtlebot usando ros. `http://www.ros.org/is-ros-for-me/`. acessado em 20/08/2016.

[34] Ros user location. `https://www.google.com/maps/d/viewer?mid=1NgE8o76SI-zQaMmu5L21uxghV_Y&hl=en_US&ll=32.985370012811224%2C25.674343102924354&z=2`. acessado em 30/01/2017.

[35] Ros user location. `http://wiki.ros.org/Robots`. acessado em 30/01/2017.

[36] Open source initiative. `https://opensource.org/licenses/BSD-3-Clause`. acessado em 30/01/2017.

[37] Ros. `http://www.ros.org/`. acessado em 20/08/2016.

[38] Tully Foote. tf: The transform library. In *Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on*, Open-Source Software workshop, pages 1–6, April 2013.

[39] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, July 1968.

[40] X. Chen and J. Zhang. The three-dimension path planning of uav based on improved artificial potential field in dynamic environment. In *Intelligent Human-Machine Systems and Cybernetics (IHMSC), 2013 5th International Conference on*, volume 2, pages 144–147, Aug 2013.

[41] A. S. Kundu, O. Mazumder, R. Chattaraj, and S. Bhaumik. Close loop control of non-holonomic wmr with augmented reality and potential field. In *Engineering and Computational Sciences (RAECS), 2014 Recent Advances in*, pages 1–5, March 2014.

[42] N. Sariff and N. Buniyamin. An overview of autonomous mobile robot path planning algorithms. In *2006 4th Student Conference on Research and Development*, pages 183–188, June 2006.

[43] S. Scheggi and S. Misra. An experimental comparison of path planning techniques applied to micro-sized magnetic agents. In *2016 International Conference on Manipulation, Automation and Robotics at Small Scales (MARSS)*, pages 1–6, July 2016.

[44] How to use a kdtree to search. `http://pointclouds.org/documentation/tutorials/kdtree_search.php`. acessado em 22/06/2016.

[45] Dirk Holz, Stefan Holzer, Radu Bogdan Rusu, and Sven Behnke. *Real-Time Plane Segmentation Using RGB-D Cameras*, pages 306–317. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[46] Bruno Marques Ferreira da Silva and Luiz Marcos Garcia Goncalves. A fast feature tracking algorithm for visual odometry and mapping based on rgb-d sensors. *2014 27th*

SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI), 00:227–234, 2014.

[47] A. L. F. Castro, Y. B. d. Brito, L. A. V. d. Souto, and T. P. Nascimento. A novel approach for natural landmarks identification using rgb-d sensors. In *2016 International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 193–198, May 2016.

[48] Allison Janoch, Sergey Karayev, Yangqing Jia, Jonathan T. Barron, Mario Fritz, Kate Saenko, and Trevor Darrell. *A Category-Level 3D Object Dataset: Putting the Kinect to Work*, pages 141–165. Springer London, London, 2013.

[49] Kinect. `https://www.ifixit.com/Teardown/Microsoft+Kinect+Teardown/4066`. acessado em 01/02/2017.

[50] Coordinate spaces. `https://msdn.microsoft.com/en-us/library/hh973078.aspx#Depth_Ranges`. acessado em 01/02/2017.

[51] Pcl. `http://pointclouds.org/`. acessado em 20/08/2016.

[52] Pcl. `http://pointclouds.org/documentation/`. acessado em 20/08/2016.

[53] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[54] M. Junemann. Object detection and recognition with microsoft kinect. Freie Universitat, Berlin, Germany, 2012.

[55] Passthrough. `http://pointclouds.org/documentation/tutorials/passthrough.php`. acessado em 01/02/2017.

[56] Voxelgrid. `http://pointclouds.org/documentation/tutorials/voxel_grid.php`. acessado em 01/02/2017.

[57] Plane model segmentation. `http://www.pointclouds.org/documentation/tutorials/planar_segmentation.php`. acessado em 22/06/2016.

[58] Peter J. Rousseeuw. Least median of squares regression. *Journal of the American Statistical Association*, 79(388):871–880, 1984.

[59] David C Hoaglin, Frederick Mosteller, and John Wilder Tukey. *Understanding robust and exploratory data analysis*, volume 3. Wiley New York, 1983.

[60] Ransac. `http://pointclouds.org/documentation/tutorials/random_sample_consensus.php`. acessado em 07/02/2017.

[61] André Luiz Figueiredo de Castro. Sistema para reconhecimento de múltiplos objetos 3d. João Pessoa, Paraíba, Brasil, 2015.

[62] Kdtree. `https://en.wikipedia.org/wiki/K-d_tree#/media/File:3dtree.png`. acessado em 22/06/2016.

[63] Euclidean cluster extraction. `http://www.pointclouds.org/documentation/tutorials/cluster_extraction.php`. acessado em 22/06/2016.

[64] freenect launch. `http://wiki.ros.org/freenect_launch`. acessado em 01/02/2017.

Apêndice A

Código

A seguir são apresentados os dois principais nodes (ROS) desse trabalho. O primeiro node (*manhattan.cpp*), o de visão, é responsável pela obtenção, filtragem e clusterização da nuvem de pontos de entrada, além de calcular a distância de cada ponto do cluster resultante para o Kinect. Em seguida, é calculada a força repulsiva sobre cada um desses pontos - detecção de obstáculos.

O segundo node (*moving.py*) é responsável pelo rastreamento e deslocamento do robô - obtendo dados dos odômetros ao longo do tempo e enviando comandos aos atuadores, respectivamente. Esse é o node que define o alvo (qual o destino final, em linha reta).

Código Fonte A.1: node de visão - manhattan.cpp

```
1  #include <ros/ros.h>
2  #include <nodelet/nodelet.h>
3  #include <sensor_msgs/PointCloud2.h>
4  #include <pcl_conversions/pcl_conversions.h>
5  #include <pcl/point_cloud.h>
6  #include <pcl/point_types.h>
7  #include <boost/foreach.hpp>
8  #include <sstream>
9  #include "novinho_laser/Vai.h"
10 #include <pcl/filters/filter.h>
11 #include <pcl/ModelCoefficients.h>
12 #include <pcl/point_types.h>
13 #include <pcl/io/pcd_io.h>
14 #include <pcl/filters/extract_indices.h>
15 #include <pcl/filters/voxel_grid.h>
16 #include <pcl/features/normal_3d.h>
17 #include <pcl/kdtree/kdtree.h>
18 #include <pcl/sample_consensus/method_types.h>
```

```
19 #include <pcl/sample_consensus/model_types.h>
20 #include <pcl/segmentation/sac_segmentation.h>
21 #include <pcl/segmentation/extract_clusters.h>
22 #include <cmath>
23 #include <pcl/filters/passthrough.h>
24
25 ros::Publisher pub;
26
27 using namespace pcl;
28 using namespace std;
29
30 typedef pcl::PointXYZ PointT;
31 typedef pcl::PointCloud<PointT> PointCloudT;
32
33 #define PI 3.14159265
34
35 void cloud_cb ( const sensor_msgs::PointCloud2 input )
36 {
37
38   pcl::PCLPointCloud2 pcl_pc;
39   pcl_conversions::toPCL ( input, pcl_pc );
40   pcl::PointCloud<pcl::PointXYZ>::Ptr cloud_f ( new pcl::PointCloud<pcl::PointXYZ> );
41   pcl::fromPCLPointCloud2 ( pcl_pc, *cloud_f );
42   pcl::PointCloud<pcl::PointXYZ>::Ptr cloud_cluster ( new pcl::PointCloud<pcl::PointXYZ> );
43   pcl::PointCloud<pcl::PointXYZ>::Ptr radius ( new pcl::PointCloud<pcl::PointXYZ> );
44
45   //!< Create the filtering object: downsample the dataset using a leaf size of 1cm
46   pcl::VoxelGrid<pcl::PointXYZ> vg;
47   pcl::PointCloud<pcl::PointXYZ>::Ptr cloud_filtered ( new pcl::PointCloud<pcl::PointXYZ> )
       ;
48   vg.setInputCloud ( cloud_f );   // *cloud_f
49   vg.setLeafSize   ( 0.01f,
50                      0.01f,
51                      0.01f );
52   vg.filter ( *cloud_filtered );
53
54   // Passthrough
55   pcl::PassThrough<pcl::PointXYZ> passz;
56   passz.setInputCloud (cloud_filtered);
57   passz.setFilterFieldName ("z");
58   passz.setFilterLimits (0.0, 6.0);
59   passz.filter (*cloud_filtered);
60
61   pcl::PassThrough<pcl::PointXYZ> passx;
62   passx.setInputCloud (cloud_filtered);
63   passx.setFilterFieldName ("x");
64   passx.setFilterLimits (-1.8, 1.8);
```

```
65    passx.filter (*cloud_filtered);
66
67    pcl::PassThrough<pcl::PointXYZ> passy;
68    passy.setInputCloud (cloud_filtered);
69    passy.setFilterFieldName ("y");
70    passy.setFilterLimits (-2.5, 0.2);
71    passy.filter (*cloud_filtered);
72
73    //!< Create the segmentation object for the planar model and set all the parameters
74    pcl::SACSegmentation<pcl::PointXYZ> seg;
75    pcl::PointIndices::Ptr inliers ( new pcl::PointIndices );
76    pcl::ModelCoefficients::Ptr coefficients ( new pcl::ModelCoefficients );
77    pcl::PointCloud<pcl::PointXYZ>::Ptr cloud_plane ( new pcl::PointCloud<pcl::PointXYZ> ( )
        );
78
79    seg.setOptimizeCoefficients ( true );
80    seg.setModelType  ( pcl::SACMODEL_PLANE );
81    seg.setMethodType ( pcl::SAC_RANSAC     );
82    seg.setMaxIterations ( 100 );
83    seg.setDistanceThreshold ( 0.02 );
84
85    int i = 0, nr_points = ( int ) cloud_filtered->points.size ( );
86    while ( cloud_filtered->points.size ( ) > 0.3 * nr_points )
87    {
88      //!< Segment the largest planar component from the remaining cloud
89      seg.setInputCloud ( cloud_filtered );
90      seg.segment ( *inliers, *coefficients );
91      if (inliers->indices.size ( ) == 0 ) break;
92
93      //!< Extract the planar inliers from the input cloud
94      pcl::ExtractIndices<pcl::PointXYZ> extract;
95      extract.setInputCloud ( cloud_filtered );
96      extract.setIndices    ( inliers );
97      extract.setNegative   ( false );
98
99      //!< Get the points associated with the planar surface
100     extract.filter ( *cloud_plane );
101
102     //!< Remove the planar inliers, extract the rest
103     extract.setNegative ( true );
104     extract.filter ( *cloud_f  );
105     *cloud_filtered = *cloud_f;
106   }
107
108   //!< Creating the KdTree object for the search method of the extraction
109   pcl::search::KdTree<pcl::PointXYZ>::Ptr tree ( new pcl::search::KdTree<pcl::PointXYZ> );
110   tree->setInputCloud ( cloud_filtered );
```

```
111
112     std::vector<pcl::PointIndices> cluster_indices;
113
114     pcl::EuclideanClusterExtraction<pcl::PointXYZ> ec;
115     ec.setClusterTolerance ( 0.01  ); // 2cm
116     ec.setMinClusterSize   ( 100   );
117     ec.setMaxClusterSize   ( 25000 );
118     ec.setSearchMethod ( tree );
119     ec.setInputCloud ( cloud_filtered );
120     ec.extract ( cluster_indices );
121
122     int cont = 0, cont2 = 0, j = 0;
123     float x = 0.0, y = 0.0, z = 0.0, position_x = 0.0, position_y = 0.0, position_z = 0.0,
            angulo_obst = 0.0, dezero = 1.5,
124           manhattan = 0.0, lambda = 2.7, Fobs = 0.0, Fxobs = 0.0, Fyobs = 0.0,  FrepTotal =
                0.0, AnguloRepulsiva = 0.0;
125     unsigned int n = 0;
126
127     for ( std::vector<pcl::PointIndices>::const_iterator it = cluster_indices.begin ( );
128           it != cluster_indices.end ( );
129           ++it )
130     {
131
132       for ( std::vector<int>::const_iterator pit = it->indices.begin ( );
133             pit != it->indices.end ( );
134             ++pit ){
135
136         //remove NAN points from the cloud
137         std::vector<int> indices;
138         pcl::removeNaNFromPointCloud(*cloud_cluster,*cloud_cluster, indices);
139
140         cloud_cluster->points.push_back ( cloud_filtered->points [ *pit ] );
141         cloud_cluster->width = cloud_cluster->points.size ( );
142         cloud_cluster->height = 1;
143         cloud_cluster->is_dense = true;
144
145         j++;
146       }
147
148       for (size_t ifdp = 0; ifdp < it->indices.size(); ifdp++){
149         if ( (!std::isnan ( cloud_cluster->points[ifdp].x )) && (!std::isnan ( cloud_cluster
              ->points[ifdp].y )) && (!std::isnan ( cloud_cluster->points[ifdp].z )) ) {
150           if (cloud_cluster->points[ifdp].z <= dezero){
151             x = cloud_cluster->points[ifdp].x;
152             y = cloud_cluster->points[ifdp].y;
153             z = cloud_cluster->points[ifdp].z;
154             n++;
```

```
155         }
156       }
157
158       cout << "y = " << y << endl;
159
160       manhattan = abs( x − position_x ) + abs( y − position_y ) + abs( z − position_z );
161
162       if (manhattan <= dezero){
163             Fobs = (−lambda ∗ ( ( ( 1 − manhattan / dezero ) ) ) );
164             angulo_obst = atan2(−x , z);
165             Fxobs += − sin(angulo_obst) ∗ abs(Fobs);
166             Fyobs +=  cos(angulo_obst) ∗ abs(Fobs);
167       }
168     }
169   }
170
171   j = 0;
172
173
174   FrepTotal = sqrt(pow((Fxobs), 2) + pow((Fyobs), 2) );
175   AnguloRepulsiva = atan2(Fxobs , Fyobs);  // the angle in relation to Kinect sensor
176
177   novinho_laser::Vai distance;
178   distance.frep_total = FrepTotal;
179   distance.angulo_repulsiva = AnguloRepulsiva;
180   distance.obstacle = manhattan;
181   pub.publish ( distance );
182
183   sensor_msgs::PointCloud2 output;
184   pcl_conversions::moveFromPCL(pcl_pc, output);
185 }
186
187 int main ( int argc, char** argv )
188 {
189   ros::init ( argc, argv, "visao" );
190   ros::NodeHandle nh;
191   ros::Subscriber depth_sub = nh.subscribe<sensor_msgs::PointCloud2> ( "/camera/
        depth_registered/points", 1, cloud_cb );
192   pub = nh.advertise<novinho_laser::Vai> ( "distance", 1 );
193
194   ros::Rate loop_rate (5.0);
195   while ( true )
196   {
197     ros::spinOnce ( );
198     loop_rate.sleep ( );
199   }
200
```

```
201     ros::spin();
202 }
```

Código Fonte A.2: node de movimento - moving.py

```
1  #!/usr/bin/env python
2
3  # Data: 19.10.2016 as 13:42
4
5  import rospy
6  from roslib import message
7  from geometry_msgs.msg import Twist, Point, Quaternion
8  import tf
9  from rbx1_nav.transform_utils import quat_to_angle, normalize_angle
10 from math import radians, copysign, sqrt, pow, pi, atan, atan2, cos, sin
11 from novinho_laser.msg import Vai
12
13 class Moving():
14     global frep_total
15     global angulo_repulsiva
16     global obstacle
17     frep_total = 0.0
18     angulo_repulsiva = 0.0
19     obstacle = 0.0
20
21     def __init__(self):
22         rospy.init_node("moving", anonymous=False)
23
24         rospy.on_shutdown(self.shutdown)
25
26         # Observacao: trabalhando com coordenadas da camera (RGB-D)
27
28         self.cmd_vel = rospy.Publisher('/cmd_vel_mux/input/navi', Twist, queue_size=5)    #
               Publisher to control the robot's speed
29
30         self.tf_listener = tf.TransformListener()    # Inicializa o 'tf listener'
31         rospy.sleep(2)    # Give tf some time to fill its buffer
32         self.odom_frame = '/odom'    # Set the odom frame
33
34         # Find out if the robot uses /base_link or /base_footprint (The Turtlebot uses this
                last)
35         try:
36             self.tf_listener.waitForTransform(self.odom_frame, '/base_footprint', rospy.
                   Time(), rospy.Duration(1.0))
37             self.base_frame = '/base_footprint'
38         except (tf.Exception, tf.ConnectivityException, tf.LookupException):
39             try:
```

```
40                 self.tf_listener.waitForTransform(self.odom_frame, '/base_link', rospy.Time
                       (), rospy.Duration(1.0))
41                 self.base_frame = '/base_link'
42             except (tf.Exception, tf.ConnectivityException, tf.LookupException):
43                 rospy.loginfo("Cannot find transform between /odom and /base_link or /
                       base_footprint")
44                 rospy.signal_shutdown("tf Exception")
45
46         rospy.sleep(2)
47
48         rate = 50  # How fast will we update the robot's movement?
49         r = rospy.Rate(rate)  # Set the equivalent ROS rate variable
50         self.x_threshold = rospy.get_param("~x_threshold", 1.0)  # How far away from the
               goal distance (in meters) before the robot reacts
51
52         self.max_angular_speed = rospy.get_param("~max_angular_speed", 2.0)  # The maximum
                rotation speed in radians per second
53         self.min_angular_speed = rospy.get_param("~min_angular_speed", 0.0)  # The minimum
                rotation speed in radians per second
54         self.max_linear_speed = rospy.get_param("~max_linear_speed", 0.1)  # The max
               linear speed in meters per second
55         self.min_linear_speed = rospy.get_param("~min_linear_speed", 0.0)  # The minimum
               linear speed in meters per second
56
57         self.slow_alt_factor = rospy.get_param("~slow_alt_factor", 0.8)  # Slow alt factor
                when stopping
58
59
60         position = Point()  # Initialize the position variable as a Point type
61         move_cmd = Twist()  # Inicializa o comando de movimento
62
63         (position, rotation) = self.get_odom()  # pega posicao inicial
64         print "Posicao inicial: ", position
65         print "Rotacao inicial: ", rotation
66
67         x_goal = position.x + 4.0  # define a que distancia (em linha reta) esta o alvo (
               destino final)
68         y_goal = position.y
69
70         global frep_total
71         global angulo_repulsiva
72         global obstacle
73
74         constante = 1.0
75         Fatt = 0.0
76         frep_total = 0.0
77         angulo_alvo = 0.0
```

```
78          current = 0.0
79          parou = 0.2
80
81          inicial = sqrt ( pow ( ( position.x - x_goal), 2 ) + pow ( ( position.y - y_goal),
                2) )
82
83          current = sqrt ( pow ( ( position.x - x_goal), 2 ) + pow ( ( position.y - y_goal),
                2) )
84          print "current inicial: ", current   # = 5
85
86          vantangem = inicial + parou
87
88          while current > 0.05 and not rospy.is_shutdown():
89
90              rospy.Subscriber("distance", Vai, self.depth_cb, queue_size = 1)   # 'Vai' - eh
                    a msg ROS do node 'manhattan.cpp'
91              rospy.wait_for_message('distance', Vai)
92
93              print "##################################"
94              print "obstacle distance: ", obstacle
95              print "frep_total ", frep_total
96              print "angulo_repulsiva ", angulo_repulsiva
97
98              (position, rotation) = self.get_odom()
99              angulo_odom = rotation
100             print "angulo_odom (rotation) ", angulo_odom
101             angulo_final_obst = angulo_repulsiva + angulo_odom  # angulo verdadeiro
102             print "angulo_final_obst (angulo_repulsiva + rotation) ", angulo_final_obst
103
104             # AGORA, ALVO!!
105
106             euclidiana_alvo = float ( sqrt ( pow ( ( x_goal - position.x ), 2 ) + pow ( (
                    y_goal - position.y), 2) ) )
107             print "euclidiana_alvo ",euclidiana_alvo
108
109             Fatt = abs(constante * euclidiana_alvo)
110             print "Fatt ", Fatt
111             passo_y = position.y - y_goal
112             print "passo_y ", passo_y
113             passo_x = position.x - x_goal
114             print "passo_x ", passo_x
115             angulo_alvo = (atan ( passo_y / passo_x ) )
116             print "angulo_alvo ", angulo_alvo
117
118
119             Fxatt = abs(Fatt) * cos(angulo_alvo)
120             print "Fxatt ", Fxatt
```

```
121             Fyatt = abs(Fatt) * sin(angulo_alvo)
122             print "Fyatt ", Fyatt
123
124             Forca_resultante_x = abs(Fatt) * cos(angulo_alvo) + abs(frep_total) * cos(
                    angulo_final_obst)   # Frep = FrepTotal
125             print "Forca_resultante_x ", Forca_resultante_x
126             Forca_resultante_y = abs(Fatt) * sin(angulo_alvo) + abs(frep_total) * sin(
                    angulo_final_obst)
127             print "Forca_resultante_y ", Forca_resultante_y
128
129             angulo_total = (atan2(Forca_resultante_y , Forca_resultante_x)) - rotation #
                    passos invertidos
130             print "angulo_total ", angulo_total
131
132             if current < 1.0:
133                 vantangem = vantangem * 1.05
134             print "vantangem = ", vantangem
135
136             ForcaTotal = float ( sqrt ( sqrt( pow ( ( Forca_resultante_x), 2 ) + pow (  (
                    Forca_resultante_y), 2 ) ) *( vantangem ) ) )
137             print "ForcaTotal ", ForcaTotal
138
139             angular_speed = angulo_total   #* rate
140
141             # se obstaculo > 0.6:
142             linear_speed = (ForcaTotal * 0.1) - 0.15
143             print "linear_speed = ", linear_speed
144
145             #se obstaculo > 0.4:
146             move_cmd.linear.x = copysign(max(self.min_linear_speed, min(self.
                    max_linear_speed, abs(linear_speed))), linear_speed)
147             move_cmd.angular.z = copysign(max(self.min_angular_speed, min(self.
                    max_angular_speed, abs(angular_speed))), angular_speed)
148
149             print "Saida:   linear = " + str (move_cmd.linear.x) + " angular " + str(
                    move_cmd.angular.z)
150             (position, rotation) = self.get_odom()   # pega posicao inicial
151             print "position atual ", position
152             print "rotation atual", rotation
153
154             self.cmd_vel.publish(move_cmd)
155
156             (position, rotation) = self.get_odom()
157
158             current = sqrt(pow((position.x - x_goal), 2) + pow((position.y - y_goal), 2))
159             print " current ", current
160
```

```
161                 if current <= 0.3:
162                     print "PARANDO... "
163                     move_cmd.linear.x = 0.0
164                     move_cmd.angular.z = 0.0
165                     current = 0.0;
166
167                 move_cmd = Twist()
168
169         def depth_cb(self, distance):
170             global frep_total
171             global angulo_repulsiva
172             global obstacle
173             frep_total = distance.frep_total
174             angulo_repulsiva = distance.angulo_repulsiva
175             obstacle = distance.obstacle
176
177         def get_odom(self):
178             try:
179                 (trans, rot) = self.tf_listener.lookupTransform(self.odom_frame, self.
                        base_frame, rospy.Time(0))
180             except (tf.Exception, tf.ConnectivityException, tf.LookupException):
181                 rospy.loginfo("TF Exception")
182                 return
183
184             return (Point(*trans), quat_to_angle(Quaternion(*rot)))
185
186
187         def shutdown(self):
188             rospy.loginfo("Problema com o robo!!!")
189             self.cmd_vel.publish(Twist())
190             rospy.sleep(1)
191
192 if __name__ == '__main__':
193     try:
194         Moving()
195     except:
196         rospy.loginfo("Robot node terminated.")
```

Apêndice B

Log

A seguir, um exemplo de log (resumido), do experimento 1, ilustrando como os dados se comportam na prática.

```
x: 0.0, y: 0.0, theta: 0.0
x: 0.0, y: 0.0, theta: 0.0
x: 0.0142437766615, y: 4.24258847549e-06, theta: 0.0010471975512
x: 0.045119501511, y: 2.98544258688e-05, theta: 0.0012217304764
x: 0.0740761443526, y: 6.45096715296e-05, theta: 0.0013962634016
x: 0.105591523347, y: 0.000118770125207, theta: 0.00226892802759
x: 0.141072913437, y: 0.000210127107609, theta: 0.00296705972839
x: 0.175146913074, y: 0.000323441240319, theta: 0.00366519142919
...
x: 3.69320403639, y: -0.264074183433, theta: -0.301243828894
x: 3.72638475892, y: -0.271518969135, theta: -0.0776671517137
x: 3.77260180601, y: -0.275108740365, theta: -0.0776671517137
x: 3.82179156984, y: -0.278974974286, theta: -0.0790634151153
x: 3.86642990295, y: -0.282513263163, theta: -0.0792379480405
x: 3.9085613694, y: -0.285835241061, theta: -0.0783652834145
x: 3.95316046292, y: -0.289328623786, theta: -0.0778416846389
Finish.
```